Serie

Te acompaño a programar

Volumen IV

Programación para idear y forjar soluciones

Joe Llerena Izquierdo



Universidad Politécnica Salesiana

CARRERA DE COMPUTACIÓN

Aprender programación en ingeniería implica enfrentar diversos desafíos, que van desde los conocimientos técnicos previos hasta su comprensión y aplicación.

Este libro compendia los contenidos iniciales del primer año de la materia de programación, basados en más de veinte años de experiencia del autor y en su práctica docente en la Universidad Politécnica Salesiana (Sede Guayaquil). A través de un enfoque tecno-pedagógico innovador, promovido por la Carrera de Computación, la obra busca potenciar el aprendizaje significativo.

Se invita al lector a motivarse por el aprendizaje de programación mediante la creación de artefactos algorítmicos y el desarrollo de soluciones que contribuyan, en el futuro, al bienestar social.

A partir del dominio de los conocimientos esenciales, esta obra es un aporte a la formación de profesionales capaces de reducir desigualdades y generar nuevos conocimientos para un mundo más justo y equitativo.

ISBN-13: 978-9942520609
9 789942 520609

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR

SALESIANOS DON BOSCO ECUADOR

ABYA YALA

4 EDUCACIÓN DE CALIDAD

Programación
para idear y forjar soluciones

Serie

Te acompaño a programar

Volumen IV

Programación
para idear y forjar soluciones

Joe Llerena Izquierdo

ABYA YALA

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR

2026

PROGRAMACIÓN PARA IDEAR Y FORJAR SOLUCIONES
© Joe Llerena Izquierdo

1ra edición: Universidad Politécnica Salesiana
Av. Turuhuayco 3-69 y Calle Vieja
Cuenca-Ecuador
P.B.X. (+593 7) 2050000
e-mail: publicaciones@ups.edu.ec
www.ups.edu.ec

CARRERA DE COMPUTACIÓN

Diagramación: Latex de autor

ISBN
Obra completa: 978-9978-10-428-6

ISBN impreso
Volumen IV: 978-9942-52-060-9

ISBN digital
Volumen IV: 978-9942-52-061-6

Tiraje: 300 ejemplares

DOI: https://doi.org/10.17163/abyaups.167

Impreso en Quito-Ecuador, marzo 2026

Publicación arbitrada de la Universidad Politécnica Salesiana

El contenido de este libro es de exclusiva responsabilidad del autor.



Dedico este libro a mi familia,
a mi esposa Raquel Ayala Carabajo
lo esencial de mi vida,
a mis hijos Natalia y Anthony
a quienes cuido de sus sueños y su formación.
A mis estudiantes de la Universidad Politécnica Salesiana
por las horas sacrificadas que tienen de estudio y dedicación.
A la carrera de Computación por su apoyo incondicional
en el desarrollo de estos objetos de aprendizaje.
A mis jóvenes amigos lectores que no conozco ni conoceré,
pero sé que estarán leyendo este libro para aprender
de manera tan sencilla y amena.
Apreciados jóvenes, que cada palabra escrita
en este libro lo vuelvas a estudiar
y que el contenido de cada capítulo lo puedas aplicar.

"Haz prósperas, Señor, las obras de nuestras manos"
Salmo 89.

Joe Llerena.

Un especial agradecimiento,
a la Comunidad Salesiana en Ecuador
por la formación recibida,
en mi niñez, juventud y adultez,
de permitirme conocer la vida
de San Juan Bosco y madre María Mazzarello,
de recibir el acompañamiento de los Salesianos y Salesianas,
de tener admiración a Santo Domingo Savio y Beata Laura Vicuña.
De haber colaborado con sacerdotes y coadjutores salesianos,
aquellos Don Bosco(s) vivos, a quienes llegué a conocer
y tener como amigos, confesores y superiores.
De ser exalumno federado, voluntario con sentido de pertenencia
y colaborador comprometido en las obras de mi tiempo en mi juventud,
de sentirme protegido y guiado, en cada momento,
por la presencia de madre María Auxiliadora.

¡Gracias!

"¡Aquí estoy, Señor, para hacer tu voluntad!"
Salmo 39.

Joe Llerena Izquierdo.

Prefacio

El presente libro, que constituye el cuarto volumen de la serie académica “Te acompaño a programar” producido por la carrera de Computación de la Universidad Politécnica Salesiana de la sede Guayaquil, tiene como propósito fundamental ofrecer un contenido sólido y estructurado, de los fundamentos lógicos hasta la implementación práctica de soluciones algorítmicas mediante el uso de un lenguaje de programación. El lector encontrará un análisis de los elementos que definen la programación básica, presentados con una metodología sencilla y amena que busca la comprensión de la teoría con la aplicación práctica impartida en las Carreras de Ingenierías.

En el primer capítulo, se hace un estudio detallado de dos componentes de la lógica matemática: el álgebra de Boole y la teoría de proposiciones. Estos, al ser conceptos abstractos, conforman la estructura esencial de la programación, la electrónica y la innovación tecnológica. Para consolidar este conocimiento, la sección concluye con los laboratorios 1 y 2, los cuales integran un conjunto de ejercicios resueltos diseñados para adaptar la teoría en escenarios de aplicación práctica. En el segundo capítulo, se propone un recorrido analítico por la algoritmia, rescatando la contribución de algunas figuras ejemplares cuya visión permitió el desarrollo actual de los lenguajes de programación. Se establece una ruta guiada que inicia desde la instrucción técnica, explorando cómo la capacidad de traducir el pensamiento abstracto en secuencias lógicas de pasos es parte fundamental para la creación de algoritmos. Se enfatiza la importancia de abordar problemas, cotidianos como complejos, utilizando un enfoque metódico que convierte las ideas en procesos funcionales. Esta sección formativa se complementa con los laboratorios del 3 al 6, orientados a ejercitar la lógica de programación mediante la resolución de problemas algorítmicos diversos. En el tercer capítulo, se introduce a la implementación práctica mediante un lenguaje de programación. Se analizan los fundamentos y las características distintivas de todo lenguaje, destacando la relevancia de la praxis en el desarrollo de soluciones informáticas. El objetivo primordial es que el lector adquiera la capacidad de idear desarrollos que abarquen la lógica inicial y las estructuras de mayor complejidad. El aprendizaje se complementa con el uso del lenguaje Python y de seis laboratorios (del 7 al 12), que ofrecen la utilidad y versatilidad del lenguaje en el ámbito profesional. El cuarto capítulo, dirigido a los profesores, es una breve experiencia de prácticas con el uso de herramientas TIC para el aula.

Finalmente, es importante destacar que esta obra ha sido elaborada en lenguaje LaTeX, asegurando una presentación técnica de alta calidad acorde a los estándares científicos. Este libro representa el compromiso del autor, que pertenece a la carrera de Computación, por proporcionar una herramienta de aprendizaje amena y de alto valor académico, diseñada para fomentar el interés y la excelencia en el estudio de la programación.

Contenido

Dedicatoria III

Agradecimientos V

Prefacio VII

Contenido IX

Índice de figuras XI

Índice de tablas XV

Índice de algoritmos XIX

Capítulo 1. Lógica Matemática 1

1.1 Álgebra de Boole . . . 1
1.1.1 Introducción . . . 3
1.1.2 Compuertas lógicas fundamentales . . . 4
1.1.3 Leyes y teoremas . . . 9
1.1.4 Simplificación de expresiones booleanas . . . 11
1.2 Teoría de proposiciones . . . 26
1.2.1 Introducción . . . 28
1.2.2 Conectores proposicionales . . . 31
1.2.3 Las tablas de verdad . . . 32
1.2.4 Equivalencias lógicas . . . 34
1.2.5 Tipos de proposiciones compuestas . . . 35
1.3 Ejercicios del Laboratorio 1 . . . 44
1.3.1 Sección 1 del laboratorio 1 . . . 44
1.3.2 Sección 2 del laboratorio 1. . . . 47
1.3.3 Sección 3 del laboratorio 1. . . . 51
1.3.4 Sección 4 del laboratorio 1. . . . 55
1.3.5 Sección 5 del laboratorio 1 . . . 57
1.3.6 Sección 6 del laboratorio 1 . . . 60
1.4 Ejercicios del Laboratorio 2 . . . 64
1.4.1 Sección 1 del laboratorio 2. . . . 64
1.4.2 Sección 2 del laboratorio 2. . . . 75
1.4.3 Sección 3 del laboratorio 2 . . . 81
1.4.4 Sección 4 del laboratorio 2. . . . 87

1.4.5 Sección 5 del laboratorio 2. 91

Capítulo 2. Algoritmos **97**
2.1 Importancia de la algoritmia en la ingeniería 97
2.1.1 Conceptos fundamentales 114
2.1.2 Formas de representar los algoritmos 120
2.2 Estructuras de control 126
2.2.1 Estructuras secuenciales 127
2.2.2 Estructuras condicionales (Decisión simple) 131
2.2.3 Estructuras condicionales (Decisión doble) 132
2.2.4 Estructuras repetitivas (Bucles) 141
2.2.5 Tipos de variables (Contadores y acumuladores) 148
2.3 Ejercicios del Laboratorio 3 152
2.4 Ejercicios del Laboratorio 4 158
2.5 Ejercicios del Laboratorio 5 166
2.6 Ejercicios del Laboratorio 6 178

Capítulo 3. Lenguaje de Programación **191**
3.1 Fundamentos a los lenguajes de desarrollo 192
3.1.1 Introducción a los lenguajes de desarrollo 193
3.1.2 Clasificación de los lenguajes de desarrollo 194
3.2 Introducción a Python 196
3.2.1 Tipos de datos y variables 196
3.2.2 Operadores 197
3.2.3 Estructuras secuenciales 198
3.2.4 Estructuras de control condicionales 199
3.2.5 Estructuras de control repetitivas 200
3.2.6 Funciones 201
3.2.7 Arreglos o vectores 201
3.2.8 Matrices 203
3.2.9 Gráficas en Python 204
3.3 Ejercicios del Laboratorio 7 206
3.4 Ejercicios del Laboratorio 8 220
3.5 Ejercicios del Laboratorio 9 228
3.6 Ejercicios del Laboratorio 10 236
3.7 Ejercicios del Laboratorio 11 242
3.8 Ejercicios del Laboratorio 12 246

Capítulo 4. Los recursos multimedia y su importancia **263**
4.1 Herramientas en el internet para el aprendizaje 265
4.2 Recursos multimedia al aula 267

Bibliografía **273**

Autor **297**

Índice de figuras

1.1 Imagen mejorada por la inteligencia artificial Gemini para la sección temática del Álgebra de Boole, en la imagen se muestra a George Boole y el tratado “Las leyes del pensamiento” 2
1.2 Imagen compactada por la inteligencia artificial Claude para la sección introducción del Álgebra de Boole, en la imagen se muestran los fundamentos del Álgebra de Boole. 3
1.3 Imagen creada por la inteligencia artificial Gemini para la sección temática de compuertas lógicas fundamentales, en la imagen se muestra el imaginario de las compuertas lógicas. 4
1.4 Descripción de la información de la compuerta lógica AND 5
1.5 Descripción de la información de la compuerta lógica OR 6
1.6 Descripción de la información de la compuerta lógica NOT 6
1.7 Descripción de la información de la compuerta lógica NAND 7
1.8 Descripción de la información de la compuerta lógica NOR 7
1.9 Descripción de la información de la compuerta lógica XOR 8
1.10 Descripción de la información de la compuerta lógica XNOR 8
1.11 Imagen creada por la inteligencia artificial Gemini para la sección temática de leyes y teoremas, en la imagen se muestra imaginariamente a George Boole y los axiomas. 9
1.12 Imagen creada por la inteligencia artificial Gemini para la sección temática de leyes y teoremas, en la imagen se muestra imaginariamente a George Boole y un contemporáneo. 10
1.13 Imagen creada por la inteligencia artificial Gemini para la sección temática de simplificación de expresiones booleanas, en la imagen se muestran esculturas póstumas. 11
1.14 Imagen creada por la inteligencia artificial Gemini y su edición con el programa de Corel Draw 2025 para la sección temática teoría de proposiciones, en la imagen se muestra imaginariamente a George Boole, Aristóteles y Gottlob Frege. 26
1.15 Imagen creada por la inteligencia artificial Gemini para la sección introducción, en la imagen se muestra imaginariamente a Gottlob Frege y Bertrand Russell. 28
1.16 Imagen creada por la inteligencia artificial Napkin.ai para la sección introducción, en la imagen se muestra los bloques de construcción de la lógica. 29
1.17 Imagen creada por la inteligencia artificial Napkin.ai para la sección introducción, en la imagen se muestra la lógica proposicional en IA. 30

1.18 Imagen creada por la inteligencia artificial Napkin.ai para la sección introducción, en la imagen se muestra la evolución de lógica proposicional. 30
1.19 Imagen del Centro de conocimiento e innovación deportiva del FC Barcelona que se enfoca a generar, compartir y aplicar conocimiento a través de proyectos de investigación. 31
1.20 Conectivo lógico binario fundamental, puerta lógica NOT. 32
1.21 Conectivos lógicos binarios fundamentales, incluyendo las puertas lógicas (NAND, NOR, XOR, XNOR). 32
1.22 Conectivo lógico fundamental (Verdadero/Falso), de la compuerta lógica NOT. 33
1.23 Conectivos lógicos fundamentales (Verdadero/Falso), incluyendo las compuertas lógicas (NAND, NOR, XOR, XNOR). 33
1.24 Clasificación de proposiciones lógicas 35
1.25 El programa de Logisim se descarga en sourgeforge.net. 51
1.26 Entorno de trabajo de Logisim 51
1.27 Imagen de salida del programa Logisim. 52
1.28 Proceso a seguir en el diagrama de flujo para la Ley de Consenso. 58
1.29 Imagen del programa Logisim. 60
1.30 Imagen final del circuito simplificado. 62
1.31 Imagen de la calculadora Logic Calculator. 81
1.32 Imagen del ejercicio propuesto con la calculadora Logic Calculator. . . . 81

2.1 Imagen generada por la IA Gemini de Muhammad ibn Musa al-Khwārizmī y los algoritmos. 99
2.2 Imagen generada por la IA Gemini de Ada Lovelace y los algoritmos. . . 100
2.3 Grace Murray Hopper ante el teclado UNIVAC I, hacia 1960. 103
2.4 Grace Hopper desarrolló en 1959, el lenguaje COBOL. 103
2.5 *"Atrévete a intentarlo. Atrévete a hacer las cosas de forma diferente. Atrévete a ser audaz y simplemente observa qué sucede"*, Grace Hooper . 103
2.6 Primer caso real de error detectado en un equipo de cómputo. 104
2.7 Sintaxis del lenguaje COBOL. 104
2.8 Hedy Lamarr. 105
2.9 Salto de frecuencia de espectro ensanchado. 105
2.10 Madre del Wi-Fi. 106
2.11 Su genio inventivo. 106
2.12 Diseño de patente. 107
2.13 Margaret Hamilton. 107
2.14 Módulo del Apolo 12. 108
2.15 En el Módulo Lunar. 108
2.16 Apollo Software Code. 108
2.17 Ingeniería de Software. 109
2.18 Nasa, Apolo 17. 109
2.19 Medalla Presidencial. 109
2.20 Comentarios dentro del código. 110
2.21 Katie Bouman. 110
2.22 Katie Bouman en TED. 111

2.23 Imagen del agujero negro M87. 111
2.24 Pilas de discos duros con datos de imágenes de agujeros negros. 111
2.25 Abril de 2019. 112
2.26 Katie Bouman investigadora en métodos de imagen computacional y visión artificial. 112
2.27 Pioneras en la computación. 113
2.28 Recorrido del origen de la palabra Algoritmo. 113
2.29 Origen de la palabra Algoritmo y su trascendencia. 114
2.30 Imagen desarrollada con napkin.ai para el tema: Características de un Algoritmo. 115
2.31 Imagen desarrollada con napkin.ai para el tema: Transformación de datos. 115
2.32 Imagen desarrollada con napkin.ai para el tema: Jerarquía de las estructuras. 116
2.33 Estructura secuencial de la ejecución de un archivo fuente. 116
2.34 La complejidad del algoritmo. 117
2.35 Estrategias de diseño de algoritmos basadas en paradigmas. 118
2.36 Características estratégicas de diseño de algoritmos. 119
2.37 Formas de representar algoritmos. 120
2.38 Los flujogramas. 121
2.39 El Pseudocódigo. 121
2.40 Diagramas Nassi-Shneiderman. 122
2.41 Representaciones relevantes y sus características. 122
2.42 Las constantes y las variables. 123
2.43 Almacenamiento en la memoria de acceso aleatorio por tipo de dato. . . . 124
2.44 Las pruebas de escritorio. 125
2.45 Las estructuras de control existentes. 126
2.46 Estructura secuencial. 127
2.47 Algoritmo que calcula el área del círculo. 128
2.48 Algoritmo que calcula el área del hexágono usando el lado. 129
2.49 Algoritmo que calcula el área del hexágono usando el lado y el apotema. . 130
2.50 Algoritmo que verifica si la edad de una persona es mayor e igual a 18. . . 131
2.51 Algoritmo que verifica si la calificación es mayor e igual a 70. 132
2.52 Algoritmo que verifica si un número ingresado es par o impar. 133
2.53 Algoritmo que clasifica la edad según el rango. 136
2.54 Estructuras repetitivas iniciales. 141
2.55 Algoritmo que genera una tabla de multiplicar. 142
2.56 Algoritmo que genera una suma controlada por centinela. 143
2.57 Algoritmo que utiliza la estructura mientras con un valor centinela. 144
2.58 Diagrama de flujo que utiliza la estructura mientras con un valor centinela para opciones de menú. 146
2.59 Diagrama de flujo que utiliza la estructura Para. 147
2.60 Las variables nos ayudan a controlar el ciclo o a guardar resultados. . . . 148
2.61 Uso de diferentes tipos de variables para controlar el ciclo y guardar resultados. 150

3.1 Los lenguajes de programación del índice TIOBE expresados en una nube de palabras. 191

3.2 Características de un lenguaje de programación. 192
3.3 Elementos fundamentales de un lenguaje de programación. 193
3.4 Clasificación de los lenguajes de programación. 194
3.5 Clasificación de los lenguajes de programación. 195
3.6 Programación secuencial. 198
3.7 Programación secuencial. 199
3.8 Estructuras de control. 200
3.9 Imagen creada por la inteligencia artificial Napkin.ai, en la imagen se muestran los elementos importantes de los arreglos. 202
3.10 Matrices en Python. 203

Índice de tablas

1.1 Operaciones definidas de las compuertas lógicas. 5
1.2 Axiomas y teoremas esenciales para la simplificación lógica. 10
1.3 Compendio de las leyes y teoremas esenciales para la simplificación lógica. 12
1.4 Simplificación de $F = (A + \overline{B})(A + B)\overline{C} + \overline{A}$ 13
1.5 Simplificación de $F = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$ 14
1.6 Simplificación de $F = AB\overline{C} + ABC + AB\overline{D} + \overline{A}B\overline{C}$ 14
1.7 Simplificación de $F = \overline{A}B + AB + \overline{A}BC$ 15
1.8 Simplificación de $F = A(\overline{A} + B) + B(\overline{B} + C)$ 15
1.9 Simplificación de $F = AB + A\overline{B} + \overline{A}C$ 16
1.10 Simplificación de $F = AC + A\overline{C} + BC$ 16
1.11 Simplificación de $F = ABC + AB\overline{C} + A\overline{B}D + \overline{A}BD$ 17
1.12 Tabla de verdad de $F(A, B, C, D)$ con $F = AB + \overline{B}D$ 17
1.13 Simplificación de $F = AB + A\overline{B}C + \overline{A}BC + \overline{A}B\overline{C}$ 19
1.14 Simplificación de $F = \overline{A}BD + ABD + A\overline{B}D + A\overline{B}\overline{C}$ 20
1.15 Simplificación de $F = \overline{A}\overline{B}C + \overline{A}BC + ABC + AB\overline{C}$ 21
1.16 Simplificación de $F = A\overline{B}C + ABC + \overline{A}B\overline{C} + \overline{A}BC$ 22
1.17 Tabla de verdad de $F(A, B, C, D)$ con $F = AC + \overline{A}B$ 22
1.18 Simplificación de $F = \overline{A}B\overline{C}D + \overline{A}B\overline{C}D + AB\overline{C}D + ABC\overline{D}$ 24
1.19 Tabla de verdad de $F(A, B, C, D)$ con $F = \overline{C}D(\overline{A} + B) + ABC\overline{D}$ 24
1.20 Distinción entre proposiciones (con valor de verdad) y expresiones no proposicionales . 27
1.21 Conectivos lógicos, resumen 27
1.22 Conectivos lógicos proposicionales 31
1.23 Leyes y equivalencias lógicas fundamentales 34
1.24 Comportamiento en la columna final de la tabla de verdad 35
1.25 Tabla de verdad de $(p \wedge q) \vee (q \wedge \neg r)$ 36
1.26 Tabla de verdad de $(\neg p \wedge q) \rightarrow r$ 36
1.27 Tabla de verdad de $(p \leftrightarrow q) \wedge \neg p$ 36
1.28 Tabla de verdad de $(\neg p \vee q) \wedge \neg q$ 37
1.29 Tabla de verdad de $((p \wedge q) \rightarrow r) \leftrightarrow \neg s$ 37
1.30 Tabla de verdad de $p \rightarrow (q \wedge \neg r)$ 37
1.31 Tabla de verdad de $(\neg p \wedge q) \rightarrow (p \rightarrow q)$ 38
1.32 Tabla de verdad de $[(p \rightarrow q) \wedge \neg q] \rightarrow \neg p$ 38
1.33 Tabla de verdad de $[(\neg p \vee q) \wedge (p \vee r)] \rightarrow (q \vee r)$ 38
1.34 Tabla de verdad de $\neg(p \vee q) \rightarrow (\neg p \wedge \neg q)$ 38
1.35 Tabla de verdad de $p \wedge \neg p$ 39
1.36 Tabla de verdad de $(p \leftrightarrow q) \wedge (p \oplus q)$ 39
1.37 Tabla de verdad de $(p \wedge q) \wedge \neg(p \wedge r)$ 39

1.38 Tabla de verdad de $(p \to q) \land (p \land \neg q)$ . . . 39
1.39 Tabla de verdad de $(p \land q \land r \land s) \land \neg(p \land q \land r \land s)$ . . . 40
1.40 Tabla de verdad de $(p \land q \land r \land s \land t) \land \neg(p \land q \land r \land s \land t)$ . . . 40
1.41 Tabla de verdad de $(p \land \neg q) \to r$ . . . 41
1.42 Tabla de verdad de $q \to (\neg p \lor p)$ . . . 41
1.43 Tabla de verdad de $(\neg q \land p) \land \neg p$ . . . 42
1.44 Tabla de verdad de $(p \lor q) \to r$ . . . 42
1.45 Comparativa de la utilidad del Álgebra Booleana en diferentes artefactos . . . 44
1.46 Resumen de las funciones y tipos de las compuertas lógicas . . . 45
1.47 Correspondencia entre estados lógicos y rangos de voltaje en electrónica digital . . . 46
1.48 Verificación de la igualdad $x'z + x'yz = x'z$ . . . 55
1.49 Verificación de la igualdad $x(x + y) = x$ . . . 55
1.50 Verificación de la igualdad $\overline{A}B\overline{C}D + \overline{A}B\overline{C}\overline{D} + AB\overline{C}D + A\overline{B}\overline{C}D = \overline{C}D$ . . . 56
1.51 Verificación detallada de la simplificación (continuación) . . . 56
1.52 Conversión de $E = ((a'b')c)'((a' + c)(b' + c'))'$ a Suma de Productos . . . 57
1.53 Simplificación de $F = (A + B)(\overline{A} + C)(B + C)$ a su mínima expresión SOP . . . 58
1.54 Simplificación de $G = AB + \overline{A}C + BC$ a su mínima expresión SOP . . . 59
1.55 Simplificación de $H = (A + \overline{B}C)(A + \overline{B}C')$ . . . 59
1.56 Simplificación de la expresión Y . . . 60
1.57 Tabla de verdad para la expresión simplificada $\mathbf{Y = AB + A'C}$ . . . 61
1.58 Mapa de Karnaugh y Simplificación de $Y = AB + \overline{A}C$ . . . 61
1.59 Simplificación de $Y = ACB + AC'B + A'C$ . . . 62
1.60 Traducción de premisas y conclusión del enunciado 1 . . . 64
1.61 Evaluación completa de la proposición de 4 variables del enunciado 1 . . . 65
1.62 Traducción de premisas y conclusión del enunciado 2 . . . 65
1.63 Evaluación completa de la proposición de 4 variables del enunciado 2 . . . 66
1.64 Traducción de premisas y conclusión del enunciado 3 . . . 66
1.65 Evaluación completa de la proposición de 5 variables . . . 67
1.66 Traducción de premisas y conclusión del enunciado 4 . . . 67
1.67 Evaluación completa de la proposición de 3 variables . . . 68
1.68 Evaluación de $T = (P \to Q) \to (\neg P \lor Q)$ . . . 68
1.69 Traducción de premisas y conclusión del enunciado 6 . . . 69
1.70 Evaluación de $T = [((P \land Q) \to R) \land (\neg P \to \neg Q)] \to (R \to Q)$ . . . 69
1.71 Traducción de premisas y conclusión del enunciado 7 . . . 70
1.72 Evaluación de $T = [((P \land Q) \to R) \land (R \to S) \land (\neg P \lor \neg S)] \to (\neg R \lor \neg Q)$ . . . 70
1.73 Traducción de premisas y conclusión del enunciado 8 . . . 71
1.74 Evaluación enunciado 8 - $W = [P_1 \land P_2 \land P_3 \land P_4] \to$ Conclusión . . . 71
1.75 Traducción de premisas y conclusión del enunciado 9 . . . 72
1.76 Evaluación enunciado 9 - $W = [P_1 \land P_2 \land P_3 \land P_4] \to$ Conclusión . . . 72
1.77 Traducción de premisas y conclusión del enunciado 10 . . . 73
1.78 Evaluación enunciado 10 - $T = [P_1 \land P_2 \land P_3] \to$ Conclusión . . . 73
1.79 Traducción de premisas y conclusión del enunciado 11 . . . 74
1.80 Evaluación enunciado 11 - $T = [P_1 \land P_2 \land P_3] \to$ Conclusión . . . 74
1.81 Traducción de premisas y conclusión del ejercicio 1 . . . 75
1.82 Traducción de premisas y conclusión del ejercicio 2 . . . 76
1.83 Traducción de premisas y conclusión del ejercicio 3 . . . 77

1.84 Traducción de premisas y conclusión del ejercicio 4 . . . 78
1.85 Traducción de premisas y conclusión del ejercicio 5 . . . 79
1.86 Traducción de premisas y conclusión del ejercicio 6 . . . 80
1.87 Valoración de verdad del argumento, ejercicio 1 . . . 82
1.88 Valoración de verdad del argumento, ejercicio 2 . . . 82
1.89 Valoración de verdad de la fórmula, ejercicio 3 . . . 82
1.90 Valoración de verdad de la fórmula, ejercicio 4 . . . 83
1.91 Valoración de verdad de la fórmula, ejercicio 5 . . . 83
1.92 Valoración de verdad de la fórmula, ejercicio 6 (con 6 variables) . . . 84
1.93 Valoración de verdad de la fórmula, ejercicio 7 . . . 85
1.94 Valoración de verdad de la fórmula, ejercicio 8 . . . 85
1.95 Valoración de verdad de la fórmula, ejercicio 9 . . . 85
1.96 Valoración de verdad de la fórmula, ejercicio 10 . . . 86
1.97 Valoración de verdad de la fórmula, ejercicio 11 . . . 86
1.98 Valoración de verdad de la fórmula, ejercicio 12 . . . 86

2.1 Impacto algorítmico . . . 113
2.2 Principales fechas en la representación de algoritmos . . . 120
2.3 Rangos de edad y clasificación . . . 135
2.4 Valores para los rangos de edad y clasificación . . . 135
2.5 Operadores aritméticos, de relación y lógicos . . . 139
2.6 Todos los códigos son lógicamente equivalentes y producen el mismo resultado. . . . 140
2.7 Clasificación y función de las variables de control. . . . 149
2.8 Ejemplos de operaciones de división . . . 161
2.9 Tabla de verdad (Operador AND para rango) . . . 171
2.10 Tabla de Verdad (Operador OR para rangos disjuntos) . . . 172

4.1 Video de bienvenida al contenido del libro . . . 263
4.2 Canal de videos para las guías de aprendizaje por capítulos y sus laboratorios . . . 264
4.3 Sitios de Internet para el aprendizaje por capítulos . . . 266
4.4 Datos principales de los videos sugeridos en cada capítulo del libro . . . 268
4.5 Documentales para el aprendizaje por capítulos . . . 269

Índice de algoritmos

2.1 Pseudocódigo PSeInt: Cálculo del año de nacimiento 127
2.2 Pseudocódigo PSeInt: Área del círculo 128
2.3 Pseudocódigo PSeInt: Área de la figura regular usando solo el lado 129
2.4 Pseudocodigo PSeInt: Área de hexágono regular (lado y apotema) 130
2.5 Pseudocódigo PSeInt: Verificación de mayoría de edad 131
2.6 Pseudocódigo PSeInt: Evaluación de calificación 132
2.7 Pseudocódigo PSeInt: Detección de valor par o impar 133
2.8 Pseudocódigo PSeInt: Clasificación de edad con anidamiento simple y doble 134
2.9 Pseudocódigo PSeInt: Clasificación de etapas de la vida (Selectiva Múltiple) 137
2.10 Pseudocódigo PSeInt: Clasificación de etapas con Si-Sino Si (Mejor para rangos) 137
2.11 Pseudocódigo PSeInt: Uso de la estructura Según 138
2.12 Pseudocódigo PSeInt: Menú de opciones con la estructura Según 138
2.13 Pseudocódigo PSeInt: Generar tabla de multiplicar 141
2.14 Pseudocódigo PSeInt: Suma controlada por centinela 142
2.15 Pseudocódigo PSeInt: Validar un número en un rango (Repetir-Hasta) . . 144
2.16 Pseudocódigo PSeInt: Menú interactivo con Repetir-Hasta 145
2.17 Pseudocódigo PSeInt: Tabla de multiplicar con el ciclo Para 147
2.18 Pseudocódigo PSeInt: Ejemplo de todas las variables útiles en un algoritmo 148
2.19 Pseudocódigo PSeInt: Cálculo del volumen de un cono 152
2.20 Pseudocódigo PSeInt: Cálculo de volumen de un cono con inicialización de variables 152
2.21 Pseudocódigo PSeInt: Conversión de temperatura 153
2.22 Pseudocódigo PSeInt: Segundos a horas y minutos 153
2.23 Pseudocódigo PSeInt: Cálculo de nota final) 154
2.24 Pseudocódigo PSeInt: Evaluación de expresión matemática compleja . . . 154
2.25 Pseudocódigo PSeInt: Cálculo del área de triángulo por Ley del Seno . . 155
2.26 Pseudocodigo PSeInt: Evaluacion de formula con constantes (Final) . . . 156
2.27 Pseudocodigo PSeInt: Mensaje de saludo simple 158
2.28 Pseudocodigo PSeInt: Cálculo del área de un cuadrado 158
2.29 Pseudocodigo PSeInt: Computo de una compra con impuesto al valor agregado 158
2.30 Pseudocodigo PSeInt: Computo del valor de la variable 159
2.31 Pseudocodigo PSeInt: Cómputo del área y perímetro de un círculo 159
2.32 Pseudocodigo PSeInt: Suma de dos números 160
2.33 Pseudocodigo PSeInt: Resta de dos números 160
2.34 Pseudocodigo PSeInt: Cociente de dos números 160
2.35 Pseudocodigo PSeInt: Cómputo del área e hipotenusa del triángulo rectángulo 161

2.36 Pseudocodigo PSeInt: Operaciones combinadas 162
2.37 Pseudocodigo PSeInt: Ley de Ohm (Voltaje y Potencia) 162
2.38 Pseudocodigo PSeInt: Computo del área y perímetro de un rectángulo . . 163
2.39 Pseudocodigo PSeInt: Cómputo del área y perímetro de un rombo 163
2.40 Pseudocodigo PSeInt: Cómputo de área y perímetro de un paralelogramo 163
2.41 Pseudocodigo PSeInt: Calculo del promedio final de Programación 164
2.42 Pseudocodigo PSeInt: Determinación de Mayor o Menor de Edad 166
2.43 Pseudocodigo PSeInt: Determinación de Mayor o Menor de Edad con salida concatenada 166
2.44 Pseudocodigo PSeInt: Determinación de pertenencia a un rango simple . . 167
2.45 Pseudocodigo PSeInt: Determinación de pertenencia a un rango menor estricto 167
2.46 Pseudocodigo PSeInt: Determinación de pertenencia a un rango mayor estricto 167
2.47 Pseudocodigo PSeInt: Determinación de pertenencia a un rango mayor o igual 168
2.48 Pseudocodigo PSeInt: Determinación de pertenencia a una unión de intervalos (OR) 168
2.49 Pseudocodigo PSeInt: Determinación de pertenencia a una intersección de intervalos (AND) 169
2.50 Pseudocodigo PSeInt: Determinación de pertenencia a una intersección (AND) de intervalos 169
2.51 Pseudocodigo PSeInt: Determinación de pertenencia a una unión de intervalos (OR) 170
2.52 Pseudocodigo PSeInt: Conversor de unidades de temperatura 170
2.53 Pseudocodigo PSeInt: Verificación de rango con AND 171
2.54 Pseudocodigo PSeInt: Verificación de rango con OR 172
2.55 Pseudocodigo PSeInt: Verificacion de valor Armstrong 173
2.56 Pseudocodigo PSeInt: Búsqueda de números Armstrong de tres cifras . . 174
2.57 Pseudocodigo PSeInt: Verificacion de número Armstrong de cuatro cifras 174
2.58 Pseudocodigo PSeInt: Búsqueda de números Armstrong de cuatro cifras . 175
2.59 Pseudocodigo PSeInt: Búsqueda de números Armstrong de cinco cifras . 176
2.60 Pseudocodigo PSeInt: Imprimir serie numérica con Para (Incremento de 4) 178
2.61 Pseudocodigo PSeInt: Imprimir serie numérica con Mientras (Incremento de 4) 178
2.62 Pseudocodigo PSeInt: Imprimir serie numérica con Repetir-Hasta (Incremento de 4) 178
2.63 Pseudocodigo PSeInt: Imprimir serie numérica descendente con Para (Decremento de 4) 179
2.64 Pseudocodigo PSeInt: Imprimir serie numérica descendente con Mientras (Decremento de 4) 179
2.65 Pseudocodigo PSeInt: Imprimir serie numérica descendente con Repetir-Hasta (Decremento de 4) 179
2.66 Pseudocodigo PSeInt: Imprimir serie numérica (Incremento de 3) con Contador 180

2.67 Pseudocodigo PSeInt: Imprimir serie numérica con Contador (Usando Mientras) . . . 180
2.68 Pseudocodigo PSeInt: Imprimir serie numérica (Incremento de 5) con Contador . . . 180
2.69 Pseudocodigo PSeInt: Generar tabla de multiplicar . . . 181
2.70 Pseudocodigo PSeInt: Cálculo de consumo eléctrico y cobro . . . 181
2.71 Pseudocodigo PSeInt: Generar Serie de Fibonacci hasta N terminos . . . 183
2.72 Pseudocodigo PSeInt: Generar serie de Ulam o Collatz . . . 183
2.73 Pseudocodigo PSeInt: Generar N números Primos . . . 184
2.74 Pseudocodigo PSeInt: Verificación de conjunto Diofántico . . . 185
2.75 Pseudocodigo PSeInt: Serie de sumas de impares (Números cuadrados) . . . 186
2.76 Pseudocodigo PSeInt: Cálculo de Potencia . . . 187
2.77 Pseudocodigo PSeInt: Cálculo del Factorial (N!) . . . 188
3.1 Ejemplo básico de Entrada y Salida en Python . . . 199
3.2 Programa para mostrar figura usando nombres como variables . . . 206
3.3 Manipulación de cadenas y formateo en Python . . . 206
3.4 Invertir una cadena de texto en Python . . . 207
3.5 Operadores aritmeticos básicos en Python . . . 207
3.6 Elevar al cuadrado e invertir dígitos con espacios . . . 207
3.7 Cálculo de Área y Perímetro para Rectángulo y Círculo . . . 208
3.8 Redondeo y Truncamiento de valores en Python . . . 208
3.9 Precisión de punto flotante en sumas y multiplicaciones . . . 209
3.10 Extraer y ordenar consonantes únicas de una frase . . . 210
3.11 Potencias de 2 menores a 10000 con formato de 4 dígitos . . . 210
3.12 Extraer e imprimir iniciales en mayúscula (Acrónimo) . . . 211
3.13 Triángulo creciente y decreciente con texto . . . 211
3.14 Uso de condicionales anidados en Python . . . 212
3.15 Uso de bucles y condicionales combinados . . . 212
3.16 Función para determinar si un número es par o impar . . . 213
3.17 Función para comparar dos números . . . 213
3.18 Juego de adivinar número con límite de intentos . . . 213
3.19 Función para invertir cada palabra de un texto . . . 214
3.20 Imprimir letras de dos palabras fila por fila . . . 215
3.21 Función para verificar si una tupla esta ordenada . . . 215
3.22 Suma de los N primeros números enteros . . . 216
3.23 Suma de los N primeros números enteros en otra versión . . . 216
3.24 Generador de tablas de multiplicar . . . 216
3.25 Generador de tablas de multiplicar en otra versión . . . 216
3.26 Algoritmo para verificar números primos . . . 216
3.27 Algoritmo para verificar números primos en otra versión . . . 217
3.28 Búsqueda secuencial en un arreglo de 10 números . . . 217
3.29 Sumatoria con visualizacion ascendente y descendente . . . 218
3.30 Sumatoria dinámica con visualización dual . . . 218
3.31 Contador ascendente del 1 al 10 con nuevas variables . . . 220
3.32 Contador descendente del 10 al 1 . . . 220
3.33 Cuenta regresiva del 10 al 1 usando bucle while . . . 220

3.34 Simulación de un bucle Hacer-Mientras del 10 al 1 221
3.35 Sumatoria acumulada desde 1 hasta N 221
3.36 Generación de la secuencia de Ulam o Collatz 221
3.37 Generador de la serie de Fibonacci 222
3.38 Cálculo del factorial de un número N 222
3.39 Cálculo de potencia mediante bucles 223
3.40 Menú interactivo para ejercicios matemáticos 223
3.41 Aproximación de e^x usando Series de Taylor 224
3.42 Aproximación de Seno(x) usando Taylor 225
3.43 Aproximacion de Coseno(x) usando Taylor 225
3.44 Método de Multiplicación Rusa 226
3.45 Determinación de números primos 228
3.46 Secuencia con bucle for 228
3.47 Secuencia con bucle while 228
3.48 Secuencia simulando do-while 229
3.49 Números correlativos con saltos de línea cada 5 unidades 229
3.50 Generación de números aleatorios 230
3.51 Redondeo hacia arriba y hacia abajo 230
3.52 Números pares entre 10 y 20 inclusive 230
3.53 Programa para captura de nombre 231
3.54 Contador de vocales en una palabra 231
3.55 Contador de espacios en blanco 231
3.56 Validador de cedula ecuatoriana 231
3.57 Juego de Adivina el valor con calificación 232
3.58 Juego de Adivina el valor con mensajes de guia 233
3.59 Generador de Tabla de Bingo 75 valores 234
3.60 Conversor de Dólares a Euros 236
3.61 Cálculo de promedio de calificaciones 236
3.62 Comparación de dos edades 236
3.63 Conversor de número a día de la semana 237
3.64 Bucle While del 25 al 58 237
3.65 Emulación de Do-While del 74 al 96 237
3.66 Números del 11 al 99 en bloques de 10 238
3.67 Clasificador de números 238
3.68 Sumatoria de números del 20 al 30 239
3.69 Simulación de lanzamiento de dos dados 239
3.70 Juego: Tres veces Siete 239
3.71 Simulador de intentos hasta ganar 240
3.72 Proyecto de misión parte 1 242
3.73 Proyecto de misión parte 2 243
3.74 Proyecto de misión parte 2 243
3.75 Nueva Misión a completar el siguiente código 244
3.76 Descomposición de Matriz Cuadrada 246
3.77 Producto de elementos pares por fila 246
3.78 Suma de elementos fuera de las diagonales 247
3.79 Simulador del Cuadrado Semi-Magico 4x4 248

3.80 Relleno de matriz basado en promedio de bordes 249
3.81 Intercambio de diagonales en matriz cuadrada 250
3.82 Conteo de pares por columna 251
3.83 Identificar columna con más pares 251
3.84 Suma de cifras y búsqueda de máximo 252
3.85 Verificación de propiedad de números impares 253
3.86 Cálculo del n-esimo término de Fibonacci 253
3.87 Calculo del n-esimo termino de Tribonacci 254
3.88 Búsqueda aleatoria de números simétricos 255
3.89 Simetría numérica mediante lógica matemática 255
3.90 Simulación de lanzamientos hasta obtener n 256
3.91 Cálculo de términos necesarios para exceder x 257
3.92 Modificación de elementos de una lista al cuadrado 257
3.93 Transformación de lista usando List Comprehension 258
3.94 Cuadrados de una lista de tamaño n 258
3.95 Sustitución de negativos por cero 258
3.96 Lista de los n primeros primos sin usar append 258
3.97 Eliminación de índices pares 259
3.98 Uso de is para filtrar elementos 259
3.99 Uso recomendado de is con None 260
3.100 Verificar si es el mismo objeto lista 260

DOI: 10.17163/abyaups.167.1

Capítulo 1
Lógica Matemática

Resultado de aprendizaje

Diseña soluciones a problemas de lógica empleando álgebra de Boole y teoría de proposiciones.

Indicador de logro

Aplica lógica matemática para resolver problemas.

1.1. Álgebra de Boole

El álgebra booleana (o Álgebra de Boole) proviene del trabajo realizado por el matemático de origen inglés llamado George Boole (1815 - 1864). El trabajo de George Boole ha trascendido en la historia, debido a que sus aportaciones permitieron reducir la lógica formal a un álgebra más práctica (dificultad que tenían las matemáticas en aquel entonces). Él logró una analogía con los símbolos algebraicos y las expresiones de sus formas lógicas en la Universidad de Cork, en Cork, Irlanda, centro de estudios booleanos. Allí, desde 1849 hasta su muerte en 1864, George Boole fue profesor de matemáticas (siendo el primero), nombrado en la fundación del Queen's College, en una época victoriana conocida como «la semilla del progreso moderno» [Crilly, 2025].

George Boole logró tanto en el ámbito matemático, gracias a una red social que lo apoyó. Su historia pone de manifiesto la importancia de los profesores a la hora de ayudar a los jóvenes matemáticos, así como la influencia que estos tienen. En el caso de George Boole, se aprecia el apoyo de dos personajes, la figura de su padre, y la nobleza obligada que le brindó el aristócrata local Sir Edward French Bromhead. Este hombre (Bromhead), que había pasado por el sistema *Tripos de Matemáticas de Cambridge*, pudo aconsejar a Boole sobre qué leer y estudiar en matemáticas.

George Boole tuvo acceso a maestros como Laplace y Lagrange, y esto se debió a la generosidad de Bromhead, que le permitió a su aprendiz utilizar su biblioteca en Thurlby Hall, tal y como había hecho veinte años antes con el igualmente pobre George Green, de Nottingham [Waszek, 2025]. Pero mientras Green pasó a la Universidad de Cambridge, Boole perdió esta oportunidad, muy a su pesar, aunque el sistema orientado a los exámenes que prevalecía entonces en Cambridge podría haber frenado su estilo natural. Las matemáticas en la época victoriana tenían una ventaja notable como objeto de estudio para los autodidactas de clase social más baja. No necesitaban costosos equipos científicos

ni un laboratorio, y era una materia en la que el talento y la dedicación implícitos podían superar las barreras de clase y económicas.

De forma particular, era posible, como demostró Boole, que siendo hijo de un zapatero se labrara una reputación científica. En esta línea, podemos citar otros ejemplos en el contexto británico, «estudiantes» que obtuvieron una beca de la Royal Society de Londres. Thomas P. Kirkman fue uno de ellos, elegido en la misma promoción que Boole en 1857. Anteriormente, el irlandés Robert Murphy, también hijo de un zapatero, fue elegido miembro en 1834 [Kyriazis, 2025].

Al principio de su carrera, George Boole recibió una medalla de oro por el trabajo matemático más significativo presentado a la Royal Society en el periodo comprendido entre junio de 1841 a junio de 1844, lo que sin duda supuso un estímulo vital para un joven, aunque el reconocimiento no le llegó fácilmente. George Boole tenía dos defensores. Thomas S. Davies, geómetra elegido miembro de la Royal Society en 1833, se ganaba la vida como tutor privado antes de ser nombrado profesor de matemáticas en la Real Academia Militar de Woolwich. Argumentó que *ser pobre no era una razón válida para que se rechazara su trabajo*. El otro apoyo vino de Escocia, en la persona de Philip Kelland, profesor de matemáticas en Edimburgo. Era un antiguo wrangler senior de Cambridge que podía comprender el valor del trabajo de Boole y cuyo juicio fue probablemente decisivo en la concesión de la medalla [Crilly, 2025].

George Boole publicó un libro sobre *"Las Leyes del pensamiento"*, presentando el sustento de las teorías de la lógica y la probabilidad, ver Fig. 1.1.

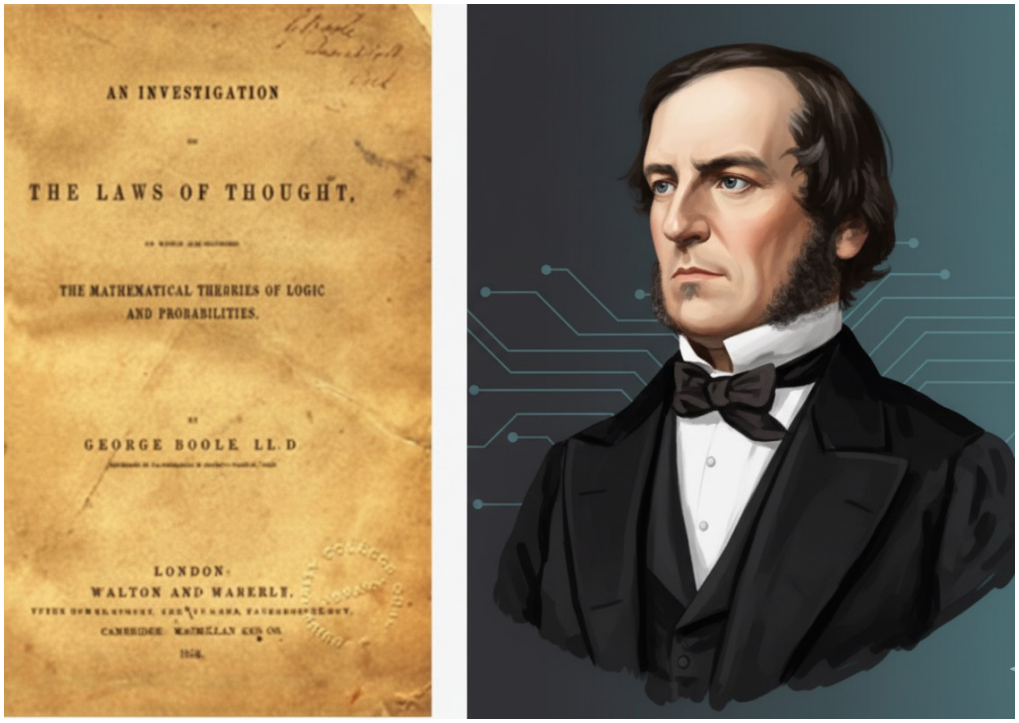


Figura 1.1: Imagen mejorada por la inteligencia artificial Gemini para la sección temática del Álgebra de Boole, en la imagen se muestra a George Boole y el tratado "Las leyes del pensamiento"

El álgebra booleana o de Boole, consiste en un método para resolver problemas de lógica, empleando para ello los valores binarios de 1 y 0 con el uso de tres operadores: AND (y), OR (o) y NOT (no), así como sus combinaciones.

1.1.1. Introducción

El álgebra booleana se lo utiliza en la creación de componentes electrónicos de las computadoras, placas y dispositivos. Fue desarrollada para manipular proposiciones lógicas de forma algebraica. Hoy es comprensible saber que los sistemas modernos trabajan con diseños de lógica binaria. Desde una condición inicial, el enfoque del álgebra booleana es abstracto, pero en 1938, la Bell Telephone Company en Estados Unidos la utilizó para analizar los circuitos de su red telefónica. Además, en ese mismo año, Claude Shannon creó el Álgebra de Conmutación a partir del álgebra booleana, lo que demostró que esta se adapta perfectamente al diseño y representación de circuitos lógicos de control.

Un sinnúmero de aplicaciones tiene el álgebra booleana y se muestran visibles en la implementación de los circuitos en los equipos de cómputo, en sistemas telefónicos, en la robótica y en cualquier operación automatizada de una empresa. En términos generales, el álgebra booleana se puede emplear en cualquier sistema en que cada variable posea dos estados (1 y 0).

- Se emplea en las computadoras modernas donde realizan sus operaciones y almacenan la información empleando valores binarios (1 y 0).
- Se emplea en el diseño de circuitos electrónicos para la elaboración de componentes eléctricos que ayudan a la conversión de lo analógico a lo digital (0V a 5V).
- Se emplea ampliamente en la programación en el momento de utilizar condiciones lógicas y validaciones que requieren una respuesta de un falso o un verdadero.

El álgebra booleana trabaja con señales binarias (un tipo de señal digital). A diferencia de una señal analógica (cambio continuo de magnitud), la señal binaria tiene solo dos valores posibles, conectado / desconectado o verdadero / falso (1 o 0) ver Fig. 1.2.

Para analizar y profundizar

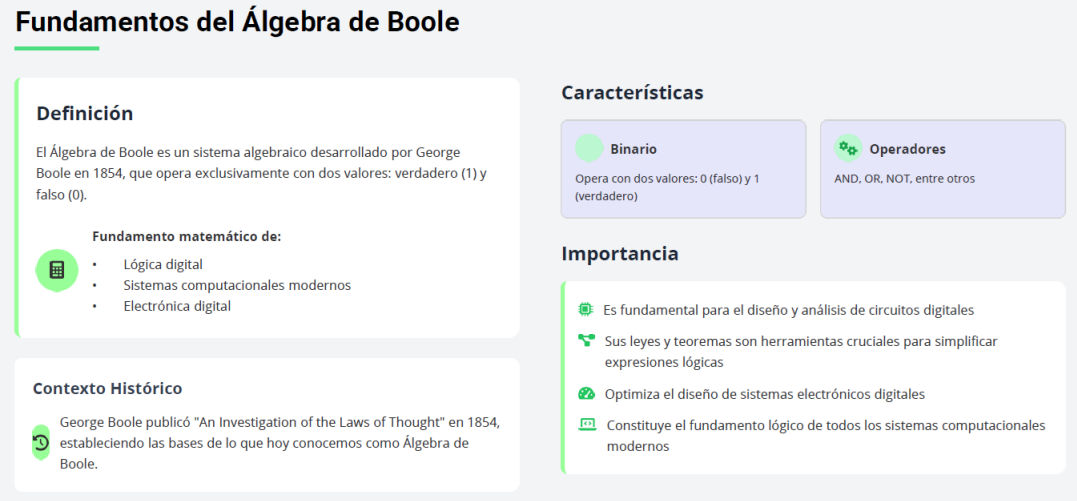


Figura 1.2: Imagen compactada por la inteligencia artificial Claude para la sección introducción del Álgebra de Boole, en la imagen se muestran los fundamentos del Álgebra de Boole.

1.1.2. Compuertas lógicas fundamentales

Las compuertas lógicas se constituyen en los elementos de construcción fundamental en la lógica combinatoria, con esto, debemos considerar lo siguiente, las compuertas trabajan con dos valores de "verdad", los números, 1 y 0, y al igual que en la lógica proposicional, es posible la construcción de tablas de verdad para cada compuerta y, sus combinaciones con otras compuertas. Las variables de entrada se utilizan para representar el nivel de voltaje en un medio de cable eléctrico o en las entradas y salidas de un circuito electrónico (0V ó 5V).

Las compuertas son elementos de un circuito electrónico que se emplean para construir la mayoría de las expresiones lógicas formales (expresiones booleanas elementales). Existen tres compuertas principales: OR, AND y NOT. De éstas compuertas se derivan otras como son las compuertas NAND, NOR, XOR (OR exclusiva) y XNOR (NOR exclusiva) ver Fig. 1.3.



Figura 1.3: Imagen creada por la inteligencia artificial Gemini para la sección temática de compuertas lógicas fundamentales, en la imagen se muestra el imaginario de las compuertas lógicas.

Las compuertas lógicas, AND ($\wedge$) o Conjunción, OR ($\vee$) o Disyunción, y NOT ($\neg$) Negación, son utilizadas en sistemas digitales (base de los circuitos lógicos), circuitos electrónicos (diseño de sistemas de control automático), Bases de datos (consultas booleanas), Programación (condiciones lógicas y control de flujo) o Inteligencia artificial (sistemas de inferencia). En los sistemas digitales las variables booleanas representan niveles de voltaje, el número cero (o 0 lógico) representa al falso, apagado o bajo de voltaje (un valor en voltios de 0-0.8V), y el número uno (o un 1 lógico) representa al verdadero, encendido o alto de voltaje (un valor en voltios de 2-5V) y con esto se establece la ecuación, ver Tabla 1.1.

Compuertas lógicas y su fundamento del diseño digital

Tabla 1.1: Operaciones definidas de las compuertas lógicas.

Compuerta	Ecuación	Salida X	Tabla de Verdad
AND ($\wedge$)	$X = A \cdot B$	Salida 1 solo si todas las entradas son 1.	$0 \cdot 0 = 0, 0 \cdot 1 = 0, 1 \cdot 0 = 0, 1 \cdot 1 = 1.$
OR ($\vee$)	$X = A + B$	Salida 1 si al menos una entrada es 1.	$0 + 0 = 0, 0 + 1 = 1, 1 + 0 = 1, 1 + 1 = 1.$
NOT ($\neg$)	$X = \neg A = \overline{A}$	Salida es el inverso de la entrada.	$0 \rightarrow 1, 1 \rightarrow 0.$
NAND	$X = \overline{A \cdot B}$	Inverso de AND.	$0 \cdot 0 = 0 \rightarrow 1, 0 \cdot 1 = 0 \rightarrow 1, 1 \cdot 0 = 0 \rightarrow 1, 1 \cdot 1 = 1 \rightarrow 0.$
NOR	$X = \overline{A + B}$	Inverso de OR.	$0 + 0 = 0 \rightarrow 1, 0 + 1 = 1 \rightarrow 0, 1 + 0 = 1 \rightarrow 0, 1 + 1 = 1 \rightarrow 0.$
XOR ($\oplus$)	$X = A \cdot \overline{B} + \overline{A} \cdot B$	Salida 1 cuando las entradas son diferentes.	$00 \rightarrow 0, 01 \rightarrow 1, 10 \rightarrow 1, 11 \rightarrow 0.$
XNOR	$X = A \cdot B + \overline{A} \cdot \overline{B}$	Salida 1 cuando las entradas son iguales.	$00 \rightarrow 1, 01 \rightarrow 0, 10 \rightarrow 0, 11 \rightarrow 1.$

De acuerdo con estas operaciones, se formaliza una conexión entre los principios de la electrónica y la abstracción del Álgebra de Boole, que es de importancia para el diseño en la ingeniería. Comprender la estructura de las compuertas lógicas consiste en comprender el concepto físico del transistor y su representación abstracta como un artefacto conocido como interruptor lógico. El transistor actúa como un interruptor que al llegar energía o recibir una variación de voltaje alto permite o niega el flujo de corriente. Al llegar un voltaje alto, se representa un estado de verdad o un 1 lógico (circuito cerrado), mientras que la ausencia de voltaje, se representa un estado falso o un 0 lógico (circuito abierto).

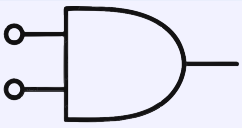
La compuerta lógica AND se la puede analizar desde diferentes ámbitos, como por ejemplo la multiplicación binaria o producto elemento a elemento (en álgebra lineal, al operar con vectores), la intersección (en teoría de conjuntos) o la conjunción (en lógica formal) (ver Fig. 1.4).

Compuerta lógica AND

La compuerta lógica AND realiza una multiplicación lógica, donde la salida es verdadera (1) únicamente cuando todas sus entradas son verdaderas (1).

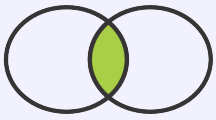
Tabla de verdad AND

A	B	Operación	Salida
0	0	$0 \cdot 0$	0
0	1	$0 \cdot 1$	0
1	0	$1 \cdot 0$	0
1	1	$1 \cdot 1$	1



$X=A \cdot B$

AND o Y
Multiplicación
de Componentes ($\cdot$)
Intersección ($\cap$)
Conjunción ($\wedge$)



En los lenguajes de programación se lo define como el ampersend & (uno) o && (dos seguidos).

Figura 1.4: Descripción de la información de la compuerta lógica AND

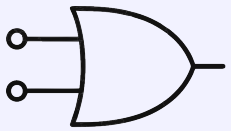
La compuerta lógica OR (OR Inclusiva u O inclusiva), cuya salida es 1 si al menos una entrada es 1, tiene analogías en diversas áreas de las matemáticas y la lógica. En la *Lógica Formal*, corresponde a la *disyunción* ($\vee$). En *Teoría de Conjuntos*, el operador OR es equivalente a la *Unión* ($\cup$) de conjuntos, recuerda que la unión de dos conjuntos $A \cup B$ incluye elementos que están en A o en B. Finalmente, en el *Álgebra Binaria y Lineal*, la compuerta lógica OR se modela como una *suma binaria con saturación*, donde el resultado de $1 + 1$ es forzado a 1 (a menudo representado como $\max(a, b)$ en operaciones componente a componente con vectores) (ver Fig. 1.5).

Compuerta lógica OR

La compuerta lógica NOT (Negación o Inversora), cuya salida es siempre el valor opuesto (inverso) de su única entrada.

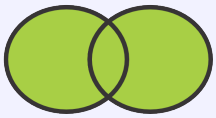
Tabla de verdad OR

A	B	Operación	Salida
0	0	$0+0$	0
0	1	$0+1$	1
1	0	$1+0$	1
1	1	$1+1$	1



X=A+B

OR o O
Sumar o juntar (+, ||)
Unión ($\cup$)
Disyunción ($\vee$)



En los lenguajes de programación se lo define como concatenar, unir o juntar.

Figura 1.5: Descripción de la información de la compuerta lógica OR

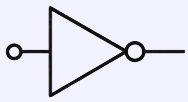
La compuerta lógica NOT, *NOT (Negación o Inversora)*, cuya salida es el valor opuesto de su única entrada, tiene analogías en diversas áreas de las matemáticas y la lógica. En la *Lógica Formal*, se corresponde con la *Negación* ($\neg$), donde si una proposición es verdadera, su negación es falsa, y viceversa. En *Teoría de Conjuntos*, el operador NOT es equivalente al *Complemento* ($\overline{A}$), representando todos los elementos que no pertenecen al conjunto A. Finalmente, en el *Álgebra Binaria y Aritmética*, la compuerta NOT se modela como la *resta de la unidad* ($1 - A$), realizando la inversión del valor (ver Fig. 1.6).

Compuerta lógica NOT

La compuerta lógica NOT realiza una inversión, donde la salida es siempre el opuesto lógico de su única entrada.

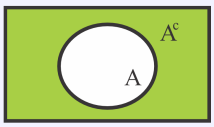
Tabla de verdad NOT

A	$\overline{A}$	Operación	Salida
0	1	$\overline{0}$ o $1-0$	1
1	0	$\overline{1}$ o $1-1$	0



X= ¬A

NOT o no
Negación (¬A, ~A o !A)
Complemento (A^c o $\overline{A}$)
Inversión o complemento a uno ($\overline{A}$ o A')
Resta de la unidad (1–A)



En lenguajes de programación se lo define como negar o invertir.

Figura 1.6: Descripción de la información de la compuerta lógica NOT

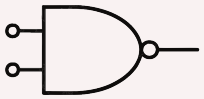
La compuerta lógica NAND (o AND negado), cuya salida es el inverso lógico de la compuerta AND, la salida es 0 solo cuando todas las entradas son 1. Se la puede analizar desde diferentes ámbitos, como por ejemplo la *negación del producto lógico* o *resta de la unidad al producto* (en álgebra binaria y lineal), la *negación de la intersección* (en teoría de conjuntos) o la *incompatibilidad* (en lógica formal) (ver Fig. 1.7).

Compuerta lógica NAND

La compuerta lógica NAND realiza la negación de la multiplicación lógica, donde la salida es falsa (0) únicamente cuando todas sus entradas son verdaderas (1).

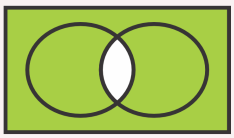
Tabla de verdad NAND

A	B	Operación	Salida
0	0	$\overline{0 \cdot 0}$	1
0	1	$\overline{0 \cdot 1}$	1
1	0	$\overline{1 \cdot 0}$	1
1	1	$\overline{1 \cdot 1}$	0



$X=\overline{A \cdot B}$

NAND
Incompatibilidad
(Negación de la conjunción)
$\neg(A \wedge B)$ o $A \uparrow B$
Complemento de la Intersección $\overline{(A \cap B)}$
Negación del Producto Lógico $\overline{(A \cdot B)}$
Resta de la Unidad del Producto $1-(A \cdot B)$



En lenguajes de programación se representa comúnmente como la negación de la conjunción: `!(A && B)`.

Figura 1.7: Descripción de la información de la compuerta lógica NAND

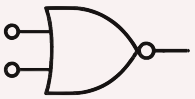
La compuerta lógica NOR (negación de la compuerta lógica OR), cuya salida es el inverso lógico de la compuerta OR, se puede analizar desde diferentes ámbitos, como por ejemplo la *negación de la suma lógica* o *resta de la unidad a la suma saturada* (en álgebra binaria y lineal), el *complemento de la unión* (en teoría de conjuntos) o la *negación de la disyunción* (en lógica formal) (ver Fig. 1.8).

Compuerta lógica NOR

La compuerta lógica NOR realiza la negación de la suma lógica, donde la salida es verdadera (1) únicamente cuando todas sus entradas son falsas (0).

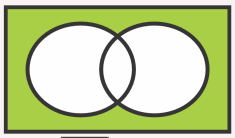
Tabla de verdad NOR

A	B	Operación	Salida
0	0	$\overline{0+0}$	1
0	1	$\overline{0+1}$	0
1	0	$\overline{1+0}$	0
1	1	$\overline{1+1}$	0



$X=\overline{A+B}$

NOR
Negación de la disyunción
(o Inclusión)
$\neg(A \vee B)$ o $A \downarrow B$
Complemento de la Unión $\overline{(A \cup B)}$
Negación de la Suma Lógica $\overline{(A+B)}$
Resta de la Unidad de la suma saturada $1-\max(A,B)$



Se representa comúnmente como la negación de la disyunción: `!(A || B)` o $A \downarrow B$ (flecha de Peirce).

Figura 1.8: Descripción de la información de la compuerta lógica NOR

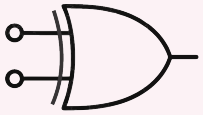
La compuerta lógica XOR (OR Exclusiva u O exclusiva), cuya salida es 1 si y solo si las entradas son diferentes, se puede analizar desde diferentes ámbitos, como por ejemplo la *suma módulo 2* o *detector de diferencia* (en álgebra binaria y lineal), la *diferencia simétrica* (en teoría de conjuntos) o la *disyunción exclusiva* (en lógica formal) (ver Fig. 1.9).

Compuerta lógica XOR

La compuerta lógica XOR realiza una suma binaria sin acarreo (módulo 2), donde la salida es verdadera (1) únicamente cuando las entradas tienen valores diferentes.

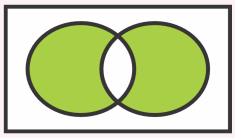
Tabla de verdad XOR

A	B	Operación	Salida
0	0	$0 \oplus 0$	0
0	1	$0 \oplus 1$	1
1	0	$1 \oplus 0$	1
1	1	$1 \oplus 1$	0



$X = A \oplus B$

XOR o O exclusivo
Símbolos ($\oplus$, $\veebar$, $\equiv/$)
Disyunción Exclusiva
(Antivalencia) $A \oplus B$ o $A/\leftrightarrow B$
Diferencia Simétrica
$(A \Delta B$ o $A \oplus B)$



Suma Módulo 2 (o Antivalencia) $A \oplus B$
Multiplicación y Suma Binaria $(A \cdot \overline{B}) + (\overline{A} \cdot B)$

En lenguajes de programación se lo representa con el símbolo de acento circunflejo: `A ^ B`, o en otros como `XOR` o símbolos como $\oplus$, $\veebar$, $\not\equiv$.

Figura 1.9: Descripción de la información de la compuerta lógica XOR

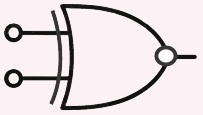
La compuerta lógica XNOR (OR Exclusiva Negada o Equivalencia), cuya salida es 1 si y solo si las entradas son iguales, se puede analizar desde diferentes ámbitos, como por ejemplo la *coincidencia* o *negación de la suma módulo 2* (en álgebra binaria y lineal), el *complemento de la diferencia simétrica* (en teoría de conjuntos) o la *equivalencia* o *bicondicional* (en lógica formal) (ver Fig. 1.10).

Compuerta lógica XNOR

La compuerta lógica XNOR realiza una coincidencia lógica, donde la salida es verdadera (1) únicamente cuando las entradas tienen el mismo valor lógico (ambas 0 o ambas 1).

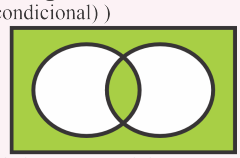
Tabla de verdad XNOR

A	B	Operación	Salida
0	0	$\overline{0 \oplus 0}$	1
0	1	$\overline{0 \oplus 1}$	0
1	0	$\overline{1 \oplus 0}$	0
1	1	$\overline{1 \oplus 1}$	1



$X = A \leftrightarrow B$

XNOR o O exclusivo negado
Símbolos ($\odot$, $\equiv$, $\leftrightarrow$ (bicondicional))
Bicondicional
o Equivalencia
$A \leftrightarrow B$ o $A \equiv B$
Complemento de
la Diferencia Simétrica
$\overline{A \Delta B}$



Coincidencia o Negación de la Suma Módulo 2
$\overline{A \oplus B}$ o $A \odot B$
Fórmula de Equivalencia $(A \cdot B) + (\overline{A} \cdot \overline{B})$

En lenguajes de programación se lo representa como la negación del operador XOR: `!(A ^ B)` o `XNOR`.

Figura 1.10: Descripción de la información de la compuerta lógica XNOR

1.1.3. Leyes y teoremas

George Boole nació en una familia de clase baja, donde su padre siendo zapatero le inculcó la motivación por las matemáticas y su aplicación a la construcción de dispositivos ópticos e instrumentos científicos. Sus padres fueron John Boole y Mary Ann Joyce, y siendo el primero de cuatro hermanos, fue en gran medida un autodidacta[1]. Sin una educación formal, tuvo que abordar los problemas de manera un poco convencional. Tuvo que elegir nuevos enfoques de estudio en vez de seguir los métodos tradicionales. Buscó la verdad en la analogía entre el álgebra numérica y la lógica simbólica, ver Fig. 1.11.

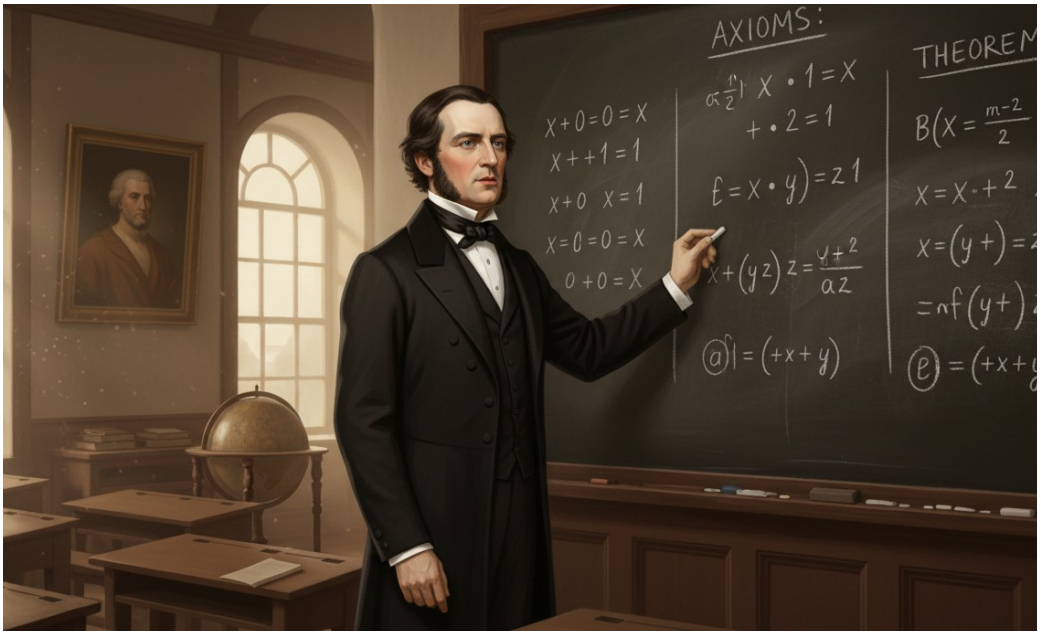


Figura 1.11: Imagen creada por la inteligencia artificial Gemini para la sección temática de leyes y teoremas, en la imagen se muestra imaginariamente a George Boole y los axiomas.

A partir de esta situación en su vida, su inventiva y genialidad no se limitó por la falta de un sistema educativo tradicional, sino que a su vez, se desarrolló a través de la exploración independiente y la disciplina personal. Ya en el siglo XIX, la manera de interpretar la lógica era mediante la filosofía. George Boole entendió que la lógica podía ser desarrollada e interpretada con métodos algebraicos.

Con esto escribió su obra *"Análisis Matemático de la Lógica"* (1847), un breve folleto en el que demostró que la lógica debía ser estudiada como una rama de las matemáticas. Esta idea evidenció su profundo pensamiento innovador sobre la universalidad de las estructuras matemáticas, que podían aplicarse a espacios que tradicionalmente se consideraban ajenos a ellas. George Boole estaba seguro de que la mente del ser humano opera siguiendo un conjunto de leyes formales, y su álgebra booleana era un intento de decodificar, precisar y formalizar esas *"leyes del pensamiento"*.

El álgebra booleana se enfoca al establecimiento de las bases de un proceso matemático teórico completo que permite describir la lógica binaria. Así, un axioma llega a ser un principio claro o una proposición que se considera evidente por sí misma sin necesidad de una demostración. Es la estructura conceptual base para una teoría, área de estudio o

[1] Biografía de George Boole

diseño de un sistema lógico. Para George Boole el principio de la dualidad establece que a partir de cualquier expresión booleana o ecuación de identidad mantiene su validez aun si se intercambian los operadores de la disyunción (OR o +) o de la conjunción (AND o ·), ver Tabla 1.2.

Leyes Fundamentales del Álgebra de Boole

Tabla 1.2: Axiomas y teoremas esenciales para la simplificación lógica.

Leyes	Operación Suma (+)	Operación Producto (·)
Identidad	$A + 0 = A$	$A \cdot 1 = A$
Complemento	$A + \overline{A} = 1$	$A \cdot \overline{A} = 0$
Dominación	$A + 1 = 1$	$A \cdot 0 = 0$
Idempotencia	$A + A = A$	$A \cdot A = A$
Conmutativas	$A + B = B + A$	$A \cdot B = B \cdot A$
Asociativas	$A + (B + C) = (A + B) + C$	$A \cdot (B \cdot C) = (A \cdot B) \cdot C$
Distributivas	$A \cdot (B + C) = A \cdot B + A \cdot C$	$A+(B \cdot C) = (A+B) \cdot (A+C)$
De Morgan	$(A + B)' = \overline{A} \cdot \overline{B}$	$(A \cdot B)' = \overline{A} + \overline{B}$
Doble Negación	$(\overline{\overline{A}}) = A$	

George Boole observó que los símbolos algebraicos para números seguían patrones que obedecen a ciertas leyes, como $x^2 = x$, una ley que en el álgebra numérica solo se cumple para el cero o el uno. Esta observación fue el punto importante para reducir las expresiones lógicas a un cálculo de solo dos valores. Él interpretó 1 como el *"universo"* (lo que contiene todo) y 0 como la *"nada"*. Esta intuición, que conectaba una propiedad algebraica numérica con la naturaleza binaria de la lógica, es un ejemplo significativo de su capacidad para ver conexiones profundas entre disciplinas aparentemente diferentes, ver Fig. 1.12.



Figura 1.12: Imagen creada por la inteligencia artificial Gemini para la sección temática de leyes y teoremas, en la imagen se muestra imaginariamente a George Boole y un contemporáneo.

Él sentó las bases para el concepto de bit en la era digital. Por eso los axiomas definen el comportamiento de las operaciones fundamentales anteriormente conocidas (OR, AND y NOT). Estos principios y las reglas de inferencia, representan el sistema axiomático a partir del cual, George Boole, deriva todos sus demás teoremas.

La importancia del trabajo de George Boole tuvo relevancia muchos años después de su muerte. Fue Claude Shannon, en 1938, quien aplicó el álgebra booleana a los circuitos de conmutación electrónica, lo que sentó las bases de la electrónica digital y la informática moderna. Él no estaba simplemente resolviendo un problema de su tiempo, estaba construyendo un marco teórico para un futuro tecnológico que aún no existía. Su pensamiento era tan fundamental y muy adelantado a su tiempo que su verdadero impacto se hizo evidente muchos años después. Su trabajo es un testimonio de vida y de la importancia de la investigación teórica fundamental, incluso cuando sus aplicaciones prácticas no son inmediatamente visibles en su tiempo.

1.1.4. Simplificación de expresiones booleanas

George Boole transformó la lógica (a partir de la filosofía), utilizando un sistema de expresiones algebraicas, permitiendo un análisis matemático formal de las proposiciones lógicas. Este desarrollo, el álgebra booleana, se convirtió en un sistema base de la informática moderna. A pesar de la admiración póstuma, sus contemporáneos no entendieron su trabajo revolucionario, que aplicaba a problemas matemáticos como el análisis de ecuaciones diferenciales y métodos de probabilidad, ver Fig. 1.13.



Figura 1.13: Imagen creada por la inteligencia artificial Gemini para la sección temática de simplificación de expresiones booleanas, en la imagen se muestran esculturas póstumas.

Boole cambió la forma de tratar la lógica, su artículo *"Sobre un método general de análisis"*, aplicó métodos algebraicos a las ecuaciones diferenciales, lo que le valió la Medalla Real de Matemáticas en 1844, la primera otorgada a un matemático. Este reconocimiento lo llevó a conseguir una cátedra en el Queen's College Cork. Boole no vivió

para ver el impacto de su obra. Décadas más tarde, Claude Shannon reconoció la aplicabilidad del álgebra booleana al diseño de circuitos eléctricos, sentando las bases para la era digital.

A partir de la aplicación de las leyes, teoremas y axiomas, George Boole propuso una forma de procesar las expresiones lógicas de tal manera que estas se puedan reducir a una forma más sencilla o equivalente, y con un número menor de términos o variables. La simplificación lógica se puede utilizar en dos ámbitos, en el diseño y fabricación de placas con circuitos digitales y en la programación.

En el primer ámbito, al diseñar circuitos con menos compuertas y con la misma función, se traduce en la disminución del costo de fabricación, el menor consumo de energía y en la reducción de la complejidad del diseño. En el segundo ámbito, al realizar programas, la simplificación permite reducir expresiones lógicas muy complejas en el código, a versiones más pequeñas y legibles, con buen rendimiento y eficientes. Existen varias formas para realizar la simplificación lógica, entre ellas están, el álgebra booleana y sus ecuaciones de equivalencia (reglas), mapas de Karnaugh (K-map o método gráfico) o el algoritmo de Quine-McCluskey (o método tabular), ver Tabla 1.3.

Leyes y teoremas fundamentales del Álgebra de Boole

Tabla 1.3: Compendio de las leyes y teoremas esenciales para la simplificación lógica.

Categoría / Ley	Operación Suma (OR: $+$)	Operación Producto (AND: $\cdot$)
I. Leyes axiomáticas y básicas		
Identidad	$A + 0 = A$	$A \cdot 1 = A$
Complemento (Inversión)	$A + \overline{A} = 1 \equiv A + A' = 1$	$A \cdot \overline{A} = 0 \equiv A \cdot A' = 0$
Dominación (Anulación)	$A + 1 = 1$	$A \cdot 0 = 0$
Idempotencia	$A + A = A$	$A \cdot A = A$
II. Leyes estructuras y negación		
Conmutativas	$A + B = B + A$	$A \cdot B = B \cdot A$
Asociativas	$A + (B + C) = (A + B) + C$	$A \cdot (B \cdot C) = (A \cdot B) \cdot C$
Distributivas	$A \cdot (B + C) = A \cdot B + A \cdot C$	$A + (B \cdot C) = (A + B) \cdot (A + C)$
Doble Negación	$\overline{\overline{A}} = A$, equivale escribir a $(A')' = A \equiv A'' = A$	
Negación de Constantes	$\overline{1} = 0$	$\overline{0} = 1$
Unicidad del Complemento	Si $A + X = 1$ y $A \cdot X = 0$, entonces $X = \overline{A}$	
III. Teoremas de simplificación		
De Morgan	$(A + B)' = \overline{A} \cdot \overline{B}$	$(A \cdot B)' = \overline{A} + \overline{B}$
Absorción	$A + A \cdot B = \mathbf{A}$	
Desarrollo	$A + A \cdot B = A \cdot (1 + B) = A \cdot 1 = \mathbf{A}$	
Absorción Dual	$A \cdot (A + B) = \mathbf{A}$	
Desarrollo	$A \cdot (A + B) = A \cdot A + A \cdot B = A + A \cdot B = \mathbf{A}$	
Simplificación Binomial (Unificación)	$A \cdot B + A \cdot B' = \mathbf{A}$	
Desarrollo	$A \cdot B + A \cdot B' = A \cdot (B + B') = A \cdot 1 = \mathbf{A}$	
Simplificación Binomial Dual	$(A + B) \cdot (A + B') = \mathbf{A}$	
Desarrollo	$(A + B) \cdot (A + B') = A + B \cdot B' = A + 0 = \mathbf{A}$	
Consenso (Eliminación)	$A \cdot B + A' \cdot C + B \cdot C = A \cdot B + A' \cdot C$	
Término Eliminado	El término $\mathbf{B \cdot C}$ es el consenso redundante de $A \cdot B$ y $A' \cdot C$.	

A continuación, se presenta un ejemplo de simplificación de una expresión booleana

siendo su expresión original:

$$F = ABC + ABC' + A'BC$$

Entonces su desarrollo y justificación sería el siguiente:

Paso 1: **Factorizar AB en los dos primeros términos.**

$$F = AB(C + C') + A'BC \qquad \text{(Factorización)}$$

Paso 2: **Aplicar la Ley del Complemento ($X + X' = 1$).**

$$F = AB(1) + A'BC \qquad \text{(Ley del Complemento: } C + C' = 1)$$

Paso 3: **Aplicar la Ley de Identidad ($X \cdot 1 = X$).**

$$F = AB + A'BC \qquad \text{(Ley de Identidad)}$$

Paso 4: **Factorizar B.**

$$F = B(A + A'C) \qquad \text{(Factorización)}$$

Paso 5: **Aplicar el Teorema de Absorción Extendido ($X + X'Y = X + Y$).**

$$F = B(A + C) \qquad \text{(Teorema de Absorción Extendido: } A + A'C = A + C)$$

Resultado simplificado:

$$\mathbf{F = B(A + C) = AB + BC}$$

A continuación, se presenta otro ejemplo de simplificación de una expresión booleana siendo su expresión original:

$$F = (A + B')(A + B)C' + A'$$

Entonces su desarrollo y justificación sería el siguiente (ver Tabla 1.4):

Ejemplo de simplificación de expresiones booleanas (Método algebraico)

Tabla 1.4: Simplificación de $F = (A + \overline{B})(A + B)\overline{C} + \overline{A}$

Paso	Expresión Resultante	Ley o Teorema Aplicado
Inicial	$F = (A + \overline{B})(A + B)\overline{C} + \overline{A}$	Expresión Original
1	$F = [A + (\overline{B} \cdot B)]\overline{C} + \overline{A}$	**Ley Distributiva** inversa: $(A + X)(A + Y) = A + (X \cdot Y)$
2	$F = [A + 0]\overline{C} + \overline{A}$	**Ley de Complemento** (AND): $B \cdot \overline{B} = 0$
3	$F = A \cdot \overline{C} + \overline{A}$	**Ley de Identidad** (OR): $A + 0 = A$
4	$F = \overline{A} + A\overline{C}$	**Ley Conmutativa** (OR): $X + Y = Y + X$
5	$F = (\overline{A} + A)(\overline{A} + \overline{C})$	**Ley Distributiva**: $X + (Y \cdot Z) = (X + Y)(X + Z)$
6	$F = (1)(\overline{A} + \overline{C})$	**Ley de Complemento** (OR): $\overline{A} + A = 1$
7	$F = \overline{A} + \overline{C}$	**Ley de Identidad** (AND): $1 \cdot X = X$

Resultado final simplificado:

$$\mathbf{F = \overline{A} + \overline{C}}$$

A continuación, se presenta otro ejemplo de simplificación de una expresión booleana, utilizando las leyes y teoremas fundamentales, siendo la expresión original es: $F = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$, entonces su desarrollo y justificación sería el siguiente (ver Tabla 1.5):

Ejemplo de simplificación de expresiones booleanas (Método algebraico)

Tabla 1.5: Simplificación de $F = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$

Paso	Expresión Resultante	Ley o Teorema Aplicado
Inicial	$F = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$	Expresión Original
1	$F = (\overline{A}BC + ABC) + (A\overline{B}C + ABC) + (AB\overline{C} + ABC)$	**Ley de Idempotencia**: ABC se duplica para agrupar (Consenso).
2	$F = (\overline{A} + A)BC + (\overline{B} + B)AC + (\overline{C} + C)AB$	**Ley Distributiva**: Factor común en cada par de términos.
3	$F = (1)BC + (1)AC + (1)AB$	**Ley de Complemento** (OR): $X + \overline{X} = 1$
4	$F = BC + AC + AB$	**Ley de Identidad** (AND): $X \cdot 1 = X$

Resultado final simplificado:

$$\mathbf{F = AC + AB + BC}$$

A continuación, se presenta otro ejemplo de simplificación de una expresión booleana, utilizando las leyes y teoremas fundamentales, siendo la expresión original es: $F = AB\overline{C} + ABC + AB\overline{D} + \overline{A}BC$, entonces su desarrollo y justificación sería el siguiente (ver Tabla 1.6):

Ejemplo de simplificación de expresiones booleanas (Método algebraico)

Tabla 1.6: Simplificación de $F = AB\overline{C} + ABC + AB\overline{D} + \overline{A}BC$

Paso	Expresión Resultante	Ley o Teorema Aplicado
Inicial	$F = AB\overline{C} + ABC + AB\overline{D} + \overline{A}BC$	Expresión Original
1	$F = AB(\overline{C} + C) + AB\overline{D} + \overline{A}BC$	**Ley Distributiva**: Factor común AB en los primeros dos términos.
2	$F = AB(1) + AB\overline{D} + \overline{A}BC$	**Ley de Complemento** (OR): $X + \overline{X} = 1$
3	$F = AB + AB\overline{D} + \overline{A}BC$	**Ley de Identidad** (AND): $X \cdot 1 = X$
4	$F = AB(1 + \overline{D}) + \overline{A}BC$	**Ley Distributiva**: Factor común AB en los dos primeros términos.
5	$F = AB(1) + \overline{A}BC$	**Ley de Dominación** (OR): $1 + X = 1$
6	$F = AB + \overline{A}BC$	**Ley de Identidad** (AND): $X \cdot 1 = X$

Resultado final simplificado:

$$\mathbf{F = AB + \overline{ABC}}$$

A continuación, se presenta otro ejemplo de simplificación de una expresión booleana, utilizando las leyes y teoremas fundamentales, siendo la expresión original: $F = \overline{A}B + AB + \overline{AB}C$, entonces su desarrollo y justificación sería el siguiente (ver Tabla 1.7):

Ejemplo de simplificación de expresiones booleanas (Método algebraico)

Tabla 1.7: Simplificación de $F = \overline{A}B + AB + \overline{AB}C$

Paso	Expresión Resultante	Ley o Teorema Aplicado
Inicial	$F = \overline{A}B + AB + \overline{AB}C$	Expresión Original (Reordenada por Conmutativa)
1	$F = B(\overline{A} + A) + \overline{AB}C$	**Ley Distributiva**: Factor común B en los primeros dos términos.
2	$F = B(1) + \overline{AB}C$	**Ley de Complemento** (OR): $\overline{A} + A = 1$
3	$F = B + \overline{AB}C$	**Ley de Identidad** (AND): $B \cdot 1 = B$
4	$F = B + \overline{A}C$	**Teorema de Absorción**: $X + \overline{X}Y = X + Y$. Aquí, $X = B$ y $Y = \overline{A}C$.

Resultado final simplificado:

$$\mathbf{F = B + \overline{A}C}$$

A continuación, se presenta otro ejemplo de simplificación de una expresión booleana, utilizando las leyes y teoremas fundamentales, siendo la expresión original: $F = A(\overline{A} + B) + B(\overline{B} + C)$, entonces su desarrollo y justificación sería el siguiente (ver Tabla 1.8):

Ejemplo de simplificación de expresiones booleanas (Método algebraico)

Tabla 1.8: Simplificación de $F = A(\overline{A} + B) + B(\overline{B} + C)$

Paso	Expresión Resultante	Ley o Teorema Aplicado
Inicial	$F = A(\overline{A} + B) + B(\overline{B} + C)$	Expresión Original
1	$F = A\overline{A} + AB + B\overline{B} + BC$	**Ley Distributiva**: $X(Y + Z) = XY + XZ$
2	$F = 0 + AB + 0 + BC$	**Ley de Complemento** (AND): $X\overline{X} = 0$
3	$F = AB + BC$	**Ley de Identidad** (OR): $X + 0 = X$
4 (Alternativa)	$F = B(A + C)$	**Ley Distributiva** (Factorización): $XY + XZ = X(Y + Z)$

Resultado final simplificado:

$$\mathbf{F = B(A + C)}$$

A continuación, se presenta otro ejemplo de simplificación de una expresión booleana, utilizando las leyes y teoremas fundamentales, siendo la expresión original: $F = AB + A\overline{B} + \overline{A}C$, entonces su desarrollo y justificación sería el siguiente (ver Tabla 1.9):

Ejemplo de simplificación de expresiones booleanas (Método algebraico)

Tabla 1.9: Simplificación de $F = AB + A\overline{B} + \overline{A}C$

Paso	Expresión Resultante	Ley o Teorema Aplicado
Inicial	$F = AB + A\overline{B} + \overline{A}C$	Expresión Original (Reordenada por Conmutativa si se desea)
1	$F = A(B + \overline{B}) + \overline{A}C$	**Ley Distributiva**: Factor común A en los dos primeros términos.
2	$F = A(1) + \overline{A}C$	**Ley de Complemento** (OR): $B + \overline{B} = 1$
3	$F = A + \overline{A}C$	**Ley de Identidad** (AND): $A \cdot 1 = A$
4	$F = A + C$	**Teorema de Absorción**: $X + \overline{X}Y = X + Y$. Aquí, $X = A$ y $Y = C$.

Resultado final simplificado:

$$\mathbf{F = A + C}$$

A continuación, se presenta otro ejemplo de simplificación de una expresión booleana, utilizando las leyes y teoremas fundamentales, siendo la expresión original: $F = AC + A\overline{C} + BC$, entonces su desarrollo y justificación sería el siguiente (ver Tabla 1.10):

Ejemplo de simplificación de expresiones booleanas (Método algebraico)

Tabla 1.10: Simplificación de $F = AC + A\overline{C} + BC$

Paso	Expresión Resultante	Ley o Teorema Aplicado
Inicial	$F = AC + A\overline{C} + BC$	Expresión Original (Reordenada por Conmutativa)
1	$F = A(C + \overline{C}) + BC$	**Ley Distributiva**: Factor común A en los dos primeros términos.
2	$F = A(1) + BC$	**Ley de Complemento** (OR): $C + \overline{C} = 1$
3	$F = A + BC$	**Ley de Identidad** (AND): $A \cdot 1 = A$
4	$F = A + C$	**Teorema de Absorción**: $X + YZ = (X + Y)(X + Z)$. Con $X = A$, $Y = B$, $Z = C$, se obtiene $A + BC = (A + B)(A + C)$ y simplificando: $F = A + C$.

Resultado final simplificado:

$$\mathbf{F = A + C}$$

A continuación, se presenta otro ejemplo de simplificación de una expresión booleana con cuatro variables, utilizando las leyes y teoremas fundamentales, siendo la expresión original: $(F = ABC + AB\overline{C} + A\overline{B}D + \overline{A}\overline{B}D)$, entonces su desarrollo y justificación sería el siguiente (ver Tabla 1.11):

Ejemplo de simplificación de expresiones booleanas (4 variables, método algebraico)

Tabla 1.11: Simplificación de $F = ABC + AB\overline{C} + A\overline{B}D + \overline{A}\overline{B}D$

Paso	Expresión Resultante	Ley o Teorema Aplicado
Inicial	$F = ABC + AB\overline{C} + A\overline{B}D + \overline{A}\overline{B}D$	Expresión Original (ordenada para agrupar términos semejantes)
1	$F = AB(C + \overline{C}) + D\left(A\overline{B} + \overline{A}\overline{B}\right)$	**Ley Distributiva**: factor común AB en los dos primeros términos y factor común D en los dos últimos.
2	$F = AB(1) + D\left(\overline{B}(A + \overline{A})\right)$	**Reagrupación y Distributiva**: extraer $\overline{B}$ en el segundo grupo.
3	$F = AB + D\left(\overline{B} \cdot 1\right)$	**Ley de Complemento**: $C + \overline{C} = 1$ y $A + \overline{A} = 1$.
4	$F = AB + D\overline{B}$	**Ley de Identidad** (AND): $\overline{B} \cdot 1 = \overline{B}$.

Resultado final simplificado:

$$\mathbf{F = AB + \overline{B}D}$$

Además se presenta la tabla de verdad resultante (ver Tabla 1.12):

Tabla de verdad de $F = AB + \overline{B}D$

Tabla 1.12: Tabla de verdad de $F(A, B, C, D)$ con $F = AB + \overline{B}D$

A	B	C	D	F
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Luego se procede a realizar el proceso de creación del mapa de Karnaugh del ejercicio.

Mapa de Karnaugh (4 variables) para F

	CD			
AB	00	01	11	10
00	0	1	1	0
01	0	0	0	0
11	1	1	1	1
10	0	1	1	0

En el mapa se identifican dos grupos mínimos:

- Toda la fila $AB = 11$, que corresponde al término AB.
- Un grupo de cuatro celdas con $B = 0$ y $D = 1$, que corresponde a $\overline{B}D$.

Por tanto la forma mínima es:

$$\mathbf{F = AB + \overline{B}D}.$$

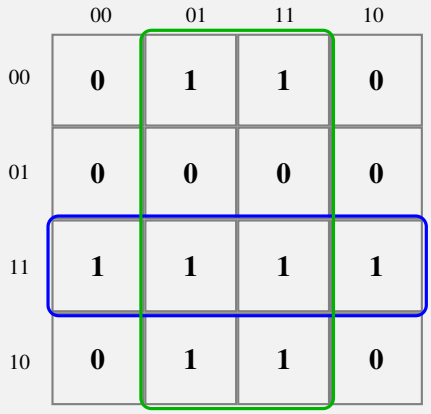
Nota sobre el consenso: El término de consenso entre AB y $\overline{B}D$ es AD. Sin embargo, este no reduce la expresión porque es redundante:

$$AB + \overline{B}D + AD = AB + \overline{B}D.$$

Por tanto la forma obtenida ya es mínima.

Luego se procede a generar el mapa de Karnaugh del ejercicio.

Mapa de Karnaugh con agrupamientos ($\mathbf{F = AB + \overline{B}D}$)



Resultado de la simplificación:

$$F = AB + \overline{B}D$$

A continuación, se presenta otro ejemplo siendo la expresión original: $F = AB + A\overline{B}C + \overline{A}BC + \overline{A}\overline{B}C$, entonces su desarrollo sería el siguiente (ver Tabla 1.13):

Ejemplo de simplificación de expresiones booleanas (Método algebraico)

Tabla 1.13: Simplificación de $F = AB + A\overline{B}C + \overline{A}BC + \overline{A}\overline{B}C$

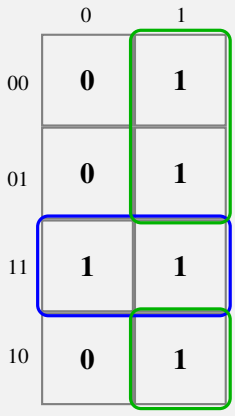
Paso	Expresión Resultante	Ley o Teorema Aplicado
Inicial	$F = AB + A\overline{B}C + \overline{A}BC + \overline{A}\overline{B}C$	Expresión Original
1	$F = AB + A\overline{B}C + C(\overline{A}B + \overline{A}\overline{B})$	**Ley Distributiva**: Factor común C en los últimos dos términos.
2	$F = AB + A\overline{B}C + C\overline{A}(B + \overline{B})$	**Ley Distributiva**: Factor común $\overline{A}$ dentro del paréntesis.
3	$F = AB + A\overline{B}C + C\overline{A}(1)$	**Ley de Complemento** (OR): $B + \overline{B} = 1$
4	$F = AB + A\overline{B}C + \overline{A}C$	**Ley de Identidad** (AND): $C\overline{A} \cdot 1 = \overline{A}C$
5	$F = AB + \overline{A}C + A\overline{B}C$	**Ley Conmutativa**: Reordenamiento de términos.
6	$F = AB + C(\overline{A} + A\overline{B})$	**Ley Distributiva**: Factor común C en los últimos dos términos.
7	$F = AB + C(\overline{A} + \overline{B})$	**Teorema de Absorción**: $X + XY = X$. Aquí, $\overline{A} + A\overline{B} = \overline{A} + \overline{B}$
8	$F = AB + C\overline{AB}$	**Ley de De Morgan**: $\overline{A} + \overline{B} = \overline{AB}$

Resultado final simplificado:

$$\mathbf{F = AB + C\overline{AB}}$$

En cambio, en este ejemplo se muestra la simplificación paso a paso de una expresión booleana de 4 términos, aplicando sistemáticamente las leyes distributiva, de complemento, de identidad, de absorción y de De Morgan para llegar a una forma más compacta con solo 2 términos. Luego generamos el mapa de Karnaugh del ejercicio.

Mapa de Karnaugh con agrupamientos ($\mathbf{F = AB + C\overline{AB}}$)



La simplificación $F = AB + C\overline{AB}$, tiene **agrupamientos** en azul, AB (fila AB=11, que cubre ambos valores de C) y en verde, $C\overline{AB}$ (columna C=1, filas AB=00, 01, 10)

A continuación, se presenta otro ejemplo siendo la expresión original: $F = \overline{A}BD + ABD + A\overline{B}D + A\overline{B}\overline{C}$, entonces su desarrollo sería el siguiente (ver Tabla 1.14):

Ejemplo de simplificación de expresiones booleanas (Método algebraico)

Tabla 1.14: Simplificación de $F = \overline{A}BD + ABD + A\overline{B}D + A\overline{B}\overline{C}$

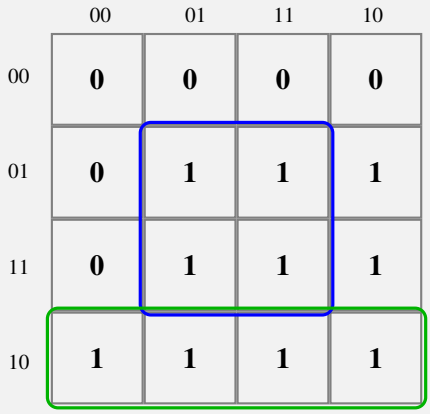
Paso	Expresión Resultante	Ley o Teorema Aplicado
Inicial	$F = \overline{A}BD + ABD + A\overline{B}D + A\overline{B}\overline{C}$	Expresión Original
1	$F = BD(\overline{A} + A) + A\overline{B}D + A\overline{B}\overline{C}$	**Ley Distributiva**: Factor común BD en los primeros dos términos.
2	$F = BD(1) + A\overline{B}D + A\overline{B}\overline{C}$	**Ley de Complemento** (OR): $\overline{A} + A = 1$
3	$F = BD + A\overline{B}D + A\overline{B}\overline{C}$	**Ley de Identidad** (AND): $BD \cdot 1 = BD$
4	$F = BD + A\overline{B}(D + \overline{C})$	**Ley Distributiva**: Factor común $A\overline{B}$ en los últimos dos términos.
5	$F = BD + A\overline{B}(\overline{C} + D)$	**Ley Conmutativa**: Reordenamiento dentro del paréntesis.
6	$F = BD + A\overline{B}\overline{C}\overline{D}$	**Ley de De Morgan**: $\overline{C} + D = \overline{C\overline{D}}$

Resultado final simplificado:

$$\mathbf{F = BD + A\overline{B}(\overline{C} + D)}$$

En este ejemplo se muestra la simplificación paso a paso de una expresión booleana de 4 términos, aplicando sistemáticamente las leyes distributiva, de complemento, de identidad, conmutativa y de De Morgan para llegar a una forma más compacta con 2 términos principales. Luego generamos el mapa de Karnaugh del ejercicio.

Mapa de Karnaugh con agrupamientos ($\mathbf{F = BD + A\overline{B}(\overline{C} + D)}$)



La simplificación $F = BD + A\overline{B}(\overline{C} + D)$, tiene **agrupamientos** en azul, BD (columnas CD=01 y CD=11 con B=1, filas AB=01, 11) y en verde, $A\overline{B}$ (fila AB=10, que cubre todos los valores de CD excepto CD=00)

A continuación, se presenta otro ejemplo de simplificación de una expresión booleana con cuatro variables, utilizando las leyes y teoremas fundamentales, siendo la expresión original: $F = \overline{A}\overline{B}C + \overline{A}BC + ABC + AB\overline{C}$, entonces su desarrollo y justificación sería el siguiente (ver Tabla 1.15):

Ejemplo de simplificación de expresiones booleanas (Método algebraico)

Tabla 1.15: Simplificación de $F = \overline{A}\overline{B}C + \overline{A}BC + ABC + AB\overline{C}$

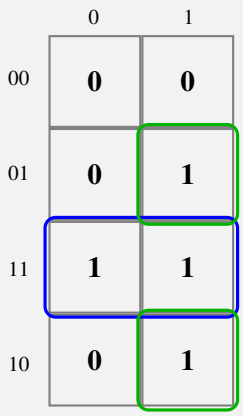
Paso	Expresión Resultante	Ley o Teorema Aplicado
Inicial	$F = \overline{A}\overline{B}C + \overline{A}BC + ABC + AB\overline{C}$	Expresión Original
1	$F = \overline{A}C(\overline{B} + B) + AB(C + \overline{C})$	**Ley Distributiva**: Factor común $\overline{A}C$ en los primeros dos términos y AB en los últimos dos.
2	$F = \overline{A}C(1) + AB(1)$	**Ley de Complemento** (OR): $\overline{B} + B = 1$ y $C + \overline{C} = 1$
3	$F = \overline{A}C + AB$	**Ley de Identidad** (AND): $\overline{A}C \cdot 1 = \overline{A}C$ y $AB \cdot 1 = AB$

Resultado final simplificado:

$$\mathbf{F = \overline{A}C + AB}$$

En este ejemplo se muestra la simplificación paso a paso de una expresión booleana de 4 términos, aplicando sistemáticamente las leyes distributiva, de complemento e identidad para llegar a una forma más compacta con solo 2 términos. Luego generamos el mapa de Karnaugh del ejercicio.

Mapa de Karnaugh con agrupamientos ($\mathbf{F = \overline{A}C + AB}$)



La simplificación $F = \overline{A}C + AB$, tiene **agrupamientos** en azul, AB (fila AB=11, que cubre ambos valores de C) y en verde, $\overline{A}C$ (columna C=1, filas AB=01, 10 donde A=0)

A continuación, se presenta otro ejemplo de simplificación de una expresión booleana con cuatro variables, utilizando las leyes y teoremas fundamentales, siendo la expresión original: $(F = A\overline{B}C + ABC + \overline{A}B\overline{C} + \overline{A}BC)$, entonces su desarrollo y justificación sería el siguiente (ver Tabla 1.16):

Ejemplo de simplificación de expresiones booleanas (4 variables, método algebraico)

Tabla 1.16: Simplificación de $F = A\overline{B}C + ABC + \overline{A}B\overline{C} + \overline{A}BC$

Paso	Expresión Resultante	Ley o Teorema Aplicado
Inicial	$F = A\overline{B}C + ABC + \overline{A}B\overline{C} + \overline{A}BC$	Expresión Original (ordenada para agrupar términos semejantes)
1	$F = AC(\overline{B} + B) + \overline{A}B(\overline{C} + C)$	**Ley Distributiva**: factor común AC en los dos primeros términos y factor común $\overline{A}B$ en los dos últimos.
2	$F = AC(1) + \overline{A}B(1)$	**Ley de Complemento** (OR): $\overline{B} + B = 1$ y $\overline{C} + C = 1$.
3	$F = AC + \overline{A}B$	**Ley de Identidad** (AND): $AC \cdot 1 = AC$ y $\overline{A}B \cdot 1 = \overline{A}B$.

Resultado final simplificado:

$$\mathbf{F = AC + \overline{A}B}$$

Además se presenta la tabla de verdad resultante (ver Tabla 1.17):

Tabla de verdad de $F = AC + \overline{A}B$

Tabla 1.17: Tabla de verdad de $F(A, B, C, D)$ con $F = AC + \overline{A}B$

A	B	C	D	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

Luego se procede a realizar el proceso de creación del mapa de Karnaugh del ejercicio.

Mapa de Karnaugh (4 variables) para F

	CD			
AB	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	1	1
10	0	0	1	1

En el mapa se identifican dos grupos mínimos:

- Toda la fila $AB = 01$ (donde $A = 0$ y $B = 1$), que corresponde al término $\overline{A}B$.
- Un grupo de cuatro celdas en las columnas $CD = 11$ y $CD = 10$ con $A = 1$ y $C = 1$, que corresponde a AC.

Por tanto la forma mínima es:

$$\mathbf{F = AC + \overline{A}B}.$$

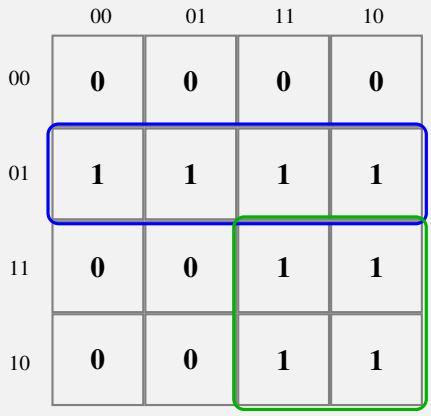
Nota sobre el consenso: El término de consenso entre AC y $\overline{A}B$ es BC. Sin embargo, este no reduce la expresión porque es redundante:

$$AC + \overline{A}B + BC = AC + \overline{A}B.$$

Por tanto la forma obtenida ya es mínima.

Luego se procede a generar el mapa de Karnaugh del ejercicio.

Mapa de Karnaugh con agrupamientos ($\mathbf{F = AC + \overline{A}B}$)



Resultado de la simplificación:

$$F = AC + \overline{A}B$$

A continuación, se presenta otro ejemplo de simplificación de una expresión booleana con cuatro variables, utilizando las leyes y teoremas fundamentales, siendo la expresión original: $(F = \overline{A}B\overline{C}D + \overline{A}B\overline{C}D + AB\overline{C}D + ABC\overline{D})$, entonces su desarrollo y justificación sería el siguiente (ver Tabla 1.18):

Ejemplo de simplificación de expresiones booleanas (4 variables, método algebraico)

Tabla 1.18: Simplificación de $F = \overline{A}B\overline{C}D + \overline{A}B\overline{C}D + AB\overline{C}D + ABC\overline{D}$

Paso	Expresión Resultante	Ley o Teorema Aplicado
Inicial	$F = \overline{A}B\overline{C}D + \overline{A}B\overline{C}D + AB\overline{C}D + ABC\overline{D}$	Expresión Original (ordenada para agrupar términos semejantes)
1	$F = \overline{C}D(\overline{A}B + \overline{A}B + AB) + ABC\overline{D}$	**Ley Distributiva**: factor común $\overline{C}D$ en los tres primeros términos.
2	$F = \overline{C}D\Big(\overline{A}(B + B) + AB\Big) + ABC\overline{D}$	**Ley Distributiva**: factor común $\overline{A}$ en los dos primeros términos del paréntesis.
3	$F = \overline{C}D\Big(\overline{A}(1) + AB\Big) + ABC\overline{D}$	**Ley de Complemento** (OR): $\overline{B} + B = 1$.
4	$F = \overline{C}D(\overline{A} + AB) + ABC\overline{D}$	**Ley de Identidad** (AND): $\overline{A} \cdot 1 = \overline{A}$.
5	$F = \overline{C}D(\overline{A} + B) + ABC\overline{D}$	**Teorema de Absorción**: $X + XY = X + Y$. Aquí, $\overline{A} + AB = \overline{A} + B$.
6	$F = \overline{C}D\overline{AB} + ABC\overline{D}$	**Ley de De Morgan**: $\overline{A} + B = \overline{A\overline{B}}$, equivalente a $\overline{AB}$ en forma simplificada.

Resultado final simplificado:

$$\mathbf{F = \overline{C}D(\overline{A} + B) + ABC\overline{D}}$$

Además se presenta la tabla de verdad resultante (ver Tabla 1.19):

Tabla de verdad de $F = \overline{C}D(\overline{A} + B) + ABC\overline{D}$

Tabla 1.19: Tabla de verdad de $F(A, B, C, D)$ con $F = \overline{C}D(\overline{A} + B) + ABC\overline{D}$

A	B	C	D	F
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Luego se procede a realizar el proceso de creación del mapa de Karnaugh del ejercicio.

Mapa de Karnaugh (4 variables) para F

	CD			
AB	00	01	11	10
00	0	1	0	0
01	0	1	0	0
11	0	1	0	1
10	0	0	0	0

En el mapa se identifican dos grupos mínimos:

- Un grupo vertical de tres celdas en la columna $CD = 01$ con filas $AB = 00, 01, 11$ (donde $C = 0$ y $D = 1$), que corresponde al término $\overline{C}D(\overline{A} + B)$.
- Una celda individual en $AB = 11, CD = 10$ (donde $A = 1, B = 1, C = 1, D = 0$), que corresponde a $ABC\overline{D}$.

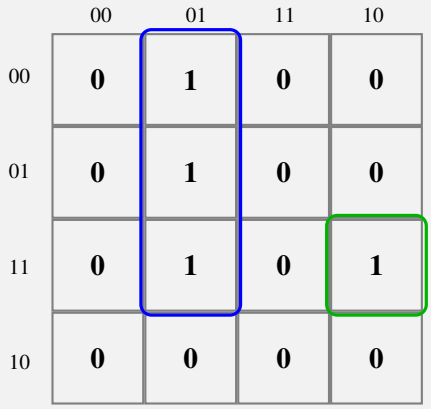
Por tanto la forma mínima es:

$$\mathbf{F = \overline{C}D(\overline{A} + B) + ABC\overline{D}}.$$

Nota: El término $\overline{C}D(\overline{A} + B)$ equivale a tres minitérminos: $\overline{A}\overline{B}\overline{C}D + \overline{A}B\overline{C}D + AB\overline{C}D$. La expresión ya se encuentra en su forma mínima.

Luego se procede a generar el mapa de Karnaugh del ejercicio.

Mapa de Karnaugh con agrupamientos ($\mathbf{F = \overline{C}D(\overline{A} + B) + ABC\overline{D}}$)



Resultado de la simplificación:

$$F = \overline{C}D(\overline{A} + B) + ABC\overline{D}$$

1.2. Teoría de proposiciones

Una proposición es un enunciado que se declara en forma cierta (verdadera) o no cierta (falsa), y con ello se introduce a la lógica proposicional. Con esto, las proposiciones se combinan mediante operadores lógicos, denotados con el caracter *"y"*, *"o"* y la palabra *"no"* para construir estructuras argumentativas y evaluar su validez. La lógica proposicional se enfoca en determinar si un razonamiento es correcto y para esto, se apoya en el conjunto de herramientas comúnmente utilizadas como son las tablas de verdad.

Es fundamental organizar y analizar la estructura argumentativa de tal forma que sea coherente. Un razonamiento es correcto o incorrecto dependiendo de su forma o estructura lógica independientemente de su contenido, a esto lo llamamos validez. La lógica proposicional es la base de las matemáticas, y con ello construir sus conceptos, leyes y teoremas. Aplicado a la computación, la lógica proposicional se enfoca en el estudio de los lenguajes formales y en la creación de diseños de sistemas lógicos, ver Fig. 1.14.



Figura 1.14: Imagen creada por la inteligencia artificial Gemini y su edición con el programa de Corel Draw 2025 para la sección temática teoría de proposiciones, en la imagen se muestra imaginariamente a George Boole, Aristóteles y Gottlob Frege.

En una oración declarativa, una proposición puede ser falsa o verdadera, pero no las dos al mismo tiempo. Para comprender esto se muestran los siguientes ejemplos: *"El cielo es azul"* (verdadero), *"El planeta Venus es plano"* (falso), a continuación se presenta un conjunto de ejemplos dispuestos por tipo de proposiciones y no proposiciones, se detalla además, su valor final de verdad o falsedad de acuerdo a la expresión o enunciado haciendo distinción un razonamiento correcto [Llerena Izquierdo, 2023a] (ver Tabla 1.20).

Asimismo, las proposiciones pueden clasificarse en dos tipos, simples (atómicas) y compuestas (moleculares). Las proposiciones simples no contienen otras proposiciones como componentes y se presentan con letras minúsculas, utilizando las ubicadas al final del abecedario, como p, q, r, s, t, u y siguientes. Mientras que las proposiciones compuestas se conforman combinando proposiciones simples y el uso de conectivos lógicos.

Ejemplos de proposiciones y no proposiciones

Tabla 1.20: Distinción entre proposiciones (con valor de verdad) y expresiones no proposicionales

¿Proposición?	Expresión o Enunciado	Valor / Razón
PROPOSICIONES (Se puede asignar V o F)		
Sí	"Quito es capital de Ecuador"	**Verdadero** (Hecho geográfico)
Sí	"Saturno es plano"	**Falso** (Hecho científico)
Sí	"El número 26 es impar"	**Falso** (Definición matemática)
Sí	"$6 \times 5 > 15$"	**Falso** (10 no es mayor que 15)
Sí	"En el año 2050, el mundo será perfecto"	**Falso** (Aseveración indemostrable y subjetiva)
NO PROPOSICIONES (No se puede asignar V o F)		
No	"¿Quién inventó el teléfono?"	**No aplica** (Pregunta o Interrogación)
No	"¡Cierra la puerta!"	**No aplica** (Imperativo u Orden)
No	"$x + 2 = 5$"	**No aplica** (Proposición Abierta, depende de x)
No	"Ojalá llueva mañana"	**No aplica** (Deseo o Exclamación)
No	"El color rojo corre más rápido"	**No aplica** (Juicio de valor/Ausencia de sentido lógico)

A continuación, presentamos unos ejemplos del tipo de conectivos lógicos fundamentales (binarios y unitarios) y sus tablas de verdad, definiendo sus símbolos formales para su lectura y utilización (ver Tabla 1.21).

Conectivos lógicos fundamentales y tablas de verdad

Tabla 1.21: Conectivos lógicos, resumen

Conectivo	Símbolo	Lectura	Definición	Tabla de Verdad
Operadores lógicos				
Negación	$\neg p$, $\sim p$	"no p"	$\neg p$ es **verdadera** cuando p es **falsa**, y viceversa.	V $\to$ F, F $\to$ V
Conjunción	$p \wedge q$	"p y q"	$p \wedge q$ es **verdadera** solo cuando **ambas** proposiciones son verdaderas.	VV $\to$ V, VF $\to$ F, FV $\to$ F, FF $\to$ F
Disyunción (Inclusiva)	$p \vee q$	"p o q"	$p \vee q$ es **verdadera** cuando al **menos una** proposición es verdadera.	VV $\to$ V, VF $\to$ V, FV $\to$ V, FF $\to$ F
Implicación	$p \to q$	"si p, entonces q"	$p \to q$ es **falsa** solo cuando p es **verdadera** (hipótesis) y q es **falsa** (conclusión).	VV $\to$ V, VF $\to$ F, FV $\to$ V, FF $\to$ V
Bicondicional	$p \leftrightarrow q$	"p si y solo si q"	$p \leftrightarrow q$ es **verdadera** cuando p y q tienen el **mismo** valor de verdad.	VV $\to$ V, VF $\to$ F, FV $\to$ F, FF $\to$ V
Disyunción Exclusiva	$p \oplus q$	"p o q, pero no ambos"	$p \oplus q$ es **verdadera** cuando **exactamente una** de las proposiciones es verdadera.	VV $\to$ F, VF $\to$ V, FV $\to$ V, FF $\to$ F

1.2.1. Introducción

La teoría de proposiciones lógicas, forma parte de una rama fundamental de las matemáticas conocida como la lógica formal. Esta se centra en el estudio del argumento razonado, teniendo la unidad fundamental en las proposiciones y las relaciones que se establecen entre ellas, abstraídas de su contenido semántico particular. Esta teoría es fundamentada a partir de los trabajos de filósofos y matemáticos de la antigüedad. Su formalización moderna y su desarrollo se dio a partir del siglo XIX y principios del XX, con figuras como Gottlob Frege y Bertrand Russell, que permitieron colocar las bases para la lógica matemática y, por extensión, la ciencia de la computación, ver Fig. 1.15.

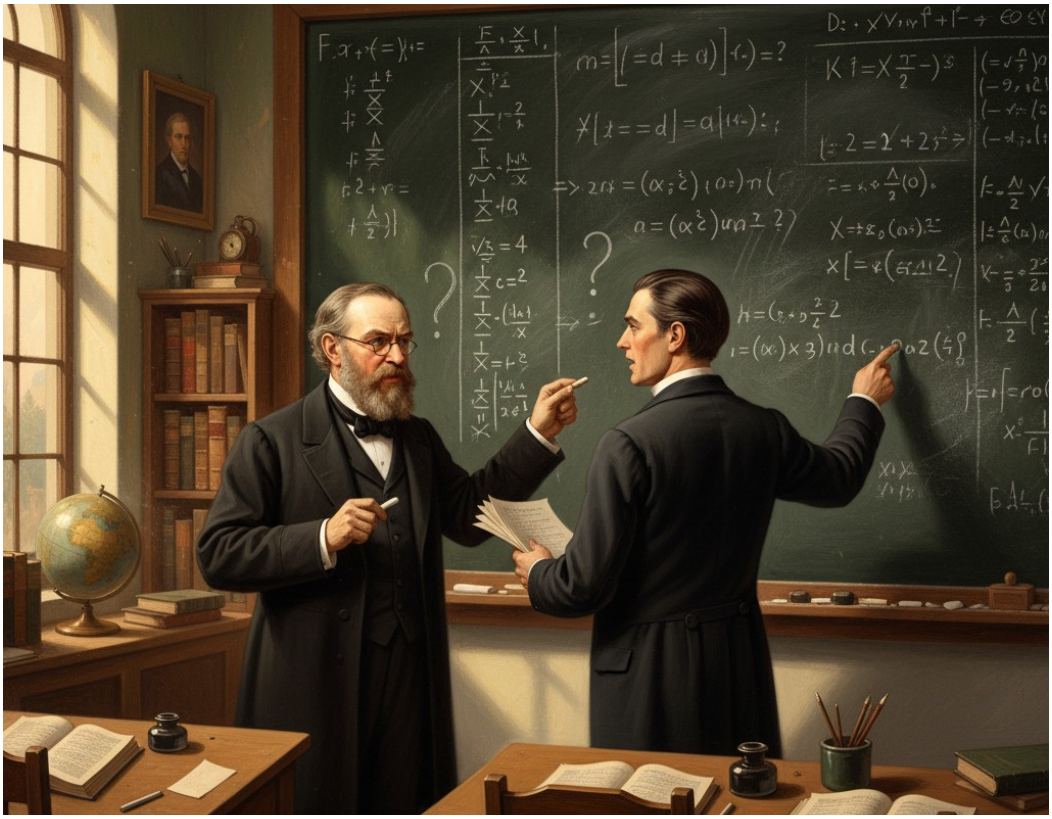


Figura 1.15: Imagen creada por la inteligencia artificial Gemini para la sección introducción, en la imagen se muestra imaginariamente a Gottlob Frege y Bertrand Russell.

En varias investigaciones actuales, en el ámbito del razonamiento computacional [Chajda & Laenger, 2025; Hu et al., 2025], han demostrado que, aunque la lógica proposicional clásica tiene limitaciones en su capacidad expresiva, la teoría proposicional sigue siendo el punto de partida para sistemas lógicos más complejos, incluyendo la lógica de predicados de primer orden. Así, la capacidad de esta teoría para modelar el razonamiento deductivo, tiene vigencia en la actualidad, y ha trascendido su origen filosófico para convertirse en una herramienta analítica indispensable en múltiples disciplinas [Llerena Izquierdo, 2023b].

El impacto de la lógica proposicional se manifiesta de manera profunda en las ciencias de la computación, donde su álgebra booleana constituye el cimiento de la arquitectura de los equipos y dispositivos (hardware), así como el diseño de circuitos digitales y los

lenguajes de programación (software) [Cao & Stepnicka, 2024; Llerena Izquierdo, 2020]. Los conectores lógicos ($\neg, \wedge, \vee$) corresponden directamente a las puertas lógicas (NOT, AND, OR), que son los componentes básicos de todos los sistemas electrónicos. Así, por ejemplo en su aplicación práctica, se evidencia en la verificación formal de sistemas, donde se utiliza para probar la corrección de diseños de hardware, garantizando que un circuito se comporte como se espera bajo todas las condiciones posibles [Fahmy-Hooke, 2024], ver Fig. 1.16.

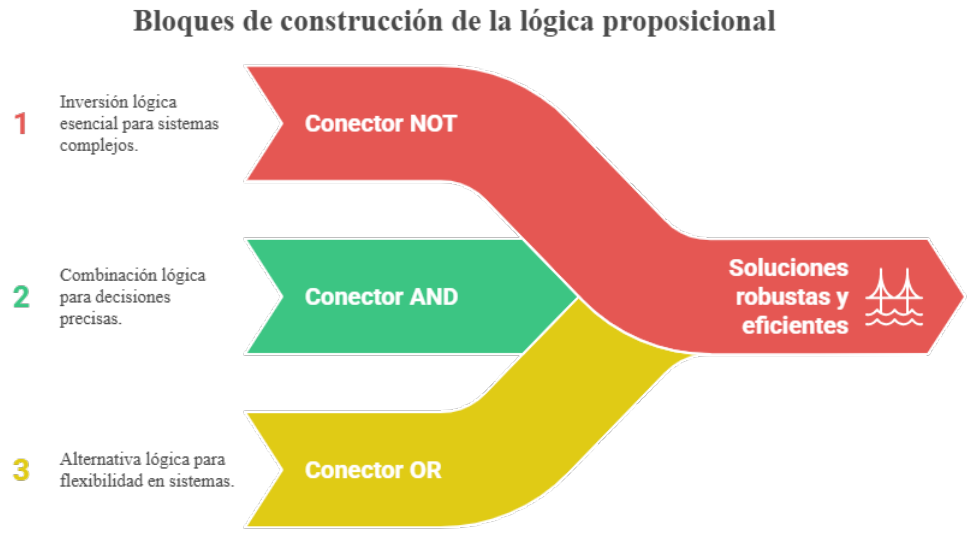


Figura 1.16: Imagen creada por la inteligencia artificial Napkin.ai para la sección introducción, en la imagen se muestra los bloques de construcción de la lógica.

Investigaciones de alto impacto, en la actualidad, destacan el uso de la lógica proposicional en el desarrollo de la inteligencia artificial, especialmente en los sistemas expertos y en la resolución de problemas lógicos dentro de los juegos, demostrando cómo su formalismo permite codificar el conocimiento humano para que sea procesado por máquinas [O'Sullivan, 2024]. El alcance actual de la lógica proposicional en la industria y la programación es amplio y a menudo de forma implícita. En la programación, las sentencias condicionales (if-then, if-else) y los bucles (while, for) se construyen directamente sobre los principios de la lógica proposicional [Faerna, 2024].

Los operadores lógicos en cualquier lenguaje de programación moderna son la base para la implementación directa de los conectivos proposicionales, permitiendo controlar el flujo de ejecución de los programas y tomar decisiones basadas en valores de verdad [Yue et al., 2024]. La utilidad del control de flujo, se extiende al diseño y optimización de algoritmos, a la creación de consultas de bases de datos que filtran información mediante expresiones lógicas y al desarrollo de sistemas de inteligencia artificial simbólica.

Además, la lógica proposicional ha trascendido a diversos ámbitos de estudio, donde su marco formal es utilizado para modelar y analizar razonamientos. En matemáticas, por ejemplo, es esencial para la construcción de demostraciones formales y la fundamentación de otras áreas como la teoría de conjuntos. En el campo de la inteligencia artificial, su aplicación es fundamental en el desarrollo de funciones de inferencia que, partiendo de un conjunto de hechos, patrones y reglas lógicas, generen nuevo conocimiento de manera algorítmica. Asimismo, en la ingeniería industrial, se utiliza para la optimización de

procesos, estableciendo relaciones lógicas entre variables para, por ejemplo, maximizar ganancias o minimizar costos en las líneas de producción, ver Fig. 1.17.

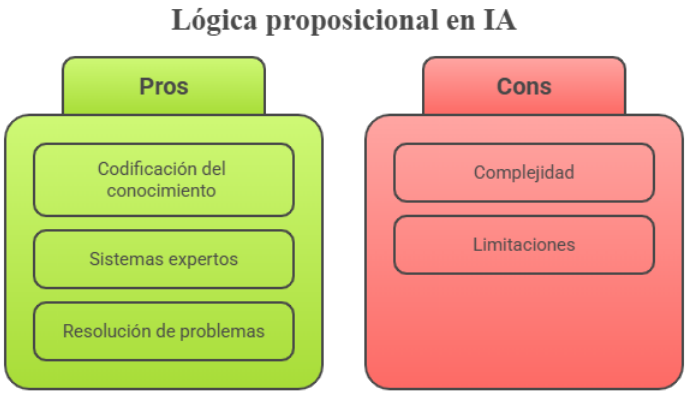


Figura 1.17: Imagen creada por la inteligencia artificial Napkin.ai para la sección introducción, en la imagen se muestra la lógica proposicional en IA.

La teoría de proposiciones lógicas ha evolucionado de un concepto filosófico a una herramienta computacional de análisis fundamental y de indiscutible relevancia en la era digital. Su impacto se evidencia en la estructura para la computación moderna, desde los circuitos más básicos hasta los sistemas de inteligencia artificial más avanzados [Cohen et al., 2024], ver Fig. 1.18.



Figura 1.18: Imagen creada por la inteligencia artificial Napkin.ai para la sección introducción, en la imagen se muestra la evolución de lógica proposicional.

Su alcance actual abarca la programación, la ingeniería y el desarrollo de software, donde sus principios rigen la toma de decisiones algorítmicas y la optimización de procesos. La trascendencia de esta teoría es tal que su formalismo lógico sigue siendo un pilar

indispensable para la innovación tecnológica y el desarrollo de nuevas áreas de investigación en la inteligencia artificial, la verificación de sistemas y la resolución de problemas complejos en una amplia gama de disciplinas.

1.2.2. Conectores proposicionales

En los últimos años, la industria de las tecnologías de información (TI) comparten soluciones y herramientas para verificar especificaciones formales de sistemas hardware y software. Se han dado cuenta del impacto y la importancia que tiene la investigación [Llerena-Izquierdo & Ayala-Carabajo, 2025b] para mejorar procesos de diseño e implementación que optimizan la planificación y la producción, ver Fig. 1.19.



Figura 1.19: Imagen del Centro de conocimiento e innovación deportiva del FC Barcelona que se enfoca a generar, compartir y aplicar conocimiento a través de proyectos de investigación.

Se requiere de formación en los conectivos lógicos para que los estudiantes tengan competencias en el uso, razonamiento y potenciación de sistemas (ver Tabla 1.22).

Conectivos lógicos fundamentales y tablas de verdad

Tabla 1.22: Conectivos lógicos proposicionales

Conectivo	Símbolo	Lectura	Definición	Tabla de Verdad
Operadores lógicos				
Negación	$\neg p$	"no p"	$\neg p$ es **verdadera** cuando p es **falsa**, y viceversa.	V $\rightarrow$ F, F $\rightarrow$ V
Conjunción o AND	$p \wedge q$	"p y q"	$p \wedge q$ es **verdadera** solo cuando **ambas** proposiciones son verdaderas.	VV $\rightarrow$ V, VF $\rightarrow$ F, FV $\rightarrow$ F, FF $\rightarrow$ F
Disyunción u OR	$p \vee q$	"p o q" (inclusivo)	$p \vee q$ es **verdadera** cuando al **menos una** proposición es verdadera.	VV $\rightarrow$ V, VF $\rightarrow$ V, FV $\rightarrow$ V, FF $\rightarrow$ F
Implicación	$p \rightarrow q$	"si p, entonces q"	$p \rightarrow q$ es **falsa** solo cuando p es **verdadera** (hipótesis) y q es **falsa** (conclusión).	VV $\rightarrow$ V, VF $\rightarrow$ F, FV $\rightarrow$ V, FF $\rightarrow$ V
Bicondicional o XNOR	$p \leftrightarrow q$	"p si y solo si q"	$p \leftrightarrow q$ es **verdadera** cuando p y q tienen el **mismo** valor de verdad.	VV $\rightarrow$ V, VF $\rightarrow$ F, FV $\rightarrow$ F, FF $\rightarrow$ V
Disyunción Exclusiva o XOR	$p \oplus q$	"p o q, pero no ambos"	$p \oplus q$ es **verdadera** cuando **exactamente una** de las proposiciones es verdadera.	VV $\rightarrow$ F, VF $\rightarrow$ V, FV $\rightarrow$ V, FF $\rightarrow$ F

1.2.3. Las tablas de verdad

Las tablas de verdad permiten determinar los posibles resultados de combinaciones de verdadero o falso de una estructura proposicional propuesta. Sea λ un lenguaje que contiene dos proposiciones: p y q. Si consideramos el principio de bivalencia, p se puede representar mediante la siguiente imagen (ver Fig. 1.20).

Conectivo Lógico NOT (Formato Binario)

Tabla de verdad NOT ($\neg p$)

p	Operación	¬ p
1	$\neg 1$	0
0	$\neg 0$	1

Figura 1.20: Conectivo lógico binario fundamental, puerta lógica NOT.

Además de las diferentes tablas de conectivos lógicos en formato binario (ver Fig. 1.21).

Conectivos Lógicos (Formato Binario)

Tabla de verdad AND ($p \wedge q$)

p	q	Operación	p ∧ q
1	1	$1 \wedge 1$	1
1	0	$1 \wedge 0$	0
0	1	$0 \wedge 1$	0
0	0	$0 \wedge 0$	0

Tabla de verdad NAND ($\neg(p \wedge q)$)

p	q	Operación	p NAND q
1	1	$\neg(1 \wedge 1)$	0
1	0	$\neg(1 \wedge 0)$	1
0	1	$\neg(0 \wedge 1)$	1
0	0	$\neg(0 \wedge 0)$	1

Tabla de verdad OR ($p \vee q$)

p	q	Operación	p ∨ q
1	1	$1 \vee 1$	1
1	0	$1 \vee 0$	1
0	1	$0 \vee 1$	1
0	0	$0 \vee 0$	0

Tabla de verdad NOR ($\neg(p \vee q)$)

p	q	Operación	p NOR q
1	1	$\neg(1 \vee 1)$	0
1	0	$\neg(1 \vee 0)$	0
0	1	$\neg(0 \vee 1)$	0
0	0	$\neg(0 \vee 0)$	1

Tabla de verdad Implicación ($p \rightarrow q$)

p	q	Operación	p → q
1	1	$1 \rightarrow 1$	1
1	0	$1 \rightarrow 0$	0
0	1	$0 \rightarrow 1$	1
0	0	$0 \rightarrow 0$	1

Tabla de verdad XOR ($p \oplus q$)

p	q	Operación	p ⊕ q
1	1	$1 \oplus 1$	0
1	0	$1 \oplus 0$	1
0	1	$0 \oplus 1$	1
0	0	$0 \oplus 0$	0

Tabla de verdad Bicondicional ($p \leftrightarrow q$)

p	q	Operación	p ↔ q
1	1	$1 \leftrightarrow 1$	1
1	0	$1 \leftrightarrow 0$	0
0	1	$0 \leftrightarrow 1$	0
0	0	$0 \leftrightarrow 0$	1

Tabla de verdad XNOR ($p \leftrightarrow q$ ó $\overline{p \oplus q}$)

p	q	Operación	p XNOR q
1	1	$1 \leftrightarrow 1$	1
1	0	$1 \leftrightarrow 0$	0
0	1	$0 \leftrightarrow 1$	0
0	0	$0 \leftrightarrow 0$	1

Figura 1.21: Conectivos lógicos binarios fundamentales, incluyendo las puertas lógicas (NAND, NOR, XOR, XNOR).

Las tablas de verdad de conectivos lógicos en formato lógico de verdadero y falso se representan como se muestra a continuación (ver Fig. 1.22).

Conectivos Lógicos (Formato Lógico)

Tabla de verdad Negación ($\neg$p)

p	Operación	$\neg$ p
V	$\neg$ V	F
F	$\neg$ F	V

Figura 1.22: Conectivo lógico fundamental (Verdadero/Falso), de la compuerta lógica NOT.

La tablas de verdad muestran en forma sistemática los valores de verdad de una proposición compuesta en función de todas las combinaciones posibles de los valores de verdad de las proposiciones que la componen (ver Fig. 1.23).

Conectivos Lógicos (Formato Lógico)

Tabla de verdad Conjunción ($p \wedge q$)

p	q	Operación	$p \wedge q$
V	V	$V \wedge V$	V
V	F	$V \wedge F$	F
F	V	$F \wedge V$	F
F	F	$F \wedge F$	F

Tabla de verdad NAND ($\neg(p \wedge q)$)

p	q	Operación	p NAND q
V	V	$\neg$ (V $\wedge$ V)	F
V	F	$\neg$ (V $\wedge$ F)	V
F	V	$\neg$ (F $\wedge$ V)	V
F	F	$\neg$ (F $\wedge$ F)	V

Tabla de verdad Disyunción ($p \vee q$)

p	q	Operación	$p \vee q$
V	V	$V \vee V$	V
V	F	$V \vee F$	V
F	V	$F \vee V$	V
F	F	$F \vee F$	F

Tabla de verdad NOR ($\neg(p \vee q)$)

p	q	Operación	p NOR q
V	V	$\neg$ (V $\vee$ V)	F
V	F	$\neg$ (V $\vee$ F)	F
F	V	$\neg$ (F $\vee$ V)	F
F	F	$\neg$ (F $\vee$ F)	V

Tabla de verdad Implicación ($p \rightarrow q$)

p	q	Operación	$p \rightarrow q$
V	V	$V \rightarrow V$	V
V	F	$V \rightarrow F$	F
F	V	$F \rightarrow V$	V
F	F	$F \rightarrow F$	V

Tabla de verdad XOR ($p \oplus q$)

p	q	Operación	$p \oplus q$
V	V	$V \oplus V$	F
V	F	$V \oplus F$	V
F	V	$F \oplus V$	V
F	F	$F \oplus F$	F

Tabla de verdad Bicondicional ($p \leftrightarrow q$)

p	q	Operación	$p \leftrightarrow q$
V	V	$V \leftrightarrow V$	V
V	F	$V \leftrightarrow F$	F
F	V	$F \leftrightarrow V$	F
F	F	$F \leftrightarrow F$	V

Tabla de verdad XNOR ($\overline{p \oplus q}$)

p	q	Operación	p XNOR q
V	V	$V \leftrightarrow V$	V
V	F	$V \leftrightarrow F$	F
F	V	$F \leftrightarrow V$	F
F	F	$F \leftrightarrow F$	V

Figura 1.23: Conectivos lógicos fundamentales (Verdadero/Falso), incluyendo las compuertas lógicas (NAND, NOR, XOR, XNOR).

Si analizamos las proposiciones p y q en conjunto, podremos observar que la tabla permite expresar todos los posibles valores de verdad de las mismas.

1.2.4. Equivalencias lógicas

El análisis de proposiciones requiere de formular las expresiones mediante un proceso de simplificación de los argumentos para obtener una conclusión (ver Tabla 1.23).

Leyes, Axiomas y Teoremas de la Teoría de Proposiciones

Tabla 1.23: Leyes y equivalencias lógicas fundamentales

Nombre de la Ley	Expresión Lógica	Ejemplo de Aplicación	Tipo
I. Leyes Fundamentales (Axiomas)			
Idempotencia	$p \land p \equiv p \quad p \lor p \equiv p$	$(p \lor r) \land (p \lor r) \equiv p \lor r$	Axioma
Doble Negación	$\neg(\neg p) \equiv p$	$\neg(\neg(q \lor r)) \equiv q \lor r$	Axioma
Conmutativa	$p \land q \equiv q \land p$ $p \lor q \equiv q \lor p$	$(p \lor r) \land q \equiv q \land (p \lor r)$	Axioma
Asociativa	$p \land (q \land r) \equiv (p \land q) \land r$	$p \lor (q \lor r) \equiv (p \lor q) \lor r$	Axioma
Distributiva	$p \land (q \lor r) \equiv$ $(p \land q) \lor (p \land r)$	$p \lor (q \land r) \equiv (p \lor q) \land (p \lor r)$	Axioma
II. Leyes de Identidad y Complemento			
Identidad	$p \land V \equiv p \quad p \lor F \equiv p$	$(p \land q) \land V \equiv p \land q$	Teorema
Dominación	$p \lor V \equiv V$ $p \land F \equiv F$	$(p \lor q) \lor V \equiv V$	Teorema
Complemento	$p \lor \neg p \equiv V$ (Tercero Excluido) $p \land \neg p \equiv F$ (No Contradicción)	$\neg(p \land q) \lor (p \land q) \equiv V$ $\neg(p \land q) \land (p \land q) \equiv F$	Teorema Teorema
III. Teoremas de Simplificación (Equivalencias)			
Absorción 1	$p \lor (p \land q) \equiv p$	$\neg q \lor (\neg q \land r) \equiv \neg q$	Teorema
Absorción 2	$p \land (\neg p \lor q) \equiv p \land q$	$p \land (\neg p \lor \neg r) \equiv p \land \neg r$	Teorema
De Morgan (Conj.)	$\neg(p \land q) \equiv \neg p \lor \neg q$	$\neg((\neg p) \land r) \equiv p \lor \neg r$	Equivalencia
De Morgan (Disy.)	$\neg(p \lor q) \equiv \neg p \land \neg q$	$\neg(\neg p \lor r) \equiv p \land \neg r$	Equivalencia
IV. Equivalencias de Conectivos (Artificios Lógicos)			
Implicación	$p \to q \equiv \neg p \lor q$	$(p \land q) \to r \equiv \neg(p \land q) \lor r$	Equivalencia
Bicondicional	$p \leftrightarrow q \equiv (p \to q) \land (q \to p)$	$p \leftrightarrow q \equiv (\neg p \lor q) \land (\neg q \lor p)$	Equivalencia
Contraposición	$p \to q \equiv \neg q \to \neg p$	$\neg p \to q \equiv \neg q \to p$	Equivalencia
V. Teoremas del Consenso (Resolución)			
Consenso	$(p \lor q) \land (\neg p \lor r) \land (q \lor r) \equiv (p \lor q) \land (\neg p \lor r)$	Si se tiene $A \lor B$ y $\neg A \lor C$, la cláusula $B \lor C$ es el consenso. $(p \lor q) \land (\neg p \lor r) \land (q \lor r)$ se reduce a $(p \lor q) \land (\neg p \lor r)$	Teorema
VI. Equivalencias de la Disyunción Exclusiva ($\oplus$)			
Disyunción Excl.	$p \oplus q \equiv (p \land \neg q) \lor (\neg p \land q)$	Se usa para eliminar el conectivo $\oplus$ y simplificar con $\land/\lor$.	Equivalencia
Negación de $\oplus$	$\neg(p \oplus q) \equiv p \leftrightarrow q$	La negación de la disyunción exclusiva es el bicondicional.	Equivalencia
VII. Reglas de Inferencia (Formas de Razonamiento)			
Modus Ponens	$[(p \to q) \land p] \to q$	$[(p \to q) \land p]$ puede ser reemplazado por q en la derivación.	Inferencia
Modus Tollens	$[(p \to q) \land \neg q] \to \neg p$	Si la conclusión es falsa, la hipótesis original debe ser falsa.	Inferencia
Silogismo Hipotético	$[(p \to q) \land (q \to r)] \to (p \to r)$	Si "A $\to$ B" y "B $\to$ C", entonces "A $\to$ C".	Inferencia
Simplificación	$(p \land q) \to p$	Si es verdad que $(p \land q)$, se puede inferir p como verdad.	Inferencia

1.2.5. Tipos de proposiciones compuestas

Las fórmulas contingentes, las contradicciones y las tautologías son conceptos muy importantes del Cálculo Proposicional Clásico (CPC). *Una tautología (Ley Lógica)* es una proposición compuesta que es siempre verdadera (V), sin importar los valores de verdad de las proposiciones simples que la componen. La columna final contiene solo valores Verdaderos (V) o 1. Representa una Ley Lógica cuya verdad es incuestionable en el sistema. Un ejemplo de ella es, la Ley del Tercero Excluido $p \vee \neg p$. *Una Contradicción (Falacia)* es una proposición compuesta que es siempre falsa (F), sin importar los valores de verdad de las proposiciones simples que la componen. La columna final contiene solo valores Falsos (F) o 0. Representa una imposibilidad o una inconsistencia lógica. Un ejemplo es, la Ley de no contradicción negada $p \wedge \neg p$. *Una Contingencia (Fórmula Neutra)* es una proposición compuesta cuyo valor de verdad depende de los valores de verdad de las proposiciones simples. No es ni una tautología ni una contradicción. La columna final contiene una mezcla de valores Verdaderos (V) y Falsos (F). Su verdad no está garantizada lógicamente; depende de las valoraciones de las fórmulas atómicas o proposiciones simples. Un ejemplo de ella es, Una simple implicación $p \to q$. Será V si p es F ($P \to F$). Será F si p es V y q es F ($V \to F$) (ver Tabla 1.24).

Clasificación Fundamental de Proposiciones Lógicas

Tabla 1.24: Comportamiento en la columna final de la tabla de verdad

Criterio	Tautología (✓)	Contradicción (×)	Contingencia (△)
Valor Final	**Siempre Verdadera (V)**	**Siempre Falsa (F)**	**Depende (V y F)**
Definición	Ley Lógica: Verdad garantizada.	Falacia Lógica: Falsedad garantizada.	Fórmula Neutra: Su verdad es circunstancial.
Composición T.V.	La columna final contiene **solo** valores **V** o **1**.	La columna final contiene **solo** valores **F** o **0**.	La columna final contiene **ambos** valores **V** y **F**.
Ejemplo	$p \vee \neg p$	$p \wedge \neg p$	$p \to q$

Entonces el análisis y la clasificación debe empezar primero por las proposiciones atómicas para establecer el valor final de la proposición compuesta, ver Fig. 1.24.

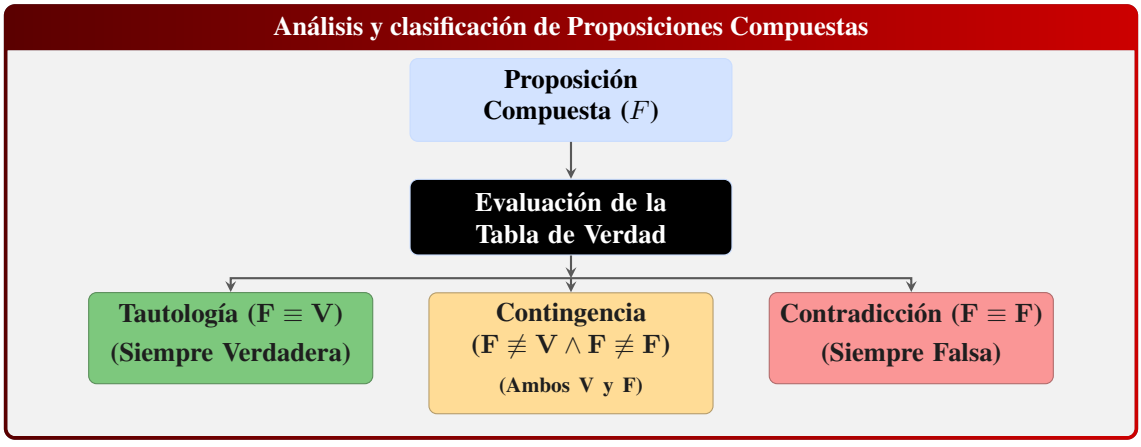


Figura 1.24: Diagrama de flujo que ilustra la clasificación de una proposición compuesta según el resultado de su tabla de verdad.

A continuación te presento un ejercicio donde se evidencia la evaluación de la proposición compuesta y su respuesta que es una **Contingencia** (ver Tabla 1.25).

Tabla de Verdad: Análisis de la Contingencia

Tabla 1.25: Tabla de verdad de $(p \wedge q) \vee (q \wedge \neg r)$

p	q	r	$p \wedge q$	$\neg r$	$q \wedge \neg r$	$(p \wedge q) \vee (q \wedge \neg r)$
V	V	V	V	F	F	**V**
V	V	F	V	V	V	**V**
V	F	V	F	F	F	F
V	F	F	F	V	F	F
F	V	V	F	F	F	F
F	V	F	F	V	V	**V**
F	F	V	F	F	F	F
F	F	F	F	V	F	F

A continuación te presento otro ejercicio donde se evidencia la evaluación de la proposición compuesta y su respuesta que es una **Contingencia** (ver Tabla 1.26).

Tabla de Verdad: Análisis de la Contingencia

Tabla 1.26: Tabla de verdad de $(\neg p \wedge q) \rightarrow r$

p	q	r	$\neg p$	$\neg p \wedge q$	$(\neg p \wedge q) \rightarrow r$
V	V	V	F	F	**V**
V	V	F	F	F	**V**
V	F	V	F	F	**V**
V	F	F	F	F	**V**
F	V	V	V	V	**V**
F	V	F	V	V	**F**
F	F	V	V	F	**V**
F	F	F	V	F	**V**

A continuación te presento otro ejercicio donde se evidencia la evaluación de la proposición compuesta y su respuesta que es una **Contingencia** (ver Tabla 1.27).

Tabla de Verdad: Análisis de la Contingencia

Tabla 1.27: Tabla de verdad de $(p \leftrightarrow q) \wedge \neg p$

p	q	$p \leftrightarrow q$	$\neg p$	$(p \leftrightarrow q) \wedge \neg p$
V	V	V	F	F
V	F	F	F	F
F	V	F	V	F
F	F	V	V	**V**

A continuación te presento otro ejercicio donde se evidencia la evaluación de la proposición compuesta y su respuesta que es una **Contingencia** (ver Tabla 1.28).

Tabla de Verdad: Análisis de la Contingencia

Tabla 1.28: Tabla de verdad de $(\neg p \vee q) \wedge \neg q$

p	q	$\neg p$	$\neg q$	$\neg p \vee q$	$(\neg p \vee q) \wedge \neg q$
V	V	F	F	V	F
V	F	F	V	F	F
F	V	V	F	V	F
F	F	V	V	V	V

A continuación te presento un ejercicio con cuatro variables proposicionales donde se evidencia la evaluación de la proposición compuesta y su respuesta que es una **Contingencia** (ver Tabla 1.29).

Tabla de Verdad: Contingencia con Cuatro Variables

Tabla 1.29: Tabla de verdad de $((p \wedge q) \rightarrow r) \leftrightarrow \neg s$

p	q	r	s	$p \wedge q$	$(p \wedge q) \rightarrow r$	$\neg s$	$((p \wedge q) \rightarrow r) \leftrightarrow \neg s$
V	V	V	V	V	V	F	F
V	V	V	F	V	V	V	V
V	V	F	V	V	F	F	V
V	V	F	F	V	F	V	F
V	F	V	V	F	V	F	F
V	F	V	F	F	V	V	V
V	F	F	V	F	V	F	F
V	F	F	F	F	V	V	V
F	V	V	V	F	V	F	F
F	V	V	F	F	V	V	V
F	V	F	V	F	V	F	F
F	V	F	F	F	V	V	V
F	F	V	V	F	V	F	F
F	F	V	F	F	V	V	V
F	F	F	V	F	V	F	F
F	F	F	F	F	V	V	V

A continuación te presento un ejercicio donde se evidencia la evaluación de la proposición compuesta y su respuesta que es una **Contingencia** (ver Tabla 1.30).

Tabla de Verdad: Análisis de la Contingencia

Tabla 1.30: Tabla de verdad de $p \rightarrow (q \wedge \neg r)$

p	q	r	$\neg r$	$q \wedge \neg r$	$p \rightarrow (q \wedge \neg r)$
V	V	V	F	F	F
V	V	F	V	V	V
V	F	V	F	F	F
V	F	F	V	F	F
F	V	V	F	F	V
F	V	F	V	V	V
F	F	V	F	F	V
F	F	F	V	F	V

A continuación te presento un ejercicio donde se evidencia la evaluación de la proposición compuesta y su respuesta que es una **Tautología** (ver Tabla 1.31).

Tabla de Verdad: Análisis de la Tautología

Tabla 1.31: Tabla de verdad de $(\neg p \wedge q) \rightarrow (p \rightarrow q)$

p	q	$\neg p$	$\neg p \wedge q$	$p \rightarrow q$	$(\neg p \wedge q) \rightarrow (p \rightarrow q)$
V	V	F	F	V	**V**
V	F	F	F	F	**V**
F	V	V	V	V	**V**
F	F	V	F	V	**V**

A continuación te presento otro ejercicio donde se evidencia la evaluación de la proposición compuesta y su respuesta que es una **Tautología** (ver Tabla 1.32).

Tabla de Verdad: Análisis de la Tautología (Modus Tollens)

Tabla 1.32: Tabla de verdad de $[(p \rightarrow q) \wedge \neg q] \rightarrow \neg p$

p	q	$\neg p$	$\neg q$	$p \rightarrow q$	$(p \rightarrow q) \wedge \neg q$	$[(p \rightarrow q) \wedge \neg q] \rightarrow \neg p$
V	V	F	F	V	F	**V**
V	F	F	V	F	F	**V**
F	V	V	F	V	F	**V**
F	F	V	V	V	V	**V**

A continuación te presento un ejercicio con tres variables que evidencia la evaluación de la proposición compuesta y su respuesta, que es una **Tautología** (ver Tabla 1.33).

Tabla de Verdad: Análisis de la Tautología (Principio de Resolución)

Tabla 1.33: Tabla de verdad de $[(\neg p \vee q) \wedge (p \vee r)] \rightarrow (q \vee r)$

p	q	r	$\neg p$	$\neg p \vee q$	$p \vee r$	$(\neg p \vee q) \wedge (p \vee r)$	$[(\neg p \vee q) \wedge (p \vee r)] \rightarrow (q \vee r)$
V	V	V	F	V	V	V	**V**
V	V	F	F	V	V	V	**V**
V	F	V	F	F	V	F	**V**
V	F	F	F	F	V	F	**V**
F	V	V	V	V	V	V	**V**
F	V	F	V	V	F	F	**V**
F	F	V	V	V	V	V	**V**
F	F	F	V	V	F	F	**V**

A continuación te presento un ejercicio con dos variables que evidencia la evaluación de la proposición compuesta y su respuesta, que es una **Tautología** (ver Tabla 1.34).

Tabla de Verdad: Análisis de la Tautología (De Morgan Condicional)

Tabla 1.34: Tabla de verdad de $\neg(p \vee q) \rightarrow (\neg p \wedge \neg q)$

p	q	$p \vee q$	$\neg(p \vee q)$	$\neg p$	$\neg q$	$\neg p \wedge \neg q$	$\neg(p \vee q) \rightarrow (\neg p \wedge \neg q)$
V	V	V	F	F	F	F	**V**
V	F	V	F	F	V	F	**V**
F	V	V	F	V	F	F	**V**
F	F	F	V	V	V	V	**V**

A continuación te presento un ejercicio con dos variables que evidencia la evaluación de la proposición compuesta y su respuesta, que es una **Contradicción** (ver Tabla 1.35).

Tabla de Verdad: Análisis de la Contradicción

Tabla 1.35: Tabla de verdad de $p \wedge \neg p$

p	$\neg p$	$p \wedge \neg p$
V	F	F
F	V	F

A continuación te presento otro ejercicio con dos variables que evidencia la evaluación de la proposición compuesta y su respuesta, que es una **Contradicción** (ver Tabla 1.36).

Tabla de Verdad: Análisis de la Contradicción

Tabla 1.36: Tabla de verdad de $(p \leftrightarrow q) \wedge (p \oplus q)$

p	q	$p \leftrightarrow q$	$p \oplus q$	$(p \leftrightarrow q) \wedge (p \oplus q)$
V	V	V	F	F
V	F	F	V	F
F	V	F	V	F
F	F	V	F	F

A continuación te presento un ejercicio con tres variables que evidencia la evaluación de la proposición compuesta y su respuesta, que es una **Contradicción** (ver Tabla 1.37).

Tabla de Verdad: Análisis de la Contradicción

Tabla 1.37: Tabla de verdad de $(p \wedge q) \wedge \neg(p \wedge r)$

p	q	r	$p \wedge q$	$p \wedge r$	$\neg(p \wedge r)$	$(p \wedge q) \wedge \neg(p \wedge r)$
V	V	V	V	V	F	F
V	V	F	V	F	V	F
V	F	V	F	V	F	F
V	F	F	F	F	V	F
F	V	V	F	F	V	F
F	V	F	F	F	V	F
F	F	V	F	F	V	F
F	F	F	F	F	V	F

A continuación te presento un ejercicio con dos variables que evidencia la evaluación de la proposición compuesta y su respuesta, que es una **Contradicción** (ver Tabla 1.38).

Tabla de Verdad: Análisis de la Contradicción

Tabla 1.38: Tabla de verdad de $(p \rightarrow q) \wedge (p \wedge \neg q)$

p	q	$p \rightarrow q$	$\neg q$	$p \wedge \neg q$	$(p \rightarrow q) \wedge (p \wedge \neg q)$
V	V	V	F	F	F
V	F	F	V	V	F
F	V	V	F	F	F
F	F	V	V	F	F

A continuación te presento un ejercicio con cuatro variables que evidencia la evaluación de la proposición compuesta y su respuesta, que es una **Contradicción** (ver Tabla 1.39).

Tabla de Verdad: Contradicción con Cuatro Variables

Tabla 1.39: Tabla de verdad de $(p \wedge q \wedge r \wedge s) \wedge \neg(p \wedge q \wedge r \wedge s)$

p	q	r	s	$P = (p \wedge q \wedge r \wedge s)$	$\neg P$	$P \wedge \neg P$
V	V	V	V	V	F	F
V	V	V	F	F	V	F
V	V	F	V	F	V	F
V	V	F	F	F	V	F
V	F	V	V	F	V	F
V	F	V	F	F	V	F
V	F	F	V	F	V	F
V	F	F	F	F	V	F
F	V	V	V	F	V	F
F	V	V	F	F	V	F
F	V	F	V	F	V	F
F	V	F	F	F	V	F
F	F	V	V	F	V	F
F	F	V	F	F	V	F
F	F	F	V	F	V	F
F	F	F	F	F	V	F

A continuación te presento un ejercicio con cinco variables que evidencia la evaluación de la proposición compuesta y su respuesta, que es una **Contradicción** (ver Tabla 1.40).

Tabla de Verdad: Contradicción con Cinco Variables

Tabla 1.40: Tabla de verdad de $(p \wedge q \wedge r \wedge s \wedge t) \wedge \neg(p \wedge q \wedge r \wedge s \wedge t)$

p	q	r	s	t	$P = (\cdots \wedge t)$	$\neg P$	$P \wedge \neg P$
V	V	V	V	V	V	F	F
V	V	V	V	F	F	V	F
V	V	V	F	V	F	V	F
V	V	V	F	F	F	V	F
V	V	F	V	V	F	V	F
V	V	F	V	F	F	V	F
V	V	F	F	V	F	V	F
V	V	F	F	F	F	V	F
V	F	V	V	V	F	V	F
V	F	V	V	F	F	V	F
V	F	V	F	V	F	V	F
V	F	V	F	F	F	V	F
V	F	F	V	V	F	V	F
V	F	F	V	F	F	V	F
V	F	F	F	V	F	V	F
V	F	F	F	F	F	V	F
F	V	V	V	V	F	V	F
F	V	V	V	F	F	V	F
F	V	V	F	V	F	V	F
F	V	V	F	F	F	V	F
F	V	F	V	V	F	V	F
F	V	F	V	F	F	V	F
F	V	F	F	V	F	V	F
F	V	F	F	F	F	V	F
F	F	V	V	V	F	V	F
F	F	V	V	F	F	V	F
F	F	V	F	V	F	V	F
F	F	V	F	F	F	V	F
F	F	F	V	V	F	V	F
F	F	F	V	F	F	V	F
F	F	F	F	V	F	V	F
F	F	F	F	F	F	V	F

A continuación te presento un ejercicio donde la proposición compuesta se extrae de una situación cotidiana, y su evaluación final resulta ser una **Contingencia** (ver Tabla 1.41).

1. Proposición Narrativa: La proposición compuesta se extrae de la siguiente situación cotidiana: *"Si hoy es lunes y no he tenido una reunión, entonces me iré a casa temprano."*
2. Definición de Variables: Para la formalización, se definen las siguientes variables, p: Hoy es lunes; q: He tenido una reunión; r: Me iré a casa temprano.
3. Lenguaje Formal: Con estas definiciones, la proposición se traduce al lenguaje formal como $(\mathbf{p} \wedge \neg \mathbf{q}) \rightarrow \mathbf{r}$.

Tabla de Verdad: Análisis de la Proposición Narrativa (Contingencia)

Tabla 1.41: Tabla de verdad de $(p \wedge \neg q) \rightarrow r$

p	q	r	$\neg q$	$p \wedge \neg q$	$(p \wedge \neg q) \rightarrow r$
V	V	V	F	F	**V**
V	V	F	F	F	**V**
V	F	V	V	V	**V**
V	F	F	V	V	F
F	V	V	F	F	**V**
F	V	F	F	F	**V**
F	F	V	V	F	**V**
F	F	F	V	F	**V**

A continuación te presento un ejercicio donde la proposición compuesta se extrae de una situación cotidiana y su evaluación final resulta ser una **Tautología** (ver Tabla 1.42).

1. Proposición Narrativa: La proposición compuesta se extrae de la siguiente situación cotidiana: *"Si hay tráfico en la ciudad, entonces la universidad tendrá clases o la universidad no tendrá clases."*
2. Definición de Variables: Para la formalización, se definen las siguientes variables: p: La universidad tendrá clases; q: Hay tráfico en la ciudad.
3. Lenguaje Formal: Con estas definiciones, la proposición se traduce al lenguaje formal como $\mathbf{q} \rightarrow (\neg \mathbf{p} \vee \mathbf{p})$.

Tabla de Verdad: Análisis de la Proposición Narrativa (Tautología)

Tabla 1.42: Tabla de verdad de $q \rightarrow (\neg p \vee p)$

p	q	$\neg p$	$\neg p \vee p$	$q \rightarrow (\neg p \vee p)$
V	V	F	**V**	**V**
V	F	F	**V**	**V**
F	V	V	**V**	**V**
F	F	V	**V**	**V**

A continuación te presento un ejercicio donde la proposición compuesta se extrae de una situación cotidiana y su evaluación final resulta ser una **Contradicción** (ver Tabla 1.43).

1. Proposición Narrativa: La proposición compuesta se extrae de la siguiente situación cotidiana: *"Hoy no estoy de vacaciones y estoy trabajando, y al mismo tiempo no estoy trabajando."*
2. Definición de Variables: Para la formalización, se definen las siguientes variables: p: Estoy trabajando; q: Estoy de vacaciones.
3. Lenguaje Formal: Con estas definiciones, la proposición se traduce al lenguaje formal como $(\neg \mathbf{q} \land \mathbf{p}) \land \neg \mathbf{p}$.

Tabla de Verdad: Análisis de la Proposición Narrativa (Contradicción)

Tabla 1.43: Tabla de verdad de $(\neg q \land p) \land \neg p$

p	q	$\neg q$	$\neg p$	$\neg q \land p$	$(\neg q \land p) \land \neg p$
V	V	F	F	F	F
V	F	V	F	V	F
F	V	F	V	F	F
F	F	V	V	F	F

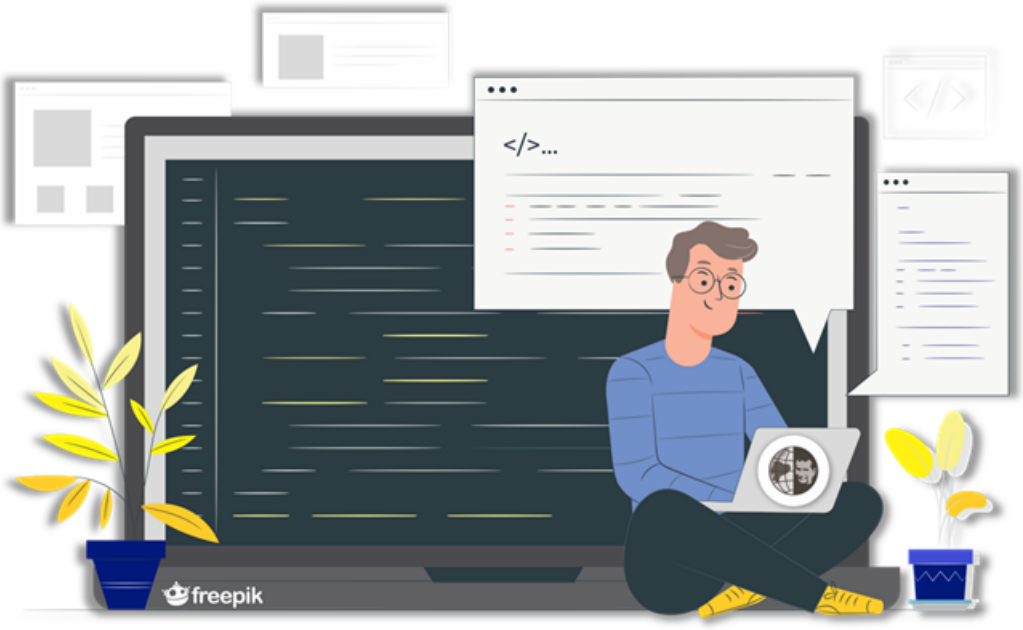
A continuación te presento un ejercicio donde la proposición compuesta se extrae de una situación cotidiana y su evaluación final resulta ser una **Contingencia** (ver Tabla 1.44).

1. Proposición Narrativa: La proposición compuesta se extrae de la siguiente situación cotidiana: *"Si hace sol o tenemos tiempo libre, entonces iremos a la playa."*
2. Definición de Variables: Para la formalización, se definen las siguientes variables: p: Hace sol; q: Tenemos tiempo libre; r: Iremos a la playa.
3. Lenguaje Formal: Con estas definiciones, la proposición se traduce al lenguaje formal como $(\mathbf{p} \lor \mathbf{q}) \to \mathbf{r}$.

Tabla de Verdad: Análisis de la Proposición Narrativa (Contingencia)

Tabla 1.44: Tabla de verdad de $(p \lor q) \to r$

p	q	r	$p \lor q$	$(p \lor q) \to r$
V	V	V	V	**V**
V	V	F	V	F
V	F	V	V	**V**
V	F	F	V	F
F	V	V	V	**V**
F	V	F	V	F
F	F	V	F	**V**
F	F	F	F	**V**



Laboratorio 1

Capítulo 1

1.3. Ejercicios del Laboratorio 1

A continuación empezamos con los ejercicios del laboratorio 1 de las prácticas de experimentación del capítulo 1.

1.3.1. Sección 1 del laboratorio 1

A continuación, responde las siguientes preguntas luego de observar y comparar las diferentes respuestas existentes para consolidar los conceptos fundamentales.

Para la **pregunta 1. ¿Qué es el Álgebra de Boole?**, te presento un conjunto de resúmenes que debes comparar y analizar para escribir tu respuesta, ver Tabla 1.45.

Análisis consolidado a la pregunta ¿Qué es el Álgebra de Boole?

Tabla 1.45: Comparativa de la utilidad del Álgebra Booleana en diferentes artefactos

Fuente	Respuestas	Explicación de utilidad y propósito
Gemini	Opera con 0 (Falso) y 1 (Verdadero). Utiliza las operaciones básicas AND ($\wedge$), OR ($\vee$) y NOT ($\neg$).	Es el *sistema matemático fundamental* de la lógica digital y la informática. Su importancia radica en que sus operaciones se implementan en *compuertas lógicas* y el *hardware* de la computadora.
Claude	Opera con **0 / 1** e incluye operaciones: AND ($\cdot$), OR ($+$), NOT ($\overline{A}$), además de Leyes de De Morgan y de Complemento, etc.	Es la *base fundamental* de la computación digital moderna. Se utiliza para *diseñar y optimizar circuitos* (procesadores, memorias) y escribir programas mediante expresiones lógicas.
Copilot	Trabaja con valores *Binarios* (**0** y **1**). Sus operaciones (AND, OR, NOT) se basan en la teoría de la electrónica digital.	Constituye la *base teórica* de la electrónica digital. Clave para la creación de *compuertas lógicas*, tablas de verdad y la *simplificación de circuitos* y lógica de sistemas.
Deepseek	Manipula variables con **1/0** (V/F) a partir de las operaciones lógicas fundamentales: AND, OR, NOT.	Rama matemática base del diseño de *circuitos digitales*, *arquitectura de computadoras* y *programación*, permitiendo modelar y simplificar la lógica del hardware y software.
ChatGPT	Trabaja con **0** y **1** (Falso o Verdadero, Apagado/Encendido), utilizando las operaciones lógicas básicas (AND, OR, NOT).	Su importancia radica en que constituye la *base del diseño de circuitos digitales* y de la *computación moderna*, permitiendo combinar y simplificar expresiones lógicas.

Ahora, responde con tus propias palabras y trata de recordar lo aprendido.

Para la **pregunta 2. ¿Qué es una compuerta lógica?**, te presento un conjunto de resúmenes que debes comparar y analizar para escribir tu respuesta, ver Tabla 1.46.

Análisis consolidado a la pregunta ¿Qué es una compuerta lógica?

Tabla 1.46: Resumen de las funciones y tipos de las compuertas lógicas

Fuente	Respuestas	Explicación de implementación
Gemini	Es el *bloque de construcción fundamental* de un circuito digital. Realiza una *operación lógica* elemental sobre entradas binarias (**1** o **0**) con una única salida, siguiendo el Álgebra de Boole.	Son implementadas físicamente mediante *transistores*. Se combinan para construir *funciones complejas*, permitiendo a los dispositivos electrónicos tomar decisiones y procesar información.
Claude	Dispositivo electrónico que implementa físicamente una operación booleana (AND, OR, NOT, NAND, NOR, XOR, XNOR) con una o más señales de entrada binarias.	Son los *bloques de construcción fundamentales* de *todos los sistemas digitales modernos*: procesadores, memorias, calculadoras. Se implementan usando *transistores* en circuitos integrados (chips).
Copilot	Dispositivo electrónico fundamental que realiza *operaciones lógicas* sobre una o más señales de entrada para producir una salida, basándose en reglas del Álgebra de Boole.	Se usan para construir *sistemas complejos* como procesadores y memorias, ya que permiten tomar *decisiones binarias* (**0** o **1**) según las combinaciones de entrada.
Deepseek	Circuito electrónico que opera con *señales binarias* (**0** y **1**) para implementar las *funciones básicas* del Álgebra de Boole (AND, OR, NOT).	Sirven como *bloques de construcción esenciales* para formar *circuitos digitales más complejos* (microprocesadores, memorias) y todos los sistemas digitales modernos.
ChatGPT	Dispositivo electrónico que realiza *operaciones lógicas básicas* (AND, OR y NOT) a partir de una o varias *señales de entrada binarias* (**0** y **1**) para producir una salida también binaria.	Son los *bloques fundamentales* en el diseño de *circuitos digitales* y permiten procesar información en computadoras, calculadoras y sistemas electrónicos en general.

Ahora, responde con tus propias palabras y trata de recordar lo aprendido.

Para la **pregunta 3. ¿Qué son los 0s y 1s lógicos y cuáles son los rangos de voltajes de cada uno?**, te presento un conjunto de resúmenes que debes comparar y analizar para escribir tu respuesta, ver Tabla 1.47.

Análisis consolidado: 0s y 1s lógicos y rangos de voltaje

Tabla 1.47: Correspondencia entre estados lógicos y rangos de voltaje en electrónica digital

Fuente	Definición del estado lógico	Rangos de voltaje y tecnologías
Gemini	*0 Lógico*: Estado bajo, falso o apagado. *1 Lógico*: Estado alto, verdadero o encendido. Ambos son la base de la señal digital en Álgebra de Boole.	*0 Lógico*: Voltaje cercano a $\mathbf{0\,V}$. *1 Lógico*: Voltaje alto; típicamente $\mathbf{5\,V}$ (TTL) o $\mathbf{3{,}3\,V}$ (CMOS).
Claude	*0 Lógico*: "Falso", "apagado", "bajo". *1 Lógico*: "Verdadero", "encendido", "alto". Son representaciones binarias de estados discretos.	*Tecnología TTL (5V):* $\mathbf{0\,L}$ es $\mathbf{0\,V}$ a $\mathbf{0{,}8\,V}$. $\mathbf{1\,L}$ es $\mathbf{2\,V}$ a $\mathbf{5\,V}$. Menciona la *zona de incertidumbre* y tecnologías menores ($\mathbf{3{,}3\,V}, \mathbf{1{,}8\,V}$).
Copilot	Representan los dos estados posibles (nivel bajo: $\mathbf{0}$ y nivel alto: $\mathbf{1}$) para procesar y almacenar información binaria en sistemas digitales.	*TTL:* $\mathbf{0\,L}$ es $\mathbf{0\,V}$ a $\mathbf{0{,}8\,V}$. $\mathbf{1\,L}$ es $\mathbf{2\,V}$ a $\mathbf{5\,V}$. *CMOS:* $\mathbf{0\,L}$ es $< \mathbf{1/3}$ de V_{CC} y $\mathbf{1\,L}$ es $> \mathbf{2/3}$ de V_{CC}.
Deepseek	Representaciones abstractas de dos estados discretos ($\mathbf{0}$, falso o bajo y $\mathbf{1}$, verdadero o alto). Se materializan físicamente como rangos de voltaje.	*TTL:* $\mathbf{0\,L}$ es $\mathbf{0}$ a $\mathbf{0{,}8\,V}$. $\mathbf{1\,L}$ es $\mathbf{2}$ a $\mathbf{5\,V}$. *CMOS (3.3V):* $\mathbf{0\,L}$ es $\mathbf{0}$ a $\mathbf{0{,}9\,V}$. Existe una *región prohibida* para garantizar claridad.
ChatGPT	*0 Lógico*: Estado bajo, falso o apagado. *1 Lógico*: Estado alto, verdadero o encendido. Ambos son los dos estados posibles de una señal digital en el álgebra de Boole.	*0 Lógico*: Voltaje cercano a $\mathbf{0\,V}$. *1 Lógico*: Voltaje alto; típicamente $\mathbf{5\,V}$ (TTL) o $\mathbf{3{,}3\,V}$ (CMOS).

Ahora, responde con tus propias palabras y trata de recordar lo aprendido.

1.3.2. Sección 2 del laboratorio 1.

A continuación, investiga toda información que te permita comprender el funcionamiento de las siguientes compuertas.

Consolidación del aprendizaje

Compuertas Lógicas Digitales

[Investiga, observa, reflexiona, analiza y completa la información de las principales compuertas.]

Tabla: Compuerta NOT

Compuerta: NOT

Describe su funcionamiento:

Símbolo	Diagrama lógico	Tabla de verdad (2 entradas)	Circuito integrado	Código TTL

Tabla: Compuerta AND

Compuerta: AND

Describe su funcionamiento:

Símbolo	Diagrama lógico	Tabla de verdad (2 entradas)	Circuito integrado	Código TTL

Tabla: Compuerta OR

Compuerta: OR

Describe su funcionamiento:

Símbolo	Diagrama lógico	Tabla de verdad (2 entradas)	Circuito integrado	Código TTL

Tabla: Compuerta NAND

Compuerta: NAND

Describe su funcionamiento:

Símbolo	Diagrama lógico	Tabla de verdad (2 entradas)	Circuito integrado	Código TTL

Tabla: Compuerta NOR

Compuerta: NOR

Describe su funcionamiento:

Símbolo	Diagrama lógico	Tabla de verdad (2 entradas)	Circuito integrado	Código TTL

Tabla: Compuerta XOR

Compuerta: XOR

Describe su funcionamiento:

Símbolo	Diagrama lógico	Tabla de verdad (2 entradas)	Circuito integrado	Código TTL

Tabla: Compuerta XNOR

Compuerta: XNOR

Describe su funcionamiento:

Símbolo	Diagrama lógico	Tabla de verdad (2 entradas)	Circuito integrado	Código TTL

1.3.3. Sección 3 del laboratorio 1.

Ahora utilice un simulador de circuitos (puede usar Logisim u otro) y presente cinco ejemplos de circuitos combinatorios que usen compuertas lógicas e indicando qué hacen (ver Fig. 1.25).

Simulador de circuitos combinatorios

Logisim[a] es un programa que permite diseñar y simular circuitos digitales. Pensado como una herramienta educativa para aprender cómo funcionan los circuitos y compuertas lógicas.

[a] Desacrga Logisim en sourceforge.net

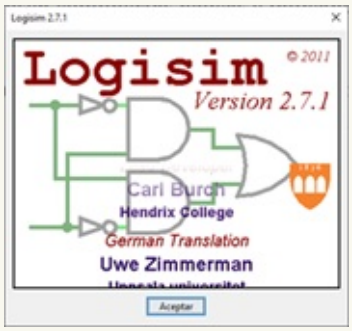


Figura 1.25: El programa de Logisim se descarga en sourgeforge.net.

A continuación, reconocemos el ambiente de trabajo del programa y utilizamos las compuertas visibles, ver Fig. 1.26.

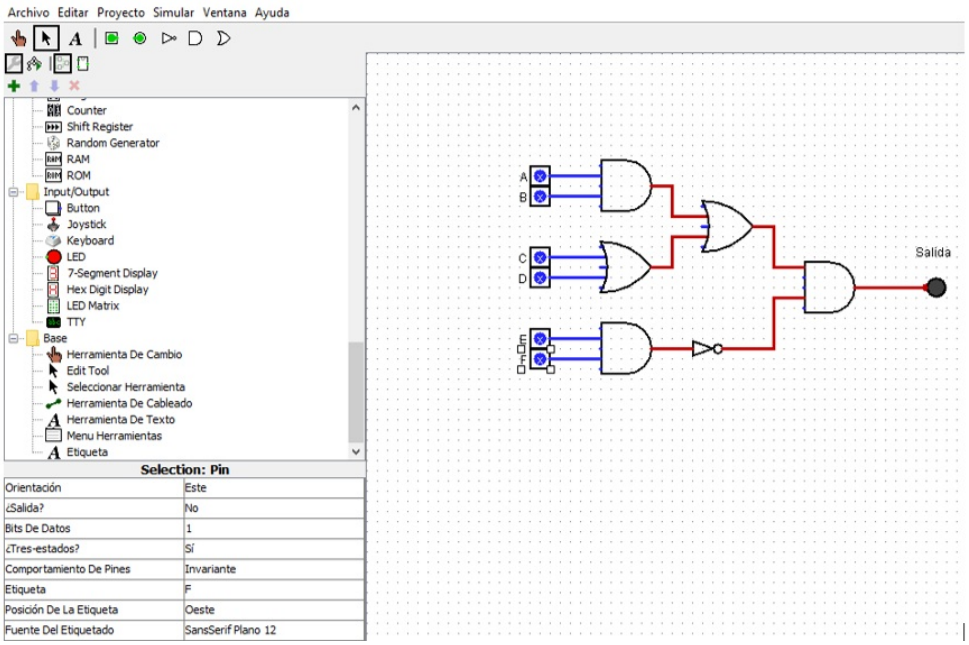


Figura 1.26: Entorno de trabajo de Logisim

En el menú del programa[2] puedes utilizar el módulo de “Ayuda” donde se encuentran elementos de asistencia para iniciar su uso. Finalmente, generamos varios esquemas con la utilización de las compuertas lógicas.

[2] Descarga Logisim de su sitio web

Se puede observar que la imagen visible en el entorno tiene diversos colores, a diferencia de la imagen cuando se exporta en formatos jpg. Gif o png, en el módulo de "Archivo", exportar imagen, ver Fig. 1.27.

Para analizar y reproducir

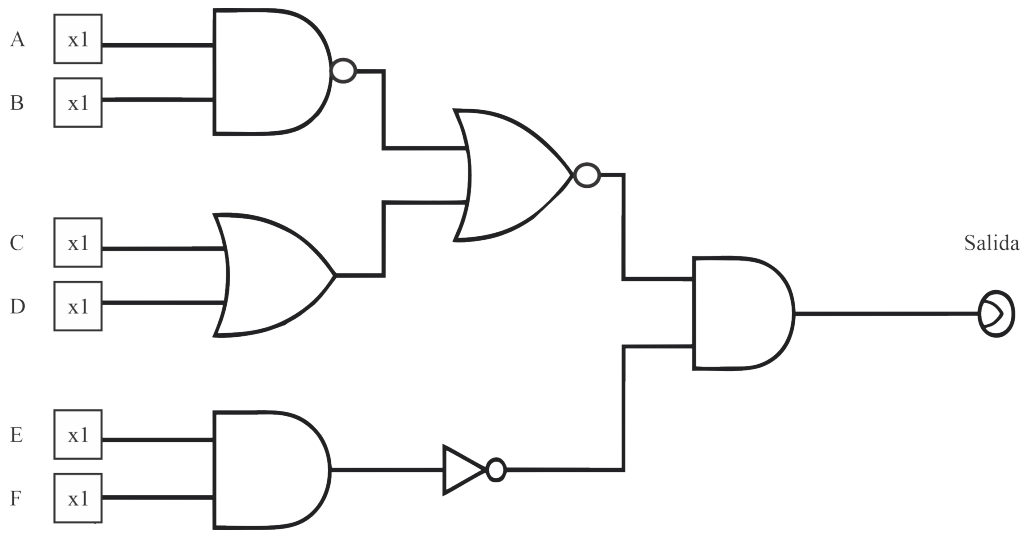


Figura 1.27: Imagen de salida del programa Logisim.

A continuación, en el espacio vacío de la siguiente caja dibuja tu diseño logrado luego del uso del programa Logisim.

Coloca tu dibujo en este espacio

A continuación, en el espacio vacío de la siguiente caja dibuja otro diseño logrado luego del uso del programa Logisim.

Coloca tu dibujo en este espacio

A continuación, en el espacio vacío de la siguiente caja dibuja un nuevo diseño logrado luego del uso del programa Logisim.

Coloca tu dibujo en este espacio

A continuación, en el espacio vacío de la siguiente caja dibuja otro diseño logrado luego del uso del programa Logisim.

Coloca tu dibujo en este espacio

A continuación, en el espacio vacío de la siguiente caja dibuja un nuevo diseño logrado luego del uso del programa Logisim.

Coloca tu dibujo en este espacio

1.3.4. Sección 4 del laboratorio 1.

Utilizando las propiedades y teoremas del Álgebra de Boole verifique si la siguiente igualdad es verdadera o falsa. Elabora las simplificaciones necesarias y enuncia cada una a medida que vayas desarrollando hacia abajo.

Para el ejercicio: $x'z + x'yz = x'z$, el proceso de desarrollo se encuentra a continuación, ver Tabla 1.48.

Verificación de igualdad booleana (Método Algebraico)

Tabla 1.48: Verificación de la igualdad $x'z + x'yz = x'z$

Paso	Expresión resultante	Ley o Teorema aplicado
L.I. Inicial	$x'z + x'yz$	Expresión Original (Lado Izquierdo)
1	$x'z(1 + y)$	**Factor Común** (Propiedad Distributiva): Se factoriza $x'z$.
2	$x'z(1)$	**Ley de Dominación (o Unidad)**: $1 + A = 1$. Cualquier término sumado a 1 es 1.
3	$x'z$	**Ley de Identidad**: $A \cdot 1 = A$. Cualquier término multiplicado por 1 es el propio término.
L.D. Final	$x'z$	Igual al Lado Derecho (L.D.). **La igualdad es VERDADERA.**

Para el ejercicio de verificación de igualdad booleana: $x(x + y) = x$, el proceso de desarrollo se encuentra a continuación, ver Tabla 1.49.

Verificación de igualdad booleana (Método Algebraico)

Tabla 1.49: Verificación de la igualdad $x(x + y) = x$

Paso	Expresión resultante	Ley o Teorema aplicado
L.I. Inicial	$x(x + y)$	Expresión Original (Lado Izquierdo)
1	$x \cdot x + x \cdot y$	**Ley Distributiva**: Se multiplica x por cada término dentro del paréntesis.
2	$x + xy$	**Ley de Idempotencia**: $A \cdot A = A$. El término $x \cdot x$ se simplifica a x.
3	$x(1 + y)$	**Factor Común** (Propiedad Distributiva): Se factoriza x de ambos términos.
4	$x(1)$	**Ley de Dominación (o Unidad)**: $1 + A = 1$. Cualquier cosa sumada a 1 es 1.
L.D. Final	x	Igual al Lado Derecho (L.D.). **La igualdad es VERDADERA.**

Utilizando las propiedades y teoremas del Álgebra de Boole, verifique si la siguiente igualdad es verdadera o falsa. Elabore las simplificaciones necesarias y enuncie cada ley o teorema aplicado a medida que desarrolle la expresión:

$$\overline{A}B\overline{C}D + \overline{A}B\overline{C}\overline{D} + AB\overline{C}D + A\overline{B}\overline{C}D = \overline{C}D$$

Nota: Considere la notación $\overline{A}$ para la negación (NOT) y la simple yuxtaposición ($\overline{A}B$) para la conjunción (AND), ver Tabla 1.50.

Verificación de igualdad booleana (4 Variables)

Tabla 1.50: Verificación de la igualdad $\overline{A}B\overline{C}D + \overline{A}B\overline{C}\overline{D} + AB\overline{C}D + A\overline{B}\overline{C}D = \overline{C}D$

Paso	Expresión resultante	Ley o Teorema aplicado
L.I. Inicial	$\overline{A}B\overline{C}D+\overline{A}B\overline{C}\overline{D}+AB\overline{C}D+A\overline{B}\overline{C}D$	Expresión Original (Lado Izquierdo)
1	$\overline{C}D(\overline{A}B + \overline{A}B + AB + A\overline{B})$	**Factor Común** (Propiedad Distributiva). Se factoriza $\overline{C}D$ de todos los términos.
2	$\overline{C}D(\overline{A}B + AB + A\overline{B})$	**Ley de Idempotencia**: $X + X = X$. El término $\overline{A}B$ se repite.
3	$\overline{C}D[B(\overline{A} + A) + A\overline{B}]$	**Factor Común** (Propiedad Distributiva). Se factoriza B de los dos primeros términos.
4	$\overline{C}D[B(1) + A\overline{B}]$	**Ley de Complemento** (OR): $\overline{A} + A = 1$.
5	$\overline{C}D(B + A\overline{B})$	**Ley de Identidad**: $X \cdot 1 = X$.
6	$\overline{C}D(B + A)$	**Teorema de Absorción**: $X + \overline{X}Y = X + Y$. Aquí, $B + A\overline{B} = B + A$.
7	$\overline{C}D$	**Ley de Dominación** (AND): $\overline{C}D \cdot (B + A)$ no simplifica a $\overline{C}D$, no es igual.
L.D. Final	$\overline{C}D$	Reagrupamos términos de la expresión original para optimizar u alternativa... (ver abajo)

A continuación, la simplificación del Lado Izquierdo ($\overline{C}D(\overline{A}B + AB + A\overline{B})$) buscando otra alternativa a la que se hizo en la Tabla 1.50, pero necesitamos verificar los últimos pasos, ver Tabla 1.51.

Continuación del análisis de la simplificación, otra alternativa

Tabla 1.51: Verificación detallada de la simplificación (continuación)

Paso	Expresión resultante	Ley o Teorema aplicado
Inicial (desde el paso 2)	$\overline{C}D \cdot (\overline{A}B + AB + A\overline{B})$	Continuación del Paso 2 de la Tabla 1.50
3	$\overline{C}D \cdot [B(\overline{A} + A) + A\overline{B}]$	**Factor Común** de B en los dos primeros términos.
4	$\overline{C}D \cdot [B(1) + A\overline{B}]$	**Ley de Complemento** (OR): $\overline{A} + A = 1$.
5	$\overline{C}D \cdot (B + A\overline{B})$	**Ley de Identidad**: $X \cdot 1 = X$.
6	$\overline{C}D \cdot (B + A)$	**Teorema de Absorción**: $X + \overline{X}Y = X + Y$. Aquí, $B + A\overline{B} = B + A$.
L.I. Simplificada	$\overline{C}D \cdot (B + A)$	**Lado Izquierdo completamente simplificado.**
L.D. Final	$\overline{C}D$	Se concluye que no son iguales.

Conclusión y corrección del enunciado: El Lado Izquierdo se simplifica a $\overline{\mathbf{C}}\mathbf{D}(\mathbf{A} + \mathbf{B})$.

- La igualdad original $\overline{\mathbf{A}}\mathbf{B}\overline{\mathbf{C}}\mathbf{D} + \overline{\mathbf{A}}\mathbf{B}\overline{\mathbf{C}}\overline{\mathbf{D}} + \mathbf{A}\mathbf{B}\overline{\mathbf{C}}\mathbf{D} + \mathbf{A}\overline{\mathbf{B}}\overline{\mathbf{C}}\mathbf{D} = \overline{\mathbf{C}}\mathbf{D}$ es **FALSA**.
- Para que la igualdad fuera **VERDADERA**, el Lado Derecho debería haber sido $\overline{\mathbf{C}}\mathbf{D}(\mathbf{A} + \mathbf{B})$.

1.3.5. Sección 5 del laboratorio 1

Utilizando las propiedades y teoremas del Álgebra de Boole escriba el enunciado E en forma de suma de productos, $E = (((a'b')c))'(((a' + c)(b' + c')))'$

Aplique los siguientes pasos: (a). Use las *leyes De Morgan* y la *involución* para eliminar los complementos de los paréntesis hasta que se aplique solamente a variables. (b). Use la *ley distributiva* para transformar E en suma de productos. (c). Use las *leyes conmutativas*, de *idempotencia* y de *complemento* transformar cada producto de E en 0 o en un producto fundamental. (d). Use la *ley de absorción* para dejar E en la forma correcta de suma de productos.

Se presenta a continuación el desarrollo paso a paso, ver Tabla 1.52.

Ejercicio de simplificación a suma de productos

Tabla 1.52: Conversión de $E = ((a'b')c)'((a' + c)(b' + c'))'$ a Suma de Productos

Paso	Expresión Resultante	Ley o Teorema Aplicado
Inicial	$E = ((a'b')c)'((a' + c)(b' + c'))'$	Expresión Original
a. Uso de Leyes De Morgan e Involución		
a.1	$E = [(a'b')' + c'] + [(a' + c)' + (b' + c')']$	**Ley De Morgan**: $(XY)' = X' + Y'$ (aplicada a la conjunción principal).
a.2	$E = [(a'b')' + c'] + [(a''c') + (b''c''')]$	**Ley De Morgan**: $(X + Y)' = X'Y'$ (aplicada dentro del segundo corchete).
a.3	$E = [(a'' + b'') + c'] + [(ac') + (bc)]$	**Ley De Morgan** y **Involución**: $(a'b')' = a'' + b''$. También: $a'' = a$ y $c''' = c'$.
a.4	$E = [a + b + c'] + [ac' + bc]$	**Ley de Involución**: $a'' = a$ y $b'' = b$.
b. Uso de Ley Distributiva (para suma de productos)		
b.1	$E = a + b + c' + ac' + bc$	Eliminando corchetes, ya que todas las operaciones restantes son OR (suma).
c. Uso de Leyes Conmutativas, Idempotencia, y Complemento		
c.1	$E = a + b + c' + ac' + bc$	La expresión ya está en forma de suma de productos (Minterms y Maxterms).
d. Uso de Ley de Absorción		
d.1	$E = a + b + c'$	**Ley de Absorción**: $X + XY = X$. • $c' + ac' = c'$ (Término ac' es absorbido por c'). • $b + bc = b$ (Término bc es absorbido por b).
Final	$\mathbf{E = a + b + c'}$	Expresión Final Simplificada en Suma de Productos.

A continuación, utilizando las propiedades y teoremas del Álgebra de Boole, simplifique la siguiente expresión lógica F a su forma mínima de Suma de Productos (SOP, sum of products), $\mathbf{F = (A + B)(\overline{A} + C)(B + C)}$.

Aplique los siguientes pasos: (a). Use la *ley distributiva* para expandir los primeros dos paréntesis. (b). Simplifique la expresión resultante utilizando la *ley de complemento* ($X\overline{X} = 0$) y la *ley de identidad* ($X + 0 = X$). (c). Use la *ley de consenso* para simplificar la expresión por completo.

Se presenta a continuación el desarrollo paso a paso, ver Tabla 1.53.

Ejercicio de simplificación a suma de productos (Ley de Consenso)

Tabla 1.53: Simplificación de $F = (A + B)(\overline{A} + C)(B + C)$ a su mínima expresión SOP

Paso	Expresión Resultante	Ley o Teorema Aplicado
Inicial	$F = (A + B)(\overline{A} + C)(B + C)$	Expresión Original
a. Expansión de los primeros dos paréntesis		
a.1	$F = (A\overline{A} + AC + B\overline{A} + BC)(B + C)$	**Ley Distributiva**: $X(Y + Z) = XY + XZ$.
b. Simplificación usando Complemento e Identidad		
b.1	$F = (0 + AC + \overline{A}B + BC)(B + C)$	**Ley de Complemento** (AND): $A\overline{A} = 0$.
b.2	$F = (AC + \overline{A}B + BC)(B + C)$	**Ley de Identidad** (OR): $X + 0 = X$.
c. Uso de la Ley de Consenso (Avanzada)		
c.1	$F = (AC + \overline{A}B)(B + C)$	**Ley de Consenso** (Forma OR): $XY + \overline{X}Z + YZ = XY + \overline{X}Z$. El término BC es redundante.
d. Conversión final a SOP		
d.1	$F = AC(B + C) + \overline{A}B(B + C)$	**Ley Distributiva** (Expandiendo la conjunción principal).
d.2	$F = ACB + ACC + \overline{A}BB + \overline{A}BC$	**Ley Distributiva** (Completando la expansión).
d.3	$F = ABC + AC + \overline{A}B + \overline{A}BC$	**Ley de Idempotencia**: $BB = B$ y $CC = C$.
d.4	$F = AC(B + 1) + \overline{A}B(1 + C)$	**Factor Común** (Agrupando términos semejantes).
d.5	$F = AC(1) + \overline{A}B(1)$	**Ley de Dominación** (Unidad): $1 + X = 1$.
Final	$\mathbf{F = AC + \overline{A}B}$	**Ley de Identidad**: $X \cdot 1 = X$.

A continuación, utilizando las propiedades y teoremas del Álgebra de Boole, simplifique la siguiente expresión lógica G a su forma mínima de Suma de Productos (SOP), $\mathbf{G = AB + \overline{A}C + BC}$.

Aplique los siguientes pasos: (a). Identifique el término redundante utilizando la *ley de consenso*. Demuestre su redundancia añadiendo el factor $1 = (A + \overline{A})$ al término. (b). Use la *ley distributiva* y las *leyes de complemento* para expandir y agrupar el término expandido con los términos originales. (c). Use la *ley de absorción* y la *ley de idempotencia* para eliminar los términos redundantes y obtener la expresión simplificada final.

Observe, analice y siga el proceso presentado en el diagrama de flujo, ver Fig. 1.28.

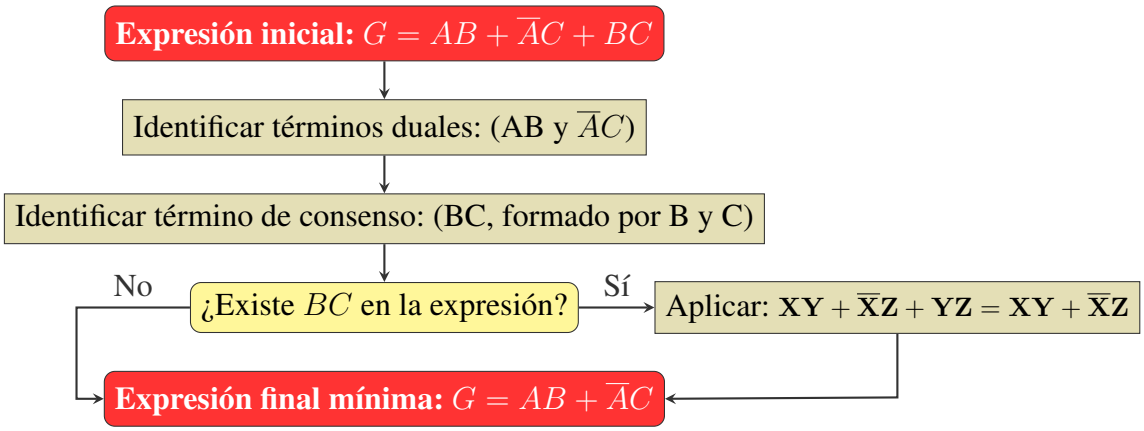


Figura 1.28: Proceso a seguir en el diagrama de flujo para la Ley de Consenso.

Se presenta a continuación el desarrollo del ejercicio paso a paso, ver Tabla 1.54.

Desarrollo: Simplificación por Ley de Consenso

Tabla 1.54: Simplificación de $G = AB + \overline{A}C + BC$ a su mínima expresión SOP

Paso	Expresión Resultante	Ley o Teorema Aplicado
Inicial	$G = AB + \overline{A}C + BC$	Expresión Original
a. Identificación y Demostración de Redundancia		
a.1	$G = AB + \overline{A}C + BC \cdot (A + \overline{A})$	**Ley de Complemento**: $A + \overline{A} = 1$. (Término BC es el consenso)
b. Expansión y Agrupación de Términos		
b.1	$G = AB + \overline{A}C + ABC + \overline{A}BC$	**Ley Distributiva** (Expandiendo el término BC).
b.2	$G = (AB + ABC) + (\overline{A}C + \overline{A}BC)$	**Ley Asociativa** (Reordenando para agrupar términos semejantes).
b.3	$G = AB(1 + C) + \overline{A}C(1 + B)$	**Factor Común** (Propiedad Distributiva).
b.4	$G = AB(1) + \overline{A}C(1)$	**Ley de Dominación** (Unidad): $1 + X = 1$.
c. Eliminación de Redundancia y Simplificación Final		
c.1	$G = AB + \overline{A}C$	**Ley de Identidad**: $X \cdot 1 = X$.
Final	$\mathbf{G = AB + \overline{A}C}$	**Ley de Consenso**: $AB + \overline{A}C + BC = AB + \overline{A}C$. El término BC es redundante.

Finalmente, utilizando las propiedades y teoremas del Álgebra de Boole, escriba el enunciado H en su forma mínima de Suma de Productos (SOP), $\mathbf{H = (A + \overline{B}C)(A + \overline{B}C')}$.

Aplique los siguientes pasos: (a). Use la *ley distributiva (inversa)* para factorizar el término común A. (b). Simplifique la expresión resultante utilizando la *ley distributiva (expansión)* y la *ley de complemento*. (c). Use la *ley de identidad* para obtener la expresión final en la forma mínima de Suma de Productos.

Se presenta a continuación el desarrollo paso a paso, ver Tabla 1.55.

Desarrollo: Simplificación por Distribución Inversa

Tabla 1.55: Simplificación de $H = (A + \overline{B}C)(A + \overline{B}C')$

Paso	Expresión Resultante	Ley o Teorema Aplicado
Inicial	$H = (A + \overline{B}C)(A + \overline{B}C')$	Expresión Original
a. Uso de Ley Distributiva (Inversa)		
a.1	$H = A + (\overline{B}C)(\overline{B}C')$	**Ley Distributiva Inversa**: $(X + Y)(X + Z) = X + YZ$.
b. Uso de Leyes de Complemento e Idempotencia		
b.1	$H = A + (\overline{B}\overline{B})(CC')$	**Ley Asociativa/Conmutativa** (Reordenando términos).
b.2	$H = A + (\overline{B})(0)$	**Ley de Idempotencia**: $\overline{B}\overline{B} = \overline{B}$. **Ley de Complemento**: $CC' = 0$.
b.3	$H = A + 0$	**Ley de Dominación** (Cero): $X \cdot 0 = 0$.
c. Uso de Ley de Identidad		
c.1	$H = A$	**Ley de Identidad**: $X + 0 = X$.
Final	$\mathbf{H = A}$	Expresión Final Mínima en Suma de Productos.

1.3.6. Sección 6 del laboratorio 1

Utilizando el siguiente circuito que se muestra en la imagen a continuación, ver Fig. 1.29.

Para analizar y desarrollar el problema

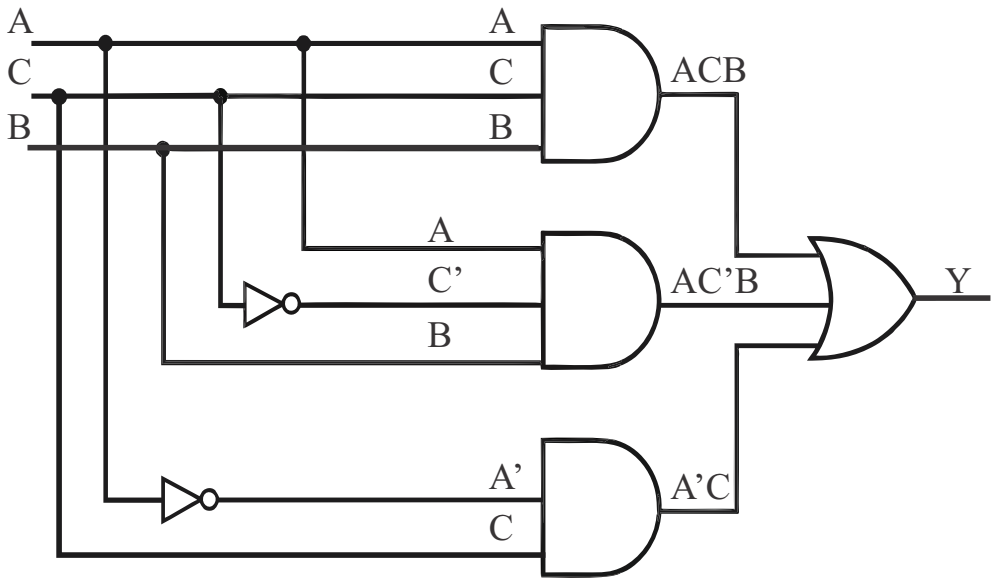


Figura 1.29: Imagen del programa Logisim.

Realice lo siguiente: a. Escriba la expresión booleana para el siguiente circuito. b. Construya la tabla de verdad. c. Simplifique la expresión resultante de la tabla de verdad utilizando los mapas de Karnaugh. d. Simplifique la expresión booleana de salida del circuito utilizando las leyes del algebra de Boole. (Las dos expresiones simplificadas deben resultar iguales). e. Diseñe el circuito simplificado utilizando Logisim, guarde el archivo e imagen con la documentación de esta tarea.

Para el literal **a. Escriba la expresión booleana para el siguiente circuito**, la respuesta es la siguiente: $\mathbf{Y = ACB + AC'B + A'C}$.

Realizando un análisis y tabla de verdad de $\mathbf{Y = ACB + AC'B + A'C}$ se tendría la siguiente información, ver Tabla 1.56.

a. Simplificación por Álgebra de Boole

Tabla 1.56: Simplificación de la expresión Y

Paso	Expresión Resultante	Ley o Teorema Aplicado
Inicial	$Y = ACB + AC'B + A'C$	Expresión Original
1	$Y = AB(C + C') + A'C$	**Ley Distributiva** (Factor Común AB).
2	$Y = AB(1) + A'C$	**Ley de Complemento**: $C + C' = 1$.
Final	$\mathbf{Y = AB + A'C}$	**Ley de Identidad**: $X \cdot 1 = X$.

Para el literal **b. Construya la tabla de verdad**, se tendría la siguiente información, ver Tabla 1.57.

b. Tabla de Verdad

Tabla 1.57: Tabla de verdad para la expresión simplificada $\mathbf{Y = AB + A'C}$

Entradas			Salidas Intermedias y Final		
A	**B**	**C**	**AB**	**A'C**	**Y = AB + A'C**
0	0	0	0	0	**0**
0	0	1	0	1	**1**
0	1	0	0	0	**0**
0	1	1	0	1	**1**
1	0	0	0	0	**0**
1	0	1	0	0	**0**
1	1	0	1	0	**1**
1	1	1	1	0	**1**

Para el literal **c. Simplifique la expresión resultante de la tabla de verdad utilizando los mapas de Karnaugh**, se tendría la siguiente información, ver Tabla 1.58.

c. Simplificación por Mapa de Karnaugh (K-map)

Tabla 1.58: Mapa de Karnaugh y Simplificación de $Y = AB + \overline{A}C$

Paso	Descripción	Expresión Mínima
Inicial	Minterms de Y: $m(1, 3, 6, 7)$.	$Y = ACB + ACB + \overline{A}C$
1. Simplificación Algebraica Previa		
a.1	$Y = AB(C + \overline{C}) + \overline{A}C$	$Y = AB + \overline{A}C$
2. Agrupación en el Mapa de Karnaugh (K-map)		
b.1	El mapa muestra los 1 en las celdas 1, 3, 6, 7.	
b.2	**Grupo 1 (Rojo):** Agrupa los 1 en $m(6, 7)$ (donde $A = 1$ y $B = 1$).	
b.3	**Grupo 2 (Azul):** Agrupa los 1 en $m(1, 3)$ (donde $A = 0$ y $C = 1$).	
3. Obtención de la expresión mínima		
Final	La suma de los dos grupos primos es la expresión mínima.	$Y_{\text{Mínima}} = AB + \overline{A}C$

El mapa de Karnaugh o K-map resultante es el siguiente:

Mapa de Karnaugh para $\mathbf{Y = AB + \bar{A}C}$

	00 ($\overline{BC}$)	01 ($\overline{B}C$)	11 (BC)	10 ($B\overline{C}$)
0 ($\overline{A}$)	**0**	**1**	**1**	**0**
1 (A)	**0**	**0**	**1**	**1**

Resultado de la simplificación por agrupación:

- **Grupo Azul** (AB)**:** Celdas (110) y (111) $\rightarrow$ Simplifica a AB.
- **Grupo Rojo** ($\overline{A}C$)**:** Celdas (001) y (011) $\rightarrow$ Simplifica a $\overline{A}C$.

$$Y_{\text{Mínima}} = AB + A'C$$

Para el literal **d. Simplifique la expresión booleana de salida del circuito utilizando las leyes del algebra de Boole. (Las dos expresiones simplificadas deben resultar iguales)**, se tendría la siguiente información, ver Tabla 1.59.

Demostración de Equivalencia y Simplificación

Tabla 1.59: Simplificación de $Y = ACB + AC'B + A'C$

Paso	Expresión Resultante	Ley o Teorema Aplicado
Inicial	$Y = ACB + AC'B + A'C$	Expresión dada
1	$Y = (AB)C + (AB)C' + A'C$	**Ley Conmutativa o Asociativa** (Reagrupando términos).
2	$Y = AB(C + C') + A'C$	**Ley Distributiva** (Factor Común AB).
3	$Y = AB(1) + A'C$	**Ley de Complemento**: $C + C' = 1$.
Final	$\mathbf{Y = AB + A'C}$	**Ley de Identidad**: $X \cdot 1 = X$.

Conclusión: La expresión simplificada es: $\mathbf{Y_{Simplificada} = AB + A'C}$.
Esta expresión simplificada ya se había obtenido previamente por el Mapa de Karnaugh (K-map), confirmando la igualdad:

$$\mathbf{Y_{Inicial} \equiv Y_{K\text{-}map} \equiv Y_{Simplificada}}$$

Para el literal **e. Diseñe el circuito simplificado utilizando Logisim, guarde el archivo e imagen con la documentación de esta tarea**, se tendría la siguiente información, ver Fig. 1.30.

Diseño final del circuito

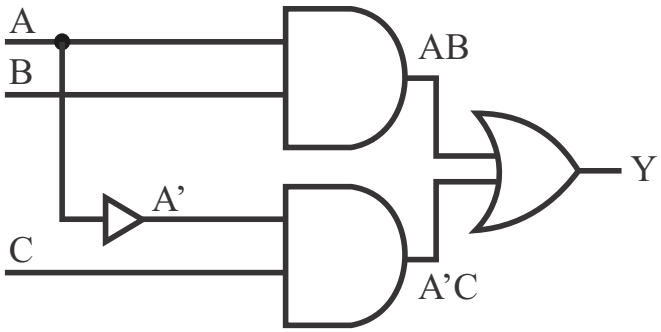
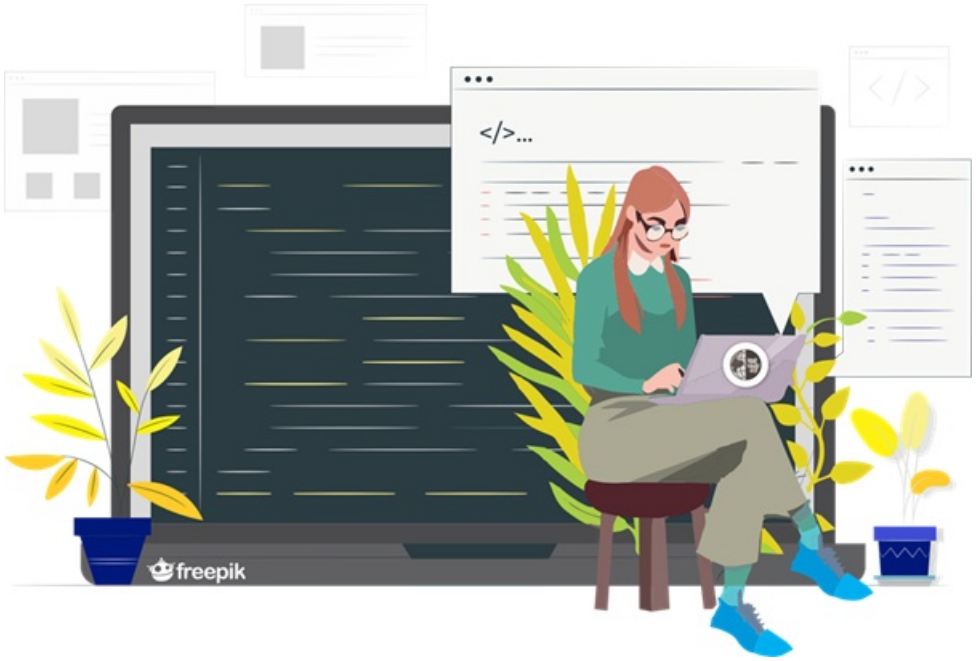


Figura 1.30: Imagen final del circuito simplificado.



Laboratorio 2

Capítulo 1

1.4. Ejercicios del Laboratorio 2

A continuación empezamos con los ejercicios del laboratorio 2 de las prácticas de experimentación del capítulo 1.

1.4.1. Sección 1 del laboratorio 2.

Enunciado 1: A continuación, representa con notación lógica el siguiente argumento.

"El programa está bien si y sólo si no tiene errores de compilación. Si no tiene errores de lógica y no tiene errores de compilación, entonces el programa está bien y los resultados son satisfactorios. Por lo tanto, si tiene errores de compilación o tiene errores de lógica, entonces el programa no está bien y los resultados no son satisfactorios."

1. Identificación de variables y notación lógica, ver Tabla 1.60

Variables proposicionales y notación

Variables definidas

P: El programa está bien.

Q: El programa tiene errores de compilación.

R: El programa tiene errores de lógica.

S: Los resultados son satisfactorios.

Notación lógica del argumento

Tabla 1.60: Traducción de premisas y conclusión del enunciado 1

Parte del enunciado	Notación lógica
Premisa 1: El programa está bien si y sólo si no tiene errores de compilación.	$\mathbf{P} \leftrightarrow \neg\mathbf{Q}$
Premisa 2: Si no tiene errores de lógica y no tiene errores de compilación, entonces el programa está bien y los resultados son satisfactorios.	$(\neg\mathbf{R} \land \neg\mathbf{Q}) \rightarrow (\mathbf{P} \land \mathbf{S})$
Conclusión: Si tiene errores de compilación o tiene errores de lógica, entonces el programa no está bien y los resultados no son satisfactorios.	$(\mathbf{Q} \lor \mathbf{R}) \rightarrow (\neg\mathbf{P} \land \neg\mathbf{S})$

Proposición compuesta total (T)

El argumento completo se formula como una implicación [Premisas] $\rightarrow$ [Conclusión]: $T = [(P \leftrightarrow \neg Q) \land ((\neg R \land \neg Q) \rightarrow (P \land S))] \rightarrow [(Q \lor R) \rightarrow (\neg P \land \neg S)]$

2. Tabla de verdad y clasificación, ver Tabla 1.61

2. Tabla de verdad y clasificación

Evaluamos: $T = [(P \leftrightarrow \neg Q) \land ((\neg R \land \neg Q) \rightarrow (P \land S))] \rightarrow [(Q \lor R) \rightarrow (\neg P \land \neg S)]$

Tabla 1.61: Evaluación completa de la proposición de 4 variables del enunciado 1

Entradas				Proposiciones y Evaluación									
P	Q	R	S	$\neg$Q	P $\leftrightarrow \neg$Q	$\neg$R	$\neg$R $\land \neg$Q	P $\land$ S	P_2	$P_1 \land P_2$	Q $\lor$ R	$\neg$P $\land \neg$S	T
F	F	F	F	V	F	V	V	F	F	F	F	V	**V**
F	F	F	V	V	F	V	V	F	F	F	F	F	**V**
F	F	V	F	V	F	F	F	F	V	F	V	V	**V**
F	F	V	V	V	F	F	F	F	V	F	V	F	**V**
F	V	F	F	F	V	V	F	F	V	V	V	V	**V**
F	V	F	V	F	V	V	F	F	V	V	V	F	**F**
F	V	V	F	F	V	F	F	F	V	V	V	V	**V**
F	V	V	V	F	V	F	F	F	V	V	V	F	**F**
V	F	F	F	V	V	V	V	F	F	F	F	F	**V**
V	F	F	V	V	V	V	V	V	V	V	F	F	**V**
V	F	V	F	V	V	F	F	F	V	V	V	F	**V**
V	F	V	V	V	V	F	F	V	V	V	V	F	**V**
V	V	F	F	F	F	V	F	F	V	F	V	F	**V**
V	V	F	V	F	F	V	F	V	V	F	V	F	**V**
V	V	V	F	F	F	F	F	F	V	F	V	F	**V**
V	V	V	V	F	F	F	F	V	V	F	V	F	**V**

Clasificación del resultado: La columna de la proposición compuesta total (**T**) muestra tanto valores *(V)* como *(F)* (filas 6 y 8). Por ende, la proposición es una **Contingencia**.

Enunciado 2: A continuación, representa con notación lógica el siguiente argumento.

"Si se realiza un buen diseño de la base de datos y se hace una buena programación, entonces se accederá rápidamente la información. Si no se hace buena programación, entonces toma mucho tiempo corregir el programa. Por lo tanto, si no se accede rápidamente la información y toma mucho tiempo corregir el programa, entonces no se ha realizado un buen diseño de la base de datos."

1. Identificación de variables y notación lógica, ver Tabla 1.62

Variables proposicionales y notación

Variables definidas

- **P:** Se realiza un buen diseño de la base de datos.
- **Q:** Se hace una buena programación.
- **R:** Se accederá rápidamente la información.
- **S:** Toma mucho tiempo corregir el programa.

Tabla 1.62: Traducción de premisas y conclusión del enunciado 2

Parte del enunciado	Notación lógica
Premisa 1: Si se realiza un buen diseño de la base de datos y se hace una buena programación, entonces se accederá rápidamente la información.	$(\mathbf{P} \land \mathbf{Q}) \rightarrow \mathbf{R}$
Premisa 2: Si no se hace buena programación, entonces toma mucho tiempo corregir el programa.	$\neg\mathbf{Q} \rightarrow \mathbf{S}$
Conclusión: Si no se accede rápidamente la información y toma mucho tiempo corregir el programa, entonces no se ha realizado un buen diseño de la base de datos.	$(\neg\mathbf{R} \land \mathbf{S}) \rightarrow \neg\mathbf{P}$

Proposición $T = [((P \land Q) \rightarrow R) \land (\neg Q \rightarrow S)] \rightarrow [(\neg R \land S) \rightarrow \neg P]$

2. Tabla de verdad y clasificación, ver Tabla 1.63

2. Tabla de verdad y clasificación

Evaluamos $T = [((P \land Q) \to R) \land (\neg Q \to S)] \to [(\neg R \land S) \to \neg P]$

Tabla 1.63: Evaluación completa de la proposición de 4 variables del enunciado 2

Entradas				Proposiciones y Evaluación							
P	Q	R	S	$\mathbf{P \land Q}$	$\mathbf{P_1}$	$\mathbf{\neg Q}$	$\mathbf{P_2}$	$\mathbf{P_1 \land P_2}$	$\mathbf{\neg R \land S}$	$\mathbf{\neg P}$	T
F	F	F	F	F	V	V	F	F	F	V	**V**
F	F	F	V	F	V	V	V	V	V	V	**V**
F	F	V	F	F	V	V	F	F	F	V	**V**
F	F	V	V	F	V	V	V	V	F	V	**V**
F	V	F	F	F	V	F	V	V	F	V	**V**
F	V	F	V	F	V	F	V	V	V	V	**V**
F	V	V	F	F	V	F	V	V	F	V	**V**
F	V	V	V	F	V	F	V	V	F	V	**V**
V	F	F	F	F	V	V	F	F	F	F	**V**
V	F	F	V	F	V	V	V	V	V	F	**F**
V	F	V	F	F	V	V	F	F	F	F	**V**
V	F	V	V	F	V	V	V	V	F	F	**V**
V	V	F	F	V	F	F	V	F	F	F	**V**
V	V	F	V	V	F	F	V	F	V	F	**V**
V	V	V	F	V	V	F	V	V	F	F	**V**
V	V	V	V	V	V	F	V	V	F	F	**V**

Resultado: La columna (**T**) contiene un valor *(F)* en la fila 10 (VFFV). En todos los demás casos es *(V)*. Por lo tanto, la proposición se clasifica como una **Contingencia**.

Enunciado 3: A continuación, representa con notación lógica el siguiente argumento.

"Si tengo conocimientos de computación y domino el inglés, entonces no tendré problemas para encontrar trabajo. Si tengo problemas para encontrar trabajo, entonces tengo más de 40 años o no me preparé lo suficiente. Por lo tanto, si me preparo lo suficiente y no tengo más de 40 años y domino el inglés, entonces no tendré problemas para encontrar trabajo."

1. Identificación de variables y notación lógica, ver Tabla 1.64

Variables proposicionales y notación

P: Tengo conocimientos de computación.
Q: Domino el inglés.
R: Tendré problemas para encontrar trabajo.
S: Tengo más de 40 años.
T: Me preparé lo suficiente.

Notación: $W = [((P \land Q) \to \neg R) \land (R \to (S \lor \neg T))] \to [(T \land \neg S \land Q) \to \neg R]$

Tabla 1.64: Traducción de premisas y conclusión del enunciado 3

Parte del enunciado	Notación lógica
Premisa 1: Si tengo conocimientos de computación y domino el inglés, entonces no tendré problemas para encontrar trabajo.	$\mathbf{(P \land Q) \to \neg R}$
Premisa 2: Si tengo problemas para encontrar trabajo, entonces tengo más de 40 años o no me preparé lo suficiente.	$\mathbf{R \to (S \lor \neg T)}$
Conclusión: Si me preparo lo suficiente y no tengo más de 40 años y domino el inglés, entonces no tendré problemas para encontrar trabajo.	$\mathbf{(T \land \neg S \land Q) \to \neg R}$

2. Tabla de verdad y clasificación, ver Tabla 1.65

2. Tabla de verdad y clasificación (5 Variables: 32 Filas)

Evaluamos $W = [((P \land Q) \to \neg R) \land (R \to (S \lor \neg T))] \to [(T \land \neg S \land Q) \to \neg R]$

Tabla 1.65: Evaluación completa de la proposición de 5 variables

Entradas					Proposiciones y Evaluación								
P	Q	R	S	T	$\neg$R	P $\land$ Q	P_1	$\neg$T	S $\lor \neg$T	P_2	$P_1 \land P_2$	C	W
F	F	F	F	F	V	F	V	V	V	V	V	V	**V**
F	F	F	F	V	V	F	V	F	F	V	V	V	**V**
F	F	F	V	F	V	F	V	V	V	V	V	V	**V**
F	F	F	V	V	V	F	V	F	V	V	V	V	**V**
F	F	V	F	F	F	F	V	V	V	V	V	V	**V**
F	F	V	F	V	F	F	V	F	F	F	F	V	**V**
F	F	V	V	F	F	F	V	V	V	V	V	V	**V**
F	F	V	V	V	F	F	V	F	V	V	V	V	**V**
F	V	F	F	F	V	F	V	V	V	V	V	V	**V**
F	V	F	F	V	V	F	V	F	F	V	V	V	**V**
F	V	F	V	F	V	F	V	V	V	V	V	V	**V**
F	V	F	V	V	V	F	V	F	V	V	V	V	**V**
F	V	V	F	F	F	F	V	V	V	V	V	V	**V**
F	V	V	F	V	F	F	V	F	F	F	F	F	**V**
F	V	V	V	F	F	F	V	V	V	V	V	V	**V**
F	V	V	V	V	F	F	V	F	V	V	V	V	**V**
V	F	F	F	F	V	F	V	V	V	V	V	V	**V**
V	F	F	F	V	V	F	V	F	F	V	V	V	**V**
V	F	F	V	F	V	F	V	V	V	V	V	V	**V**
V	F	F	V	V	V	F	V	F	V	V	V	V	**V**
V	F	V	F	F	F	F	V	V	V	V	V	V	**V**
V	F	V	F	V	F	F	V	F	F	F	F	V	**V**
V	F	V	V	F	F	F	V	V	V	V	V	V	**V**
V	F	V	V	V	F	F	V	F	V	V	V	V	**V**
V	V	F	F	F	V	V	V	V	V	V	V	V	**V**
V	V	F	F	V	V	V	V	F	F	V	V	F	**F**
V	V	F	V	F	V	V	V	V	V	V	V	V	**V**
V	V	F	V	V	V	V	V	F	V	V	V	V	**V**
V	V	V	F	F	F	V	F	V	V	V	F	V	**V**
V	V	V	F	V	F	V	F	F	F	F	F	F	**V**
V	V	V	V	F	F	V	F	V	V	V	F	V	**V**
V	V	V	V	V	F	V	F	F	V	V	F	V	**V**

Resultado: La columna (**W**) contiene un valor ***(F)***, fila 26 (VVFFV). En el resto de los casos es ***(V)***. Por lo tanto, la proposición se clasifica como una **Contingencia**.

Enunciado 4: A continuación, representa con notación lógica el siguiente argumento.

"Si hace buen tiempo, vamos a la playa. Si vamos a la playa o hay mucha gente, entonces no nos quedamos en casa. Por lo tanto, si no vamos a la playa y nos quedamos en casa, entonces no hace buen tiempo."

1. Identificación de variables y notación lógica, ver Tabla 1.66

Variables proposicionales y notación

Variables, P: Hace buen tiempo. **Q:** Vamos a la playa. y **R:** Nos quedamos en casa.
Notación $T = [(P \to Q) \land (Q \to \neg R)] \to [(\neg Q \land R) \to \neg P]$

Tabla 1.66: Traducción de premisas y conclusión del enunciado 4

Parte del enunciado	Notación lógica
Premisa 1: Si hace buen tiempo, vamos a la playa.	$\mathbf{P \to Q}$
Premisa 2: Si vamos a la playa, entonces no nos quedamos en casa.	$\mathbf{Q \to \neg R}$
Conclusión: Si no vamos a la playa y nos quedamos en casa, entonces no hace buen tiempo.	$\mathbf{(\neg Q \land R) \to \neg P}$

2. Tabla de verdad y clasificación, ver Tabla 1.67

2. Tabla de verdad y clasificación (3 Variables: 8 Filas)

Evaluamos $T = [(P \to Q) \land (Q \to \neg R)] \to [(\neg Q \land R) \to \neg P]$

Tabla 1.67: Evaluación completa de la proposición de 3 variables

Entradas			Proposiciones y Evaluación						
P	Q	R	$\mathbf{P \to Q}$	$\mathbf{\neg R}$	$\mathbf{Q \to \neg R}$	$\mathbf{P_1 \land P_2}$	$\mathbf{\neg Q \land R}$	$\mathbf{\neg P}$	T
F	F	F	V	V	V	V	F	V	**V**
F	F	V	V	F	V	V	V	V	**V**
F	V	F	V	V	V	V	F	V	**V**
F	V	V	V	F	F	F	F	V	**V**
V	F	F	F	V	V	F	F	F	**V**
V	F	V	F	F	V	F	V	F	**V**
V	V	F	V	V	V	V	F	F	**V**
V	V	V	V	F	F	F	F	F	**V**

Clasificación del resultado: La columna de la proposición compuesta total (**T**) contiene **solamente** valores *Verdaderos (V)*. Por lo tanto, la proposición se clasifica como una **Tautología**.

Enunciado 5: A continuación, representa con notación lógica el siguiente argumento.

"Si el sistema está en línea, los usuarios pueden acceder. Por lo tanto, el sistema no está en línea o los usuarios pueden acceder."

Análisis lógico completo (2 variables), ver Tabla 1.68

Análisis de la Proposición (Equivalencia Lógica)

1. Variables y notación

P: El sistema está en línea.

Q: Los usuarios pueden acceder.

Notación $T = (P \to Q) \to (\neg P \lor Q)$

2. Tabla de Verdad y clasificación

Tabla 1.68: Evaluación de $T = (P \to Q) \to (\neg P \lor Q)$

Entradas		Proposiciones y Evaluación			
P	Q	$\mathbf{P \to Q}$	$\mathbf{\neg P}$	$\mathbf{\neg P \lor Q}$	T
V	V	V	F	V	**V**
V	F	F	F	F	**V**
F	V	V	V	V	**V**
F	F	V	V	V	**V**

Clasificación del resultado: La columna de la proposición compuesta total (**T**) contiene **solamente** valores *Verdaderos (V)*. Por lo tanto, la proposición se clasifica como una **Tautología**.

Enunciado 6: A continuación, representa con notación lógica el siguiente argumento.

"Si invierto en criptomonedas y la economía es estable, gano dinero. Si no invierto en criptomonedas, la economía no es estable. Por lo tanto, si gano dinero, la economía es estable."

Análisis lógico completo (3 variables), ver Tabla 1.69 y Tabla 1.70

Análisis de la Proposición (Contingencia)

1. Variables y notación

P: Invierto en criptomonedas.
Q: La economía es estable.
R: Gano dinero.

Notación $T = [((P \land Q) \to R) \land (\neg P \to \neg Q)] \to (R \to Q)$

Tabla 1.69: Traducción de premisas y conclusión del enunciado 6

Parte del enunciado	Notación lógica
Premisa 1: Si invierto en criptomonedas y la economía es estable, gano dinero.	$(\mathbf{P} \land \mathbf{Q}) \to \mathbf{R}$
Premisa 2: Si no invierto en criptomonedas, la economía no es estable.	$\neg\mathbf{P} \to \neg\mathbf{Q}$
Conclusión: Si gano dinero, la economía es estable.	$\mathbf{R} \to \mathbf{Q}$

2. Tabla de Verdad y clasificación

Tabla 1.70: Evaluación de $T = [((P \land Q) \to R) \land (\neg P \to \neg Q)] \to (R \to Q)$

Entradas			Proposiciones y Evaluación							
P	Q	R	$\mathbf{P \land Q}$	$\mathbf{P_1}$	$\mathbf{\neg P}$	$\mathbf{\neg Q}$	$\mathbf{P_2}$	$\mathbf{P_1 \land P_2}$	$\mathbf{R \to Q}$	T
V	V	V	V	V	F	F	V	V	V	**V**
V	V	F	V	F	F	F	V	F	V	**V**
V	F	V	F	V	F	V	V	V	F	**F**
V	F	F	F	V	F	V	V	V	V	**V**
F	V	V	F	V	V	F	F	F	V	**V**
F	V	F	F	V	V	F	F	F	V	**V**
F	F	V	F	V	V	V	V	V	F	**F**
F	F	F	F	V	V	V	V	V	V	**V**

Clasificación del resultado: La columna de la proposición compuesta total (**T**) contiene valores *Verdaderos (V)* y *Falsos (F)* (en las filas 3 y 7). Por lo tanto, la proposición se clasifica como una **Contingencia**.

Enunciado 7: A continuación, representa con notación lógica el siguiente argumento.

"Si un estudiante asiste a todas las clases y estudia para el examen, entonces aprueba el curso. Si el estudiante aprueba el curso, obtiene su título. El estudiante no asiste a todas las clases o no obtiene su título. Por lo tanto, el estudiante no aprueba el curso o no estudia para el examen."

Análisis lógico completo (4 variables), ver Tabla 1.71 y Tabla 1.72

Análisis de la Proposición (Contingencia)

1. Variables y notación

- **P:** El estudiante asiste a todas las clases.
- **Q:** El estudiante estudia para el examen.
- **R:** El estudiante aprueba el curso.
- **S:** El estudiante obtiene su título.

Tabla 1.71: Traducción de premisas y conclusión del enunciado 7

Parte del enunciado	Notación lógica
Premisa 1: Si asiste a clases y estudia, entonces aprueba el curso.	$(\mathbf{P} \land \mathbf{Q}) \to \mathbf{R}$
Premisa 2: Si el estudiante aprueba el curso, obtiene su título.	$\mathbf{R} \to \mathbf{S}$
Premisa 3: El estudiante no asiste a clases o no obtiene su título.	$\neg\mathbf{P} \lor \neg\mathbf{S}$
Conclusión: El estudiante no aprueba el curso o no estudia para el examen.	$\neg\mathbf{R} \lor \neg\mathbf{Q}$

Notación $T = [((P \land Q) \to R) \land (R \to S) \land (\neg P \lor \neg S)] \to (\neg R \lor \neg Q)$

2. Tabla de Verdad y clasificación

Tabla 1.72: Evaluación de $T = [((P \land Q) \to R) \land (R \to S) \land (\neg P \lor \neg S)] \to (\neg R \lor \neg Q)$

Entradas				Proposiciones y Evaluación								
P	Q	R	S	$\mathbf{P \land Q}$	$\mathbf{P_1}$	$\mathbf{P_2}$	$\neg\mathbf{P}$	$\neg\mathbf{S}$	$\mathbf{P_3}$	PREMISAS	CONCL.	T
F	F	F	F	F	V	V	V	V	V	V	V	**V**
F	F	F	V	F	V	V	V	F	V	V	V	**V**
F	F	V	F	F	V	F	V	V	V	F	F	**V**
F	F	V	V	F	V	V	V	F	V	V	F	**F**
F	V	F	F	F	V	V	V	V	V	V	V	**V**
F	V	F	V	F	V	V	V	F	V	V	V	**V**
F	V	V	F	F	V	F	V	V	V	F	F	**V**
F	V	V	V	F	V	V	V	F	V	V	F	**F**
V	F	F	F	F	V	V	F	V	F	F	V	**V**
V	F	F	V	F	V	V	F	F	F	F	V	**V**
V	F	V	F	F	V	F	F	V	F	F	F	**V**
V	F	V	V	F	V	V	F	F	F	F	F	**V**
V	V	F	F	V	F	V	F	V	F	F	V	**V**
V	V	F	V	V	F	V	F	F	F	F	V	**V**
V	V	V	F	V	V	F	F	V	F	F	F	**V**
V	V	V	V	V	V	V	F	F	F	F	F	**V**

Clasificación del resultado: La columna de la proposición compuesta total (**T**) contiene valores *Verdaderos (V)* y *Falsos (F)* (en las filas 4, 8, 12 y 16). Por lo tanto, la proposición se clasifica como una **Contingencia**.

Enunciado 8: A continuación, representa con notación lógica el siguiente argumento.

"Si la empresa invierte en I+D, sus productos serán innovadores. Si los productos son innovadores y la publicidad es efectiva, entonces las ventas aumentan. La empresa no invierte en I+D o el mercado es favorable. Si el mercado es favorable y las ventas aumentan, entonces la publicidad es efectiva. Por lo tanto, si las ventas no aumentan, la publicidad no es efectiva y la empresa no invierte en I+D."

Análisis lógico completo (5 variables), ver Tabla 1.73 y Tabla 1.74

Análisis de la Proposición (5 Variables: 32 Filas)

La variables son: P: La empresa invierte en I+D. **Q:** Los productos son innovadores. **R:** La publicidad es efectiva. **S:** Las ventas aumentan. **T:** El mercado es favorable.

Tabla 1.73: Traducción de premisas y conclusión del enunciado 8

Parte del enunciado	Notación lógica
Premisa 1: Si la empresa invierte en I+D, sus productos serán innovadores.	$\mathbf{P \to Q}$
Premisa 2: Si los productos son innovadores y la publicidad es efectiva, entonces las ventas aumentan.	$\mathbf{(Q \wedge R) \to S}$
Premisa 3: La empresa no invierte en I+D o el mercado es favorable.	$\mathbf{\neg P \vee T}$
Premisa 4: Si el mercado es favorable y las ventas aumentan, entonces la publicidad es efectiva.	$\mathbf{(T \wedge S) \to R}$
Conclusión: Si las ventas no aumentan, la publicidad no es efectiva y la empresa no invierte en I+D.	$\mathbf{\neg S \to (\neg R \wedge \neg P)}$

$$W = [(P \to Q) \wedge ((Q \wedge R) \to S) \wedge (\neg P \vee T) \wedge ((T \wedge S) \to R)] \to (\neg S \to (\neg R \wedge \neg P))$$

Tabla 1.74: Evaluación enunciado 8 - $W = [P_1 \wedge P_2 \wedge P_3 \wedge P_4] \to$ Conclusión

Entradas					Proposiciones y Evaluación									
P	Q	R	S	T	$\mathbf{P_1}$	$\mathbf{Q \wedge R}$	$\mathbf{P_2}$	$\mathbf{P_3}$	$\mathbf{T \wedge S}$	$\mathbf{P_4}$	PREMISAS	$\neg$S	CONCL.	W
F	F	F	F	F	V	F	V	V	F	V	V	V	V	**V**
F	F	F	F	V	V	F	V	V	F	V	V	V	V	**V**
F	F	F	V	F	V	F	V	V	F	V	V	F	V	**V**
F	F	F	V	V	V	F	V	V	V	F	F	F	V	**V**
F	F	V	F	F	V	F	V	V	F	V	V	V	V	**V**
F	F	V	F	V	V	F	V	V	F	V	V	V	V	**V**
F	F	V	V	F	V	F	V	V	F	V	V	F	V	**V**
F	F	V	V	V	V	F	V	V	V	V	V	F	V	**V**
F	V	F	F	F	V	F	V	V	F	V	V	V	V	**V**
F	V	F	F	V	V	F	V	V	F	V	V	V	V	**V**
F	V	F	V	F	V	F	V	V	F	V	V	F	V	**V**
F	V	F	V	V	V	F	V	V	V	F	F	F	V	**V**
F	V	V	F	F	V	V	F	V	F	V	F	V	V	**V**
F	V	V	F	V	V	V	F	V	F	V	F	V	V	**V**
F	V	V	V	F	V	V	V	V	F	V	V	F	V	**V**
F	V	V	V	V	V	V	V	V	V	V	V	F	V	**V**
V	F	F	F	F	F	F	V	F	F	V	F	V	V	**V**
V	F	F	F	V	F	F	V	V	F	V	F	V	V	**V**
V	F	F	V	F	F	F	V	F	F	V	F	F	V	**V**
V	F	F	V	V	F	F	V	V	V	F	F	F	V	**V**
V	F	V	F	F	F	F	V	F	F	V	F	V	V	**V**
V	F	V	F	V	F	F	V	V	F	V	F	V	V	**V**
V	F	V	V	F	F	F	V	F	F	V	F	F	V	**V**
V	F	V	V	V	F	F	V	V	V	V	F	F	V	**V**
V	V	F	F	F	V	F	V	F	F	V	F	V	V	**V**
V	V	F	F	V	V	F	V	V	F	V	V	V	V	**V**
V	V	F	V	F	V	F	V	F	F	V	F	F	V	**V**
V	V	F	V	V	V	F	V	V	V	F	F	F	V	**V**
V	V	V	F	F	V	V	F	F	F	V	F	V	V	**V**
V	V	V	F	V	V	V	F	V	F	V	F	V	V	**V**
V	V	V	V	F	V	V	V	F	F	V	F	F	V	**V**
V	V	V	V	V	V	V	V	V	V	V	V	F	V	**V**

Clasificación del resultado: La columna de la proposición compuesta total (**W**) contiene *solamente* valores *Verdaderos (V)* en las 32 filas. Por lo tanto, la proposición se clasifica como una **Tautología**.

Enunciado 9: A continuación, representa con notación lógica el siguiente argumento.

"La empresa implementa IA generativa (P) y la calidad del código (Q) es alta. La satisfacción del cliente (S) aumenta si la interfaz de usuario (R) se mejora o la calidad del código no es alta. Si el gasto en capacitación (T) es alto, entonces la empresa implementa IA generativa. Si la satisfacción del cliente aumenta, entonces el gasto en capacitación es alto. Por lo tanto, la interfaz de usuario se mejora o la satisfacción del cliente aumenta."

Análisis lógico completo (5 variables), ver Tabla 1.75 y Tabla 1.76

Análisis de la Proposición (5 Variables: 32 Filas)

Variables. P: La empresa implementa IA generativa. **Q:** La calidad del código es alta. **R:** La interfaz de usuario se mejora. **S:** La satisfacción del cliente aumenta. **T:** El gasto en capacitación es alto.

Tabla 1.75: Traducción de premisas y conclusión del enunciado 9

Parte del enunciado	Notación lógica
Premisa 1 (P_1)**:** La empresa implementa IA generativa y la calidad del código es alta.	$\mathbf{P \wedge Q}$
Premisa 2 (P_2)**:** La satisfacción del cliente aumenta si la interfaz de usuario se mejora o la calidad del código no es alta.	$\mathbf{(R \vee \neg Q) \rightarrow S}$
Premisa 3 (P_3)**:** Si el gasto en capacitación es alto, entonces la empresa implementa IA generativa.	$\mathbf{T \rightarrow P}$
Premisa 4 (P_4)**:** Si la satisfacción del cliente aumenta, entonces el gasto en capacitación es alto.	$\mathbf{S \rightarrow T}$
Conclusión: Por lo tanto, la interfaz de usuario se mejora o la satisfacción del cliente aumenta.	$\mathbf{R \vee S}$

Notación: $W = [(P \wedge Q) \wedge ((R \vee \neg Q) \rightarrow S) \wedge (T \rightarrow P) \wedge (S \rightarrow T)] \rightarrow (R \vee S)$

Tabla 1.76: Evaluación enunciado 9 - $W = [P_1 \wedge P_2 \wedge P_3 \wedge P_4] \rightarrow$ Conclusión

Entradas					Proposiciones y Evaluación								
P	Q	R	S	T	$\mathbf{P_1}$	$\mathbf{\neg Q}$	$\mathbf{R \vee \neg Q}$	$\mathbf{P_2}$	$\mathbf{P_3}$	$\mathbf{P_4}$	PREMISAS	$\mathbf{R \vee S}$	W
F	F	F	F	F	F	V	V	F	V	V	F	F	**V**
F	F	F	F	V	F	V	V	F	F	F	F	F	**V**
F	F	F	V	F	F	V	V	V	V	F	F	V	**V**
F	F	F	V	V	F	V	V	V	F	V	F	V	**V**
F	F	V	F	F	F	V	V	F	V	V	F	V	**V**
F	F	V	F	V	F	V	V	F	F	V	F	V	**V**
F	F	V	V	F	F	V	V	V	V	F	F	V	**V**
F	F	V	V	V	F	V	V	V	F	V	F	V	**V**
F	V	F	F	F	F	F	F	V	V	V	F	F	**V**
F	V	F	F	V	F	F	F	V	F	F	F	F	**V**
F	V	F	V	F	F	F	F	V	V	F	F	V	**V**
F	V	F	V	V	F	F	F	V	F	V	F	V	**V**
F	V	V	F	F	F	F	V	F	V	V	F	V	**V**
F	V	V	F	V	F	F	V	F	F	V	F	V	**V**
F	V	V	V	F	F	F	V	V	V	F	F	V	**V**
F	V	V	V	V	F	F	V	V	F	V	F	V	**V**
V	F	F	F	F	F	V	V	F	V	V	F	F	**V**
V	F	F	F	V	F	V	V	F	V	F	F	F	**V**
V	F	F	V	F	F	V	V	V	V	F	F	V	**V**
V	F	F	V	V	F	V	V	V	V	V	F	V	**V**
V	F	V	F	F	F	V	V	F	V	V	F	V	**V**
V	F	V	F	V	F	V	V	F	V	V	F	V	**V**
V	F	V	V	F	F	V	V	V	V	F	F	V	**V**
V	F	V	V	V	F	V	V	V	V	V	F	V	**V**
V	V	F	F	F	V	F	F	V	V	V	F	F	**F**
V	V	F	F	V	V	F	F	V	V	F	V	F	**F**
V	V	F	V	F	V	F	F	V	V	F	V	V	**V**
V	V	F	V	V	V	F	F	V	V	V	V	V	**V**
V	V	V	F	F	V	F	V	F	V	V	V	V	**V**
V	V	V	F	V	V	F	V	F	V	V	V	V	**V**
V	V	V	V	F	V	F	V	V	V	F	V	V	**V**
V	V	V	V	V	V	F	V	V	V	V	V	V	**V**

Clasificación del resultado: La columna (**W**) contiene valores *Verdaderos (V)* y *Falsos (F)* en las 32 filas. Por lo tanto, la proposición se clasifica como una **Contingencia**.

Enunciado 10: Representa con notación lógica el siguiente argumento.

"Si la IA Generativa personaliza el aprendizaje (P), el compromiso del estudiante (Q) aumenta. Los educadores tienen capacitación adecuada (R) y la IA Generativa se implementa en el aula (S). Si el compromiso del estudiante no aumenta y la IA se implementa en el aula, entonces el gasto en tecnología (P) se mantiene bajo o el compromiso del estudiante aumenta. Por lo tanto, si los educadores no tienen capacitación, la IA no personaliza el aprendizaje o el gasto en tecnología se mantiene bajo."

Análisis lógico completo (4 variables), ver Tabla 1.77 y Tabla 1.78

Análisis de la Proposición (4 Variables: 16 Filas)

Las variables son: P: La IA Generativa personaliza el aprendizaje (Gasto en tecnología se mantiene bajo $\neg P$). **Q:** El compromiso del estudiante aumenta. **R:** Los educadores tienen capacitación adecuada. **S:** La IA Generativa se implementa en el aula.

Tabla 1.77: Traducción de premisas y conclusión del enunciado 10

Parte del enunciado	Notación lógica
Premisa 1 (P_1)**:** Si la IA Generativa personaliza el aprendizaje, el compromiso del estudiante aumenta.	$\mathbf{P \rightarrow Q}$
Premisa 2 (P_2)**:** Los educadores tienen capacitación adecuada y la IA Generativa se implementa en el aula.	$\mathbf{R \wedge S}$
Premisa 3 (P_3)**:** Si el compromiso del estudiante no aumenta y la IA se implementa en el aula, entonces el gasto en tecnología se mantiene bajo o el compromiso del estudiante aumenta.	$(\neg \mathbf{Q} \wedge \mathbf{S}) \rightarrow (\neg \mathbf{P} \vee \mathbf{Q})$
Conclusión: Por lo tanto, si los educadores no tienen capacitación, la IA no personaliza el aprendizaje o el gasto en tecnología se mantiene bajo.	$\neg \mathbf{R} \rightarrow (\neg \mathbf{P} \vee \neg \mathbf{S})$

Notación: $T = [(P \rightarrow Q) \wedge (R \wedge S) \wedge ((\neg Q \wedge S) \rightarrow (\neg P \vee Q))] \rightarrow (\neg R \rightarrow (\neg P \vee \neg S))$

Tabla 1.78: Evaluación enunciado 10 - $T = [P_1 \wedge P_2 \wedge P_3] \rightarrow$ Conclusión

Entradas				Proposiciones y Evaluación						
P	Q	R	S	$\mathbf{P_1}$	$\mathbf{P_2}$	$\neg \mathbf{Q} \wedge \mathbf{S}$	$\mathbf{P_3}$	PREMISAS	CONCL.	T
F	F	F	F	V	F	F	V	F	V	**V**
F	F	F	V	V	F	V	F	F	V	**V**
F	F	V	F	V	F	F	V	F	V	**V**
F	F	V	V	V	V	V	V	V	V	**V**
F	V	F	F	V	F	F	V	F	V	**V**
F	V	F	V	V	F	F	V	F	V	**V**
F	V	V	F	V	F	F	V	F	V	**V**
F	V	V	V	V	V	F	V	V	V	**V**
V	F	F	F	F	F	F	V	F	V	**V**
V	F	F	V	F	F	V	V	F	V	**V**
V	F	V	F	F	F	F	V	F	V	**V**
V	F	V	V	F	V	V	V	F	V	**V**
V	V	F	F	V	F	F	V	F	V	**V**
V	V	F	V	V	F	F	V	F	V	**V**
V	V	V	F	V	F	F	V	F	V	**V**
V	V	V	V	V	V	F	V	V	V	**V**

Clasificación del resultado: La columna de la proposición compuesta total (**T**) contiene *solamente* valores *Verdaderos (V)* en las 16 filas. Por lo tanto, la proposición se clasifica como una **Tautología.**

Enunciado 11: Representa con notación lógica el siguiente argumento.

"Si el contenido generado por IA es malicioso (P), la desinformación (Q) aumenta. La regulación es estricta (R) si la IA no es transparente. Si la desinformación aumenta y la IA no es transparente, entonces la confianza pública (S) se deteriora. Por lo tanto, si la confianza pública se deteriora, el contenido generado por IA no es malicioso o la regulación no es estricta."

Análisis lógico completo (4 variables), ver Tabla 1.79 y Tabla 1.80

Análisis de la Proposición (4 Variables: 16 Filas)

La variables son: P: El contenido generado por IA es malicioso. **Q:** La desinformación aumenta. **R:** La regulación es estricta. **S:** La IA es transparente ($\neg S$: Confianza pública se deteriora).

Tabla 1.79: Traducción de premisas y conclusión del enunciado 11

Parte del enunciado	Notación lógica
Premisa 1 (P_1)**:** Si el contenido generado por IA es malicioso, la desinformación aumenta.	$\mathbf{P \to Q}$
Premisa 2 (P_2)**:** La regulación es estricta si la IA no es transparente.	$\mathbf{\neg S \to R}$
Premisa 3 (P_3)**:** Si la desinformación aumenta y la IA no es transparente, entonces la confianza pública se deteriora.	$\mathbf{(Q \wedge \neg S) \to \neg S}$
Conclusión: Por lo tanto, si la confianza pública se deteriora, el contenido generado por IA no es malicioso o la regulación no es estricta.	$\mathbf{\neg S \to (\neg P \vee \neg R)}$

Notación: $T = [(P \to Q) \wedge (\neg S \to R) \wedge ((Q \wedge \neg S) \to \neg S)] \to (\neg S \to (\neg P \vee \neg R))$

Tabla 1.80: Evaluación enunciado 11 - $T = [P_1 \wedge P_2 \wedge P_3] \to$ Conclusión

Entradas				Proposiciones y Evaluación							
P	Q	R	S	$\mathbf{P_1}$	$\mathbf{\neg S}$	$\mathbf{P_2}$	$\mathbf{Q \wedge \neg S}$	$\mathbf{P_3}$	PREMISAS	CONCL.	T
F	F	F	F	V	V	F	F	V	F	F	**V**
F	F	F	V	V	F	V	F	V	V	V	**V**
F	F	V	F	V	V	V	F	V	V	V	**V**
F	F	V	V	V	F	V	F	V	V	V	**V**
F	V	F	F	V	V	F	V	V	F	F	**V**
F	V	F	V	V	F	V	F	V	V	V	**V**
F	V	V	F	V	V	V	V	V	V	V	**V**
F	V	V	V	V	F	V	F	V	V	V	**V**
V	F	F	F	F	V	F	F	V	F	F	**V**
V	F	F	V	F	F	V	F	V	F	V	**V**
V	F	V	F	F	V	V	F	V	F	V	**V**
V	F	V	V	F	F	V	F	V	F	V	**V**
V	V	F	F	V	V	F	V	V	F	F	**V**
V	V	F	V	V	F	V	F	V	V	V	**V**
V	V	V	F	V	V	V	V	V	V	V	**V**
V	V	V	V	V	F	V	F	V	V	V	**V**

Clasificación del resultado: La columna de la proposición compuesta total (**T**) contiene *solamente* valores *Verdaderos (V)* en las 16 filas. Por lo tanto, la proposición se clasifica como una **Tautología.**

1.4.2. Sección 2 del laboratorio 2.

Ejercicio 1: Representa cada una de las notaciones lógicas en forma de enunciados.
Notación lógica: $[(p \vee q) \implies \neg r] \wedge [r \iff s] \implies [(r \wedge s) \implies (\neg p \vee \neg q)]$
Considere las siguientes proposiciones: variable p: Compré un automóvil, variable q: Me levanté más temprano, variable r: Llegué tarde al trabajo, variable s: Fui despedido.

Fase 1: Traducción y representación de la proposición

Argumento y notación lógica (4 variables)

Variables proposicionales:

- **p**: Compré un automóvil.
- **q**: Me levanté más temprano.
- **r**: Llegué tarde al trabajo.
- **s**: Fui despedido.

Notación lógica:

$$T = [(p \vee q) \implies \neg r] \wedge [r \iff s] \implies [(r \wedge s) \implies (\neg p \vee \neg q)]$$

Estructura del argumento ver Tabla 1.81:

- **Premisa** (Pr): $[(p \vee q) \implies \neg r] \wedge [r \iff s]$
- **Conclusión** (C): $(r \wedge s) \implies (\neg p \vee \neg q)$

Tabla 1.81: Traducción de premisas y conclusión del ejercicio 1

Parte del enunciado	Notación lógica
Premisa 1 (P_1)**:** Si compré un automóvil o me levanté más temprano, entonces no llegué tarde al trabajo.	$(\mathbf{p} \vee \mathbf{q}) \implies \neg \mathbf{r}$
Premisa 2 (P_2)**:** Llegué tarde al trabajo si y solo si fui despedido.	$\mathbf{r} \iff \mathbf{s}$
Conclusión (C)**:** Si llegué tarde al trabajo y fui despedido, entonces no compré un automóvil o no me levanté más temprano.	$(\mathbf{r} \wedge \mathbf{s}) \implies (\neg \mathbf{p} \vee \neg \mathbf{q})$

Fase 2: Estructura del argumento

Argumento estructurado

Notación lógica a descomponer:

$$T = [(p \vee q) \implies \neg r] \wedge [r \iff s] \implies [(r \wedge s) \implies (\neg p \vee \neg q)]$$

Traducción estructurada

$$\mathbf{T} = \underbrace{[(p \vee q) \implies \neg r] \wedge [r \iff s]}_{\text{PREMISA (Pr)}} \implies \underbrace{[(r \wedge s) \implies (\neg p \vee \neg q)]}_{\text{CONCLUSIÓN (C)}}$$

Significado: La verdad de todo el argumento (**T**) depende de la implicación entre la conjunción de las dos premisas y la conclusión.

Fase 3: Enunciado final

Enunciado final completo del argumento lógico traducido a lenguaje natural

"**Si es cierto que** *si compré un automóvil* **o** *me levanté más temprano*, **entonces** *no llegué tarde* al trabajo**.** Llegué tarde al trabajo **si y solo si** *fui despedido*. **Por lo tanto**, *si llegué tarde al trabajo* **y** *fui despedido*, **entonces** *no compré un automóvil* **o** *no me levanté* más temprano."

Ejercicio 2: Representa cada una de las notaciones lógicas en forma de enunciados.
Notación lógica: $[(\neg p \land q) \implies r] \lor [(r \lor s) \iff \neg q] \implies [r \implies (\neg p \land s)]$

Considere las siguientes proposiciones: variable p: La vigilancia digital es mínima, variable q: La privacidad del usuario se respeta, variable r: Las leyes son insuficientes, variable s: La seguridad de los datos es robusta.

Fase 1: Traducción y representación de la proposición

Argumento y notación lógica (4 variables)

Variables proposicionales:

- **p**: La vigilancia digital es mínima.
- **q**: La privacidad del usuario se respeta.
- **r**: Las leyes son insuficientes.
- **s**: La seguridad de los datos es robusta.

Notación lógica:

$$T = [(\neg p \land q) \implies r] \lor [(r \lor s) \iff \neg q] \implies [r \implies (\neg p \land s)]$$

Estructura del argumento ver Tabla 1.82:

- **Premisa** (Pr): $[(\neg p \land q) \implies r] \lor [(r \lor s) \iff \neg q]$
- **Conclusión** (C): $[r \implies (\neg p \land s)]$

Tabla 1.82: Traducción de premisas y conclusión del ejercicio 2

Parte del enunciado	Notación lógica
Premisa 1 (P_1)**:** Si la vigilancia digital no es mínima y la privacidad del usuario se respeta, entonces las leyes son insuficientes.	$(\neg \mathbf{p} \land \mathbf{q}) \implies \mathbf{r}$
Premisa 2 (P_2)**:** Las leyes son insuficientes o la seguridad de los datos es robusta, si y solo si la privacidad del usuario no se respeta.	$(\mathbf{r} \lor \mathbf{s}) \iff \neg \mathbf{q}$
Conclusión (C)**:** Si las leyes son insuficientes, entonces la vigilancia digital no es mínima y la seguridad de los datos es robusta.	$\mathbf{r} \implies (\neg \mathbf{p} \land \mathbf{s})$

Fase 2: Estructura del argumento

Argumento estructurado

Notación lógica a descomponer:

$$T = [(\neg p \land q) \implies r] \lor [(r \lor s) \iff \neg q] \implies [r \implies (\neg p \land s)]$$

Traducción estructurada

$$\mathbf{T} = \underbrace{[(\neg p \land q) \implies r] \lor [(r \lor s) \iff \neg q]}_{\text{PREMISA (Pr)}} \implies \underbrace{[r \implies (\neg p \land s)]}_{\text{CONCLUSIÓN (C)}}$$

Significado: La verdad de todo el argumento (**T**) depende de la implicación entre la disyunción de las dos premisas y la conclusión.

Fase 3: Enunciado final

Enunciado final completo del argumento lógico traducido a lenguaje natural

> "**Si es cierto que** (*si la vigilancia digital no es mínima* **y** *la privacidad del usuario se respeta*, **entonces** *las leyes son insuficientes*) **o** (*las leyes son insuficientes* **o** *la seguridad de los datos es robusta*, **si y solo si** *la privacidad del usuario no se respeta*). **Por lo tanto**, *si las leyes son insuficientes*, **entonces** *la vigilancia digital no es mínima* **y** *la seguridad de los datos es robusta*."

Ejercicio 3: Representa cada una de las notaciones lógicas en forma de enunciados.
Notación: $(\neg p \implies \neg q) \wedge [\neg r \implies \neg s] \wedge [(\neg q \vee \neg s) \implies t] \implies [\neg t \implies (p \wedge r)]$
Considere las siguientes proposiciones: variable p: Acelero la motocicleta, variable q: La motocicleta corre, variable r: Freno la motocicleta, variable s: La motocicleta se detiene, variable t: La motocicleta falla.

Fase 1: Traducción y representación de la proposición

Argumento y notación lógica (5 variables)

Variables proposicionales:

- **p**: Acelero la motocicleta.
- **q**: La motocicleta corre.
- **r**: Freno la motocicleta.
- **s**: La motocicleta se detiene.
- **t**: La motocicleta falla.

Notación lógica:

$$T = (\neg p \implies \neg q) \wedge [\neg r \implies \neg s] \wedge [(\neg q \vee \neg s) \implies t] \implies [\neg t \implies (p \wedge r)]$$

Estructura del argumento ver Tabla 1.83:

- **Premisa** (Pr): $(\neg p \implies \neg q) \wedge [\neg r \implies \neg s] \wedge [(\neg q \vee \neg s) \implies t]$
- **Conclusión** (C): $\neg t \implies (p \wedge r)$

Tabla 1.83: Traducción de premisas y conclusión del ejercicio 3

Parte del enunciado	Notación lógica
Premisa 1 (P_1)**:** Si no acelero la motocicleta, la motocicleta no corre.	$\neg \mathbf{p} \implies \neg \mathbf{q}$
Premisa 2 (P_2)**:** Si no freno la motocicleta, la motocicleta no se detiene.	$\neg \mathbf{r} \implies \neg \mathbf{s}$
Premisa 3 (P_3)**:** Si la motocicleta no corre o la motocicleta no se detiene, la motocicleta falla.	$(\neg \mathbf{q} \vee \neg \mathbf{s}) \implies \mathbf{t}$
Conclusión (C)**:** Si la motocicleta no falla, entonces acelero la motocicleta y freno la motocicleta.	$\neg \mathbf{t} \implies (\mathbf{p} \wedge \mathbf{r})$

Fase 2: Estructura del argumento

Argumento estructurado (5 variables)

Notación lógica a descomponer:

$$T = [(\neg p \implies \neg q) \wedge (\neg r \implies \neg s) \wedge ((\neg q \vee \neg s) \implies t)] \implies [\neg t \implies (p \wedge r)]$$

Traducción estructurada

$$\mathbf{T} = \underbrace{(\neg p \implies \neg q) \wedge [\neg r \implies \neg s] \wedge [(\neg q \vee \neg s) \implies t]}_{\text{PREMISA (Pr)}} \implies \underbrace{[\neg t \implies (p \wedge r)]}_{\text{CONCLUSIÓN (C)}}$$

Significado: La verdad de todo el argumento (**T**) depende de la implicación entre la conjunción de las tres premisas y la conclusión.

Fase 3: Enunciado final

Enunciado final completo del argumento lógico traducido a lenguaje natural

"**Si** (*no acelero la motocicleta*, *la motocicleta no corre*) **y** (*si no freno la motocicleta*, *la motocicleta no se detiene*) **y** (*si la motocicleta no corre* **o** *la motocicleta no se detiene*, **entonces** *la motocicleta falla*); **por lo tanto**, *si la motocicleta no falla*, **entonces** *acelero la motocicleta* **y** *freno la motocicleta.*"

Ejercicio 4: Representa cada una de las notaciones lógicas en forma de enunciados.
Notación lógica: $[(p \land \neg q) \implies r] \land [(r \iff s) \lor \neg p] \implies [\neg s \implies (\neg q \land r)]$

Considere las siguientes proposiciones: variable p: El contenido generado por IA es supervisado, variable q: La desinformación se propaga rápidamente, variable r: La regulación legal es estricta, variable s: Los derechos de autor están protegidos.

Fase 1: Traducción y representación de la proposición

Argumento y notación lógica (4 variables)

Variables proposicionales:

- **p**: El contenido generado por IA es supervisado.
- **q**: La desinformación se propaga rápidamente.
- **r**: La regulación legal es estricta.
- **s**: Los derechos de autor están protegidos.

Notación lógica:

$$T = [(p \land \neg q) \implies r] \land [(r \iff s) \lor \neg p] \implies [\neg s \implies (\neg q \land r)]$$

Estructura del argumento ver Tabla 1.84:

- **Premisa** (Pr): $[(p \land \neg q) \implies r] \land [(r \iff s) \lor \neg p]$
- **Conclusión** (C): $\neg s \implies (\neg q \land r)$

Tabla 1.84: Traducción de premisas y conclusión del ejercicio 4

Parte del enunciado	Notación lógica
Premisa 1 (P_1)**:** Si el contenido generado por IA es supervisado y la desinformación no se propaga rápidamente, entonces la regulación legal es estricta.	$(\mathbf{p} \land \neg \mathbf{q}) \implies \mathbf{r}$
Premisa 2 (P_2)**:** La regulación legal es estricta si y solo si los derechos de autor están protegidos, o el contenido generado por IA no es supervisado.	$(\mathbf{r} \iff \mathbf{s}) \lor \neg \mathbf{p}$
Conclusión (C)**:** Si los derechos de autor no están protegidos, entonces la desinformación no se propaga rápidamente y la regulación legal es estricta.	$\neg \mathbf{s} \implies (\neg \mathbf{q} \land \mathbf{r})$

Fase 2: Estructura del argumento

Argumento estructurado (4 variables)

Notación lógica a descomponer:

$$T = [(p \land \neg q) \implies r] \land [(r \iff s) \lor \neg p] \implies [\neg s \implies (\neg q \land r)]$$

Traducción estructurada

$$\mathbf{T} = \underbrace{[(p \land \neg q) \implies r] \land [(r \iff s) \lor \neg p]}_{\text{PREMISA (Pr)}} \implies \underbrace{[\neg s \implies (\neg q \land r)]}_{\text{CONCLUSIÓN (C)}}$$

Significado: La verdad de todo el argumento (**T**) depende de la implicación entre la conjunción de las dos premisas y la conclusión.

Fase 3: Enunciado final

Enunciado final completo del argumento lógico traducido a lenguaje natural

"**Si es cierto que** (*si el contenido generado por IA es supervisado* **y** *la desinformación no se propaga* rápidamente, **entonces** *la regulación legal es estricta*) **y** (*la regulación legal es estricta* **si y solo si** *los derechos de autor están protegidos*, **o** *el contenido generado por IA no es supervisado*). **Por lo tanto**, *si los derechos de autor no están protegidos*, **entonces** *la desinformación no se propaga* rápidamente **y** *la regulación legal es estricta*."

Ejercicio 5: Representa cada una de las notaciones lógicas en forma de enunciados.
Notación lógica: $[\neg p \implies (q \wedge \neg r)] \vee [r \iff (\neg q \wedge s)] \implies [\neg s \implies (r \vee p)]$

Considere las siguientes proposiciones: variable p: Las sanciones económicas occidentales son efectivas, variable q: La ayuda militar a Ucrania es constante, variable r: Las negociaciones de paz se retoman, variable s: Rusia avanza significativamente en el territorio ucraniano.

Fase 1: Traducción y representación de la proposición

Argumento y notación lógica (4 variables)

Variables proposicionales:

- **p**: Las sanciones económicas occidentales son efectivas.
- **q**: La ayuda militar a Ucrania es constante.
- **r**: Las negociaciones de paz se retoman.
- **s**: Rusia avanza significativamente en el territorio ucraniano.

Notación lógica:

$$T = [\neg p \implies (q \wedge \neg r)] \vee [r \iff (\neg q \wedge s)] \implies [\neg s \implies (r \vee p)]$$

Estructura del argumento ver Tabla 1.85:

- **Premisa** (Pr): $[\neg p \implies (q \wedge \neg r)] \vee [r \iff (\neg q \wedge s)]$
- **Conclusión** (C): $\neg s \implies (r \vee p)$

Tabla 1.85: Traducción de premisas y conclusión del ejercicio 5

Parte del enunciado	Notación lógica
Premisa 1 (P_1)**:** Si las sanciones económicas no son efectivas, entonces la ayuda militar a Ucrania es constante y las negociaciones de paz no se retoman.	$\neg \mathbf{p} \implies (\mathbf{q} \wedge \neg \mathbf{r})$
Premisa 2 (P_2)**:** Las negociaciones de paz se retoman si y solo si la ayuda militar a Ucrania no es constante y Rusia avanza significativamente.	$\mathbf{r} \iff (\neg \mathbf{q} \wedge \mathbf{s})$
Conclusión (C)**:** Si Rusia no avanza significativamente en el territorio ucraniano, entonces las negociaciones de paz se retoman o las sanciones son efectivas.	$\neg \mathbf{s} \implies (\mathbf{r} \vee \mathbf{p})$

Fase 2: Estructura del argumento

Argumento estructurado (4 variables)

Traducción estructurada

$$\mathbf{T} = \underbrace{[\neg p \implies (q \wedge \neg r)] \vee [r \iff (\neg q \wedge s)]}_{\text{PREMISA (Pr)}} \implies \underbrace{[\neg s \implies (r \vee p)]}_{\text{CONCLUSIÓN (C)}}$$

Fase 3: Enunciado final

Enunciado final completo del argumento lógico traducido a lenguaje natural

"**Si es cierto que** (*si las sanciones económicas no son efectivas*, **entonces** *la ayuda militar a Ucrania es constante* **y** *las negociaciones de paz no se retoman*) **o** (*las negociaciones de paz se retoman* **si y solo si** *la ayuda militar a Ucrania no es constante* **y** *Rusia avanza significativamente en el territorio ucraniano*). **Por lo tanto**, *si Rusia no avanza* significativamente en el territorio ucraniano, **entonces** *las negociaciones de paz se retoman* **o** *las sanciones económicas occidentales son efectivas*."

Ejercicio 6: Representa cada una de las notaciones lógicas en forma de enunciados.
Notación lógica: $[(p \wedge \neg q) \implies r] \vee [\neg r \iff (s \vee q)] \implies [\neg s \implies (p \wedge r)]$

Considere las siguientes proposiciones: variable p: Ecuador gana el próximo partido de Eliminatorias, variable q: El equipo se clasifica directamente al Mundial, variable r: El Director Técnico mantiene su puesto, variable s: El goleador histórico anota en el partido.

Fase 1: Traducción y representación de la proposición

Argumento y notación lógica (4 variables)

Variables proposicionales:

- **p**: Ecuador gana el próximo partido de Eliminatorias.
- **q**: El equipo se clasifica directamente al Mundial.
- **r**: El Director Técnico mantiene su puesto.
- **s**: El goleador histórico anota en el partido.

Notación lógica:

$$T = [(p \wedge \neg q) \implies r] \vee [\neg r \iff (s \vee q)] \implies [\neg s \implies (p \wedge r)]$$

Estructura del argumento ver Tabla 1.86:

- **Premisa** (Pr): $[(p \wedge \neg q) \implies r] \vee [\neg r \iff (s \vee q)]$
- **Conclusión** (C): $\neg s \implies (p \wedge r)$

Tabla 1.86: Traducción de premisas y conclusión del ejercicio 6

Parte del enunciado	Notación lógica
Premisa 1 (P_1)**:** Si Ecuador gana el próximo partido de Eliminatorias y el equipo no se clasifica directamente al Mundial, entonces el Director Técnico mantiene su puesto.	$(\mathbf{p} \wedge \neg \mathbf{q}) \implies \mathbf{r}$
Premisa 2 (P_2)**:** El Director Técnico no mantiene su puesto si y solo si el goleador histórico anota en el partido o el equipo se clasifica directamente al Mundial.	$\neg \mathbf{r} \iff (\mathbf{s} \vee \mathbf{q})$
Conclusión (C)**:** Si el goleador histórico no anota en el partido, entonces Ecuador gana el próximo partido y el Director Técnico mantiene su puesto.	$\neg \mathbf{s} \implies (\mathbf{p} \wedge \mathbf{r})$

Fase 2: Estructura del argumento

Argumento estructurado (4 variables)

Traducción estructurada

$$\mathbf{T} = \underbrace{[(p \wedge \neg q) \implies r] \vee [\neg r \iff (s \vee q)]}_{\text{PREMISA (Pr)}} \implies \underbrace{[\neg s \implies (p \wedge r)]}_{\text{CONCLUSIÓN (C)}}$$

Fase 3: Enunciado final

Enunciado final completo del ergumento lógico traducido a lenguaje natural

"**Si es cierto que** (*si Ecuador gana el próximo partido de Eliminatorias* **y** *el equipo no se clasifica* directamente al Mundial, **entonces** *el Director Técnico mantiene su puesto*) **o** (*el Director Técnico no mantiene* su puesto **si y solo si** *el goleador histórico anota en el partido* **o** *el equipo se clasifica directamente al Mundial*). **Por lo tanto**, *si el goleador histórico no anota* en el partido, **entonces** *Ecuador gana el próximo partido de Eliminatorias* **y** *el Director Técnico mantiene su puesto*."

1.4.3. Sección 3 del laboratorio 2

En esta sección se va a determinar si las siguientes proposiciones son fórmulas contingentes, contradicciones o tautologías. Para obtener la respuesta debe generarlas con el uso de la calculadora lógica[3] que se encuentra en el siguiente enlace: https://sourceforge.net/projects/logiccalculator/?source=typ_redirect. Para usar la calculadora debe tener instalado Java[4] en su computadora. Sitio de descarga: https://java.com/es/download/. Y para ejecutar la calculadora, dar un clic con el botón derecho del ratón en el archivo *"LogicCalculator.jar"* y seleccionar la opción *"Oracle Java Runtime"* o *"Java™ Platform SE binary"* ver Fig. 1.31.

Logic Calculator

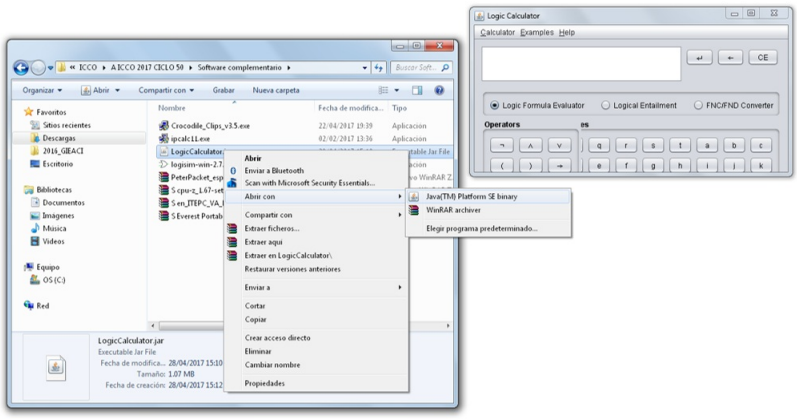


Figura 1.31: Imagen de la calculadora Logic Calculator.

Indicar por qué la proposición es una fórmula contingente, una contradicción o una tautología. Y una vez que se haya generado la tabla de verdad escoger una de las filas de la tabla de verdad y presentar un ejemplo de enunciado. Valor ingresado: $(p \wedge q) \implies (\neg p \vee \neg q)$, ver Fig. 1.32.

Logic Calculator

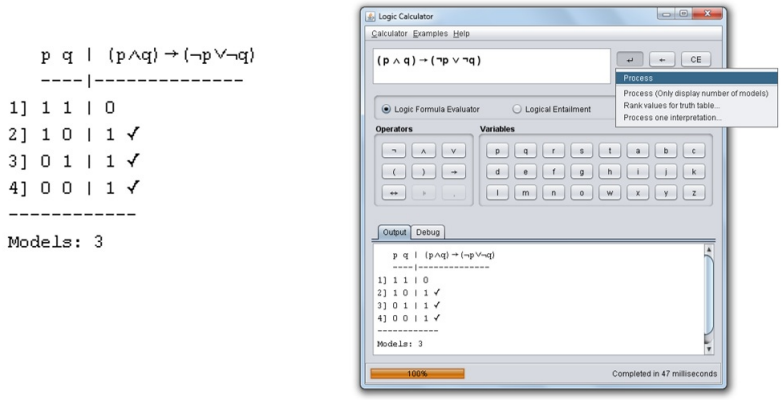


Figura 1.32: Imagen del ejercicio propuesto con la calculadora Logic Calculator.

[3] Descargar LogicCalculator
[4] Descargar Java

Ejercicio 1: $[\neg p \iff q] \land [\neg q \implies (r \lor q)] \land p \implies [\neg r \implies \neg p]$

Tabla de verdad del ejercicio 1

Notación lógica: $\mathbf{T} = \underbrace{[(\neg p \iff q) \land (\neg q \implies (r \lor q)) \land p]}_{\mathbf{Pr}} \implies \underbrace{[\neg r \implies \neg p]}_{\mathbf{C}}$

Tabla 1.87: Valoración de verdad del argumento, ejercicio 1

p	q	r	$\neg p$	$\neg q$	$\neg r$	$\neg p \iff q$	$r \lor q$	$\neg q \implies (r \lor q)$	Pr	C	T (Pr $\implies$ C)
V	V	V	F	F	F	F	V	V	F	V	V
V	V	F	F	F	V	F	V	V	F	F	V
V	F	V	F	V	F	V	V	V	V	V	V
V	F	F	F	V	V	V	F	F	F	F	V
F	V	V	V	F	F	V	V	V	V	V	V
F	V	F	V	F	V	V	V	V	V	V	V
F	F	V	V	V	F	F	V	V	F	V	V
F	F	F	V	V	V	F	F	F	F	V	V

Resultado: El argumento es una **Tautología**, ya que el valor de verdad final (**T**) es siempre Verdadero (V) en todas las filas.

Ejercicio 2: $[(q \lor \neg p) \implies r] \land [\neg r \implies \neg q] \implies [p \implies \neg q]$

Tabla de verdad del ejercicio 2

Notación lógica: $\mathbf{T} = \underbrace{[(q \lor \neg p) \implies r] \land [\neg r \implies \neg q]}_{\mathbf{Pr}} \implies \underbrace{[p \implies \neg q]}_{\mathbf{C}}$

Tabla 1.88: Valoración de verdad del argumento, ejercicio 2

p	q	r	$\neg p$	$\neg q$	$q \lor \neg p$	$(q \lor \neg p) \implies r$	$\neg r$	$\neg r \implies \neg q$	Pr	T (Pr $\implies$ C)
V	V	V	F	F	V	V	F	V	V	F
V	V	F	F	F	V	F	V	F	F	V
V	F	V	F	V	F	V	F	V	V	V
V	F	F	F	V	F	V	V	V	V	V
F	V	V	V	F	V	V	F	V	V	F
F	V	F	V	F	V	F	V	F	F	V
F	F	V	V	V	V	V	F	V	V	V
F	F	F	V	V	V	F	V	V	F	V

Resultado: El argumento es una **Contingencia**, ya que el valor de verdad final (**T**) contiene valores Verdaderos (V) y Falsos (F).

Ejercicio 3: $(p \implies q) \land (q \implies r) \land \neg(p \implies r)$

Tabla de verdad del ejercicio 3

Notación lógica: $\mathbf{T} = \underbrace{(p \implies q)}_{\mathbf{P_1}} \land \underbrace{(q \implies r)}_{\mathbf{P_2}} \land \underbrace{\neg(p \implies r)}_{\mathbf{P_3}}$

Tabla 1.89: Valoración de verdad de la fórmula, ejercicio 3

p	q	r	$p \implies q$	$q \implies r$	$p \implies r$	$\neg(p \implies r)$	$\mathbf{P_1} \land \mathbf{P_2}$	T
V	V	V	V	V	V	F	V	F
V	V	F	V	F	F	V	F	F
V	F	V	F	V	V	F	F	F
V	F	F	F	V	F	V	F	F
F	V	V	V	V	V	F	V	F
F	V	F	V	F	V	F	F	F
F	F	V	V	V	V	F	V	F
F	F	F	V	V	V	F	V	F

Resultado: El argumento es una **Contradicción**.

Ejercicio 4: $[(p \wedge \neg q) \implies (r \vee s)] \wedge [\neg r \iff (q \wedge \neg s)]$

Tabla de verdad del ejercicio 4

Notación lógica: $\mathbf{T} = \underbrace{[(p \wedge \neg q) \implies (r \vee s)]}_{\mathbf{P_1}} \wedge \underbrace{[\neg r \iff (q \wedge \neg s)]}_{\mathbf{P_2}}$

Tabla 1.90: Valoración de verdad de la fórmula, ejercicio 4

p	q	r	s	$\neg q$	$\neg r$	$\neg s$	$p \wedge \neg q$	$r \vee s$	P_1	$q \wedge \neg s$	P_2	T
V	V	V	V	F	F	F	F	V	V	F	V	V
V	V	V	F	F	F	V	F	V	V	V	F	F
V	V	F	V	F	V	F	F	V	V	F	F	F
V	V	F	F	F	V	V	F	F	V	V	V	V
V	F	V	V	V	F	F	V	V	V	F	V	V
V	F	V	F	V	F	V	V	V	V	F	V	V
V	F	F	V	V	V	F	V	V	V	F	F	F
V	F	F	F	V	V	V	V	F	F	F	F	F
F	V	V	V	F	F	F	F	V	V	F	V	V
F	V	V	F	F	F	V	F	V	V	V	F	F
F	V	F	V	F	V	F	F	V	V	F	F	F
F	V	F	F	F	V	V	F	F	V	V	V	V
F	F	V	V	V	F	F	F	V	V	F	V	V
F	F	V	F	V	F	V	F	V	V	F	V	V
F	F	F	V	V	V	F	F	V	V	F	F	F
F	F	F	F	V	V	V	F	F	V	F	F	F

Resultado: El argumento es una **Contingencia**, ya que el valor de verdad final (**T**) contiene valores Verdaderos (V) y Falsos (F).

Ejercicio 5: $[(p \implies \neg q) \wedge (\neg r \vee q)] \implies \neg(p \wedge r)$

Tabla de verdad del ejercicio 5

Notación lógica: $\mathbf{T} = \underbrace{[(p \implies \neg q) \wedge (\neg r \vee q)]}_{\mathbf{Pr}} \implies \underbrace{\neg(p \wedge r)}_{\mathbf{C}}$

Tabla 1.91: Valoración de verdad de la fórmula, ejercicio 5

p	q	r	$\neg q$	$\neg r$	$p \implies \neg q$	$\neg r \vee q$	Pr	$p \wedge r$	C	T
V	V	V	F	F	F	V	F	V	F	V
V	V	F	F	V	F	V	F	F	V	V
V	F	V	V	F	V	F	F	V	F	V
V	F	F	V	V	V	V	V	F	V	V
F	V	V	F	F	V	V	V	F	V	V
F	V	F	F	V	V	V	V	F	V	V
F	F	V	V	F	V	F	F	F	V	V
F	F	F	V	V	V	V	V	F	V	V

Resultado: El argumento es una **Tautología**, ya que el valor de verdad final (**T**) es siempre Verdadero (V) en todas las filas.

Ejercicio 6: $[\neg(p \vee q) \iff (r \wedge s)] \wedge [(t \implies u) \wedge \neg t] \implies [(\neg r \wedge p) \vee \neg u]$

Tabla de verdad del ejercicio 6 (64 filas)

Notación lógica: $\mathbf{T} = \underbrace{[\neg(p \vee q) \iff (r \wedge s)]}_{\mathbf{P_1}} \wedge \underbrace{[(t \implies u) \wedge \neg t]}_{\mathbf{P_2}} \implies \underbrace{[(\neg r \wedge p) \vee \neg u]}_{\mathbf{C}}$

Tabla 1.92: Valoración de verdad de la fórmula, ejercicio 6 (con 6 variables)

p	q	r	s	t	u	$\neg(p \vee q)$	$r \wedge s$	$\mathbf{P_1}$	$t \implies u$	$\neg t$	$\mathbf{P_2}$	$\neg r$	$\neg u$	**C**	**T**
V	V	V	V	V	V	F	V	F	V	F	F	F	F	F	V
V	V	V	V	V	F	F	V	F	F	F	F	F	V	V	V
V	V	V	V	F	V	F	V	F	V	V	V	F	F	F	V
V	V	V	V	F	F	F	V	F	V	V	V	F	V	V	V
V	V	V	F	V	V	F	F	V	V	F	F	F	F	F	V
V	V	V	F	V	F	F	F	V	F	F	F	F	V	V	V
V	V	V	F	F	V	F	F	V	V	V	V	F	F	F	F
V	V	V	F	F	F	F	F	V	V	V	V	F	V	V	V
V	V	F	V	V	V	F	F	V	V	F	F	V	F	V	V
V	V	F	V	V	F	F	F	V	F	F	F	V	V	V	V
V	V	F	V	F	V	F	F	V	V	V	V	V	F	V	V
V	V	F	V	F	F	F	F	V	V	V	V	V	V	V	V
V	V	F	F	V	V	F	F	V	V	F	F	V	F	V	V
V	V	F	F	V	F	F	F	V	F	F	F	V	V	V	V
V	V	F	F	F	V	F	F	V	V	V	V	V	F	V	V
V	V	F	F	F	F	F	F	V	V	V	V	V	V	V	V
V	F	V	V	V	V	F	V	F	V	F	F	F	F	F	V
V	F	V	V	V	F	F	V	F	F	F	F	F	V	V	V
V	F	V	V	F	V	F	V	F	V	V	V	F	F	F	V
V	F	V	V	F	F	F	V	F	V	V	V	F	V	V	V
V	F	V	F	V	V	F	F	V	V	F	F	F	F	F	V
V	F	V	F	V	F	F	F	V	F	F	F	F	V	V	V
V	F	V	F	F	V	F	F	V	V	V	V	F	F	F	F
V	F	V	F	F	F	F	F	V	V	V	V	F	V	V	V
V	F	F	V	V	V	F	F	V	V	F	F	V	F	V	V
V	F	F	V	V	F	F	F	V	F	F	F	V	V	V	V
V	F	F	V	F	V	F	F	V	V	V	V	V	F	V	V
V	F	F	V	F	F	F	F	V	V	V	V	V	V	V	V
V	F	F	F	V	V	F	F	V	V	F	F	V	F	V	V
V	F	F	F	V	F	F	F	V	F	F	F	V	V	V	V
V	F	F	F	F	V	F	F	V	V	V	V	V	F	V	V
V	F	F	F	F	F	F	F	V	V	V	V	V	V	V	V
F	V	V	V	V	V	F	V	F	V	F	F	F	F	F	V
F	V	V	V	V	F	F	V	F	F	F	F	F	V	V	V
F	V	V	V	F	V	F	V	F	V	V	V	F	F	F	V
F	V	V	V	F	F	F	V	F	V	V	V	F	V	V	V
F	V	V	F	V	V	F	F	V	V	F	F	F	F	F	V
F	V	V	F	V	F	F	F	V	F	F	F	F	V	V	V
F	V	V	F	F	V	F	F	V	V	V	V	F	F	F	F
F	V	V	F	F	F	F	F	V	V	V	V	F	V	V	V
F	V	F	V	V	V	F	F	V	V	F	F	V	F	V	V
F	V	F	V	V	F	F	F	V	F	F	F	V	V	V	V
F	V	F	V	F	V	F	F	V	V	V	V	V	F	V	V
F	V	F	V	F	F	F	F	V	V	V	V	V	V	V	V
F	V	F	F	V	V	F	F	V	V	F	F	V	F	V	V
F	V	F	F	V	F	F	F	V	F	F	F	V	V	V	V
F	V	F	F	F	V	F	F	V	V	V	V	V	F	V	V
F	V	F	F	F	F	F	F	V	V	V	V	V	V	V	V
F	F	V	V	V	V	V	V	V	V	F	F	F	F	F	V
F	F	V	V	V	F	V	V	V	F	F	F	F	V	V	V
F	F	V	V	F	V	V	V	V	V	V	V	F	F	F	F
F	F	V	V	F	F	V	V	V	V	V	V	F	V	V	V
F	F	V	F	V	V	V	F	F	V	F	F	F	F	F	V
F	F	V	F	V	F	V	F	F	F	F	F	F	V	V	V
F	F	V	F	F	V	V	F	F	V	V	V	F	F	F	V
F	F	V	F	F	F	V	F	F	V	V	V	F	V	V	V
F	F	F	V	V	V	V	F	F	V	F	F	V	F	F	V
F	F	F	V	V	F	V	F	F	F	F	F	V	V	V	V
F	F	F	V	F	V	V	F	F	V	V	V	V	F	F	V
F	F	F	V	F	F	V	F	F	V	V	V	V	V	V	V
F	F	F	F	V	V	V	F	F	V	F	F	V	F	F	V
F	F	F	F	V	F	V	F	F	F	F	F	V	V	V	V
F	F	F	F	F	V	V	F	F	V	V	V	V	F	F	V
F	F	F	F	F	F	V	F	F	V	V	V	V	V	V	V

Resultado (Análisis): Este argumento es una **Contingencia**, ya que el valor de verdad final (**T**) es Falso en 7 de las 64 combinaciones (las filas marcadas con **FALSO** en la columna **T** interna).

Ejercicio 7: $[p \implies q] \iff [\neg q \implies \neg p]$

Tabla de verdad del ejercicio 7

Notación lógica: $\mathbf{T} = \underbrace{[p \implies q]}_{\mathbf{P}} \iff \underbrace{[\neg q \implies \neg p]}_{\mathbf{C}}$

Tabla 1.93: Valoración de verdad de la fórmula, ejercicio 7

p	q	$\neg p$	$\neg q$	$p \implies q$	$\neg q \implies \neg p$	T
V	V	F	F	V	V	V
V	F	F	V	F	F	V
F	V	V	F	V	V	V
F	F	V	V	V	V	V

Resultado: El argumento es una **Tautología**.

Ejercicio 8: $[p \implies (q \wedge r)] \wedge [\neg p \implies \neg q] \implies [\neg r \implies p]$

Tabla de verdad del ejercicio 8

Notación lógica: $\mathbf{T} = \underbrace{[p \implies (q \wedge r)] \wedge [\neg p \implies \neg q]}_{\mathbf{Pr}} \implies \underbrace{[\neg r \implies p]}_{\mathbf{C}}$

Tabla 1.94: Valoración de verdad de la fórmula, ejercicio 8

p	q	r	$\neg p$	$\neg q$	$\neg r$	$q \wedge r$	$p \implies (q \wedge r)$	$\neg p \implies \neg q$	Pr	C	T
V	V	V	F	F	F	V	V	V	V	V	V
V	V	F	F	F	V	F	F	V	F	V	V
V	F	V	F	V	F	F	F	V	F	V	V
V	F	F	F	V	V	F	F	V	F	V	V
F	V	V	V	F	F	V	V	F	F	V	V
F	V	F	V	F	V	F	V	F	F	F	V
F	F	V	V	V	F	F	V	V	V	V	V
F	F	F	V	V	V	F	V	V	V	F	F

Resultado: El argumento es una **Contingencia**.

Ejercicio 9: $[(p \vee q) \implies (r \wedge \neg s)] \wedge [\neg r \implies p] \implies [s \implies \neg q]$

Tabla de verdad del ejercicio 9

Notación lógica: $\mathbf{T} = \underbrace{[(p \vee q) \implies (r \wedge \neg s)] \wedge [\neg r \implies p]}_{\mathbf{Pr}} \implies \underbrace{[s \implies \neg q]}_{\mathbf{C}}$

Tabla 1.95: Valoración de verdad de la fórmula, ejercicio 9

p	q	r	s	$\neg q$	$\neg r$	$\neg s$	$p \vee q$	$r \wedge \neg s$	$\mathbf{P_1}$	$\mathbf{P_2}$	Pr	C	T
V	V	V	V	F	F	F	V	F	F	V	F	F	V
V	V	V	F	F	F	V	V	V	V	V	V	V	V
V	V	F	V	F	V	F	V	F	F	V	F	F	V
V	V	F	F	F	V	V	V	F	F	V	F	V	V
V	F	V	V	V	F	F	V	F	F	V	F	V	V
V	F	V	F	V	F	V	V	V	V	V	V	V	V
V	F	F	V	V	V	F	V	F	F	V	F	V	V
V	F	F	F	V	V	V	V	F	F	V	F	V	V
F	V	V	V	F	F	F	V	F	F	V	F	F	V
F	V	V	F	F	F	V	V	V	V	V	V	V	V
F	V	F	V	F	V	F	V	F	F	F	F	F	V
F	V	F	F	F	V	V	V	F	F	F	F	V	V
F	F	V	V	V	F	F	F	F	V	V	V	V	V
F	F	V	F	V	F	V	F	V	V	V	V	V	V
F	F	F	V	V	V	F	F	F	V	F	F	V	V
F	F	F	F	V	V	V	F	F	V	F	F	V	V

Resultado: El argumento es una **Tautología**.

Ejercicio 10: $[(p \land \neg q) \lor (r \implies \neg s)] \land [\neg p \lor s] \implies [q \iff r]$

Tabla de verdad del ejercicio 10

Notación lógica: $\mathbf{T} = \underbrace{[(p \land \neg q) \lor (r \implies \neg s)] \land [\neg p \lor s]}_{\mathbf{Pr}} \implies \underbrace{[q \iff r]}_{\mathbf{C}}$

Tabla 1.96: Valoración de verdad de la fórmula, ejercicio 10

p	q	r	s	$\neg p$	$\neg q$	$\neg s$	$p \land \neg q$	$r \implies \neg s$	$\mathbf{P_1}$	$\mathbf{P_2}$	Pr	C	T
V	V	V	V	F	F	F	F	F	F	V	F	V	V
V	V	V	F	F	F	V	F	V	V	V	V	V	V
V	V	F	V	F	F	F	F	V	V	V	V	F	F
V	V	F	F	F	F	V	F	V	V	V	V	F	F
V	F	V	V	F	V	F	V	F	V	V	V	F	F
V	F	V	F	F	V	V	V	V	V	V	V	F	F
V	F	F	V	F	V	F	V	V	V	V	V	V	V
V	F	F	F	F	V	V	V	V	V	V	V	V	V
F	V	V	V	V	F	F	F	F	F	V	F	V	V
F	V	V	F	V	F	V	F	V	V	V	V	V	V
F	V	F	V	V	F	F	F	V	V	V	V	F	F
F	V	F	F	V	F	V	F	V	V	V	V	F	F
F	F	V	V	V	V	F	F	F	F	V	F	F	V
F	F	V	F	V	V	V	F	V	V	V	V	F	F
F	F	F	V	V	V	F	F	V	V	V	V	V	V
F	F	F	F	V	V	V	F	V	V	V	V	V	V

Resultado: El argumento es una **Contingencia**.

Ejercicio 11: $[(p \land q) \implies r] \land [(\neg r \lor p) \implies q] \implies r$

Tabla de verdad del ejercicio 11

Notación lógica: $\mathbf{T} = \underbrace{[(p \land q) \implies r] \land [(\neg r \lor p) \implies q]}_{\mathbf{Pr}} \implies \underbrace{r}_{\mathbf{C}}$

Tabla 1.97: Valoración de verdad de la fórmula, ejercicio 11

p	q	r	$\neg r$	$p \land q$	$\mathbf{P_1}$	$\neg r \lor p$	$\mathbf{P_2}$	Pr	C	T
V	V	V	F	V	V	V	V	V	V	V
V	V	F	V	V	F	V	V	F	F	V
V	F	V	F	F	V	V	F	F	V	V
V	F	F	V	F	V	V	F	F	F	V
F	V	V	F	F	V	F	V	V	V	V
F	V	F	V	F	V	V	V	V	F	F
F	F	V	F	F	V	F	V	V	V	V
F	F	F	V	F	V	V	F	F	F	V

Resultado: El argumento es una **Contingencia**.

Ejercicio 12: $[(p \implies q) \land \neg p] \implies \neg q$

Tabla de verdad del ejercicio 12

Notación lógica: $\mathbf{T} = \underbrace{[(p \implies q) \land \neg p]}_{\mathbf{Pr}} \implies \underbrace{\neg q}_{\mathbf{C}}$

Tabla 1.98: Valoración de verdad de la fórmula, ejercicio 12

p	q	$\neg p$	$\neg q$	$p \implies q$	Pr	T
V	V	F	F	V	F	V
V	F	F	V	F	F	V
F	V	V	F	V	V	F
F	F	V	V	V	V	V

Resultado: El argumento es una **Contingencia**.

1.4.4. Sección 4 del laboratorio 2.

Análisis proposicional, ejemplo 1. La proposición a evaluar es: $T = (p \wedge q) \implies (\neg p \vee \neg q)$

1. **Traducción lógica:** La proposición T es lógicamente equivalente a la ley de De Morgan: $(p \wedge q) \implies \neg(p \wedge q)$.
2. **Análisis de resultados:** La proposición es falsa solo cuando su antecedente $(p \wedge q)$ es Verdadero (V) y su consecuente $(\neg p \vee \neg q)$ es Falso (F). Esto ocurre exclusivamente en la primera fila.

p	q	$\neg p$	$\neg q$	$p \wedge q$	$\neg p \vee \neg q$	T
V	V	F	F	V	F	**F**
V	F	F	V	F	V	V
F	V	V	F	F	V	V
F	F	V	V	F	V	V

Resultado: La proposición es una **Contingencia** porque los valores de verdad se combinan entre Verdadero (V) y Falso (F). **Ejemplo de enunciado:**

Línea	Premisas y Conclusión	Valor
1	p: Tengo novia	1
2	q: Me quiero casar	1
Antecedente	$(p \wedge q)$: (Tengo novia y Me quiero casar)	$1 \wedge 1 = 1$
Consecuente	$(\neg p \vee \neg q)$: (No tengo novia o No Me quiero casar)	$0 \vee 0 = 0$
3	$(p \wedge q) \implies (\neg p \vee \neg q)$: (Tengo novia y Me quiero casar) $\implies$ (No tengo novia o No Me quiero casar)	$1 \implies 0 = \mathbf{0}$

Conclusión del enunciado: La proposición completa es **Falsa** (0), lo cual tiene sentido, pues no es lógico que "Tengo novia y me quiero casar" implique que "No tengo novia o no me quiero casar".

Derivación de conclusiones: El *Modus Ponendus Ponens (MPP)* tiene la forma: $(A \implies B) \wedge A \implies B$.

- Argumento 1:
  a. $p \vee q \implies r$
  b. $p \vee q$
  c. r (Por MPP en a y b)
- Argumento 2:
  a. $\neg p \implies \neg r$
  b. $\neg p$
  c. $\neg r$ (Por MPP en a y b)
- Argumento 3:
  a. $\neg r$
  b. $\neg r \implies q \wedge p$
  c. $q \wedge p$ (Por MPP en b y a)

Análisis proposicional, ejemplo 2: La proposición a evaluar es un argumento sobre los requisitos para adoptar la IAGen de forma efectiva en un centro educativo: $T = [(\neg p \vee q) \wedge r] \implies (p \implies r)$. Donde: **p**: Los profesores reciben capacitación en IAGen. **q**: La IAGen se integra al currículo de manera efectiva. **r**: La IAGen mejora el rendimiento estudiantil.

1. **Traducción lógica:** La proposición T evalúa si la existencia de *capacitación O implementación* efectiva, junto con la *mejora de rendimiento*, necesariamente implica que *si hay capacitación, hay mejora de rendimiento*.

2. **Análisis de resultados:** La proposición es falsa cuando su antecedente $[(\neg p \vee q) \wedge r]$ es (V) y su consecuente $(p \implies r)$ es (F). En este caso, no ocurre ninguna vez.

p	q	r	$\neg p$	$\neg$p $\vee$ q	Pr	$p \implies r$	T
V	V	V	F	V	V	V	V
V	V	F	F	V	F	F	V
V	F	V	F	F	F	V	V
V	F	F	F	F	F	F	V
F	V	V	V	V	V	V	V
F	V	F	V	V	F	V	V
F	F	V	V	V	V	V	V
F	F	F	V	V	F	V	V

Resultado: La proposición es una **Tautología** porque todos los valores de verdad finales (**T**) son Verdaderos (V). **Ejemplo de enunciado:** (Usando la fila V F V, donde T=V)

Línea	Premisas y Conclusión	Valor
1	p: Los profesores reciben **capacitación**	1
2	q: IAGen se integra al **currículo** efectivamente	0
3	r: IAGen **mejora** el rendimiento	1
Antecedente	$[(\neg p \vee q) \wedge r]$: (No hay capacitación O no hay currículo) $\wedge$ hay mejora	$0 \wedge 1 = \mathbf{0}$
Consecuente	$(p \implies r)$: Si hay capacitación, entonces hay mejora	$1 \implies 1 = \mathbf{1}$
3	Antecedente $\implies$ Consecuente	$0 \implies 1 = \mathbf{1}$

Conclusión del enunciado: La proposición completa es **Verdadera** (1), demostrando la validez del argumento incluso bajo estas condiciones.

Derivación de conclusiones: El *Modus Ponendus Ponens (MPP)* tiene la forma: $(A \implies B) \wedge A \implies B$.

- Argumento 1:
  a. $(p \wedge q) \implies \neg r$
  b. $p \wedge q$
  c. $\neg r$ (Por MPP en a y b)

- Argumento 2:
  a. $\neg r \implies \neg p$
  b. $\neg r$
  c. $\neg p$ (Por MPP en a y b)

- Argumento 3:
  a. $\neg p \vee q$
  b. $(\neg p \vee q) \implies r$
  c. r (Por MPP en b y a)

Análisis proposicional, ejemplo 3. La proposición a evaluar es: $T = [(p \implies q) \land \neg q] \implies \neg p$

1. **Traducción lógica:** La proposición T evalúa si Si A implica B, Y B es Falso, necesariamente implica que A también es Falso. (Modus Tollens).

2. **Análisis de resultados:** La proposición es falsa solo cuando su antecedente $[(p \implies q) \land \neg q]$ es Verdadero (V) y su consecuente $\neg p$ es Falso (F). Esto no ocurre en ninguna fila.

p	q	$\neg p$	$\neg q$	$p \implies q$	Pr	T
V	V	F	F	V	F	V
V	F	F	V	F	F	V
F	V	V	F	V	F	V
F	F	V	V	V	V	V

Resultado: La proposición es una **Tautología** porque todos los valores de verdad finales (**T**) son Verdaderos (V). **Ejemplo de enunciado.** Donde: **p**: Aumenta la temperatura media global. **q**: Los polos se deshielan.

Línea	Premisas y Conclusión	Valor
1	$\neg q$: Los polos **no se deshielan**	1
2	$p \implies q$: Si aumenta la temperatura, entonces los polos se deshielan	1
Antecedente	$(p \implies q) \land \neg q$: Premisa 2 y Premisa 1	$1 \land 1 = \mathbf{1}$
Consecuente	$\neg p$: La temperatura media global no aumenta	**1**
3	$[(p \implies q) \land \neg q] \implies \neg p$: Antecedente $\implies$ Consecuente	$1 \implies 1 = \mathbf{1}$

Conclusión del enunciado: La proposición completa es **Verdadera** (1), confirmando que el Modus Tollens es válido.

Derivación de conclusiones: El *Modus Ponendus Ponens (MPP)* tiene la forma: $(A \implies B) \land A \implies B$.

- Argumento 1:
  a. $p \lor q \implies \neg r$
  b. $p \lor q$
  c. $\neg r$ (Por MPP en a y b)

- Argumento 2:
  a. $\neg r \implies p$
  b. $\neg r$
  c. p (Por MPP en a y b)

- Argumento 3:
  a. q
  b. $q \implies p \land \neg r$
  c. $p \land \neg r$ (Por MPP en b y a)

Análisis proposicional, ejemplo 4. La proposición a evaluar es: $T = [(p \wedge \neg q) \vee (\neg r \implies q)] \implies (r \wedge p)$

1. **Traducción lógica:** La proposición T evalúa si *El chatbot responde rápido Y mal, O si no hay satisfacción implica respuesta correcta,* necesariamente implica que *El cliente queda satisfecho Y la respuesta fue rápida.*

2. **Análisis de resultados:** La proposición es falsa cuando el antecedente $[(p \wedge \neg q) \vee (\neg r \implies q)]$ es Verdadero (V) y el consecuente $(r \wedge p)$ es Falso (F). Esto ocurre en varias filas, confirmando que es una Contingencia.

p	q	r	$\neg q$	$\neg r$	$p \wedge \neg q$	$\neg r \implies q$	Pr	$r \wedge p$	T
V	V	V	F	F	F	V	V	V	V
V	V	F	F	V	F	V	V	F	V
V	F	V	V	F	V	V	V	V	V
V	F	F	V	V	V	F	V	F	**F**
F	V	V	F	F	F	V	V	F	**F**
F	V	F	F	V	F	V	V	F	**F**
F	F	V	V	F	F	V	V	F	**F**
F	F	F	V	V	F	F	F	F	V

Resultado: La proposición es una **Contingencia**. **Ejemplo de enunciado:** Usando la fila **V F F** (Rápido=V, Correcto=F, Satisfecho=F).

Línea	Premisas y Conclusión	Valor
1	p: El chatbot proporciona **respuesta rápida**	1
2	q: La respuesta es **correcta**	0
3	r: El cliente queda **satisfecho**	0
Antecedente	$[(p \wedge \neg q) \vee (\neg r \implies q)]$	$(1 \wedge 1) \vee (1 \implies 0) = 1 \vee 0 = \mathbf{1}$
Consecuente	$(r \wedge p)$: El cliente queda satisfecho Y la respuesta fue rápida	$0 \wedge 1 = \mathbf{0}$
3	Antecedente $\implies$ Consecuente	$1 \implies 0 = \mathbf{0}$

Conclusión del enunciado: La proposición completa es **Falsa** (0), lo cual tiene sentido ya que es posible tener un servicio rápido pero incorrecto (Antecedente V), sin que ello implique que el cliente esté satisfecho (Consecuente F).

Derivación de conclusiones: El *Modus Ponendus Ponens (MPP)* tiene la forma: $(A \implies B) \wedge A \implies B$.

- Argumento 1:
  a. $(p \wedge \neg q) \implies \neg r$
  b. $p \wedge \neg q$
  c. $\neg r$ (Por MPP en a y b)

- Argumento 2:
  a. $\neg r \implies q$
  b. $\neg r$
  c. q (Por MPP en a y b)

- Argumento 3:
  a. $\neg q \vee \neg p$
  b. $\neg q \vee \neg p \implies r$
  c. r (Por MPP en b y a)

1.4.5. Sección 5 del laboratorio 2.

Análisis de inferencia ejercicio 1, Modus Tollendo Tollens (MTT): Si la luz fuera simplemente una onda de movimiento continuo, entonces la luz intensa pudiera siempre causar electrones que escapen con mayor energía que aquella que causaría la luz tenue. La luz intensa no siempre causa electrones que escapen con mayor energía que la que causa la luz tenue.

1. **Variables proposicionales:**

   P: La luz es simplemente una onda de movimiento continuo.
   Q: La luz intensa puede siempre causar electrones que escapen con mayor energía que aquella que causaría la luz tenue.

2. **Regla de inferencia:**

Forma	Nombre
$(P \implies Q) \land \neg Q$	Modus Tollendo Tollens (MTT)
$\therefore \neg P$	

3. **Simbolización y derivación:**

   a. $\mathbf{P1}: P \implies Q$ (Si la luz fuera una onda, causaría Q)
   b. $\mathbf{P2}: \neg Q$ (La luz intensa **no** causa Q)
   c. $\mathbf{C}: \neg P$ (**Por MTT** en a y b)

Conclusión ($\neg P$)**:** La luz **no** es simplemente una onda de movimiento continuo.

Análisis de inferencia ejercicio 2, Modus Tollendo Tollens (MTT): Si el ángulo de un triángulo es mayor que 90 grados, entonces la suma de los otros dos ángulos es menor que 90 grados. La suma de los otros dos ángulos no es menor que 90 grados.

1. **Variables proposicionales:**

   P: El ángulo de un triángulo es mayor que 90 grados.
   Q: La suma de los otros dos ángulos es menor que 90 grados.

2. **Regla de inferencia:**

Forma	Nombre
$(P \implies Q) \land \neg Q$	Modus Tollendo Tollens (MTT)
$\therefore \neg P$	

3. **Simbolización y derivación:**

   a. $\mathbf{P1}: P \implies Q$ (Si un ángulo es $> 90°$, la suma es $< 90°$)
   b. $\mathbf{P2}: \neg Q$ (La suma de los otros dos ángulos **no** es menor que $90°$)
   c. $\mathbf{C}: \neg P$ (**Por MTT** en a y b)

Conclusión ($\neg P$)**:** El ángulo de un triángulo **no** es mayor que 90 grados.

Análisis de inferencia ejercicio 3, Modus Tollendo Tollens (MTT): Si llovió la noche anterior, entonces las carreteras fueron lavadas. Las carreteras no han sido lavadas.

1. **Variables proposicionales:**

   P: Llovió la noche anterior.
   Q: Las carreteras fueron lavadas.

2. **Regla de inferencia:**

Forma	Nombre
$(P \implies Q) \land \neg Q$	Modus Tollendo Tollens (MTT)
$\therefore \neg P$	

3. **Simbolización y derivación:**

   a. $\mathbf{P1}$: $P \implies Q$ (Si llovió, entonces las carreteras fueron lavadas)
   b. $\mathbf{P2}$: $\neg Q$ (Las carreteras **no** han sido lavadas)
   c. $\mathbf{C}$: $\neg P$ (**Por MTT** en a y b)

Conclusión ($\neg P$): No llovió la noche anterior.

Análisis de inferencia ejercicio 4: Modus Tollendo Tollens (MTT): Si la sonda espacial aterrizó con éxito, entonces el centro de control recibió una señal de telemetría. El centro de control no ha recibido una señal de telemetría.

1. **Variables proposicionales:**

   P: La sonda espacial aterrizó con éxito.
   Q: El centro de control recibió una señal de telemetría.

2. **Regla de inferencia:**

Forma	Nombre
$(P \implies Q) \land \neg Q$	Modus Tollendo Tollens (MTT)
$\therefore \neg P$	

3. **Simbolización y derivación:**

   a. $\mathbf{P1}$: $P \implies Q$ (Si la sonda aterrizó, el control recibe telemetría)
   b. $\mathbf{P2}$: $\neg Q$ (El centro de control **no** ha recibido una señal de telemetría)
   c. $\mathbf{C}$: $\neg P$ (**Por MTT** en a y b)

Conclusión ($\neg P$): La sonda espacial **no** aterrizó con éxito.

Análisis de inferencia ejercicio 5, Modus Tollendo Tollens (MTT): Si una noticia es verificada por tres fuentes independientes, entonces es legítima. La noticia no es legítima.

1. **Variables proposicionales:**
   - **P:** Una noticia es verificada por tres fuentes independientes.
   - **Q:** La noticia es legítima (no es una fake news).
2. **Regla de inferencia:**

Forma	Nombre
$(P \implies Q) \land \neg Q$	Modus Tollendo Tollens (MTT)
$\therefore \neg P$	

3. **Simbolización y derivación:**
   a. $\mathbf{P1}: P \implies Q$ (Si es verificada, es legítima)
   b. $\mathbf{P2}: \neg Q$ (La noticia **no** es legítima)
   c. $\mathbf{C}: \neg P$ (**Por MTT** en a y b)

Conclusión ($\neg P$): La noticia **no** fue verificada por tres fuentes independientes.

Análisis de inferencia ejercicio 6, Modus Tollendo Tollens (MTT): Si el cometa 3I/ATLAS (detectado por la NASA) alcanza su perihelio, entonces su brillo aparente superará la magnitud +5. Su brillo aparente no superó la magnitud +5.

1. **Variables proposicionales:**
   - **P:** El cometa 3I/ATLAS (detectado por la NASA) alcanza su perihelio.
   - **Q:** Su brillo aparente superará la magnitud $+5$.
2. **Regla de inferencia:**

Forma	Nombre
$(P \implies Q) \land \neg Q$	Modus Tollendo Tollens (MTT)
$\therefore \neg P$	

3. **Simbolización y derivación:**
   a. $\mathbf{P1}: P \implies Q$ (Si alcanza el perihelio, el brillo es $> +5$)
   b. $\mathbf{P2}: \neg Q$ (Su brillo aparente **no** superó la magnitud $+5$)
   c. $\mathbf{C}: \neg P$ (**Por MTT** en a y b)

Conclusión ($\neg P$): El cometa 3I/ATLAS (detectado por la NASA) **no** alcanzó su perihelio.

Análisis de inferencia ejercicio 7, Modus Tollendo Tollens (MTT): Si la velocidad de conexión a internet es de al menos 25 Mbps, entonces Netflix transmitirá en calidad Ultra HD (4K). Netflix no está transmitiendo en calidad Ultra HD (4K).

1. **Variables proposicionales:**
   - **P:** La velocidad de conexión a internet es de al menos 25 Mbps.
   - **Q:** Netflix transmitirá en calidad Ultra HD (4K).
2. **Regla de inferencia:**

Forma	Nombre
$(P \implies Q) \land \neg Q$	Modus Tollendo Tollens (MTT)
$\therefore \neg P$	

3. **Simbolización y derivación:**
   a. $\mathbf{P1}: P \implies Q$ (Si es 25 Mbps o más, transmite en 4K)
   b. $\mathbf{P2}: \neg Q$ (Netflix **no** está transmitiendo en Ultra HD (4K))
   c. $\mathbf{C}: \neg P$ (**Por MTT** en a y b)

Conclusión ($\neg P$): La velocidad de conexión a internet **no** es de al menos 25 Mbps.

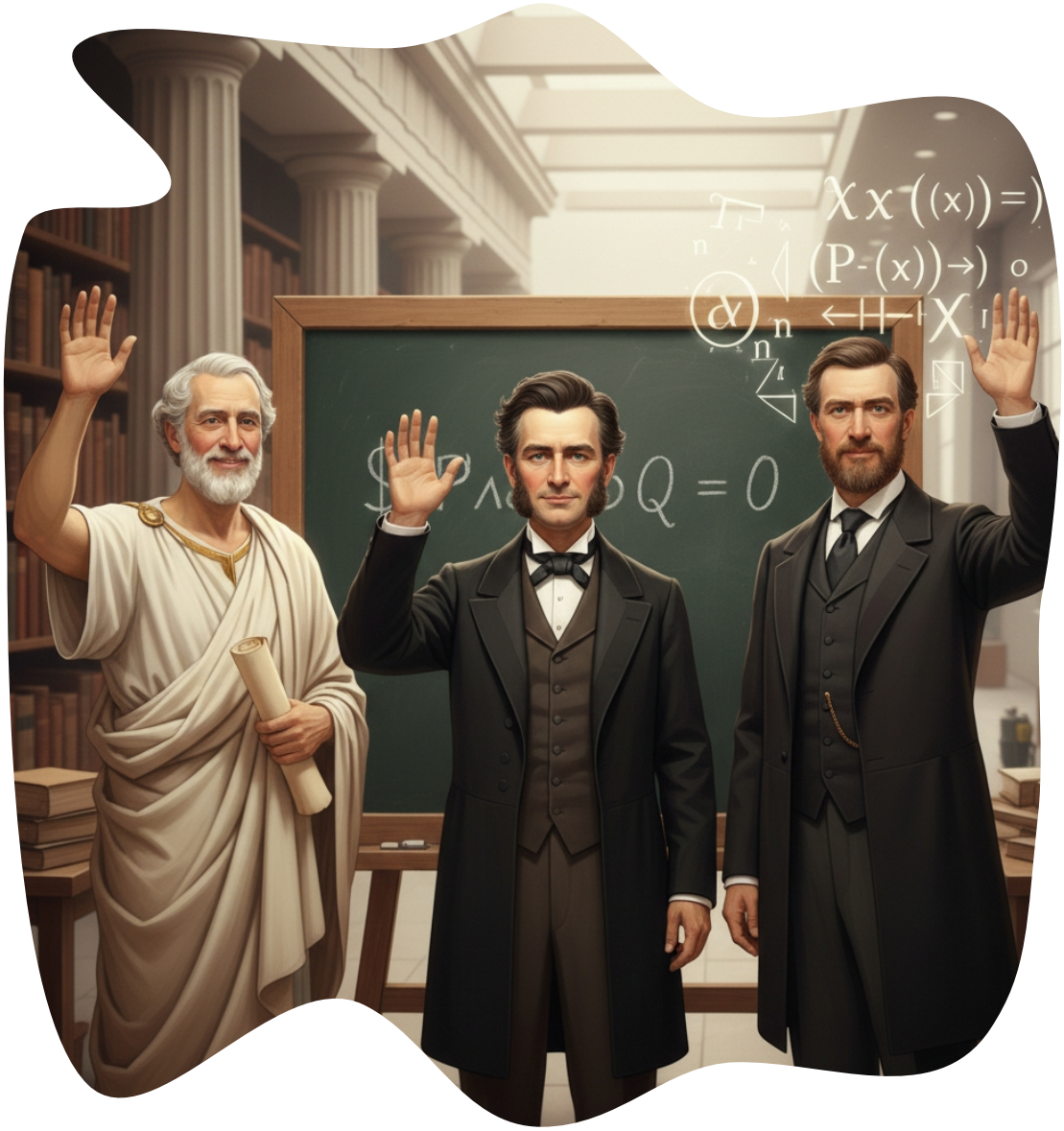
Análisis de inferencia ejercicio 8, Modus Tollendo Tollens (MTT): Si Thor, Hulk, Iron Man y Capitán América forman equipo, entonces el mundo está a salvo. El mundo no está a salvo.

1. **Variables proposicionales:**
   - **P:** Thor, Hulk, Iron Man y Capitán América forman equipo.
   - **Q:** El mundo está a salvo.
2. **Regla de inferencia:**

Forma	Nombre
$(P \implies Q) \land \neg Q$	Modus Tollendo Tollens (MTT)
$\therefore \neg P$	

3. **Simbolización y derivación:**
   a. $\mathbf{P1}: P \implies Q$ (Si los 4 héroes forman equipo, el mundo está a salvo)
   b. $\mathbf{P2}: \neg Q$ (El mundo **no** está a salvo)
   c. $\mathbf{C}: \neg P$ (**Por MTT** en a y b)

Conclusión ($\neg P$): Thor, Hulk, Iron Man y Capitán América **no** formaron equipo.



Aristóteles, George Boole y Gottlob Frege

¡Buen trabajo!

Finalización del capítulo 1

DOI: 10.17163/abyaups.167.2

Capítulo 2
Algoritmos

Resultado de aprendizaje

Diseña algoritmos para la resolución de problemas.

Indicador de logro

Utiliza estructuras de control para resolver problemas a través de algoritmos.

2.1. Importancia de la algoritmia en la ingeniería

El análisis de problemas y el diseño de soluciones a través de herramientas y sistemas computacionales trasciende la simple aplicación de un desarrollo de software, sino más bien se constituye, como una disciplina fundamental de la Ciencias de la Computación, en el área informática, que maximiza la capacidad de las computadoras para resolver desafíos complejos. Esta aproximación sistemática y eficiente se logra mediante la aplicación del pensamiento computacional (*Computational Thinking*), una habilidad cognitiva que se ha consolidado como un paradigma esencial en la educación y la práctica profesional en la actualidad [Kolczyk, 2008].



Para materializar estas soluciones en un artefacto tangible, es imprescindible el uso de un lenguaje algorítmico que sea preciso, general y eficiente. Es así como, *la algoritmia (o Pensamiento Algorítmico)* es la habilidad de formular problemas de manera que se transformen de una entrada a una salida deseada a través de una secuencia estricta de operaciones, tareas, procesos o secuencia de pasos. De esta manera, *los algoritmos* llegan a ser un concepto fundamental en la informática y representan estrategias formales para resolver problemas específicos a través de una secuencia de pasos bien definidos [Nazareth, 2023].

Más allá de la programación, se trata de un proceso cognitivo que involucra, *descomposición* de problemas complejos en subproblemas manejables, *abstracción* para centrarse en los detalles relevantes y, *evaluación y optimización* de las soluciones. Este enfoque metódico permite plantear las posibles soluciones a cualquier problema, independientemente del campo disciplinar. Su implementación se ha estudiado en áreas no tradicionales como la formación contable, en medicina, o en biología, demostrando que facilita la comprensión y solución procedimental de ejercicios

complejos a través de herramientas informáticas y su debida estructuración en un currículo formal [Llerena-Izquierdo, 2023].

Hoy en día, la necesidad de poseer conocimientos y dominio de la informática es una exigencia transversal para todo profesional, debido a la necesidad de elevar su potencial competitivo en un mercado laboral en aumento enfocado a la digitalización. Para la Ingeniería, el impacto de la informática es trascendental. Se puede evidenciar que ningún otro aspecto de la tecnología tiene un mayor impacto potencial en la ingeniería que la informática [Möller & Collignon, 2023]. Las bases de datos indexadas en áreas de Ciencias Naturales e Ingeniería, como Scopus y WoS, reflejan el volumen de producción científica en esta intersección, confirmando que temas como las ciencias computacionales aplicadas son consistentemente algunos de los más citados en la categoría [Yutong & Jin, 2024].

Los ingenieros no solo utilizan la programación y la simulación, sino que aplican el pensamiento computacional para: *Diseño Asistido por Computadora* (CAD) y *Análisis de Ingeniería Asistido por Computadora* (CAE), el *Modelado y simulación de sistemas complejos* (mecánica, resistencia de materiales, electricidad) para analizar, predecir y optimizar su comportamiento y, para la *implementación de sistemas avanzados como la Inteligencia Artificial (IA) y la Computación en la nube* para enfrentar los desafíos de la Industria 4.0, lo que permite el diseño y el desarrollo de soluciones guiadas y estandarizadas. La habilidad para concebir un plan de trabajo o de solución a un problema, que incluye el análisis y el diseño asistido por computadora de sistemas de producción, es una función directa del dominio algorítmico y el pensamiento computacional [Landman & Kohn, 2024]. Este dominio es lo que permite al ingeniero actual utilizar la herramienta, y también diseñar y automatizar el proceso de solución fundamental.



El avance de la informática ha ampliado exponencialmente el alcance de este bien generado por los primeros investigadores. Hoy, los problemas de ingeniería, medicina, educación o economía se formulan a través de estructuras algorítmicas que permiten simular, analizar y predecir resultados complejos. Estudios recientes en educación en ingeniería demuestran que integrar herramientas computacionales, como Python o MATLAB, mejora de manera significativa la capacidad de análisis y la comprensión conceptual de los estudiantes [Cuevas et al., 2025]. Esto reafirma que la programación es una destreza técnica, y un medio de estructurar el pensamiento, de modelar la realidad con precisión y creatividad [Llerena-Izquierdo et al., 2024b].

Desde los inicios de la historia de la ciencia, la humanidad ha buscado sistematizar los procesos de razonamiento que conducen a la resolución de problemas. Uno de los primeros en lograrlo fue Muhammad ibn Musa al-Khwārizmī, erudito del siglo IX que trabajó en la Casa de la Sabiduría de Bagdad. En su obra Al-kitāb al-mukhtaṣar fī ḥisāb al-jabr wa'l-muqābala, Al-Khwārizmī estableció los principios del álgebra, e introdujo una forma de pensar que consistía en descomponer un problema en pasos lógicos y secuenciales para alcanzar una solución. Este método, cuyo nombre derivó en la palabra *"algoritmo"*, se convirtió con el paso del tiempo en el corazón de las Ciencias de la Computación [Sun et al., 2024a]. La esencia de la contribución de Al-Khwārizmī consiste en haber concebido el conocimiento como un proceso de traducción del pensamiento lógico al lenguaje simbólico, algo que hoy se refleja en los lenguajes de programación [Kydyraliev

et al., 2025]. En la actualidad, el acto de programar no es más que la expresión moderna de esa antigua idea: *representar el pensamiento humano de manera precisa, formal y reproducible por una máquina de cómputo* [Fauziah, 2025].

La obra de Al-Khwārizmī, reinterpretada a la luz de la ciencia de la computación contemporánea, nos invita a ver la programación no como una habilidad aislada, sino como una forma de pensamiento universal [Roelens, 2025]. Los algoritmos, lejos de ser simples secuencias de instrucciones, son manifestaciones de una inteligencia estructurada que guía la resolución de problemas en cualquier ámbito del conocimiento [Yıldız, 2025]. Comprenderlos, crearlos y aplicarlos constituye hoy una de las tareas más nobles de la educación moderna: *preparar a los estudiantes para pensar en construir un futuro distinto, como lo hizo Al-Khwārizmī, con rigor lógico, curiosidad científica y sentido humanista,* ver Fig. 2.1.

Representación artificial



Figura 2.1: Imagen generada por la IA Gemini de Muhammad ibn Musa al-Khwārizmī y los algoritmos.

En este contexto, el algoritmo se convierte en el punto de encuentro entre la teoría y la práctica, entre la abstracción y la acción. Las investigaciones actuales coinciden en que el pensamiento algorítmico permite trascender el ámbito técnico para convertirse en una forma de razonamiento aplicable a cualquier disciplina [Sakdiyah et al., 2025].

Programar implica una forma de conocimiento que combina intuición, lógica y estética; una expresión moderna de la filosofía que Al-Khwārizmī anticipó cuando unió las matemáticas con el razonamiento sistemático [Karandikar & Warnekar, 2025].

Otro personaje importante es Augusta Ada King, Condesa de Lovelace (1815-1852), matemática británica e hija del célebre poeta Lord Byron, ella ocupa un lugar singular en la historia de la ciencia computacional como la primera persona en conceptualizar y documentar un algoritmo destinado a ser ejecutado por una máquina [Terroso Sáenz, 2025; Cavendish, 2015]. Su contribución trasciende la simple programación técnica, estableciendo las bases conceptuales del pensamiento algorítmico moderno y anticipando el potencial de las computadoras más allá del cálculo aritmético básico [Grzybowski et al., 2024; Rafiq, 2024], ver Fig. 2.2.

Representación artificial



Figura 2.2: Imagen generada por la IA Gemini de Ada Lovelace y los algoritmos.

Ada Lovelace[1] colaboró estrechamente con Charles Babbage en el desarrollo conceptual de la *Máquina Analítica*, un prototipo de computadora digital pero mecánica y diseñada en la década de 1840 [Martin, 2015; Charman-Anderson, 2020]. Este dispositivo, propulsado por vapor y programable mediante tarjetas perforadas, nunca se construyó completamente durante la vida de Babbage, pero representaba un salto conceptual extraordinario respecto a las calculadoras mecánicas de la época [Callaghan et al., 2020]. La *Máquina Analítica* incorporaba elementos fundamentales de la arquitectura computacional moderna: *una unidad de procesamiento aritmético* (el "Molino"), *memoria para almacenar datos* y resultados intermedios (el "Almacén"), y un *sistema de programación* mediante tarjetas perforadas que especificaban las operaciones determinadas en relación con las direcciones de memoria existentes [Haugtvedt & Abata, 2021; Ferry, 2015; Newton &

[1] Biografía de Ada Lovelace

Maas, 2025]. En sus notas, Ada Lovelace escribió lo que hoy se reconoce como el primer algoritmo destinado a ser procesado por una máquina, lo que la convierte en la primera programadora de la historia [Callaghan et al., 2020; Abbate et al., 2023]. Lo más notable de su trabajo, mucho más que la codificación de instrucciones, fue su visión conceptual de lo que las máquinas pueden llegar a lograr, en aquel presente y en el futuro [Rafiq, 2024]. Ella comprendió que los dispositivos computacionales (o de cómputo o de cálculo) no estaban limitados a cálculos aritméticos, ella tuvo la visión de que estos podían manipular símbolos formales de acuerdo con reglas, anticipando así el concepto de programación general [Charman-Anderson, 2015; Angluin, 2016].

El algoritmo conocido como *"La Nota G"*, fue escrito por Ada Lovelace, y contiene un conjunto de pasos diseñados para calcular los números de Bernoulli[2] utilizando la hipotética *Máquina Analítica*, y es generalmente reconocida como el primer algoritmo específicamente diseñado para una computadora [Charman-Anderson, 2015; Lovelace, 2015]. Este algoritmo apareció como parte de una serie de notas etiquetadas de la letra A, a la letra G que Lovelace anexó a su traducción al inglés del artículo de Luigi Menabrea sobre la conferencia de Babbage en la Universidad de Turín (1842) [Haugtvedt & Abata, 2021]. Los números de Bernoulli son una secuencia de números racionales con aplicaciones profundas en teoría de números, análisis matemático y combinatoria. El algoritmo de Lovelace para calcular estos números fue revolucionario porque demostró que una máquina podía realizar tareas que iban más allá de la aritmética simple, mostrando el potencial de la computación automatizada [Rice, 2024; Haugtvedt & Abata, 2021; Hénin, 2015].



Ada Lovelace imaginó una visión revolucionaria sobre si la información podía reducirse a símbolos y reglas de manipulación, entonces una máquina podría procesar cualquier tipo de información simbólica, no solo números [Haugtvedt & Abata, 2021]. Esta intuición anticipó conceptos fundamentales de la ciencia de la computación moderna, incluyendo, la computación a base de símbolos. Lovelace consideró que la *Máquina Analítica* podría manipular símbolos de acuerdo con reglas, abriendo la posibilidad de que las máquinas pudieran componer música o producir arte si los elementos del arte pudieran formalizarse adecuadamente [Gonçalves, 2024]. Otro concepto que intuyó fue de *"Programación como disciplina creativa"*, Lovelace identificó que la programación requería habilidades analíticas y creativas, estableciendo la programación como una forma de arte intelectual [Callaghan et al., 2020]. Finalmente, identificó los *"Límites de la computación mecánica"*, contrariamente, Lovelace también anticipó debates contemporáneos sobre inteligencia artificial al afirmar (1843) que *la Máquina Analítica "no tiene pretensiones de originar nada. Puede hacer cualquier cosa que sepamos ordenarle que realice"*, planteando tempranamente la cuestión de si las máquinas pueden exhibir creatividad genuina [Blackburn, 2024; Oliveira & Figueiredo, 2024].

Además, su trabajo ha sido una fuente de inspiración para generaciones de científicas y científicos, al demostrar que la programación no es solo una cuestión técnica, sino también una forma de pensamiento creativo [Callaghan et al., 2020; Case & Leggett, 2016]. Sus palabras resaltan la importancia del diseño algorítmico como puente entre la lógica humana y la ejecución mecánica [Füegi & Francis, 2015]. Después de su muerte prematura (36 años), las contribuciones de Ada Lovelace permanecieron en relativa obscuridad [Ferlier, 2025]. Su redescubrimiento en el siglo XX, particularmente tras la publicación de las biografías de Doris Langley Moore (1977) y Betty Alexandra Toole (1992), estableció su posición canónica en la historia de la computación [Haugtvedt & Abata, 2021]. El *"Lenguaje de programación Ada"*, desarrollado por el Departamento de

[2] Número de Bernoulli - Wikipedia

Defensa de Estados Unidos entre 1977 y 1983, para sistemas críticos, fue nombrado en su honor, reconociendo su contribución pionera, su búsqueda de la precisión y la fiabilidad [Füegi & Francis, 2015; Kyriazis, 2025]. Adicionalmente, el segundo martes de octubre[3] se celebra internacionalmente el *"Ada Lovelace Day"*, dedicado a reconocer los logros de las mujeres en ciencia, tecnología, ingeniería y matemáticas (STEM) [Tamboukou, 2023]. Iniciativas como el Ada Lovelace Day celebran su contribución y promueven la participación de las mujeres en STEM (ciencia, tecnología, ingeniería y matemáticas) [Strawn, 2023; Hollings et al., 2017b].

Lovelace también se adentró en un debate filosófico que sigue vigente hoy: la posibilidad de que la inteligencia artificial posea conciencia [Gonçalves, 2024]. Su afirmación más citada, conocida como la *"Objeción de Lady Lovelace"*, señala una observación, sobre cómo las máquinas reciben órdenes, y que no era una limitación de la máquina, sino una delimitación clara entre la ejecución de instrucciones (el algoritmo) y la capacidad de tener intencionalidad y creatividad autónomas [Rafiq, 2024]. Este argumento estableció, por primera vez, la distinción fundamental entre el software (las instrucciones) y el hardware (la máquina), y continúa siendo un pilar en la ética y filosofía de la IA [Charman-Anderson, 2020; Llerena-Izquierdo & Ayala-Carabajo, 2025a]. La contribución de Ada Lovelace ha cobrado nueva relevancia en la era de la inteligencia artificial [Grzybowski et al., 2024]. Su intuición sobre el poder simbólico de las máquinas se refleja en tecnologías actuales como los modelos de lenguaje, los algoritmos de aprendizaje automático y la programación orientada a objetos [Cavendish, 2015].

En consecuencia, hablar de Ada Lovelace no es solo evocar a la primera programadora, sino reflexionar sobre la dimensión humana del algoritmo [Haugtvedt & Abata, 2021]. Ella comprendió que programar es un acto de traducción entre dos mundos: el de las ideas y el de la materia, el de la mente y el de la máquina [Callaghan et al., 2020]. En esa intersección, el algoritmo deja de ser una simple secuencia de pasos y se convierte en un lenguaje del pensamiento, una manifestación simbólica de la capacidad humana para imaginar sistemas, modelar realidades y transformar el conocimiento en acción con ética [Natale & Henrickson, 2024; Llerena Izquierdo & Ayala Carabajo, 2025]. El impacto de Ada Lovelace trasciende la historia de la computación: es la raíz de una nueva forma de razonar [Haugtvedt & Abata, 2021]. Su trabajo realizado vive en cada código escrito, en cada modelo de inteligencia artificial, y en toda aproximación científica que traduzca la complejidad del mundo en estructuras lógicas comprensibles. Lovelace imaginó el primer algoritmo e imaginó el futuro [Avila et al., 2024]. El trabajo de Ada Lovelace es inconmensurable [Haugtvedt & Abata, 2021]. Al ser reconocida como la primera programadora de la historia, su pensamiento dispuso las bases para conceptualizar el computador y abrir su significado a la computación como ciencia. Su visión conecta directamente con los trabajos de Alan Turing, quien un siglo después retomaría y debatiría sus ideas sobre la inteligencia de las máquinas [Schlombs, 2021].

Hoy, su nombre es sinónimo de innovación en tecnología [Strawn, 2023]. Su figura inspira a nuevas generaciones de científicas, ingenieras y programadoras, recordándonos que la tecnología no es solo una cuestión de ingeniería, sino también de imaginación [Hollings et al., 2017a].

Ada Lovelace estableció fundamentos conceptuales de la algoritmia computacional un siglo antes de que existieran máquinas capaces de ejecutar sus ideas. Su visión trascendió las limitaciones tecnológicas de su época, anticipando el papel transformador que la computación ejercería en la civilización moderna [Ferry, 2015]. Más allá del algoritmo específico que desarrolló, su legado radica en demostrar que las máquinas podrían procesar información simbólica de formas generales, sentando las bases filosóficas de la revolución digital [Newton & Maas, 2025]. Su historia nos recuerda que los avances tecnológicos fundamentales frecuentemente provienen de saltos conceptuales y visión imaginativa tanto como de ingenio técnico, y que la diversidad de perspectivas enriquece fundamentalmente la innovación científica y tecnológica, de alguien que realmente valora, aprecia y se motiva por la ciencia [Charman-Anderson, 2020].

[3] Ada Lovelace Day

A lo largo de la historia de la computación, varias mujeres han dejado una huella imborrable en el desarrollo de la programación, la algoritmia y la tecnología digital. Entre ellas, **Grace Hopper**, **Hedy Lamarr**, **Margaret Hamilton** y **Katie Bouman**, quienes representan diferentes generaciones que han transformado, desde sus contextos, la forma en que concebimos la relación entre el pensamiento lógico y las máquinas.



Figura 2.3: Grace Murray Hopper ante el teclado UNIVAC I, hacia 1960.

Grace Brewster Murray Hopper[4] (1906-1992). La Precursora de los Lenguajes de Programación [Candler, 2020]. Nació el 9 de diciembre de 1906 en Nueva York y falleció el 1 de enero de 1992 en Arlington, Virginia (Wikipedia). Se graduó en Vassar College en 1928 con una licenciatura en Matemáticas y Física con honores, obtuvo su maestría en matemáticas de Yale en 1930 y su doctorado en 1934 donde se especializó en matemáticas relativas y física cuántica, simultáneamente, fue profesora de matemáticas en Vassar, mientras completaba su doctorado[5].

Durante la Segunda Guerra Mundial, se unió a la Reserva Naval (1943) y fue asignada al Proyecto de Computación de la Oficina de Artillería en Harvard, donde trabajó en el equipo que desarrolló la máquina analítica Harvard Mark I, una de las primeras computadoras electromecánicas [Aiken & Hopper, 1946] (ver Fig. 2.3).



Figura 2.4: Grace Hopper desarrolló en 1959, el lenguaje COBOL.

En 1949, Hopper se unió a Eckert-Mauchly Computer Corporation, donde diseñó uno de los primeros compiladores, que traducían las instrucciones de un programador en códigos de computadora, y acuñó el término *"compilador"*. Grace Hopper es reconocida como una de las fundadoras de la programación moderna.



Figura 2.5: *"Atrévete a intentarlo. Atrévete a hacer las cosas de forma diferente. Atrévete a ser audaz y simplemente observa qué sucede"*, Grace Hooper

Su contribución más destacada a la algoritmia fue el desarrollo del primer compilador informático, el sistema A-0 en 1952 (el primer compilador para lenguajes informáticos) [McDevitt & McGettigan, 2024], que traducía instrucciones simbólicas matemáticas a código binario legible por máquinas (ver Fig. 2.4).

Esta aportación significativa se convierte en una herramienta que traduce el lenguaje humano en lenguaje máquina, lo que sentó las bases de los lenguajes de alto nivel, democratizando la programación al abstraer la com-

[4] Biografía (1) Grace Hopper.
[5] Biografía (2) Grace Hopper.

plejidad del hardware y haciendo que el software fuera accesible más allá de los expertos en ensamblador [Candler, 2020] (ver Fig. 2.5).

Este avance condujo directamente al diseño del lenguaje COBOL -*lenguaje común orientado a negocios*- (1959) [Candler, 2020], que permitió que personas (en las empresas) sin formación altamente especializada, pudieran programar computadoras y escribir programas más comprensibles y eficientes[6]. Como programadora principal, trabajó en el diseño del UNIVAC I y desarrolló aplicaciones navales para COBOL [Hopper, 1952].

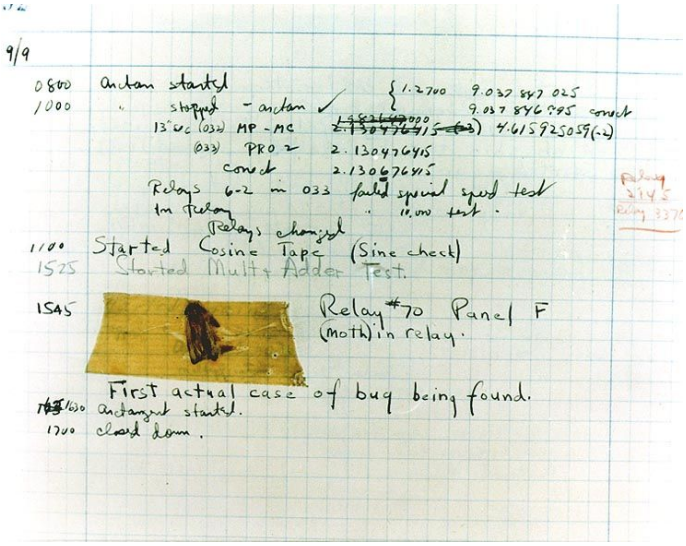


Figura 2.6: Primer caso real de error detectado en un equipo de cómputo.

Este lenguaje permitió que personas sin formación altamente especializada pudieran programar equipos [Aiken & Hopper, 1946]. En 1966, fue llamada nuevamente al servicio activo para ayudar a estandarizar los múltiples lenguajes y programas informáticos de la Marina. Así, Grace Hopper, es conocida como la "Abuela de COBOL" (uno de los primeros lenguajes de alto nivel, 1959) y como la mujer que inventó el primer compilador (A-0 System, 1952) [McDevitt & McGettigan, 2024] y el primer lenguaje de programación que utilizaba comandos similares al inglés (Flow-Matic, 1955 a 1959) [Candler, 2020]. Su visión fue fundamental para que el software fuera escrito en un lenguaje legible por humanos, revolucionando la programación automática. Una de las anécdotas más famosas de la historia de la computación involucra a Hopper. Aunque no acuñó realmente el término, ayudó a popularizarlo cuando una polilla fue encontrada en el Mark II, causando un cortocircuito [Hopper, 1952].

Mientras depuraba la Harvard Mark II, su equipo descubrió una polilla atascada en un relé, causando un mal funcionamiento. Hopper la extrajo y la pegó en el libro de registro con la nota "Primer caso real de un bug encontrado", acuñando así el término "debugging" (depuración), que aún usamos hoy, para referirse a la corrección de errores en los programas (ver Fig. 2.6). Este incidente dio origen al uso común del término "bug" (error o bicho) en programación. En 1991, el presidente George Bush le otorgó la Medalla Nacional de Tecnología por sus logros pioneros en el desarrollo de lenguajes de programación; fue la primera mujer en recibir individualmente el premio tecnológico más alto de

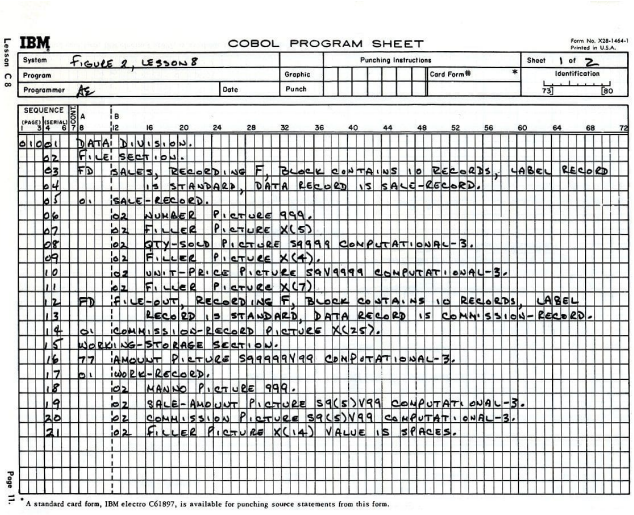


Figura 2.7: Sintaxis del lenguaje COBOL.

[6] Biografía (3) Grace Hopper

la nación Yale. En 2016, recibió póstumamente la Medalla Presidencial de la Libertad, el más alto honor civil. Hopper también fue una promotora incansable de la educación en informática, abogando por acercar la programación a un público más amplio. Hoy se la recuerda no solo por su genio técnico, sino por su visión democratizadora del conocimiento computacional. Su legado perdura a través del lenguaje COBOL (aún en uso, ver Fig. 2.7), es recordada con numerosos honores y como almirante de la Marina (retirada en 1986) [McDevitt & McGettigan, 2024], incluyendo ser la primera mujer comodoro de la Marina de los EE. UU. y el destructor USS Hopper (DDG-70) que lleva su nombre, además de su inducción póstuma al Salón de la Fama de Inventores. Inspira generaciones a través de la *Grace Hopper Celebration of Women in Computing*, un evento anual que fomenta la diversidad en STEM. Se retiró de la Marina como contraalmirante a los 79 años, siendo la oficial en servicio de mayor edad en las fuerzas armadas estadounidenses [Hopper, 1952]. Fue enterrada con todos los honores militares en el Cementerio Nacional de Arlington [Beyer, 2012].

Hedwig Eva Maria Kiesler, conocida como Hedy Lamarr[7], nació en Viena, Austria, el 9 de noviembre de 1914 y falleció en Casselberry, Florida, EE.UU. el 19 de enero de 2000 [Rosengren, 2022] (ver Fig. 2.8). Aunque más conocida como actriz de Hollywood, Hedy Lamarr fue también una inventora extraordinaria [Markstrom, 2020]. Famosa artista de cine, tuvo una formación en ingeniería y fue una autodidacta con una mente extremadamente inquisitiva. Aunque su carrera principal fue el cine, su interés por la invención era profundo. Lamarr[8] fue hija única de una familia judía acomodada. Su padre era director de un banco y su madre pianista de concierto. Su padre la llevaba a largos paseos donde discutían el funcionamiento interno de diferentes máquinas, como la prensa de impresión o los tranvías, lo que guió su pensamiento. Lamarr abandonó la escuela a los 15 años, para perseguir la actuación, debutando en el controvertido filme Ecstasy (1933), aunque su educación informal y autodidacta fue profunda [Griffiths & Farina, 2021].



Figura 2.8: Hedy Lamarr.

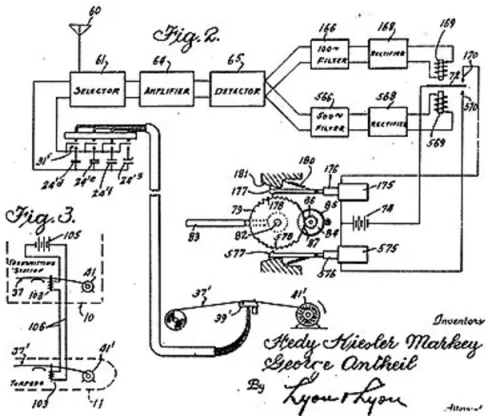


Figura 2.9: Salto de frecuencia de espectro ensanchado.

Tras una breve carrera cinematográfica en Checoslovaquia, huyó de su esposo, un rico fabricante austríaco de municiones, pero este episodio le dio la base de su conocimiento sobre sistemas de armas, incluidos sistemas de control de torpedos. Se trasladó secretamente a París. Viajó a Londres, donde conoció a Louis B. Mayer, quien le ofreció un contrato cinematográfico en Hollywood [Aguayo-Torres et al., 2024].

Su conocimiento técnico fue autodidacta, derivado de lecturas voraces en mecánica e ingeniería mientras filmaba en Hollywood. Lamarr, junto con el compositor George Antheil, inventó el sistema de *"Salto de Frecuencia"* (*Spread Spectrum*), una técnica que hoy se menciona como un desarrollo importante en el campo de las comunicaciones inalámbricas, desarrollado en 1941 [Burkland,

[7] Biografía (1) Hedy Lamarr
[8] Biografía (2) Hedy Lamarr

2021]. Este no es un algoritmo de programación como tal, sino un principio de comunicaciones que permite que la señal salte entre frecuencias de radio. Esta técnica es el fundamento teórico de la tecnología de las comunicaciones inalámbricas modernas, como el Wi-Fi, el Bluetooth y el GPS [Griffiths & Farina, 2021] (ver Fig. 2.9).



Figura 2.10: Madre del Wi-Fi.

Su sistema de comunicación por salto de frecuencia (*frequency hopping spread spectrum*), diseñado inicialmente para guiar torpedos sin ser detectados durante la Segunda Guerra Mundial derivó a su aportación algorítmica en el *"salto de frecuencia de espectro ensanchado"* (FHSS), patentado en 1942 [Burkland, 2021]. Este algoritmo sincronizaba señales de radio entre 88 frecuencias para evitar interferencias y *jamming* en torpedos, sentando precedentes para comunicaciones seguras. Desarrolló su invento durante la Segunda Guerra Mundial, preocupada por cómo torpedos controlados por radio podían ser fácilmente desviados por las fuerzas enemigas [Aguayo-Torres et al., 2024]. El salto de frecuencia de Lamarr y Antheil redujo el riesgo de detección o interferencia de torpedos controlados por radio. Se utilizaban múltiples frecuencias de radio para transmitir una señal de radio, cambiando de frecuencias a intervalos de fracciones de segundo de una manera aparentemente aleatoria que sonaría como mero ruido para cualquiera que escuchara. Este sistema fue diseñado para hacer que los torpedos guiados por radio fueran más difíciles de detectar o interferir [Griffiths & Farina, 2021].

Una anécdota emblemática revela su doble vida: durante una fiesta en su mansión de los años 40, mientras charlaba con aviadores y científicos, esbozó su idea de FHSS en una servilleta, inspirada en el piano de 88 teclas de Antheil[9]. Sin embargo, la Marina de EE.UU. rechazó la patente, argumentando que una *"estrella de cine glamorosa"* no podía ser seria inventora [Aguayo-Torres et al., 2024] (ver Fig. 2.10).



Figura 2.11: Su genio inventivo.

Durante sus sesiones de lluvia de ideas nocturnas, Hedy y Antheil jugaban un juego musical [Markstrom, 2020]. Se sentaban al piano juntos, una persona comenzaba a tocar una canción popular y la otra veía qué tan rápido podía reconocerla y comenzar a tocar junto. Fue aquí donde Richard Rhodes escribió el libro titulado *Hedy's Folly: The Life and Breakthrough Inventions of Hedy Lamarr, the Most Beautiful Woman in the World*, (La locura de Hedy: La vida y las invenciones revolucionarias de Hedy Lamarr, la mujer más bella del mundo), publicado en 2011, piensa que Hedy y Antheil concibieron por primera vez la idea del salto de frecuencia. Si dos músicos tocan la misma música, pueden saltar por el teclado juntos en perfecta sincronía. Sin embargo, si alguien escuchando no conoce la canción, no tiene idea de qué teclas se presionarán a continuación [Aguayo-Torres et al., 2024] (ver Fig. 2.11).

Lamarr no tuvo formación formal en ingeniería o programación, pero su capacidad para pen-

[9] Biografía (3) Hedy Lamarr

sar algorítmicamente —previendo secuencias de señales codificadas— refleja un entendimiento intuitivo de la lógica de la comunicación digital. Su vida combina el arte y la ciencia, desafiando los estereotipos de género de su época [Griffiths & Farina, 2021].

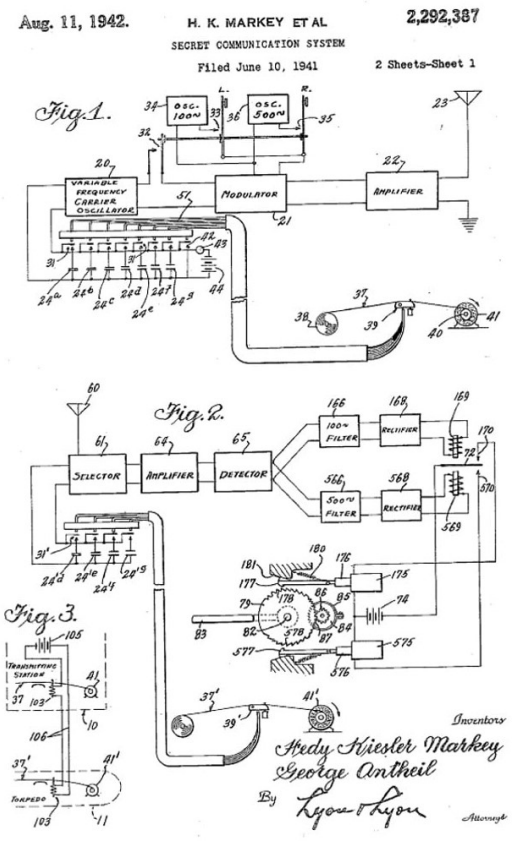


Figura 2.12: Diseño de patente.

Su faceta de inventora fue eclipsada por su fama de estrella de Hollywood durante décadas. Recién en el siglo XXI su contribución tecnológica comenzó a recibir el reconocimiento que merecía, convirtiéndose en símbolo de las mujeres invisibilizadas en la historia de la innovación. La tecnología no fue implementada por los militares hasta la década de 1960, pero desde entonces se ha convertido en la base de tecnologías modernas como Bluetooth y Wi-Fi [Burkland, 2021].

Hedy Lamarr[10] es conocida póstumamente como *"la madre del Wi-Fi"*, aunque recientemente, los intentos de abordar agravios históricos han caído en otro tipo de historia simplista [Markstrom, 2020]. En las últimas décadas se ha vuelto estándar, incluso de moda, acreditar la invención de Wi-Fi, Bluetooth e incluso teléfonos celulares a la estrella de cine de los años 1940 Hedy Lamarr, cuando en realidad múltiples inventores contribuyeron al desarrollo del salto de frecuencia (ver Fig. 2.12).

En los últimos años de Lamarr[11] recibió un premio por su invención. La Electronic Frontier Foundation otorgó conjuntamente a Lamarr y Antheil su Premio Pioneer en 1997 [Burkland, 2021]. Se convirtió en la primera mujer en recibir el Premio Bulbie Gnass Spirit of Achievement de la Convención de Invención [Griffiths & Farina, 2021].

El Día del Inventor en su Austria natal se celebra el 9 de noviembre en su honor. Ella es reconocida en el Salón de la Fama de Inventores Nacionales (2014) y en el *National Women's History Museum*. Su legado resalta la interseccionalidad entre arte y ciencia, y ha inspirado biografías y documentales que corrigen su subestimación histórica [Markstrom, 2020].



Figura 2.13: Margaret Hamilton.

Margaret Elaine Heafield Hamilton nació el 17 de agosto de 1936 en Paoli, Indiana, actualmente en 2025 sigue activa a sus 89 años [Hamilton & Hackler, 1991], (ver Fig. 2.13). Académicamente, obtuvo una licenciatura en matemáticas de Earlham College en 1958 e inició estudios de posgrado en la Universidad de Michigan, aunque los interrumpió para trabajar en el Proyecto SAGE de defensa aérea [Hamilton, 1986]. Comenzó su carrera en los años sesenta, cuando la programación aún no era considerada una disciplí-

[10] Biografía (4) Hedy Lamarr
[11] Biografía (5) Hedy Lamarr

na formal [Hamilton, 2018].

Su carrera despegó en el MIT *Instrumentation Laboratory* [Hamilton & Hackler, 2008], donde dirigió la división de ingeniería de software para NASA[12]. Fue la primera programadora contratada para el proyecto Apollo en el MIT y la primera mujer programadora en el proyecto, y posteriormente se convirtió en Directora de la División de Ingeniería de Software (Wikipedia). A principios de la década de 1960, Hamilton se unió al Laboratorio Lincoln del MIT, es allí donde participó en el proyecto *Semi-Automatic Ground Environment* (SAGE), el primer sistema de defensa aérea de Estados Unidos [Hamilton, 1986]. Escribió notablemente software para un programa para identificar aviones enemigos y el rastreo de misiles [Hamilton & Hackler, 1991].



Figura 2.14: Módulo del Apolo 12.

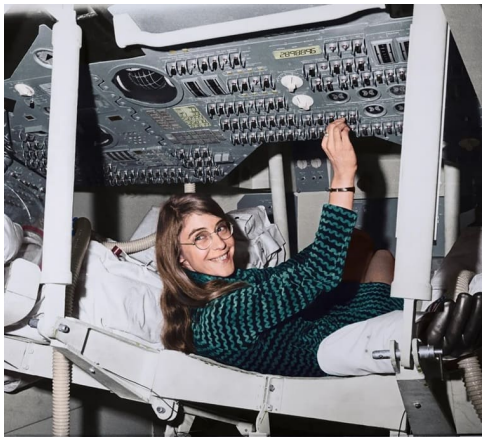


Figura 2.15: En el Módulo Lunar.

Margaret Hamilton fue la directora del equipo de desarrollo de software del MIT *Instrumentation Laboratory* que programó el sistema de navegación de las misiones Apollo de la NASA [Hamilton, 2018]. Su contribución pivotal a la algoritmia fue el diseño del software para la Computadora de Guía de Apollo (AGC) en la década de 1960, incluyendo algoritmos de programación asíncrona y manejo de interrupciones priorizadas [Hamilton & Hackler, 1991]. Estos permitieron multitarea en tiempo real y recuperación de errores, previniendo fallos catastróficos en misiones lunares [Hamilton & Hackler, 2008]. Su aporte a la algoritmia se materializó en los programas de control del módulo lunar, que incorporaban rutinas de detección y recuperación de errores, permitiendo al sistema priorizar tareas críticas [Hamilton & Hackler, 1991], (ver Fig. 2.14).

Gracias a esta lógica, la misión Apollo 11 pudo aterrizar con éxito en la Luna en 1969, a pesar de sobrecargas imprevistas del procesador [Hamilton & Hackler, 2008], (ver Fig. 2.15). Al liderar el equipo de desarrollo del software de vuelo a bordo para el Programa Apolo de la NASA (Apolo 11) [Hamilton & Hackler, 2008]. Su software fue fundamental para evitar un aborto de la misión de alunizaje al priorizar las



Figura 2.16: Apollo Software Code.

[12] Biografía (1) Margaret Hamilton

tareas esenciales del sistema en momentos críticos [Hamilton & Hackler, 1991]. Es conocida por acuñar el término *"Ingeniería de Software"* para legitimar el desarrollo de software como una disciplina de ingeniería [Hamilton, 2018].



Figura 2.17: Ingeniería de Software.

Una imagen icónica muestra a Hamilton junto a una pila de papeles que representa el código que su equipo escribió, símbolo de la magnitud del esfuerzo humano detrás de la exploración espacial[13], (ver Fig. 2.16).

Hamilton al acuñar el término *software engineering* dio reconocimiento al rigor necesario en la creación de código confiable [Hamilton & Hackler, 1991]. Ella afirmó: *"Comencé a usar el término -ingeniería de software- para distinguirlo del hardware y otros tipos de ingeniería, pero tratando cada tipo de ingeniería como parte del proceso general de ingeniería de sistemas"* [Hamilton, 2018].

A lo largo de su carrera, Hamilton[14] desarrolló los conceptos de software asíncrono, programación de prioridades y capacidad de decisión humana en el circuito, que se convirtieron en la base del diseño de software moderno [Hamilton & Hackler, 1991], (ver Fig. 2.17).



Figura 2.18: Nasa, Apolo 17.

Un error inspirado por su hija pequeña (que jugó con el simulador de vuelo y cargó un programa incorrecto justo antes del lanzamiento) llevó a Hamilton a insistir en la implementación de una sólida detección y recuperación de fallos en el software del Apolo [Hamilton, 1994]. Una anécdota singular data del 20 de julio de 1969, durante el alunizaje de Apollo 11: un error de software activó la alarma 1202, amenazando con abortar la misión [Hamilton, 2018], (ver Fig. 2.18).

Gracias al algoritmo de recuperación de errores de Hamilton[15], el sistema priorizó comandos críticos y salvó el aterrizaje, permitiendo que Neil Armstrong pisara la Luna [Hamilton, 1994]. Tres minutos antes del aterrizaje del Apollo 11, sonaron alarmas cuando la computadora se vio abrumada con datos de un sistema de radar innecesario que había sido activado accidentalmente por la tripulación [Hamilton & Hackler, 1990]. El software de Hamilton, diseñado con detección y recuperación de errores, salvó la misión [Hamilton & Hackler, 2008]. Hamilton recordó estos acontecimientos en una carta de 1971, diciendo *"si la computadora no hubiera reconocido este pro-*



Figura 2.19: Medalla Presidencial.

[13] Biografía (2) Margaret Hamilton
[14] Biografía (3) Margaret Hamilton
[15] Biografía (4) Margaret Hamilton

blema y tomado medidas de recuperación, dudo que Apollo 11 hubiera sido el exitoso aterrizaje lunar que fue" [Hamilton, 1986].

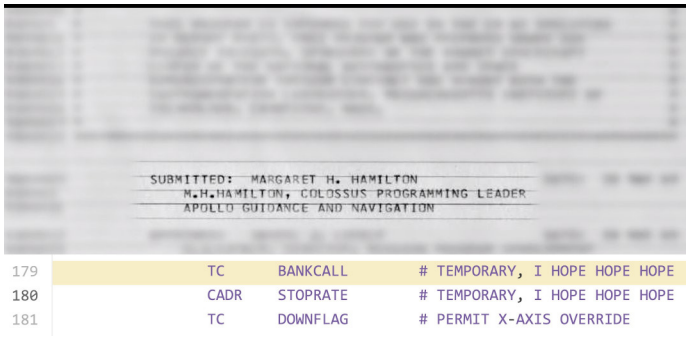


Figura 2.20: Comentarios dentro del código.

Otra anécdota revela su método único de depuración: Descubrió que el sonido podía servir como detector de errores [Hamilton, 2018]. Su programa en SAGE, notó, sonaba como la orilla del mar cuando estaba funcionando. Una vez, fue despertada por un colega, quien dijo que su programa *"¡ya no sonaba como la orilla del mar!"*. Se apresuró a trabajar ansiosa por encontrar el problema y comenzar a aplicar esta nueva forma de depuración a su trabajo [Hamilton, 1994]. En 2003, recibió el Premio Exceptional Space Act de la NASA por contribuciones científicas y técnicas [16]. El premio incluyó $37,200, la cantidad más grande otorgada a cualquier individuo en la historia de la NASA [Hamilton & Hackler, 2008]. El 22 de noviembre de 2016, Hamilton recibió la Medalla Presidencial de la Libertad de Barack Obama, el más alto honor civil de Estados Unidos, (ver Fig. 2.19). En 2017, un set de LEGO *"Women of NAS"* salió a la venta con minifiguras de Hamilton, Mae Jemison, Sally Ride y Nancy Grace Roman [Hamilton, 2018].

En 2019, para celebrar 50 años después del aterrizaje del Apollo, Google decidió hacer un tributo a Hamilton [Hamilton & Hackler, 1990]. Los espejos en la Instalación de Energía Solar Ivanpah se configuraron para crear una imagen de Hamilton y el Apollo 11 bajo la luz de la luna [Hamilton, 1994].

Hamilton sigue siendo una figura de referencia en la ética del software y en la promoción de las mujeres en STEM, además de ser una fuente de inspiración y una voz influyente [Hamilton, 2018]. Fundó dos compañías de software, Higher Order Software en 1976 y su propia empresa, Hamilton Technologies en 1986, ambas en Cambridge, Massachusetts [Hamilton & Hackler, 2008], (ver Fig. 2.20).

En la actualidad ella sigue contribuyendo al desarrollo de metodologías de software. Actualmente, Hamilton es una defensora de la ingeniería de software confiable y sigue impartiendo conferencias sobre ética en IA.



Figura 2.21: Katie Bouman.

Katherine Louise Bouman[17] nació el 9 de mayo de 1989 en West Lafayette, Indiana, Estados Unidos, de 36 años (2025) [Chanana, 2024], (ver Fig. 2.21).

Su padre, Charles Bouman, es profesor de ingeniería eléctrica e informática y de ingeniería biomédica en la Universidad de Purdue. Como estudiante de secundaria, Bouman realizó una investigación sobre imágenes en la Universidad de Purdue [D'Ignazio & Klein, 2020]. Se graduó cum laude en Ingeniería Eléctrica en la Universidad de Michigan (2011). Obtuvo su Máster (2013) y Doctorado (2017) en Ingeniería Eléctrica y Ciencias de la Computación en el Instituto Tecnológico de Massachusetts (MIT), donde se enfocó en

[16] Biografía (5) Margaret Hamilton
[17] Biografía (1) Katie Bouman

imagen computacional [Bouman et al., 2016]. Como postdoc en el Instituto Tecnológico de California (Caltech), lideró el equipo de imagen del EHT[18]. En el MIT, fue miembro del Laboratorio de Ciencias de la Computación e Inteligencia Artificial de dicha institución [Bouman et al., 2017].



Figura 2.22: Katie Bouman en TED.

Su tesis de maestría, *"Estimating Material Properties of Fabric through the Observation of Motion"*, recibió el premio Ernst Guillemin a la mejor tesis de máster en ingeniería eléctrica [Bouman et al., 2013]. Su tesis doctoral, *"Extreme imaging via physical model inversion: seeing around corners and imaging black holes"*, fue supervisada por William T. Freeman del MIT [Levis, 2025], (ver Fig. 2.22).

Su avance algorítmico fue el liderar desarrollo del programa CHIRP (Continuous High-resolution Image Reconstruction using Patch priors) en 2016 [Bouman et al., 2016, 2018], un algoritmo de aprendizaje profundo que reconstruye imágenes a partir de datos escasos y ruidosos del Telescopio del Horizonte de Sucesos (EHT) [Bouman et al., 2010, 2017], (ver Fig. 2.23).

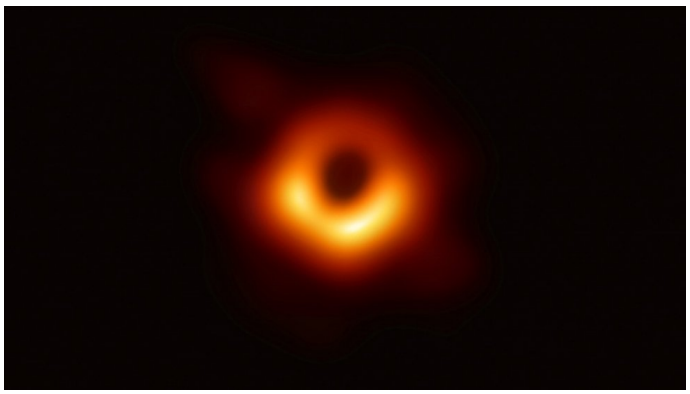


Figura 2.23: Imagen del agujero negro M87.

Este fue uno de los algoritmos de reconstrucción fundamental utilizados por el equipo internacional del Event Horizon Telescope (EHT) para sintetizar los vastos datos de radio y, en 2019 [Bouman et al., 2016, 2018], esto permitió generar la primera fotografía de un agujero negro, una imagen real del M87, fusionando radioastronomía con visión computacional [Bouman, 2020]. Su algoritmo combinaba datos incompletos de múltiples radiotelescopios y los reconstruía mediante modelos estadísticos [Bouman et al., 2010], una forma avanzada de razonamiento algorítmico que integra física, matemática y computación [Bouman et al., 2010, 2017], (ver Fig. 2.24).

Su trabajo excepcional permitió el desarrollo del algoritmo para la obtención de imágenes de agujeros negros, conocido como Reconstrucción Continua de Imágenes de Alta Resolución utilizando Prioridades de Parche (CHIRP) [Bouman et al., 2018]. Bouman[19] se incorporó al proyecto del Event Horizon Telescope en 2013. CHIRP inspiró los procedimientos de validación de imágenes utilizados en la adquisición de la primera imagen de un agujero negro en abril de 2019 [Levis, 2025], y Bouman desempeñó un papel importante en el proyecto al verificar las imágenes (aproximadamente 5 petabytes de datos) [Bouman et al., 2018],



Figura 2.24: Pilas de discos duros con datos de imágenes de agujeros negros.

[18] Biografía (2) Katie Bouman
[19] Biografía (3) Katie Bouman

seleccionar los parámetros para filtrar las imágenes tomadas por el Event Horizon y participar en el desarrollo de un marco de imagen robusto que comparaba los resultados de diferentes técnicas de reconstrucción de imágenes [Bouman et al., 2017], (ver Fig. 2.25).



Figura 2.25: Abril de 2019.

Una anécdota viral capturó su pasión: al revelarse la imagen del agujero negro M87 en abril de 2019, una foto de Bouman, con ojos desorbitados de emoción, se volvió meme como *"#KatieBug"*, acuñado por un colega [D'Ignazio & Klein, 2020]. Representó no solo su logro, sino el de un equipo global de 200 científicos [Chanana, 2024]. Con esto, ella se convirtió en un símbolo de la joven generación de científicos y programadores [Bouman et al., 2018].

Cuando se le preguntó sobre sus inicios, Bouman[20] admitió: *"Apenas sabía qué era un agujero negro, pero recuerdo pensar al final de la reunión que realmente quería trabajar en este proyecto"*. Esta honestidad ilustra cómo la pasión y la curiosidad pueden superar la falta inicial de conocimiento especializado [Bouman, 2020]. Su formación era en ciencias de la computación e ingeniería eléctrica, y se involucró en el proyecto mientras cursaba un doctorado en visión computacional [Bouman et al., 2016]. Su pasión es *"idear formas de ver o medir cosas que son invisibles"*, (ver Fig. 2.26).



Figura 2.26: Katie Bouman investigadora en métodos de imagen computacional y visión artificial.

El espacio era pequeño y caliente. En un día fatídico del verano pasado, Katie Bouman y tres compañeros investigadores se encerraron en una pequeña habitación en la Universidad de Harvard, a salvo de miradas indiscretas, para ver una imagen que había tardado años en crearse. En ese día de junio, los datos finalmente habían llegado y el equipo de Bouman presionó *"ir"*, esperando ver si el código que habían escrito realmente podía capturar lo invisible. Es Profesora Asistente de Ciencias de la Computación y Ciencias Matemáticas en el Instituto de Tecnología de California (Caltech), continuando su investigación en métodos de imagen computacional y visión artificial. Bouman dirige el Laboratorio de Imagen Computacional [Pasquali, 2020]. Su legado incluye premios como el Breakthrough Prize en Física Fundamental y charlas TED que promueven la colaboración en ciencia, inspirando a jóvenes en IA y astronomía [Levis, 2025].

Finalmente, desde los compiladores de Hopper hasta los algoritmos de imagen de Bouman, estas mujeres han tejido el tapiz algorítmico de la computación moderna. Sus anécdotas —de bugs literales a memes estelares— humanizan su genialidad, recordándonos que la innovación florece en la perseverancia [D'Ignazio & Klein, 2020]. Su legado actual no solo habita en tecnologías cotidianas, sino en un llamado a la inclusión: en un campo aún dominado por hombres, ellas iluminan el camino para futuras pioneras [Bouman et al., 2013].

[20] Biografía (4) Katie Bouman

Estas cuatro mujeres, en distintos momentos de la historia, demostraron que la algoritmia no es solo un conjunto de reglas, sino una herramienta creativa con el poder de transformar el mundo, ya sea haciendo los ordenadores más accesibles (Hopper) [Hopper, 1952], protegiendo comunicaciones esenciales (Lamarr) [Markstrom, 2020], salvando misiones espaciales (Hamilton) [Hamilton, 2018], o revelando los secretos del cosmos (Bouman) [Bouman, 2020], (ver Fig. 2.27).



Figura 2.27: Pioneras en la computación.

Ellas en conjunto, trazan una línea de tiempo que recorre casi un siglo de innovación en computación. Desde *la estructuración de los lenguajes de programación por Grace Hopper*, *la visión algorítmica de Hedy Lamarr*, *la ingeniería rigurosa de Margaret Hamilton* y *la creatividad en ciencia de datos de Katie Bouman*, cada una representa un momento importante en la evolución del *pensamiento computacional*ver Fig. 2.28.

Para analizar y profundizar

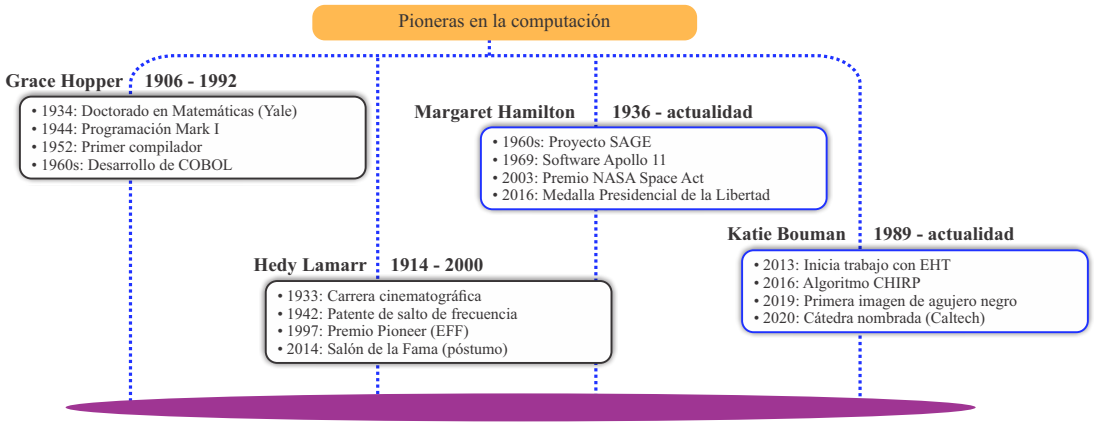


Figura 2.28: Recorrido del origen de la palabra Algoritmo.

Sus trabajos en conjunto demuestran que la programación no solo es un conjunto de códigos, sino una forma de pensar el mundo abstracto, de imaginar soluciones que no existen y de expandir los límites de lo posible desde lo imposible, ver Tabla 2.1.

Comparación entre contribuciones algorítmicas

Tabla 2.1: Impacto algorítmico

Pionera	Contribución	Impacto	Referencia
Grace Hopper	Compiladores y COBOL	Traductor de lenguaje natural a código máquina; abstracciones de alto nivel	[Hopper, 1952]
Hedy Lamarr	Salto de frecuencia	Algoritmo de sincronización pseudo-aleatoria para comunicaciones seguras	[Markstrom, 2020]
Margaret Hamilton	Software Apollo	Algoritmos de detección/recuperación de errores; programación asíncrona	[Hamilton, 2018]
Katie Bouman	Algoritmo CHIRP	Reconstrucción de imágenes mediante inversión de modelos físicos	[Bouman, 2020]

2.1.1. Conceptos fundamentales

La búsqueda de métodos para resolver problemas ha sido una constante desde la antigüedad y más aún en la era de la digitalización [Eschenburg, 2020]. Hoy lo que llamamos *"algoritmos"* proviene, como se indicó anteriormente, del nombre del matemático persa Muhammad ibn Musa al-Khwārizmī[21] (780–850 D.C.), cuyo trabajo en el siglo IX sobre el *"Compendio de cálculo por reintegración y comparación"* puso los inicios para el proceso de cálculo sistemático y, la unión de la aritmética y el álgebra, con ello el origen a lo que llamamos, algoritmo [Tatarchenko, 2023; Nabirahni et al., 2019]. Al-Khwārizmī trabajó en la Casa de la Sabiduría de Bagdad (periodo del Califato Abasí, actualmente Bagdad es capital de Irak), una institución que se convirtió en el centro del conocimiento científico del mundo islámico [Tatarchenko, 2019; Clark, 2012]. Sus métodos matemáticos representaban estructuras algorítmicas, aunque desarrollados hace más de mil años, contenían ya la esencia de lo que hoy entendemos como instrucciones precisas y finitas para resolver problemas [Malet & Panza, 2006]. Entre el año 820 al 825 D.C., publicó en Bagdad su tratado, sentando las bases para el álgebra, procedimientos de cálculo, el pensamiento algorítmico y los métodos iterativos que hoy subyacen en todo software. Así, su trabajo despierta la esencia misma de la computación y el pensamiento lógico [Parshall, 1988]. Para el año del 859, la Universidad de Al Quaraouiyine en la ciudad imperial de Fez, en Marruecos, comenzó a ser un centro de la transferencia de conocimiento, un pilar para el desarrollo posterior de los algoritmos. Este logro transformó las matemáticas en el mundo islámico medieval, que luego de traducciones al latín en la Universidad de Toledo en el siglo XII, influyó en la Revolución Científica europea, demostrando cómo un algoritmo puede trascender fronteras y épocas para alterar el curso de la historia humana [Oaks & Alkhateeb, 2007].

Los algoritmos, son secuencias precisas de instrucciones diseñadas para resolver problemas de manera sistemática, representan el centro de la computación moderna. Este proceso se define como un conjunto finito de instrucciones bien definidas, ordenadas y finitas que permiten realizar una tarea o resolver un problema [Oliveira & Figueiredo, 2024]. La noción de algoritmo estableció procedimientos sistemáticos para resolver ecuaciones, y su concepto trasciende épocas y culturas [Dunlop, 1943; Angluin, 2016]. Desde los cálculos astronómicos de los babilonios alrededor del año 2000 A.C., donde se inscribieron en tablillas cuneiformes los primeros pasos lógicos para predecir eclipses, hasta las complejas simulaciones que impulsan el Observatorio Espacial James Webb, los algoritmos han evolucionado de simples procedimientos manuales a pilares de la inteligencia artificial que procesan datos a escalas inimaginables [Bosworth, 1964; Abbate et al., 2023], ver Fig. 2.29.

Para analizar y profundizar

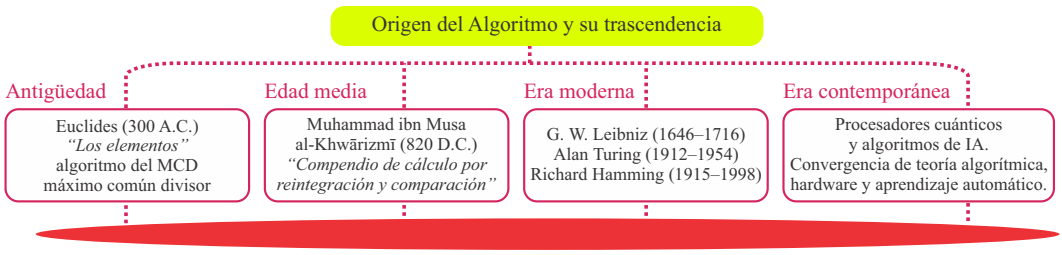


Figura 2.29: Origen de la palabra Algoritmo y su trascendencia.

Los conceptos fundamentales de los algoritmos se enmarcan en principios básicos para entenderlos, desde cómo se diseñan o representan, así cómo se analizan y aplican en la resolución de problemas reales, logrando abstraer una solución.

[21] Biografía de Al-Khwārizmī

Definición de un algoritmo

Un algoritmo es un conjunto de pasos que se reproducen en una secuencia finita bien definida, ejecutable y ordenada que, desde un punto de partida (una entrada), alcanza un punto de llegada (o produce una salida) en un tiempo realizable (finito) para conseguir (o resolver) la solución de un problema dado (o muy específico) [Cormen et al., 2022], o como *"Un conjunto finito de reglas precisas que especifican una secuencia de operaciones para resolver un problema específico", por* Knuth [2022].

Según Knuth [2022], los algoritmos tienen características esenciales. Por ejemplo, los algoritmos deben finalizar luego de un conjunto finito de pasos y en un tiempo determinado. Cada uno de estos pasos deben ser precisos (entendibles e inequívocos). A la vez, deben recibir datos de entrada (valores) y presentar al menos un dato de salida [Jiménez Toledo et al., 2023]. Es así como cada uno de los pasos debe ser realizable y efectivo (en un tiempo finito) ver Fig. 2.30.



Figura 2.30: Imagen desarrollada con napkin.ai para el tema: Características de un Algoritmo.

Las entradas y salida, conocidas como *input y output*, son los datos que se ingresan a los algoritmos (los primeros) y el resultado es el que se obtiene luego del procesamiento de los primeros (los segundos), estos se refieren a las variables [Altaher & Ferchichi, 2018], ver Fig. 2.31.

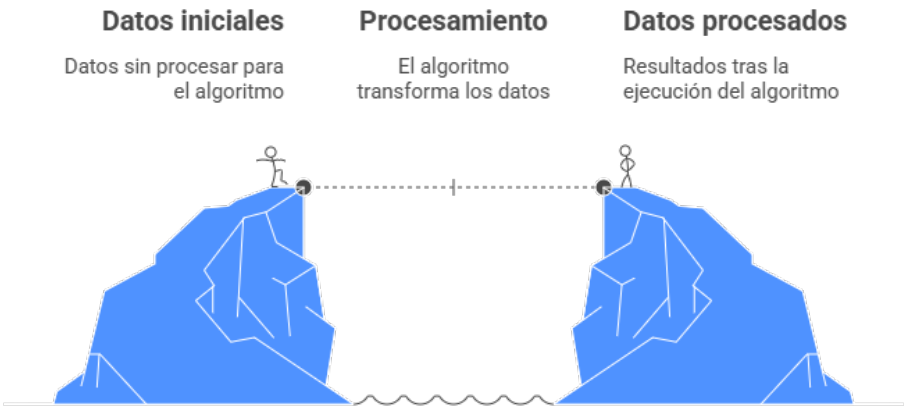


Figura 2.31: Imagen desarrollada con napkin.ai para el tema: Transformación de datos.

Otra característica de los algoritmos son las estructuras de control, bloques de instrucciones (o de código) que abordan un flujo de procesos [Vinayakumar et al., 2018; Stamouli et al., 2004], entre ellas están las estructuras secuenciales, las estructuras selectivas y las estructuras repetitivas, que guardan una jerarquía de uso [Silva-Maceda et al., 2016; Thomas et al., 2002], ver Fig. 2.32.

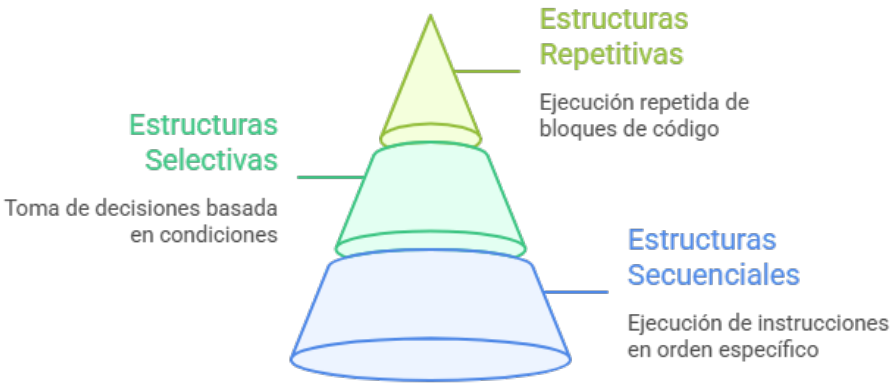


Figura 2.32: Imagen desarrollada con napkin.ai para el tema: Jerarquía de las estructuras.

Las estructuras secuenciales dirigen el flujo del procesamiento en un diseño, muchas veces utilizado una metodología de lectura de arriba hacia abajo (top-down) es decir, el computador (desde el compilador) irá leyendo línea a línea, que haya sido escrita en el archivo fuente (o principal) [Oomori et al., 2019; Watson & Li, 2014], así cada línea escrita (instrucción) será interpretada a medida que esta aparezca (editor), empezando de arriba (desde la primera línea que se escriba) hasta abajo (hasta la última línea o línea final que aparezca) [Futschek, 2006], ver Fig. 2.33.

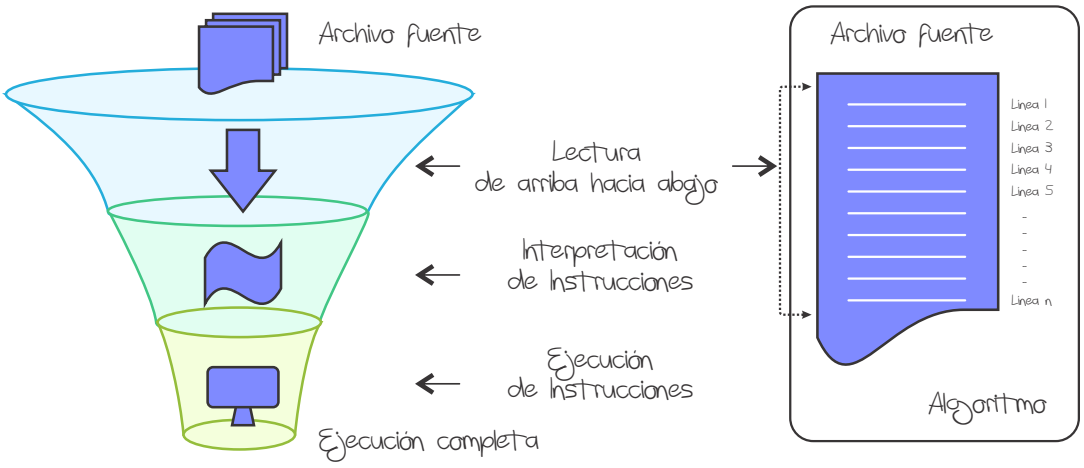


Figura 2.33: Estructura secuencial de la ejecución de un archivo fuente.

Otra característica que guardan los algoritmos es la exactitud, es decir, si es correcto, para todas las entradas válidas (variables iniciales), y que produzcan la salida esperada (variables finales) según la especificación del problema [Passi & Jackson, 2017; Hartmann et al., 2001].

Con esto, además, el análisis de eficiencia que mide los recursos que consume el algoritmo es otra característica tomada en consideración a medida que crezca el algoritmo en líneas de código e invoque recursos internos como externos haciendo uso de todos los enlaces, conexiones o nexo hacia datos, puertos de comunicación, sistemas externos [Thomas et al., 2018; Dourish, 2016].

La complejidad es otra característica llamada análisis de Complejidad de Algoritmos. La complejidad mide los recursos (tiempo y memoria) que un algoritmo requiere para ejecutarse. Es fundamental para predecir su rendimiento en función del tamaño de la entrada. La complejidad temporal ($T(n)$), mide el número de operaciones realizadas en función del tamaño de la entrada (n) [Wu et al., 2024].

Se expresa con la notación Big-O (O), que describe el límite superior del tiempo de ejecución en el peor de los casos [Sefati et al., 2025]. El Orden de Eficiencia sería: $O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(2^n)$ ver Fig. 2.34.

- $O(1)$ **(Constante):** El número de operaciones es fijo, independiente de n [Chen et al., 2025]. *Ejemplo:* Acceso directo a un elemento en un arreglo [Wang et al., 2025a; Kolmogorov, 2025].
- $O(\log n)$ **(Logarítmica):** El tiempo de ejecución crece muy lentamente [Bender et al., 2025]. El algoritmo descarta grandes porciones de la entrada en cada paso. *Ejemplo:* Búsqueda binaria [Sinnamon & Tarjan, 2025; Bhattacharya et al., 2025].
- $O(n)$ **(Lineal):** El tiempo de ejecución es directamente proporcional a n [Akitaya et al., 2025; Chernyshev et al., 2025]. *Ejemplo:* Recorrer una lista una vez [Li et al., 2025b].
- $O(n \log n)$ **(Log-Lineal):** Típica de algoritmos eficientes de "divide y vencerás" [Li et al., 2025b]. Es la complejidad más deseable para la ordenación [Wang et al., 2025c]. *Ejemplo:* Algoritmos de ordenación como *Merge Sort* o *Quick Sort* [Leverrier & Zémor, 2025].
- $O(n^2)$ **(Cuadrática):** El tiempo de ejecución es proporcional al cuadrado de n (típicamente dos bucles anidados) [Wang et al., 2025b]. *Ejemplo:* Algoritmos de ordenación ineficientes como *Bubble Sort* [Gabow, 2024].
- $O(2^n)$ **(Exponencial):** El tiempo de ejecución se duplica por cada aumento unitario de n [Malekian et al., 2025]. Solo viable para entradas minúsculas [Chapuy & Perarnau, 2025; Hirobe & Kasai, 2025].

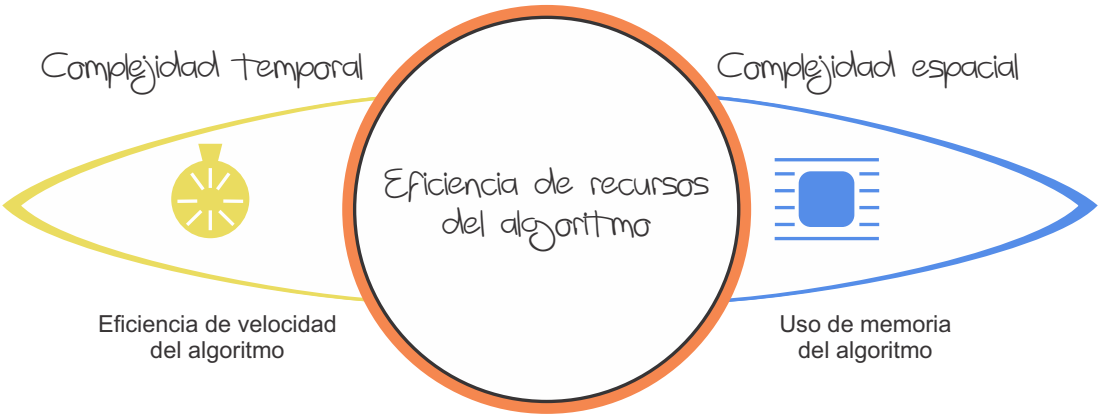


Figura 2.34: La complejidad del algoritmo.

En cambio, la Complejidad Espacial ($S(n)$) mide la cantidad de memoria adicional (espacio auxiliar) que el algoritmo necesita para ejecutarse, sin contar el espacio necesario para la entrada misma [Li et al., 2025c].

- $O(1)$ **(Constante):** La memoria adicional requerida es fija, sin importar el tamaño de n. *Ejemplo:* Un algoritmo que opera in-place sin estructuras de datos auxiliares [Assadi et al., 2025].
- $O(n)$ **(Lineal):** La memoria adicional es directamente proporcional a n. *Ejemplo:* Crear una copia auxiliar de la estructura de datos de entrada [Beelen & Neiger, 2025].

La complejidad del algoritmo se refiere a la eficiencia de los recursos para alcanzar el objetivo (rapidez de encontrar la solución y el uso disponible de la memoria para hacerlo) [Nguyen et al., 2024; Xu & Li, 2024].

Algunas estrategias comunes para crear algoritmos y se enmarcan en seis características diferenciadoras. Estas características representan a los paradigmas del diseño, ver Fig. 2.35.

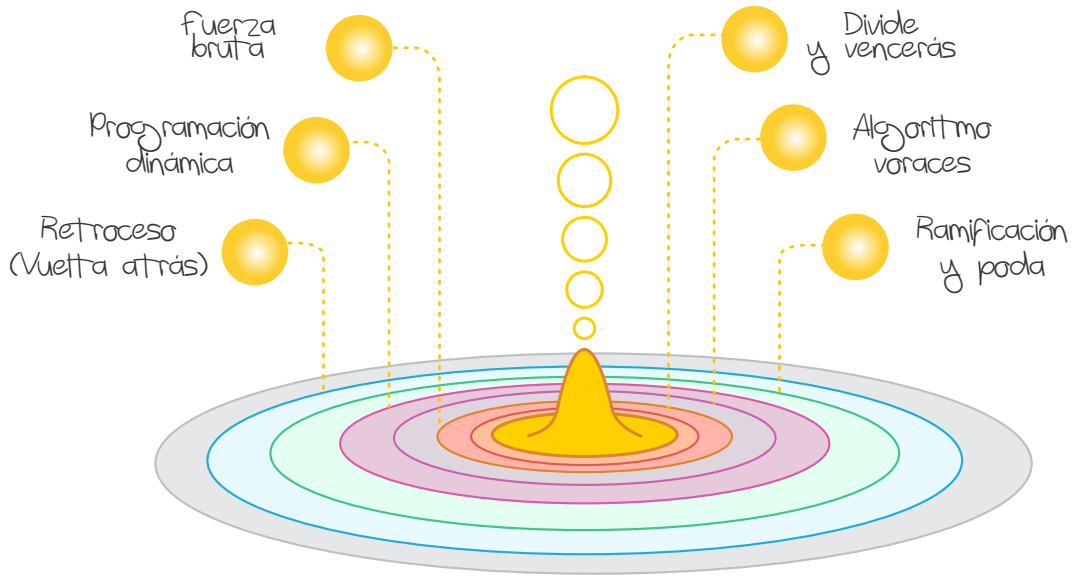


Figura 2.35: Estrategias de diseño de algoritmos basadas en paradigmas.

El paradigma de la *fuerza bruta* es una estrategia de encontrar todas las posibilidades de solución para un problema dado e ir verificando cuál es la más óptima. El ejemplo más común es adivinar una contraseña probando cada combinación de caracteres posibles [Sato et al., 2025].

El paradigma de *divide y vencerás* es una estrategia que aborda un problema de gran tamaño dividiéndolo en dos o más problemas pequeños, de esa manera se encuentra la solución juntando todas, para formar la solución al problema original. Dos ejemplos son los algoritmos de ordenamiento (Merge Sort y Quicksort), o la búsqueda binaria [Childs et al., 2025].

El paradigma de la *Programación dinámica* es una estrategia que aborda la construcción de soluciones óptimas a partir de soluciones de subproblemas que pueden estar superpuestos [Thengvall et al., 2025]. Es decir, los problemas se resuelven recursivamente, una y otra vez, almacenando las soluciones para construir progresivamente las soluciones de problemas más grandes. Los ejemplos más comunes son la secuencia de Fibonacci o el problema de *Knapsack* (mochila) [Xiao et al., 2025].

El paradigma de los *algoritmos voraces o Greedy* es una estrategia de toma de decisiones basándose en la mejor opción cercana [García-Díaz et al., 2025], por ejemplo, el algoritmo de Kruskal (árbol de expansión mínima) o el algoritmo de Dijkstra (camino o ruta más corta) [Czarnul et al., 2025].

El paradigma de los *algoritmos de retroceso o vuelta atrás* (backtracking) es una estrategia utilizada para ir encontrando soluciones parciales o diferentes, en los problemas de búsqueda,

construyendo soluciones incrementales [Stoddart et al., 2024; Chatterjee et al., 2024]. En cada ejecución, si una solución no conduce a nada *"retrocede"* al paso anterior y se prueba otra alternativa, a esto se le llama *"poda"* de la rama de soluciones [Hatami & Hoza, 2024]. Ejemplos de esta característica es, el problema de las N-Reinas, resolver el Sudoku, o encontrar caminos o rutas dentro de un laberinto [Jin et al., 2025].

El paradigma de los *algoritmos de ramificación y poda* (branch and bound) es una estrategia utilizada en problemas de optimización, para encontrar más de una solución óptima [Balcan et al., 2024; Nacar et al., 2024]. Al generar las posibles ramificaciones o soluciones *"poda"* aquellas soluciones que no pueden superar a la mejor solución actual [dos Santos & de Lima, 2024; Hien et al., 2024].

Cuatro características más que encontramos en el diseño y creación de algoritmos son: la recursión, la invariante de bucle, los casos existentes (peor, promedio y el mejor) y, la optimización y compensaciones.

La estrategia de diseñar algoritmos mediante recursividad es una técnica elegante que permite realizar la invocación a sí mismo de instancias, más pequeñas, del programa. Dos ejemplos son, el caso del factorial de un número o el problema de las torres de Hanoi [Abo Khamis et al., 2024; Qiu et al., 2025]. La estrategia de la invariante del bucle es la técnica de la afirmación que se mantiene verdadera antes y después de cada iteración para probar la exactitud de los bucles o lazos [Redondi et al., 2024; Vincent & Schwager, 2025]. La estrategia del análisis de casos, determina tres comportamientos, el mejor caso donde el algoritmo sea más eficiente o trabaje menos ($\mathcal{O}(n)$ en el algoritmo Quicksort ya ordenado) [Sun et al., 2024b]. El promedio, cuando el comportamiento se da con situaciones aleatorias. Mientras que el peor de los casos se da cunado el algoritmo es menos eficiente o trabaja más ($\mathcal{O}(n^2)$ en el algoritmo Quicksort) [Zhang & Vatedka, 2024], ver Fig. 2.36.

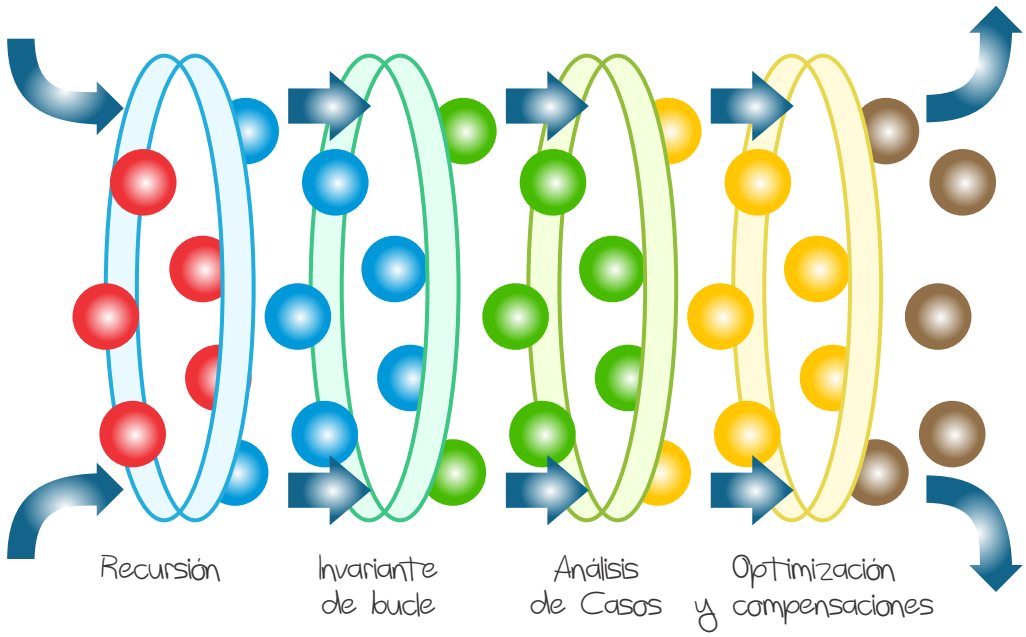


Figura 2.36: Características estratégicas de diseño de algoritmos.

Finalmente, la última estrategia es conocida como optimización y compensación, hay que recordar que todo algoritmo conlleva el compromiso del consumo adecuado del tiempo en relación con el espacio en disco o en memoria [Cuevas et al., 2025; Zhu et al., 2024].

También la necesidad de la simplicidad que se requiere en relación con la velocidad de ejecución total del algoritmo [Kundu et al., 2024; dos Santos & de Lima, 2024]. Así como la exactitud

del algoritmo en relación con la aproximación en situaciones que se requiera (por ejemplo, los algoritmos heurísticos y metaheurísticos) [Rajwar & Deep, 2025].

2.1.2. Formas de representar los algoritmos

Los algoritmos se componen de secuencias de instrucciones bien definidas y pueden ser representadas de diferentes maneras. Estas representaciones son requeridas desde sus diversos enfoques, desde lo textual hasta lo visual acorde a cada etapa del desarrollo del software.

Varios autores han ido promoviendo la claridad y la estructura en el diseño algorítmico. El teorema de estructura de Böhm & Jacopini [1966] estableció que cualquier algoritmo computable se puede expresar con solo tres estructuras básicas: estructuras secuenciales, estructuras de selección y estructuras de repetición, lo que influyó en representaciones posteriores como los diagramas estructurados. Asimismo, Dijkstra [1968a] criticó el uso descontrolado de saltos incondicionales (como GOTO), impulsando representaciones más disciplinadas [Liu et al., 2024], ver Fig. 2.37.



Figura 2.37: Formas de representar algoritmos.

Entonces, uno de los primeros pasos será la identificación del problema y definir su solución. Luego se desarrolla el algoritmo desde la solución secuencial de pasos que se han determinado para resolver el problema [Ensmenger, 2016]. El algoritmo es representado de forma visual a través de la creación de un flujo, utilizando símbolos geométricos (rectángulos, rombos, paralelogramos, etc.) [Wirth, 1973].

A continuación se describe y escribe el algoritmo mediante el uso de un código similar al lenguaje de máquina, conocido como pseudocódigo o seudolenguaje. Opcionalmente, se puede representar el algoritmo mediante los diseños de los autores Nassi & Shneiderman [1973], [Yoder & Schrag, 1978; Dijkstra, 1968b]. Se presenta a continuación un recorrido histórico de la representación de los algoritmos y sus autores, ver Tabla 2.2.

Evolución histórica de las representaciones algorítmicas

Tabla 2.2: Principales fechas en la representación de algoritmos

Año	Representación	Autores/Contribuidor	Contexto
1921	Flujogramas iniciales	Frank Gilbreth	Aplicación industrial para *eoptimización y análisis de procesos de trabajo*.
1947	Flujogramas computacionales	John von Neumann y Adele Goldstine	Representación lógica de la *secuencia de programas*, para máquinas *ENIAC*.
1966	Teorema de estructura (base para pseudocódigo y NSD)	Corrado Bohm y Giuseppe Jacopini	Comprobación matemática de solo *tres estructuras básicas* para construir cualquier algoritmo.
1968	Pseudocódigo (pidgin ALGOL)	Comunidad ALGOL (con Tony Hoare)	*Notación de alto nivel* para el diseño algorítmico, independiente de la sintaxis final del lenguaje.
1973	Diagramas Nassi-Shneiderman (NSD)	Isaac Nassi y Ben Shneiderman	Alternativa gráfica para la *programación estructurada* que elimina el uso de saltos o GOTO.
1997	Diagramas de actividad UML	Object Management Group (OMG)	*Extensión de flujogramas* para modelado orientado a objetos y *procesos de negocio complejos*.

Los flujogramas o *Flowcharts*

Los flujogramas son representaciones gráficas que emplean un conjunto de símbolos estandarizados (como óvalos para inicio/fin, rectángulos para procesos y rombos para decisiones) para ilustrar de manera visual el flujo de control lógico de un algoritmo. Aunque son muy intuitivos y valiosos en la fase de planificación inicial, pueden volverse engorrosos y difíciles de seguir en algoritmos complejos o de gran tamaño. Su origen con Gilbreth & Gilbreth [1921], los introdujo para analizar procesos industriales. Su aplicación a los algoritmos computacionales se consolidó en la década de 1940 gracias a las contribuciones de John von Neumann y Adele Goldstine [Goldstine, 1977], ver Fig. 2.38.

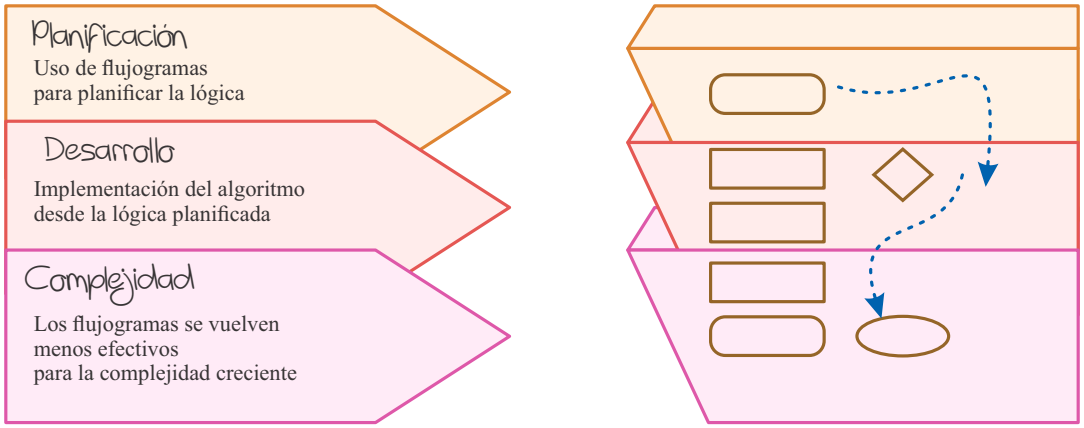


Figura 2.38: Los flujogramas.

El Pseudocódigo

El Pseudocódigo es un enfoque textual y semiformal que describe un algoritmo utilizando el uso del lenguaje natural (como el español o inglés) y construcciones sintácticas similares a las de un lenguaje de programación real, pero sin adherirse a reglas estrictas de compilación o implementación. Esta conformación abstracta lo convierte en una herramienta ideal para el diseño y la revisión formal, ya que facilita la comunicación entre desarrolladores y agiliza la traducción posterior a código fuente funcional. Surgió en la década de 1960 como un "pidgin code" influenciado por lenguajes como ALGOL [Ponder et al., 1988], ofreciendo una representación legible e independiente de la plataforma final [Hoare, 2022], ver Fig. 2.39.

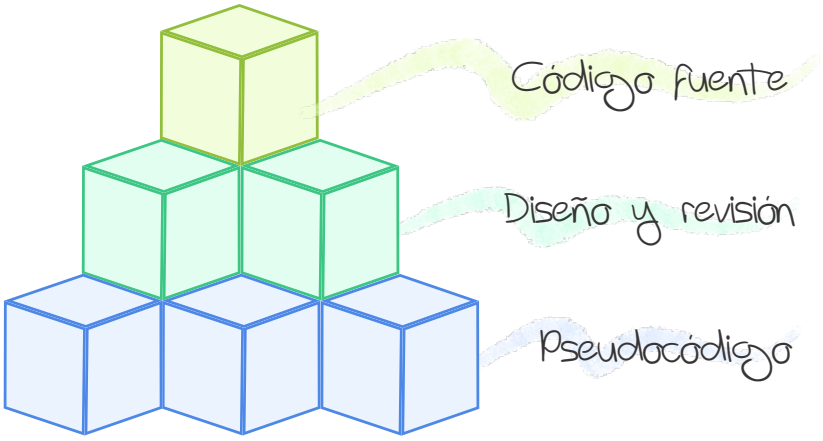


Figura 2.39: El Pseudocódigo.

Diagramas Nassi-Shneiderman (NSD)

Los diagramas Nassi-Shneiderman (NSD) son una alternativa gráfica avanzada a los flujogramas, diseñados específicamente para reforzar la programación estructurada. Estos diagramas utilizan bloques rectangulares anidados y delimitados para representar las estructuras de control fundamentales (secuencia, selección e iteración), eliminando completamente las flechas de flujo y, por lo tanto, previniendo el código *"espagueti"*. Fueron propuestos por Nassi & Shneiderman [1973] (Isaac Nassi y Ben Shneiderman en 1973)[22] como una forma más limpia y jerárquica de visualizar la lógica algorítmica. Su estructura se alinea directamente con los principios promovidos por autores clave en la programación estructurada, ver Fig. 2.40.

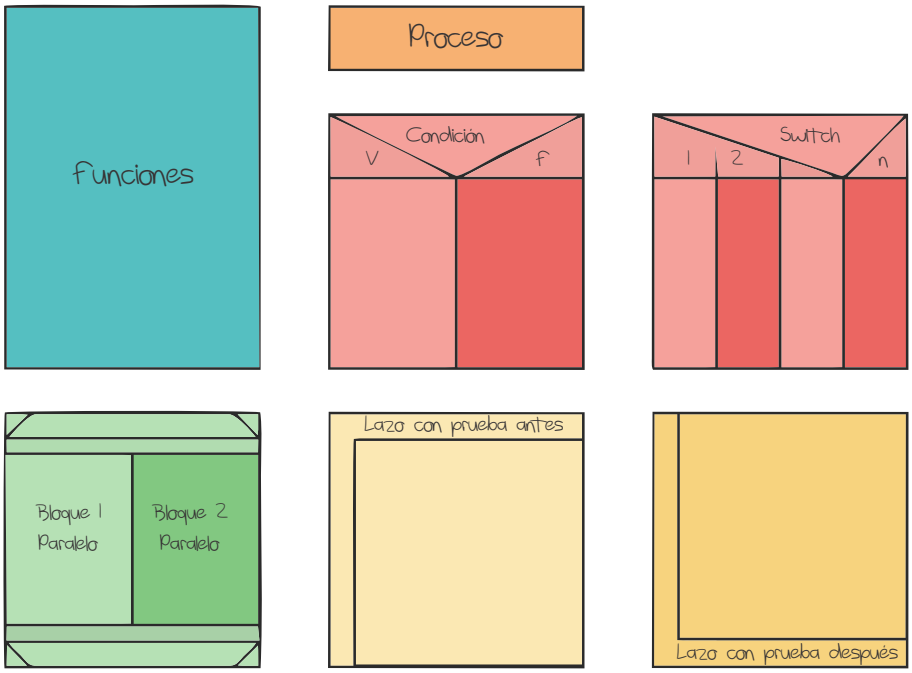


Figura 2.40: Diagramas Nassi-Shneiderman.

Otras representaciones de los algoritmos

Podemos mencionar representaciones que incluyen diagramas de actividad UML, Lenguaje Unificado de Modelado, (desarrollados a partir de 1990 por el Object Management Group para modelado orientado a objetos) y notación Bachman [2024] para bases de datos, pero las tres anteriores son fundamentales en educación y diseño algorítmico [Liu et al., 2025], ver Fig. 2.41.

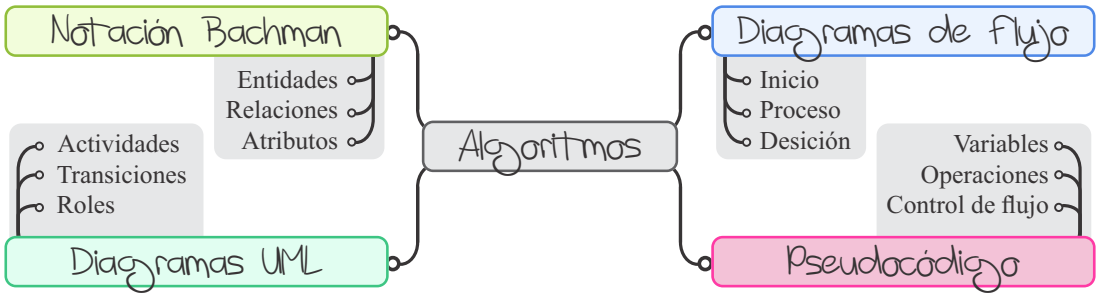


Figura 2.41: Representaciones relevantes y sus características.

[22] Disponible en el sitio de https://www.softwareideas.net/nsd-chart

De cada representación se destacan ventajas y desventajas. Por ejemplo, los flujogramas son visuales y fáciles de entender, pero escalan mal en complejidad cuando se extienden demasiado. Por el contrario, el pseudocódigo es conciso y modular, pero carece de visualidad para no programadores. Los NSD equilibran ambos, promoviendo legibilidad en entornos educativos, aunque requieren aprendizaje inicial.

Las constantes y las variables

Las *variables* y las *constantes* son los elementos necesarios para construir cualquier desarrollo computacional [Knuth, 1997]. Una *variable* es un espacio en la memoria de la computadora (como una caja de almacenamiento a medida) que guarda un valor (como un número o un texto) que está destinado a *cambiar* o ser modificado durante la ejecución del programa [Van Roy & Haridi, 2004]. Se le asigna un nombre simbólico (como "contador", "nombre", "num" o "saldo") para que el programador pueda referirse a ese espacio y manipular su contenido según lo requiera el flujo del algoritmo (la idea abstracta) [Rajaraman, 1994], permitiendo a los programas responder a condiciones dinámicas del mundo real [Stepanov & McJones, 2009].

Por el contrario, una *constante* es un espacio de memoria cuyo valor es *inmutable*, *invariable*, *que no cambia*; una vez que se define, no puede ser alterado a lo largo de la vida del programa [Apt & Wallace, 2006]. Las constantes se utilizan para almacenar valores críticos que deben ser fijos, como el número π, números atómicos, valores de densidad, el valor de la gravedad (en un planeta), tasas de impuestos o límites de seguridad, garantizando que estos datos esenciales no se modifiquen por error [Sterling & Shapiro, 1994]. Declarar un valor como constante es una buena práctica de diseño que indica a cualquier lector del código que ese valor está blindado y debe permanecer estable y sin cambios [Vagianou, 2006; Develaki, 2019].

El uso estratégico de *constantes* mejora la *legibilidad*, simplifica el *mantenimiento* y *previene errores*, especialmente al centralizar valores fundamentales para que un cambio solo deba hacerse en un único lugar [Chen et al., 2011]. Las *variables*, por el contrario, proporcionan la *flexibilidad* necesaria para implementar procesos dinámicos como bucles y acumuladores [Reas & Fry, 2007; Lin et al., 2021]. Un programador logra un equilibrio preciso entre estos dos conceptos (constantes y variables), separando lo que debe permanecer estable de lo que debe ser dinámico, lo que resulta en un *software* más robusto, elegante y fácil de mantener [Strachey, 2000], ver Fig. 2.42.

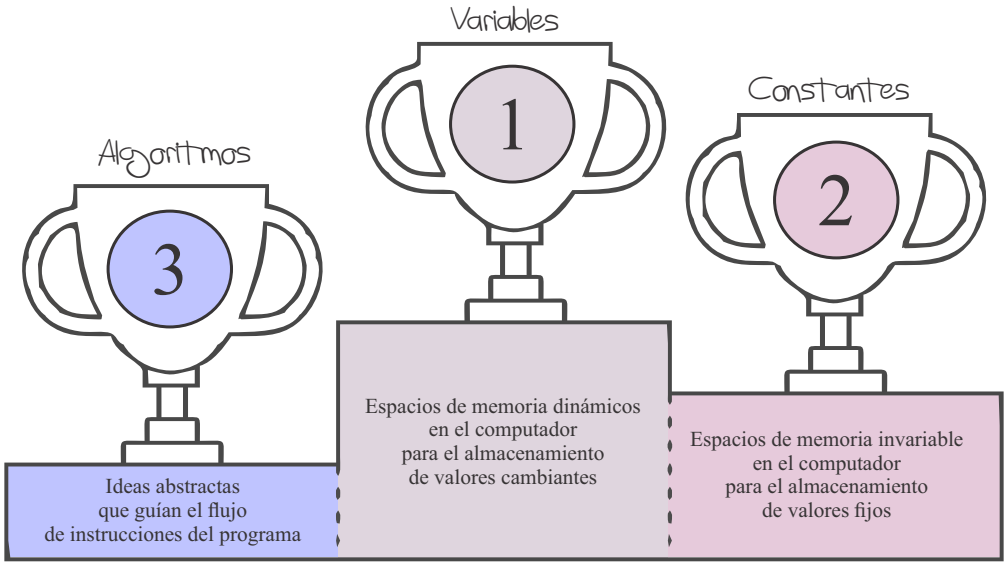


Figura 2.42: Las constantes y las variables.

Ámbito, alcance y los tipos de datos

El ámbito (o *scope*) de una variable o constante define en qué partes del programa se puede acceder a ella y durante cuánto tiempo existe en memoria [Karau et al., 2015]. Es uno de los criterios más importantes que cualquier lenguaje de programación impone para evitar conflictos de nombres y proteger la integridad del código [Lee, 2008]. Existen principalmente cuatro tipos de ámbito: **global**, **local** (o de función), de **bloque** y de **módulo** (o estático en algunos lenguajes) [Casciaro & Mammino, 2016; Singh, 2021]. Una variable global nace cuando se inicia el programa y muere cuando este termina, logrando ser vista desde cualquier función o bloque [Dalmau, 2004]; en cambio, una variable local solo existe dentro de la función o bloque donde se declara y desaparece al salir de él [Voelter et al., 2015]. El ámbito de bloque, popularizado por lenguajes como C (desde C99 con for(int i=0;...)), JavaScript (con let y const) y Python, restringe aún más la visibilidad a un par de llaves [Pym et al., 2019]. El buen uso del ámbito permite encapsular datos, reducir efectos secundarios y hacer que el código sea más predecible y fácil de depurar [Wampler & Payne, 2014; Bayle, 2013].

Los tipos de datos, por su parte, determinan qué clase de valores puede almacenar una variable o constante, qué operaciones son válidas sobre ella y cuánto espacio ocupa en memoria [Ashok, 2010]. Los lenguajes modernos los clasifican en tipos primitivos (o básicos) y tipos compuestos (o derivados) [Gerez, 1999]. Entre los primitivos más comunes están: entero (*int*), *real* o de punto flotante (*float*, *double*), carácter (*char*), booleano (*bool* o *boolean*) y, en algunos lenguajes, tipos como *byte*, *short*, *long* o *decimal* para mayor precisión [Cormen et al., 2022]. Los tipos compuestos incluyen arreglos, cadenas de texto (*string*), estructuras (*struct* o *record*), uniones, enumeraciones y punteros [Lagouvardos et al., 2020]. Además, desde finales del siglo XX, los lenguajes ofrecen tipos genéricos (como *List<T>* en Java y C#, o *templates* en C++) y tipos nulos seguros (*Option<T>* en *Rust*, *Optional<T>* en Java, o el tipo *nullable* en *Kotlin* y C#), que permiten expresar explícitamente la posible ausencia de valor y evitar el temido *NullPointerException* [Austin & Chancogne, 1999; Anami et al., 2010], ver Fig. 2.43.

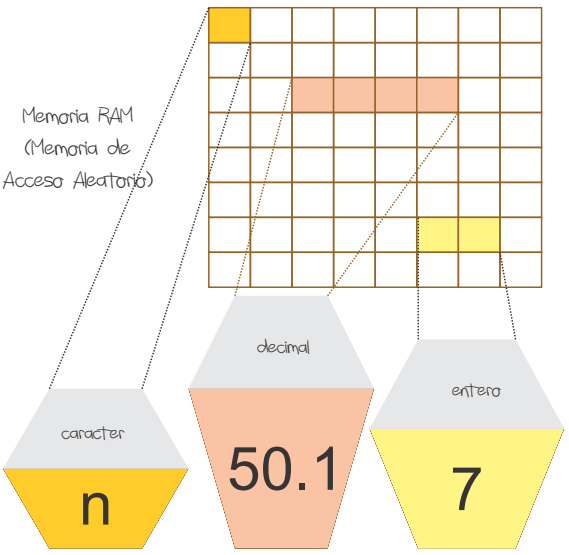


Figura 2.43: Almacenamiento en la memoria de acceso aleatorio por tipo de dato.

La relación entre ámbito y tipos de datos es profunda y bidireccional [Viswanadham & Narahari, 2015]. Un tipo estático (como en C, C++, Java o Pascal) obliga al compilador a conocer el tipo en tiempo de compilación, lo que permite detectar muchos errores antes de ejecutar el programa y optimizar el uso de memoria [Sasikumar et al., 2014]; en cambio, un tipo dinámico (Python, JavaScript, Ruby, PHP) decide el tipo en tiempo de ejecución, ofreciendo mayor flexibilidad pero trasladando la responsabilidad de la corrección al programador y al entorno de pruebas [Solomon, 1984]. A su vez, el ámbito influye directamente en la vida útil del dato: una variable local de tipo estático se destruye al salir del bloque (liberando memoria automáticamente en lenguajes con recolección de basura o mediante el *stack* en C/C++) [Yevick, 2005], mientras que una variable global o estática mantiene su valor entre llamadas, lo que puede ser tanto una ventaja (para contadores globales o configuraciones) como una fuente de errores difíciles de rastrear [Naramore et al., 2005; Kessler, 2006].

Dominar el ámbito y los tipos de datos, así como conocer la sintaxis de declaración de cada

lenguaje, es comprender la forma del diseño que busca el equilibrio entre seguridad, rendimiento y expresividad [Kurgalin & Borzunov, 2019]. Un programador que declara conscientemente const int MAX_INTENTOS = 3; en ámbito global, int contador = 0; en ámbito de función y let temporal: f64 = calcular(); ... en ámbito de bloque está, en realidad, escribiendo código más robusto, legible y preparado para crecer o escalar [Dale et al., 2000; Bantyukov et al., 2020]. En la práctica, ignorar estas reglas suele traducirse en bugs no esperados, vulnerabilidades de seguridad y costos de mantenimiento elevados; respetar las reglas, es una de las marcas distintivas del desarrollo de un software bien escrito, perdurable y mejorable [Doorenbos, 1995; Dale & Weems, 1996].

Las pruebas de escritorio

Las pruebas de escritorio (conocida como *desk checking* o *dry run*) son la forma más elemental y la más poderosa de verificar la corrección de un algoritmo antes de confiarlo a una máquina. Consisten en ejecutar manualmente, paso a paso, el pseudocódigo, flujograma o código fuente sobre papel o pizarra, siguiendo el flujo de control y actualizando el valor de cada variable en una tabla de seguimiento. Aunque hoy existen procesos de verificación visual dentro de los lenguajes de programación, en los entornos de depuración integrados, su práctica sistemática precede a la era de los computadores electrónicos demostrando repetidamente su capacidad para descubrir errores lógicos que las pruebas automáticas muchas veces pasan por alto, ver Fig. 2.44.

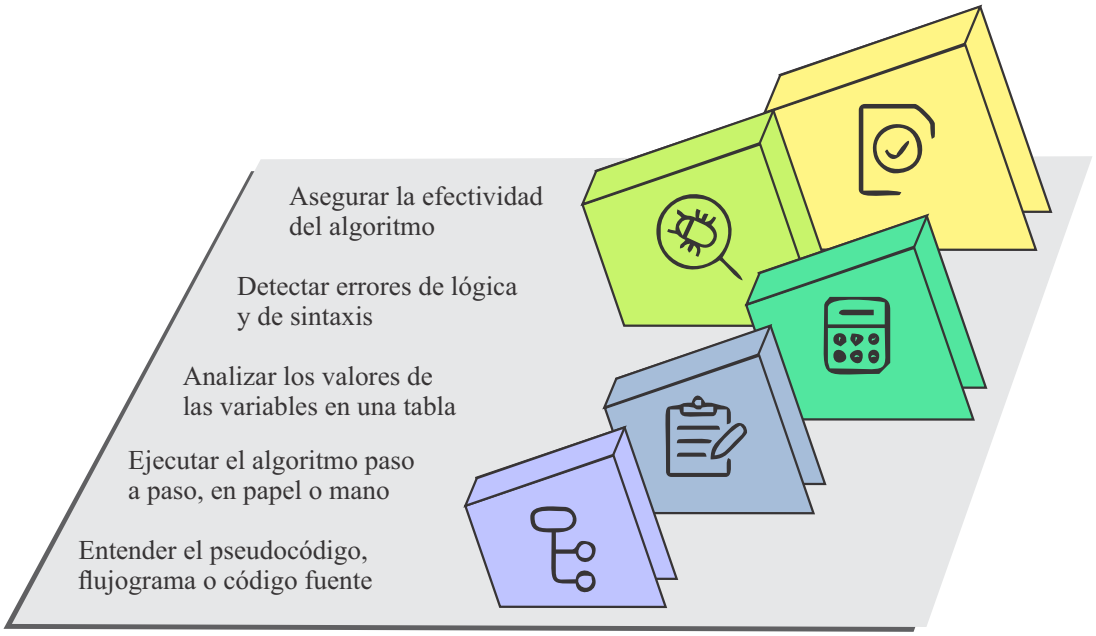


Figura 2.44: Las pruebas de escritorio.

El origen formal de esta técnica se remonta a los años 1940, cuando Goldstine et al. [1947] y Von Neumann & Goldstine [1947], en sus célebres informes sobre el diseño del EDVAC y el IAS, describían detalladamente la necesidad de verificar los diagramas de flujo mediante ejecución manual (lápiz, papel y pizarra) antes de codificarlos en la máquina. En esos documentos ya aparecían tablas de seguimiento idénticas a las que hoy enseñamos en cualquier curso introductorio de programación. Décadas después, a partir de 1970, cuando la programación estructurada luchaba por imponerse, autores como Mills [1972] y Baker [1972] incluyeron explícitamente las pruebas de escritorio dentro de las metodologías de *"inspección de código"* y *"lectura estructurada"*, demostrando empíricamente que un revisor humano siguiendo el algoritmo con lápiz y papel

detectaba entre un 60 % y un 80 % más de defectos que la simple lectura pasiva [Mills, 1976; Linger et al., 1979; Knight & Myers, 1993]. Más tarde, en los años 1980 y 1990, investigadores como Basili & Selby [1987] confirmaron en experimentos controlados que la combinación de pruebas de escritorio con inspecciones formales superaba en relación costo-efectividad a las pruebas funcionales y estructurales tradicionales [Basili, 1997; Selby et al., 1987; Basili & Selby, 1991]. A pesar del auge de los depuradores gráficos, *coverage tools* y *property-based testing*, la literatura científica sigue evidenciando su vigencia. Un estudio de Li et al. [2019] mostró que estudiantes que practicaban pruebas de escritorio sistemáticas antes de escribir código en Python cometían un 43 % menos errores lógicos en exámenes finales que aquellos que dependían exclusivamente del intérprete interactivo.

Otro trabajo de Michaeli & Romeike [2019], publicado en ACM Transactions on Computing Education (indexado en Web of Science), demostró que la habilidad de realizar pruebas de escritorio de forma manual correlaciona fuertemente con la capacidad de comprender y depurar programas ajenos, una competencia fundamental en la industria actual donde la mayoría del tiempo se dedica a mantener código legado [Marwan et al., 2020].

Así, las pruebas de escritorio son una actividad que obliga al programador a razonar con precisión sobre estados, flujo de control y condiciones de borde [Hsu et al., 2018]. Goldstine y von Neumann las necesitaban porque cada ejecución real costaba horas de configuración de cables y tubos de vacío [Goldstine & Von Neumann, 1951]; hoy las necesitamos porque cada desarrollo puede costar millones [Agbo et al., 2019; Tikva & Tambouris, 2021]. El lápiz y la tabla de seguimiento siguen siendo, una de las herramientas más económicas y efectivas para producir software correcto desde la primera versión [Espinal et al., 2024; Buitrago Flórez et al., 2017].

2.2. Estructuras de control

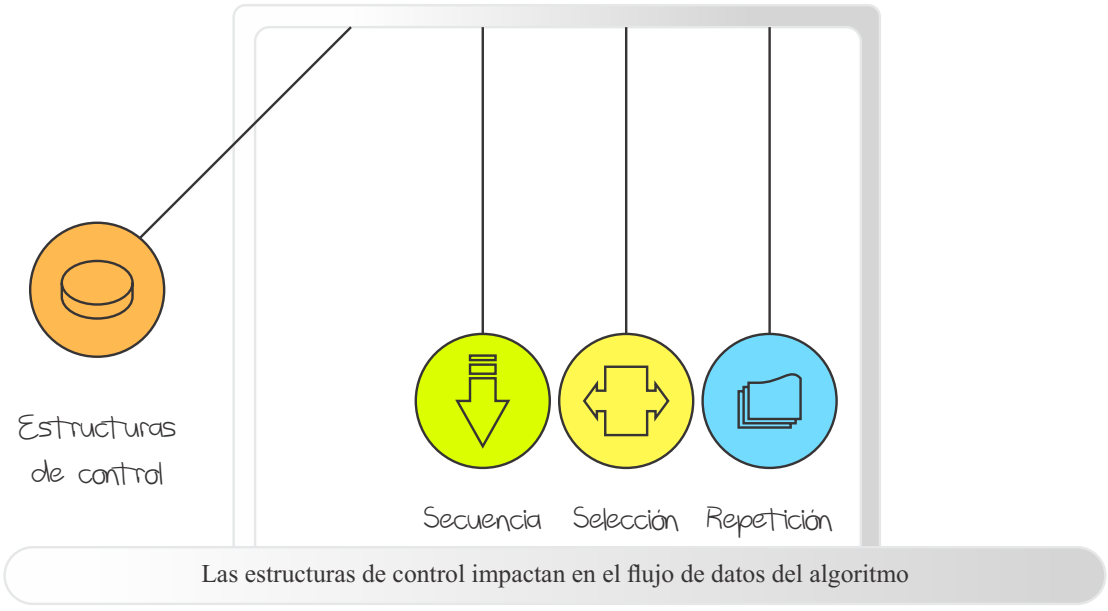


Figura 2.45: Las estructuras de control existentes.

Las estructuras de control son, en sí, aquellas secciones o bloques del algoritmo que permiten dirigir el flujo de datos, que sigue una lógica de solución, ver Fig. 2.45. Böhm & Jacopini [1966] demostraron mediante las matemáticas que todo algoritmo puede elaborarse usando únicamente tres formas de control: la secuencia, la selección y la repetición. Las tres estructuras no son

arbitrarias: reflejan directamente la forma en que los seres humanos razonamos sobre procesos. Ejecutamos pasos uno tras otro (secuencia), tomamos decisiones según condiciones (selección) y repetimos acciones o instrucciones mientras algo sea cierto o hasta alcanzar un objetivo (repetición). Al limitar el flujo de control a estas formas anidadas sin solaparse, se consigue lo que Dijkstra [1968a] llamó, *"programas transparentes"*, donde su comportamiento puede entenderse y demostrarse sin necesidad de simular mentalmente caminos infinitos.

2.2.1. Estructuras secuenciales

La estructura secuencial consiste en ejecutar instrucciones una tras otra, en el orden exacto en que aparecen. No hay bifurcaciones ni bucles: cada instrucción se ejecuta exactamente una vez y el flujo pasa automáticamente a la siguiente. En pseudocódigo se representa simplemente escribiendo las acciones línea por línea. La secuencia de instrucciones es la base de todo algoritmo y sobre la que se construyen las demás estructuras (condicionales y repetitivas).

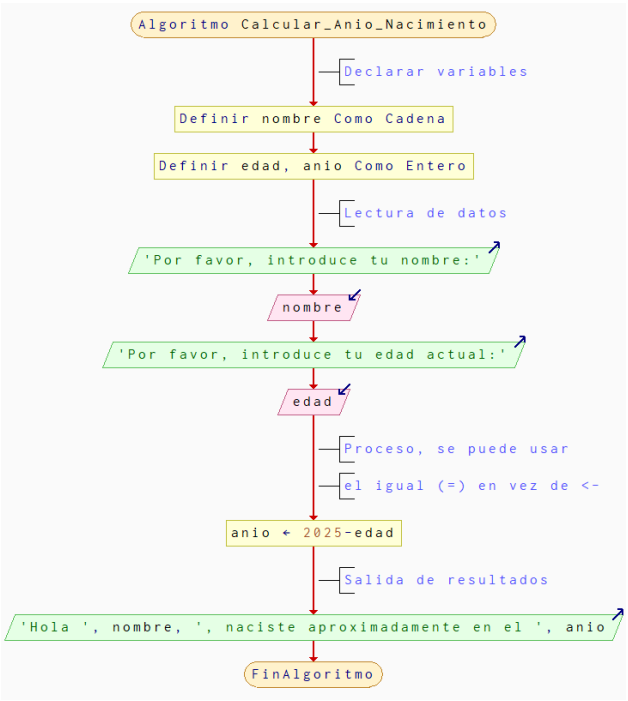


Figura 2.46: Estructura secuencial.

Sin una ejecución ordenada de pasos, efectiva y predecible, ninguna selección o repetición tendría sentido su uso. A continuación, se presentan un conjunto de ejercicios que demuestran la utilidad de ejecutar las secuencias de instrucciones paso a paso. Además, en los siguientes algoritmos, el uso de las dos barras (//) se los conoce como comentarios, mensajes o anotaciones para comprender la sintaxis de las instrucciones, se irá colocando en comentarios de línea para dejar ideas y buenas prácticas de al momento de programar. La representación gráfica de los algoritmos se los conoce como diagramas de flujo, en ellos se utilizan formas geométricas sencillas para visualizar el flujo o recorrido de los datos, ver Fig. 2.46.

Algoritmo 2.1: Pseudocódigo PSeInt: Cálculo del año de nacimiento

```
1  Algoritmo Calcular_Anio_Nacimiento
2     // Declarar variables
3     Definir nombre Como Caracter
4     Definir edad, anio Como Entero
5     // Lectura de datos
6     Escribir "Por favor, introduce tu nombre:"
7     Leer nombre
8     Escribir "Por favor, introduce tu edad actual:"
9     Leer edad
10    // Proceso, se puede usar el igual (=) en vez de <-
11    anio <- 2025 - edad
12    // Salida de resultados
13    Escribir "Hola ", nombre, ", naciste aproximadamente en el ", anio
14 FinAlgoritmo
```

A continuación se presenta un problema para ejemplificar a una estructura secuencial en pseudocódigo y su diagrama de flujo. El área (A) de un círculo se calcula multiplicando el valor de Pi (π) por el radio (r) elevado al cuadrado.

Algoritmo para su revisión en pseudocódigo:

Algoritmo 2.2: Pseudocódigo PSeInt: Área del círculo

```
1  Algoritmo Calcular_Area_Circulo
2     // Declarar variables
3     Definir radio, area Como Real
4     Definir P Como Real
5     // Proceso, en PSeInt PI ya es constante, puedes usar PI en el area
6     P <- 3.14159265
7     Escribir "Introduce el radio:"
8     Leer radio
9     area <- P * (radio * radio)
10    // Salida
11    Escribir "El resultado es: ", area
12 FinAlgoritmo
```

Se puede observar con la primera sección (líneas 3 y 4) se definen las variables que intervienen el problema. P es una constante, radio es la variable de entrada y area (sin acento) es la variable de salida, ver Fig. 2.47.

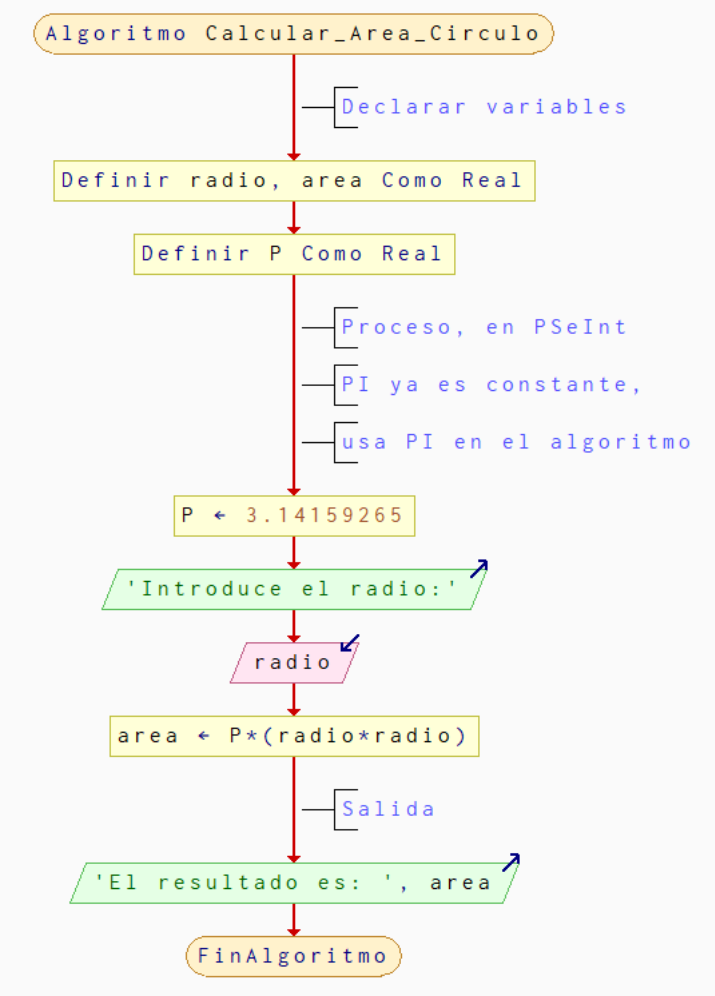


Figura 2.47: Algoritmo que calcula el área del círculo.

Otro ejemplo para representar una estructura secuencial se muestra a continuación. El área de un hexágono usando solo el lado (L). Este algoritmo es práctico entenderlo, ya que solo pide la longitud de un lado como variable de entrada. Utiliza la constante $\sqrt{3} \approx 1{,}73205$ para evitar el uso de una función raíz o usar la función raiz(x) que trae PSeInt.

Algoritmo para su revisión en pseudocódigo:

Algoritmo 2.3: Pseudocódigo PSeInt: Área de la figura regular usando solo el lado

```
1  Algoritmo Calcular_Area_Hexagono_Lado
2     // Declarar variables. Se usan 'Real' para decimales.
3     Definir lado, area, raiz_tres Como Real
4     // Asignar el valor de la constante Pi o usar PI
5     // Nota: 1.73205 es aproximada del radicando de 3.
6     raiz_tres <- 1.73205
7     // Lectura de datos
8     Escribir "Introduce la longitud del lado (L) de la figura regular:"
9     Leer lado
10    // Proceso: A = (3 * raiz(3) / 2) * L^2
11    // Se traduce: (3 * raiz_tres * (lado * lado)) / 2
12    // Se recomienda calcular (lado * lado)
13    // antes de multiplicar por 3 * raiz_tres
14    area <- (3 * raiz_tres * (lado * lado)) / 2
15    // Salida de resultados
16    Escribir "El resultado es: ", area
17 FinAlgoritmo
```

Este algoritmo define tres nombres para dos variables y una constante. Se observa que las variables llamadas lado y area (sin tilde) son las variables de entrada y salida, mientras raiz_tres es la constante visibles en la sección de declarar variables. En la sección de lectura de datos se evidencia el mensaje y la lectura del lado. Seguidamente, se encuentra la sección del proceso donde la línea 14 se presenta la fórmula utilizada en el algoritmo. Finalmente, la sección de salida de resultados se presenta la variable area (sin tilde) para mostrar el resultado. A continuación, se presenta el algoritmo en su representación de diagrama de flujo, ver Fig. 2.48.

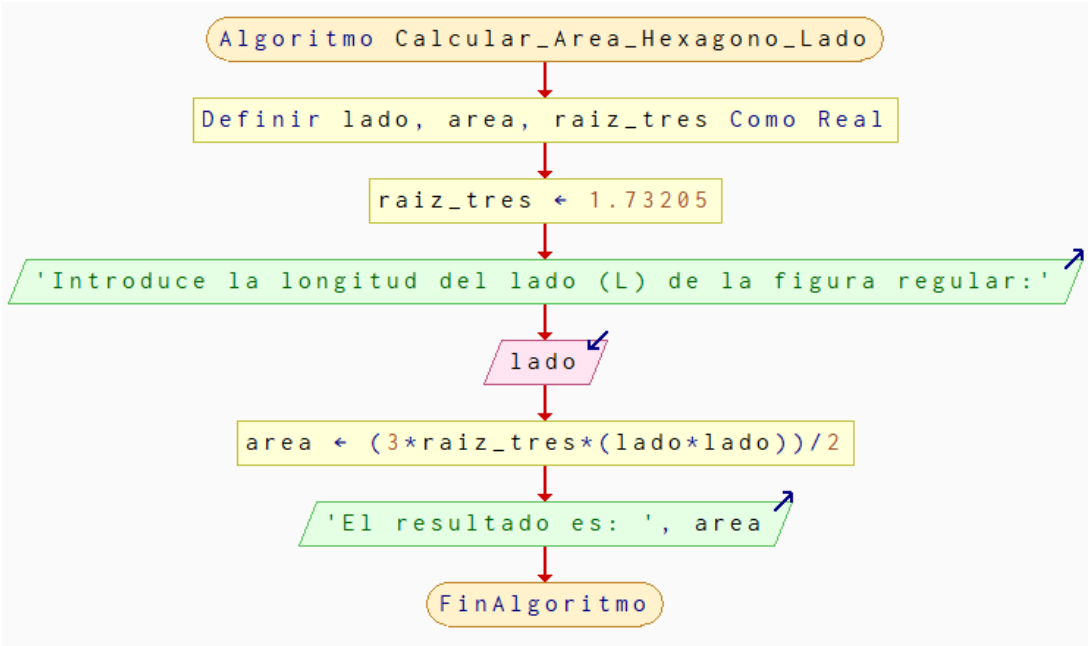


Figura 2.48: Algoritmo que calcula el área del hexágono usando el lado.

Otro ejemplo para representar una estructura secuencial se muestra a continuación. Por ejemplo, si solicitan el área de un Hexágono usando el lado (L) y apotema (a). Este algoritmo sigue la fórmula general para polígonos regulares: $A = (P \cdot a)/2$.

Algoritmo para su revisión en pseudocódigo:

Algoritmo 2.4: Pseudocodigo PSeInt: Área de hexágono regular (lado y apotema)

```
Algoritmo Calcular_Area_Hexagono_Apotema
   // Declarar variables. Se usan 'Real' para decimales.
   Definir lado, apotema, perimetro, area Como Real
   // Lectura de datos
   Escribir "Introduce la longitud de la figura regular:"
   Leer lado
   Escribir "Introduce la longitud del apotema:"
   Leer apotema
   // Proceso:
   // 1. Calcular la periferia: P = 6 * Lado
   perimetro <- 6 * lado
   // 2. Calcular la superficie: A = (P * Apotema) / 2
   area <- (perimetro * apotema) / 2
   // Salida de resultados
   Escribir "La superficie de la figura regular es: ", area
FinAlgoritmo
```

Se puede observar que todas las fórmulas utilizadas en la sección del proceso (en los algoritmos) deben escribirse de manera lineal (utilizando los paréntesis y respetando la jerarquía de los operadores). Además, las variables pueden escribirse usando letras y números, pero evita caracteres especiales, letras especiales y el uso de la tilde o acento en las vocales. A continuación, se presenta el algoritmo en su representación de diagrama de flujo, ver Fig. 2.49.

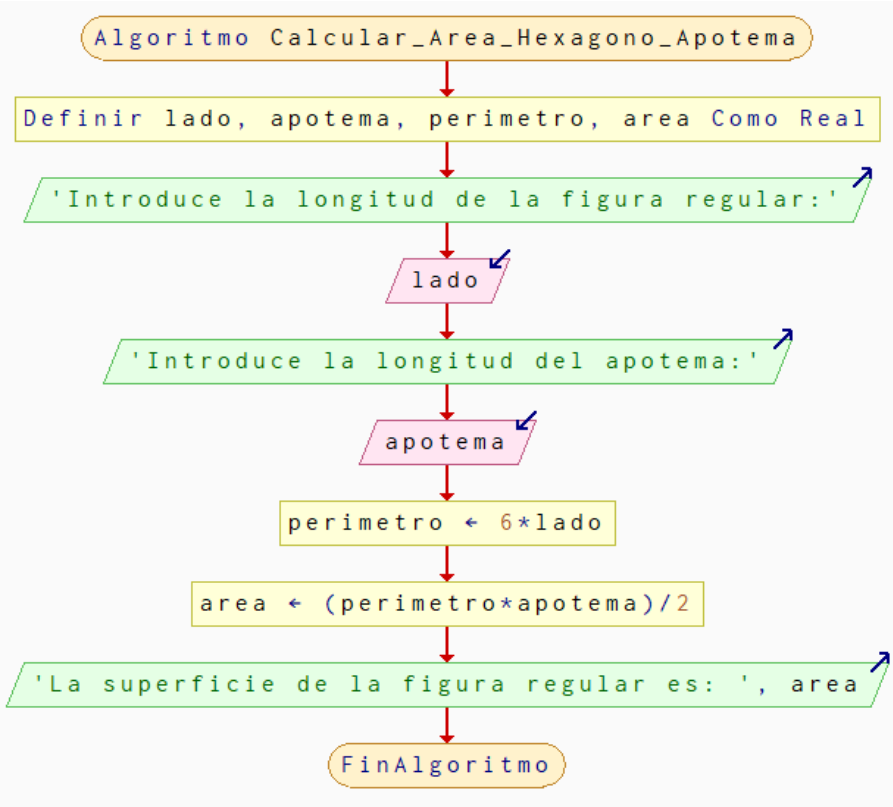


Figura 2.49: Algoritmo que calcula el área del hexágono usando el lado y el apotema.

2.2.2. Estructuras condicionales (Decisión simple)

Las estructuras condicionales permiten que el flujo de datos elija entre uno, dos o más caminos según el valor de una expresión lógica. Las formas existentes son condicional simple, condicional doble y condicional múltiple. El propósito de estas estructuras es modelar decisiones del mundo real, por ejemplo, "si llueve, llevo paraguas; si no, llevo gafas de sol; en cualquier otro caso, voy sin nada". El teorema de Böhm & Jacopini [1966] demuestra que una sola construcción si-entonces-si_no (if-then-else) basta para expresar cualquier selección, pero los lenguajes modernos ofrecen variantes más legibles y seguras. Un error común entre programadores principiantes es anidar demasiados if-else creando el llamado "infierno de flechas"; la solución pasa por usar estructuras anidadas verticales (elseif), tablas de decisión o, en casos complejos, patrones como polimorfismo.

Algoritmo 2.5: Pseudocódigo PSeInt: Verificación de mayoría de edad

```
Algoritmo Verificar_Edad
   // Declarar variables
   Definir edad Como Entero
   // Lectura de datos
   Escribir "Introduce tu edad actual:"
   Leer edad
   // Estructura Selectiva Simple (Si-Entonces)
   Si edad >= 18 Entonces
      Escribir "Eres mayor de edad por eso tienes acceso."
   FinSi
FinAlgoritmo
```

Como se observa en el pseudocódigo anterior, la sección de salida se encuentra dentro de la estructura selectiva simple. Además, esta estructura solo tiene habilitado la opción por VERDADERO, es decir, solo interesa enviar un mensaje si es mayor e igual a 18, caso contrario no hay mensaje alguno. A continuación, se presenta el algoritmo en su representación de diagrama de flujo, ver Fig. 2.50.

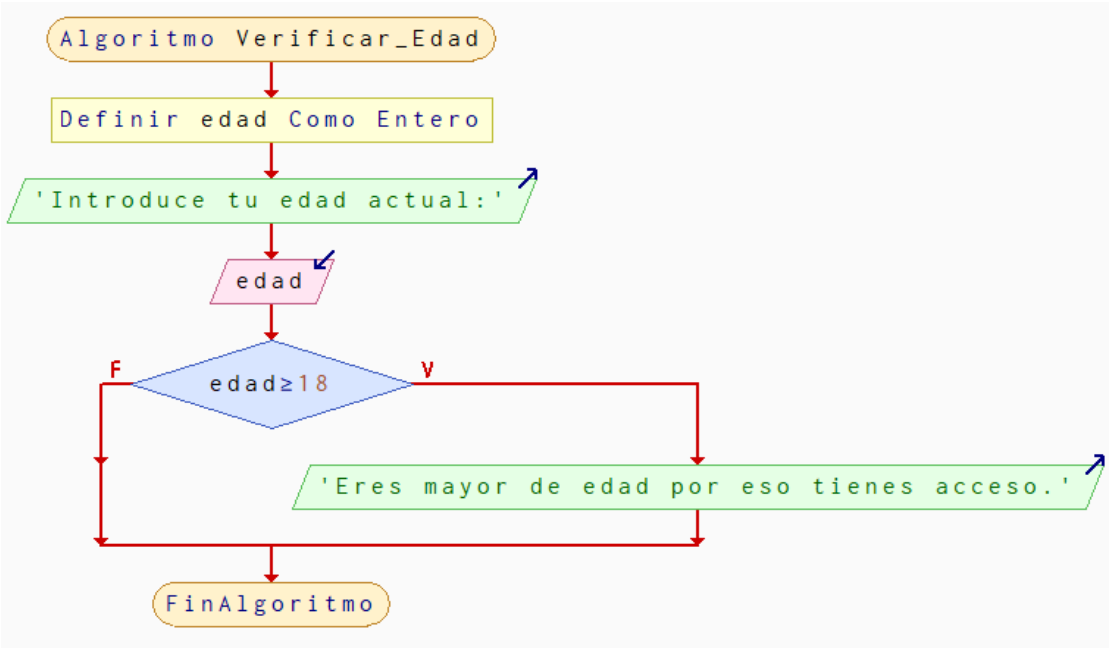


Figura 2.50: Algoritmo que verifica si la edad de una persona es mayor e igual a 18.

2.2.3. Estructuras condicionales (Decisión doble)

Un ejercicio de una estructura doble, puede ser este ejemplo. Un algoritmo que pide la calificación de un estudiante y determina si está Aprobado (nota ≥ 70) o Reprobado (nota < 70).

Algoritmo para su revisión en pseudocódigo:

Algoritmo 2.6: Pseudocódigo PSeInt: Evaluación de calificación

```
1  Algoritmo Evaluar_Aprobacion
2     // Declarar variables
3     Definir calificacion Como Entero
4     // Lectura de datos
5     Escribir "Introduce la nota del estudiante (0-100):"
6     Leer calificacion
7     // Estructura Selectiva Doble (Si-Entonces-Sino)
8     Si calificacion >= 70 Entonces
9        // Bloque VERDADERO
10       Escribir "Felicidades, el estudiante ha APROBADO."
11    Sino
12       // Bloque FALSO
13       Escribir "El estudiante ha REPROBADO por eso debe repetir."
14    FinSi
15    Escribir "Fin."
16 FinAlgoritmo
```

Es indiscutible que con la llegada de la estructura condicional, en la línea 8, se presenta el uso de operadores de relación. Los bloques de código que podemos obtener dependerán si el resultado de la condición es VERDADERO, o es FALSO. Así, la lógica del flujo de los datos irá dependiendo del resultado de evaluar la decisión. Si es mayor e igual a 70, entonces la respuesta por verdadero realizará el conjunto de instrucciones por verdadero. En caso contrario, si es falso, se ejecutará el conjunto de instrucciones o bloque por falso. Para el diagrama de flujo, la decisión toma una forma de rombo, las entradas y salidas de forma de paralelogramo (con flecha de ingreso o de salida a la figura respectivamente) y los procesos con un rectángulo. A continuación, se presenta el algoritmo en su representación de diagrama de flujo, ver Fig. 2.51.

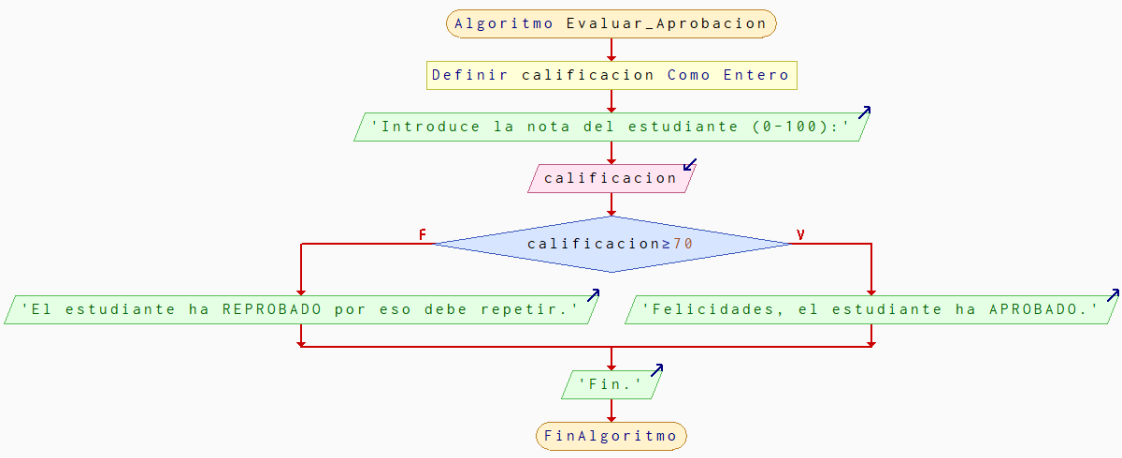


Figura 2.51: Algoritmo que verifica si la calificación es mayor e igual a 70.

Otro ejercicio para ejemplificar un algoritmo con estructura doble se presenta a continuación. Se solicita un algoritmo que verifique si un número es par o impar, para eso ingresamos un número por teclado. Luego utilizamos una operación matemática para determinar el residuo de un número.

Algoritmo para su revisión en pseudocódigo:

Algoritmo 2.7: Pseudocódigo PSeInt: Detección de valor par o impar

```
Algoritmo Verificar_Par_Impar
   // Declarar variables
   Definir numero Como Entero
   // Lectura de datos
   Escribir "Introduce un valor:"
   Leer numero
   // Estructura Selectiva Doble (Si-Entonces-Sino)
   // Usamos el operador MOD (residuo) para obtener
   // el residuo de dividir por 2.
   // El residuo al ser 0, el valor es par.
   Si (numero MOD 2) = 0 Entonces
      // Bloque VERDADERO: Residuo es 0
      Escribir "El valor ", numero, " es PAR."
   Sino
      // Bloque FALSO: Residuo es 1
      Escribir "El valor ", numero, " es IMPAR."
   FinSi
   Escribir "Verificado."
FinAlgoritmo
```

Estructura Selectiva

En el pseudocódigo se puede observar, la estructura selectiva doble, con el uso de la condición. Esta estructura tiene dos bloques, uno por verdadero y otro por falso. En la condición se usa el operador MOD que es el valor del residuo. Si el residuo es cero significa que el dividendo es divisible para el divisor (valor de 2) y por esta razón el número (dividendo) al dividir para 2 y su residuo al ser cero podemos entonces decir que el número es par. A continuación, se presenta el algoritmo en su representación de diagrama de flujo, ver Fig. 2.52.

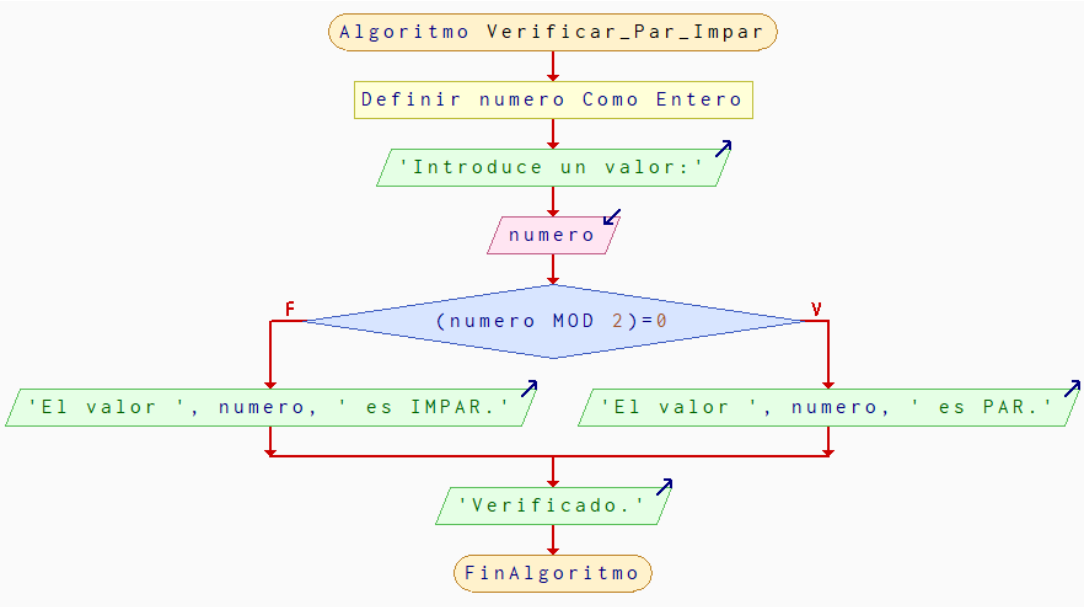


Figura 2.52: Algoritmo que verifica si un número ingresado es par o impar.

A continuación se presenta un problema para ejemplificar a una estructura selectiva anidada. En este ejercicio el uso de rangos es determinante.

Clasificación de edad usando estructuras de control, selectiva y anidada

Elabore un algoritmo que permita el ingreso de un valor entero que represente la edad de una persona. Si se ingresa un valor menor a cero o mayor a 100, se muestra un mensaje de “Incorrecto”. Si se ingresa un valor igual a cero se muestra un mensaje de “Nene” Si se ingresa un valor mayor a cero y menor a doce se muestra un mensaje de “Es infante/a” Si se ingresa un valor mayor igual a doce y menor a dieciocho se muestra un mensaje de “Es joven” Si se ingresa un valor mayor igual a dieciocho y menor a sesenta se muestra un mensaje de “Es adulto” Si se ingresa un valor mayor igual a sesenta y menor igual a cien se muestra un mensaje de “Es adulto mayor”

La primera solución, siguiendo tal como indica el problema, con una situación de no considerar optimización, y posible redundancia, sería:

Algoritmo 2.8: Pseudocódigo PSeInt: Clasificación de edad con anidamiento simple y doble

```
Algoritmo Clasificar_Edad_Anidada
   // Declarar variables
   Definir edad Como Entero
   // Lectura de datos
   Escribir "Introduce un valor que represente la edad:"
   Leer edad
   // Estructura anidada principal para manejar todos los casos
   Si edad < 0 O edad > 100 Entonces
      // 1. Requisito de Error (Rango principal, condicionas el error)
      Escribir "ERROR: Valor Incorrecto. La edad debe estar de 0 a 100."
   Sino
      // 2. Al ser la edad verdadera, clasificamos
      // Estructura Selectiva SIMPLE
      Si edad = 0 Entonces
         Escribir "Es un nene"
      Sino
         // 3. Clasificamos
         // Estructura Selectiva DOBLE pero puede ser SIMPLE
         Si edad > 0 Y edad < 12 Entonces
         // Implica edad > 0, no es necesario
            Escribir "Es infante/a"
         Sino
            // 4. Estructura Selectiva DOBLE
            Si edad >= 12 Y edad < 18 Entonces
               Escribir "Es joven"
            Sino
               // 5. Estructura Selectiva DOBLE
               Si edad >= 18 Y edad < 60 Entonces
                  Escribir "Es adulto"
               Sino
                  // 6. Caso final para (60 a 100)
                  // Estructura Selectiva DOBRE solo con VERDADERO
                  Si edad >= 60 Y edad <= 100 Entonces
                     Escribir "Es adulto mayor"
                  FinSi // Fin del Si 6
               FinSi // Fin del Si 5
            FinSi // Fin del Si 4
         FinSi // Fin del Si 3
      FinSi // Fin del Si 2
   FinSi // Fin del Si 1 (principal)
   Escribir "Fin del programa."
FinAlgoritmo
```

De acuerdo con el algoritmo anterior es visible como en la línea 33 (el último sí) es notable que solo queda la alternativa por verdadero, además en la línea 19 puede no usarse una condición, ya que se sobreentiende o ya se implica.

A tener en cuenta en la clasificación de edad usando estructuras de control

Problema: Elabore un algoritmo que permita el ingreso de un valor entero que represente la edad de una persona. Si se ingresa un valor menor a cero o mayor a 100, se muestra un mensaje de "Incorrecto". Si se ingresa un valor igual a cero se muestra un mensaje de "Nene" Si se ingresa un valor mayor a cero y menor a doce se muestra un mensaje de "Es infante/a" Si se ingresa un valor mayor igual a doce y menor a dieciocho se muestra un mensaje de "Es joven" Si se ingresa un valor mayor igual a dieciocho y menor a sesenta se muestra un mensaje de "Es adulto" Si se ingresa un valor mayor igual a sesenta y menor igual a cien se muestra un mensaje de "Es adulto mayor"
Variable de entrada:

- **Edad (E o edad)**: Variable de valor entero que representa la edad de una persona.

Consideración de rangos a usar ver Tabla 2.3:

Tabla 2.3: Rangos de edad y clasificación

Rango de edad	Condición lógica	Mensaje de salida
Inválido	$E < 0 \vee E > 100$	**Incorrecto**
Cero	$E = 0$	**Bebé / Nene**
Infantil	$E > 0 \wedge E < 12$	**Es niño/a | infante/a**
Adolescente	$12 \leq E < 18$	**Es joven**
Adulto	$18 \leq E < 60$	**Es adulto**
Adulto Mayor	$60 \leq E \leq 100$	**Es adulto mayor**

Prueba de escritorio ver Tabla 2.4:

Tabla 2.4: Valores para los rangos de edad y clasificación

Valor a utilizar	$E < 0 \vee E > 100$	$E = 0$	$E > 0 \wedge E < 12$	Mensaje de salida
200	Verdadero			**Incorrecto**
Cero		Verdadero		**Bebé / Nene**
5			Verdadero	**Es niño/a | infante/a**

En el diagrama de flujo, se muestra cómo con el uso de varias figuras geométricas sencillas, entre ellas, los paralelogramos, permiten ser útiles para los mensajes y variables de entrada. Por ejemplo, se muestra a continuación, cómo se solicita la variable edad, utilizando un paralelogramo con flecha hacia dentro y de color rosa. Mientras que los paralelogramos, que se utilizan para los mensajes, tienen la flecha hacia afuera y son de color verde. Además, todas las decisiones se concentran en rombos, de color azul. Estos rombos presentan el flujo que deben seguir los datos, si el resultado de la decisión es verdadero (V) o es (F). Es decir, que si una estructura es anidada, significa que en una de las opciones (V o F) se vuelve a colocar una estructura similar a la anterior. Así, se van anidando estructuras hasta llegar a un valor final.

El uso de estructuras anidadas debe usarse con cuidado, tratando de seguir un patrón de lógica para colocar los operadores de decisión. Si se observa la siguiente imagen, podrás notar el uso de los operadores de relación mayores y menores, así como el uso del igual en ellos y sin ellos. Además, puedes notar que se adjuntan con ellos los operadores lógicos, tanto el Y como el O, y que su uso mejora la comprensión de los rangos. El uso práctico de las decisiones, permiten mantener una lógica comprensiva de clasificar un número, yendo de un número menor a otro mayor, así como usar la recta numérica para establecer límites, y si estos se incluyen o no. Finalmente, es recomendable dibujar, estableciendo un color a los límites, el límite con un punto oscuro (si se incluye el número) y un punto sin color (si no se incluye), de acuerdo con esto podrás saber si se usa un operador de relación u otro.

A continuación, se presenta el algoritmo en su representación de diagrama de flujo, clasifica la edad según el rango y la ubica donde pertenece, a continuación presenta el mensaje correspondiente, ver Fig. 2.53.

Algoritmo para su revisión en pseudocódigo:

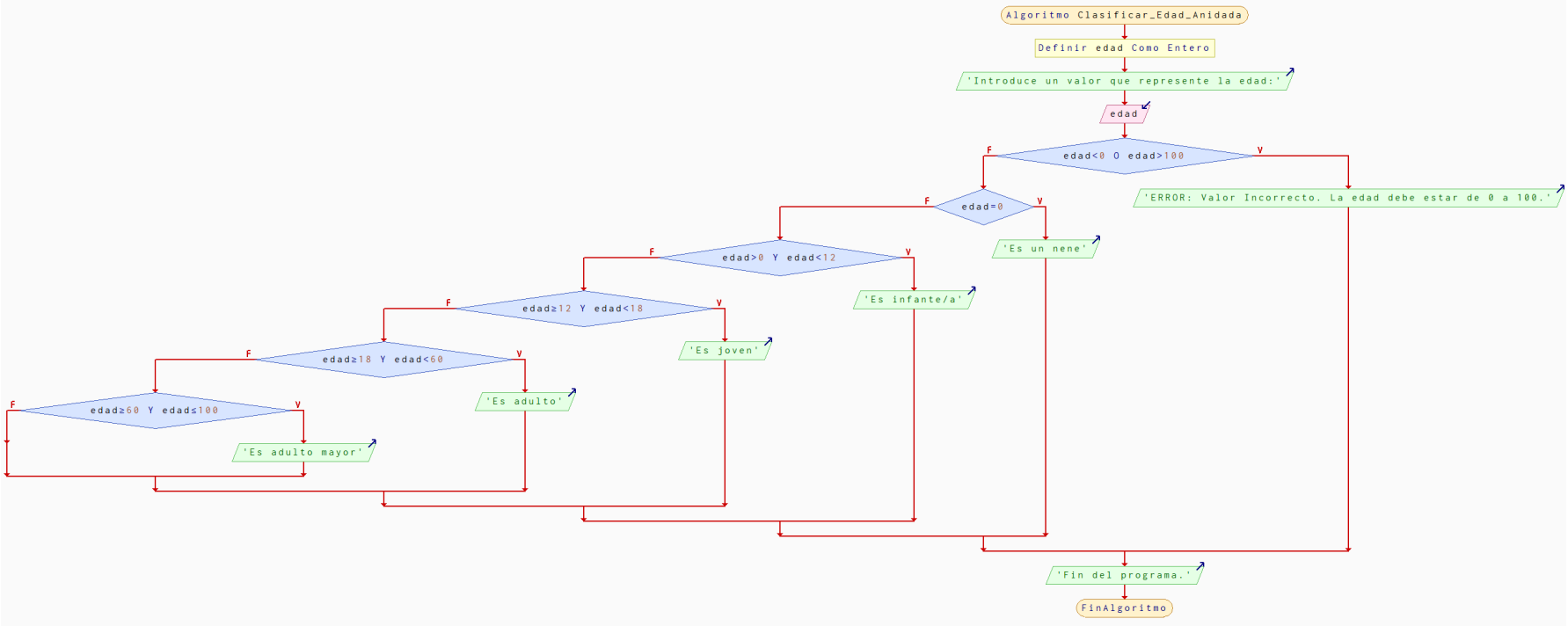


Figura 2.53: Algoritmo que clasifica la edad según el rango.

En la imagen anterior debe observarse que el autor ha decidido anidar las estructuras por el valor falso de la decisión. Y emitir el mensaje por verdadero. También es visible que en las decisiones, las expresiones de la derecha del operador lógico, son aquellos que se encuentran de lado izquierdo en la recta numérica, mientras los que están de lado derecho, son aquellos valores de mayor valor.

Una segunda versión mejorando su escritura.

Algoritmo para su análisis, revisión y comparación:

Algoritmo 2.9: Pseudocódigo PSeInt: Clasificación de etapas de la vida (Selectiva Múltiple)

```
Algoritmo Clasificar_Edad_Anidada
   // Declarar variables
   Definir edad Como Entero
   // Lectura de datos
   Escribir "Introduce un valor que represente la edad:"
   Leer edad
   // Estructura Selectiva Multi (Si-SinoSi)
   Si edad < 0 O edad > 100 Entonces
      // 1. Error
      Escribir "ERROR: Valor Incorrecto. La edad debe estar de 0 a 100."
   Sino Si edad = 0 Entonces
      // 2. Nene
      Escribir "Nene"
   Sino Si edad > 0 Y edad < 12 Entonces
      // 3. Infante/a
      Escribir "Es infante/a"
   Sino Si edad >= 12 Y edad < 18 Entonces
      // 4. Joven
      Escribir "Es joven"
   Sino Si edad >= 18 Y edad < 60 Entonces
      // 5. Adulto
      Escribir "Es adulto"
   Sino // Esto cubre el rango final identificado (>= 60 y <= 100)
      // 6. Adulto Mayor
      Escribir "Es adulto mayor"
   FinSi
   Escribir "Fin del programa."
FinAlgoritmo
```

Ahora te presentamos una tercera versión mejorada optimizando las condiciones implicando o sobreentendiendo rangos.

Algoritmo para su análisis, revisión y comparación:

Algoritmo 2.10: Pseudocódigo PSeInt: Clasificación de etapas con Si-Sino Si (Mejor para rangos)

```
Algoritmo Clasificar_Edad_SiSinoSi
   // Declarar variables
   Definir edad Como Entero
   // Lectura de datos
   Escribir "Introduce un valor que represente la edad:"
   Leer edad
   // Selectiva Multi de Rango (Evaluar varias condiciones)
   Si edad < 0 O edad > 100 Entonces
      // 1. Se verifica el Error (Invar)
      Escribir "ERROR: Valor Incorrecto. La edad debe estar entre 0 y 100."
   Sino Si edad = 0 Entonces
      // 2. Caso especial
      Escribir "Nene"
   Sino Si edad < 12 Entonces // Implica: edad > 0
      // 3. Rango
      Escribir "Es infante/a"
   Sino Si edad < 18 Entonces // Implica: edad >= 12
      // 4. Rango
```

```
      Escribir "Es joven"
   Sino Si edad < 60 Entonces // Implica: edad >= 18
      // 5. Rango
      Escribir "Es adulto"
   Sino // Implica: edad >= 60 Y edad <= 100
      // 6. Rango Final
      Escribir "Es adulto mayor"
   FinSi
   Escribir "Fin del programa."
FinAlgoritmo
```

Finalmente, la estructura selectiva múltiple con "Según" es menos recomendada para rangos. El comando Según (switch) en PSeInt está diseñado para manejar un número limitado de valores exactos (discretos), no rangos. Para clasificar por rangos, tendríamos que usar un arreglo o funciones complejas o, como se mostró, anidar el "Según" dentro de un Si-Sino Si, lo que lo vuelve menos eficiente que la solución anterior. Por lo tanto, la forma didácticamente correcta de mostrar "Según" es clasificando previamente la edad en una variable que represente la etapa, o usándolo para un problema distinto. Para el problema de la edad, la única forma de implementarlo directamente es listar cada valor discreto dentro de cada rango, lo cual sería impracticable (listando edades hasta 100 casos: $12, 13, 14, ..., 57, 58, ...$).

Vamos profundizando

Utiliza Si-Sino Si para la mayoría de los problemas que involucran rangos numéricos, como edades, notas o precios. Utiliza "Según" cuando la variable tiene un conjunto limitado y conocido de valores exactos (ej: día de la semana, opción de menú, código de estado).

Un ejemplo usando "Según" donde se presentan los días de la semana según el ingreso numérico, valor entero, correspondiente al día.

Algoritmo 2.11: Pseudocódigo PSeInt: Uso de la estructura Según

```
Algoritmo Mostrar_Dia_Semana
   // Declarar variables
   Definir dia Como Entero
   Escribir "Introduce un valor del 1 al 7:"
   Leer dia
   // Selectiva Multi por valor discreto
   Segun dia Hacer
      1: Escribir "Monday"
      2: Escribir "Tuesday"
      3: Escribir "Wednesday"
      4: Escribir "Thursday"
      5: Escribir "Friday"
      6: Escribir "Saturday"
      7: Escribir "Sunday"
      De Otro Modo:
         Escribir "Valor incorrecto."
   FinSegun
FinAlgoritmo
```

Ahora se presenta un algoritmo que simula un cajero automático simple, donde la acción (depositar, retirar, consultar) depende de la opción exacta que ingrese el usuario, utilizando para ellos un número entero del 1 al 4.

Algoritmo 2.12: Pseudocódigo PSeInt: Menú de opciones con la estructura Según

```
Algoritmo Cajero_Automatico
```

```
    // Declarar variables
    Definir opcion Como Entero
    Definir saldo Como Real
    // Inicializar saldo
    saldo <- 1000.00
    // Mostrar servicios, elegir una
    Escribir "--- CAJERO AUTOMATIC ---"
    Escribir "1. Consultar Saldo"
    Escribir "2. Depositar"
    Escribir "3. Retirar"
    Escribir "4. Salir"
    Escribir "------------------------"
    Escribir "Seleccione una (1-4):"
    Leer opcion
    // Estructura Selectiva Multi:
    Segun opcion Hacer
       1:
          Escribir "Su saldo actual es: ", saldo
       2:
          Escribir "Elegiste 'Depositar' seleccionada."
          // Se ejecuta la idea de depositar
       3:
          Escribir "Elegiste 'Retirar' seleccionada."
          // Se ejecutar la idea de retiro
       4:
          Escribir "Gracias por usar nuestro cajero. Vuelva pronto"
       De Otro Modo:
          // Caso por defecto (cualquier valor)
          Escribir "ERROR. Intente nuevamente."
    FinSegun
FinAlgoritmo
```

Se presenta la tabla de los operadores matemáticos, relacionales y lógicos, comunes y útiles, para el uso en el desarrollo de los algoritmos. Se presenta en el contenido de la tabla, el operador, el nombre, un ejemplo y su resultado o interpretación, ver Tabla 2.5.

Tabla 2.5: Operadores aritméticos, de relación y lógicos

Tipo	Operador	Nombre	Ejemplo	Resultado / Interpretación
Aritmético	+	Suma	8 + 5	13
	-	Resta	20 - 7	13
	*	Multiplicación	6 * 9	54
	/	División (real)	15 / 4	3.75
	% ó MOD	Módulo (resto)	15 mod 4	3
	^	Potencia	5^3	125
Relacional	==	Igual que	$\mathbf{edad == 18}$	Verdadero
	<>	Diferente de	$\mathbf{nota <> 0}$	Verdadero
	<	Menor que	$\mathbf{x < 10}$	Verdadero si x es 5
	>	Mayor que	$\mathbf{y > 100}$	Verdadero si y es 150
	<=	Menor o igual	puntaje <= 60	Verdadero si puntaje es 60
	>=	Mayor o igual	edad >= 18	Si es mayor e igual a 18
Lógico	& ó Y	Y lógico (AND)	edad >= 18 Y edad <= 65	Solo si ambas son ciertas
	| ó O	O lógico (OR)	dia = 6 O dia = 7	Verdadero si es fin de semana
	~ ó NO	Negación (NOT)	$\mathbf{no(x < 0)}$	Verdad si x es positivo o cero

Cada uno de los operadores matemáticos, cumple con su función, es decir, matemáticamente

el resultado a obtener será el que dependiendo del tipo de dato (entero o real) que se ingrese y con esto, el valor se presentará en pantalla. Asimismo, los operadores relacionales, evalúan el valor o las expresiones que se encuentran de lado izquierdo y derecho de ellos. Finalmente, los operadores lógicos, o conocidos como booleanos, permiten determinar el estado lógico de verdadero o falso según el resultado final de la expresión. Dependiendo del programa o entorno de escritura, del algoritmo o ambiente de desarrollo, se utiliza el operador de acuerdo con su sintaxis.

Gracias a estos operadores, un bloque de código o una parte del algoritmo puede escribirse de diversas maneras para alcanzar el mismo resultado. La combinación de valores límite, operadores de relación y operadores lógicos, junto con la aplicación de las leyes de la teoría de proposiciones y el álgebra de Boole, nos permite una extensa combinación de estructuras lógicas equivalentes, ver Tabla 2.6.

Soluciones lógicas equivalentes en el rango [0, 24]

La misma verificación puede ser escrita de diez maneras diferentes:

Tabla 2.6: Todos los códigos son lógicamente equivalentes y producen el mismo resultado.

Soluciones directas (validación ✓)		Soluciones con negación (Lógica inversa)	
1.	Si (hora >= 0) y (hora <= 24) Escribir (“Buen ingreso”) Sino Escribir (“Mal ingreso”)	**6.**	Si (hora <0) ó (hora >24) Escribir (“Mal ingreso”) Sino Escribir (“Buen ingreso”)
2.	Si (hora >-1) y (hora <25) Escribir (“Buen ingreso”) Sino Escribir (“Mal ingreso”)	**7.**	Si negado ((hora >=0) y (hora <=24)) Escribir (“Mal ingreso ”) **Sino** Escribir (“Buen ingreso”)
3.	Si negado ((hora >-1) y (hora <25)) Escribir (“Mal ingreso”) Sino Escribir (“Buen ingreso”)	**8.**	Si negado ((hora <0) ó (hora >24)) Escribir (“Buen ingreso”) Sino Escribir (“Mal ingreso”)
4.	Si (negado (hora >-1) ó negado (hora <25)) Escribir (“Mal ingreso”) Sino Escribir (“Buen ingreso”)	**9.**	Si (negado (hora <0) y negado (hora >24)) Escribir (“Buen ingreso”) Sino Escribir (“Mal ingreso”)
5.	Si ((hora <= -1) ó (hora >= 25)) Escribir (“Mal ingreso”) Sino Escribir (“Buen ingreso”)	**10.**	Si (hora >0) y (hora <= 24) ó (hora = 0) Escribir (“Buen ingreso”) Sino Escribir (“Mal ingreso”)

Podemos observar que del conjunto de diez bloques de pseudocódigo, se demuestra la flexibilidad lógica para validar el rango de la variable hora entre $[0, 24]$, siendo todos lógicamente equivalentes. La validación directa, que se muestran en los bloques 1 y 2, representa la forma más legible comprobando si $\mathbf{0} \leq \text{hora} \leq \mathbf{24}$. El bloque 10 (la versión extendida $(\text{hora} > \mathbf{0} \wedge \text{hora} \leq \mathbf{24}) \vee (\text{hora} = \mathbf{0})$) es una alternativa prolija que produce el mismo resultado. La validación inversa, representada en los bloques 5 y 6, comprueba si la hora es inválida ($\text{hora} < \mathbf{0} \vee \text{hora} > \mathbf{24}$), invirtiendo la lógica de salida para que el bloque *“Sino”* ejecute el *“Buen ingreso”*. Finalmente, los bloques 3, 4, 7, 8, y 9 ilustran la aplicación de las *“Leyes de De Morgan”*, mostrando cómo la palabra clave negado y la inversión de los operadores ($\mathbf{y} \leftrightarrow$ ó) permiten expresar la misma condición de múltiples maneras, como se ve en el bloque 7 en $\neg((\text{hora} \geq \mathbf{0}) \wedge (\text{hora} \leq \mathbf{24}))$.

2.2.4. Estructuras repetitivas (Bucles)

Las estructuras repetitivas, de repetición o, también llamadas bucles o ciclos, son instrucciones que permiten ejecutar un bloque de código varias veces de forma automática, sin tener que escribirlo, varias veces lo mismo, o repetidamente. Son fundamentales para evitar código duplicado y para procesar listas, hacer conteos, validaciones, en otras palabras automatizar, ver Fig. 2.54.

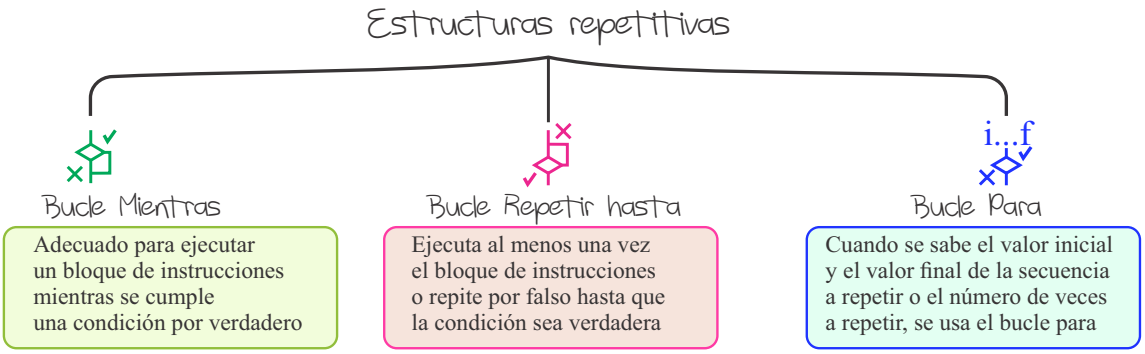


Figura 2.54: Estructuras repetitivas iniciales.

Bucle Mientras

Esta estructura ejecuta el bloque de instrucciones, mientras una condición sea verdadera. La condición se evalúa al principio del ciclo. Si la condición es falsa desde el inicio, las instrucciones no se ejecutan ni una sola vez. Se presenta a continuación el algoritmo para generar la tabla de multiplicar. Este algoritmo permite comprender la estructura de repetición *"Mientras"*.

Algoritmo 2.13: Pseudocódigo PSeInt: Generar tabla de multiplicar

```
Algoritmo Tabla_Multiplicar
   // Declarar variables
   Definir num, i Como Entero
   // Entrada de datos
   Escribir "Ingresa un valor:"
   Leer num
   // Inicializar el contador
   i <- 1
   // Ciclo se repite 10 veces
   Mientras i <= 10 Hacer
      // Muestra la secuencia de la tabla: num * i = resultado
      Escribir num, " x ", i, " = ", num * i
      // Incremento del contador (Control de ciclo)
      i <- i + 1
   FinMientras
   Escribir "Fin de la tabla."
FinAlgoritmo
```

La estructura mientras tiene dos variantes, una de ellas es el mientras controlado por un contador y otra variante es el mientras controlado por un centinela. La variable contador permite controlar las repeticiones, mientras van incrementando o decrementando, adicionan las iteraciones y llegan al valor esperado. En el diagrama de flujo siguiente, la variable i, es el contador, que es inicializado en 1 y llega al valor de 10 inclusive, por el menor e igual en la condición (figura del rombo), además se visualizan las tres partes de toda estructura repetitiva, la variable inicializada, la condición y el incremento (que se encuentra dentro del bloque de repetición, ver Fig. 2.55.

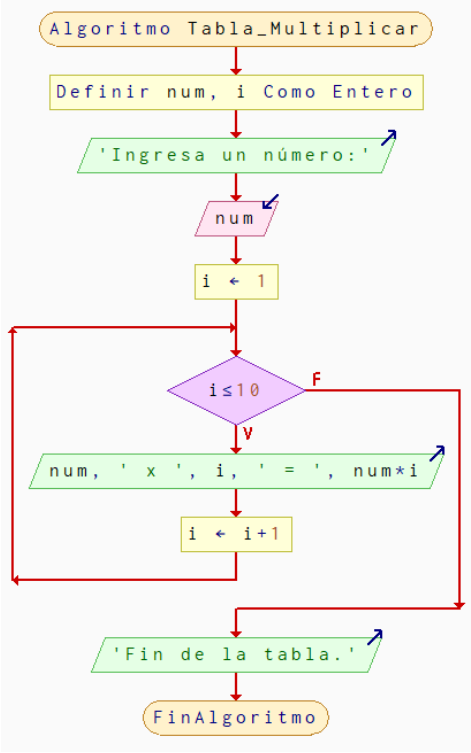


Figura 2.55: Algoritmo que genera una tabla de multiplicar.

Un ciclo Mientras controlado por centinela (o *sentinel-controlled loop*) es aquel que continúa ejecutándose hasta que se lee un valor específico, llamado el valor centinela, que indica el final de la entrada de datos. La idea de esta estructura es que debe la variable tener un valor que permita acceder a la estructura (lectura inicial o primario) esto se llama *"priming read"*. Luego las mismas líneas se colocan al final del bloque de repetición (lectura secundaria o del bloque), para la siguiente iteración.

Algoritmo 2.14: Pseudocódigo PSeInt: Suma controlada por centinela

```
Algoritmo Suma_Centinela
   // Declarar variables. Usamos REAL para permitir decimales.
   Definir numero, suma Como Real
   // El centinela es el valor quien detiene el ciclo.
   Definir CENTINELA Como Entero
   CENTINELA <- -1
   // Inicializar el acumulador de la suma
   suma <- 0
   // Lectura Primaria (Apertura): Lee el primer valor
   Escribir "Ingrese un valor (", CENTINELA, " para terminar):"
   Leer numero
   // Repite seguidamnete, en tanto el valor NO sea el centinela
   Mientras numero <> CENTINELA Hacer
      // 1. Proceso: Acumular el valor ingresado
      suma <- suma + numero
      // 2. Lectura Secundaria (Actualizar):
      // Lee el siguiente valor para controlar, condicionar e iterar
      Escribir "Ingrese nuevo valor (", CENTINELA, " para terminar):"
      Leer numero
   FinMientras
   // Muestra el resultado final (se ejecuta al ingresar el centinela)
```

```
22      Escribir "---"
23      Escribir "Se ha ingresado el centinela."
24      Escribir "La suma total es: ", suma
25  FinAlgoritmo
```

En el diagrama de flujo, se muestra el valor centinela de -1 (se detiene el algoritmo si se ingresa el número -1) mientras se ingresan números en la variable *"numero"*. Mientras no se ingrese el valor -1 (como centinela o valor diferente a todos los que se ingresan) continúa el ingreso de valores, ver Fig. 2.56.

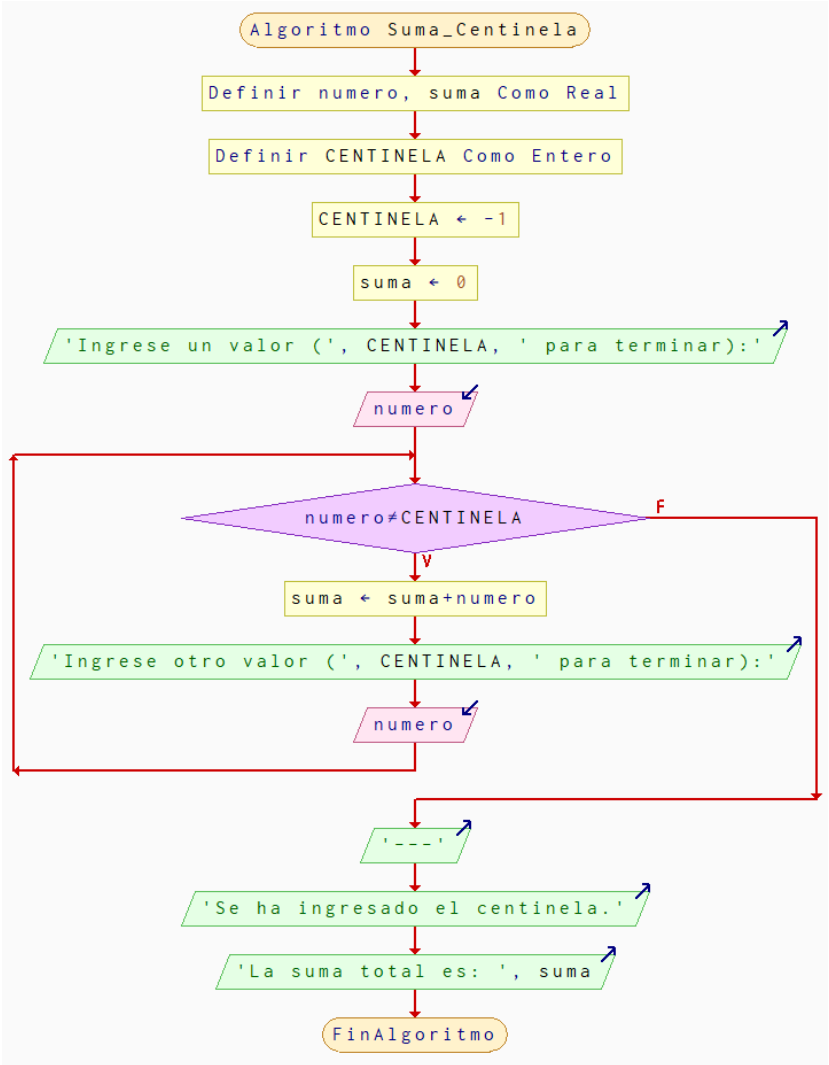


Figura 2.56: Algoritmo que genera una suma controlada por centinela.

Bucle Repetir-Hasta

El ciclo Repetir-Hasta (Repeat-Until) es otra estructura fundamental en pseudocódigo. A diferencia del ciclo Mientras, donde la condición se evalúa al inicio, el ciclo Repetir-Hasta siempre ejecuta el bloque de código al menos una vez antes de evaluar la condición al final. El ciclo continúa mientras la condición sea falsa, y se detiene hasta que la condición es verdadera. Esta estructura, en el lenguaje de programación en Java, por ejemplo, se la conoce como *"Hacer-mientras"* y repite por verdadero, cambiando su repetición por verdad (en vez de repetir por falso), en cambio, ya en el lenguaje de programación en Python, desaparece esta estructura.

Algoritmo 2.15: Pseudocódigo PSeInt: Validar un número en un rango (Repetir-Hasta)

```
Algoritmo Validacion_Rango
   // Declarar variables
   Definir numero Como Entero
   // El ciclo se ejecuta al menos una vez
   Repetir
      // Lectura de datos
      Escribir "Introduce un valor entre 1 - 10:"
      Leer numero
      // Verificar rango (Se muestra error cuando es incorrecto)
      Si (numero < 1) o (numero > 10) Entonces
         Escribir "ERROR El valor fuera del rango [1-10]."
      FinSi
   // El ciclo se repite siempre al ser falso (el valor fuera de rango)
   // Se detiene al ser verdadero
   // (el valor debe encuentrarse dentro del rango)
   Hasta Que (numero >= 1) y (numero <= 10)
   // Muestra el resultado final
   //(se ejecuta solo cuando el valor es correcto)
   Escribir "---"
   Escribir "Valor correcto ingresado: ", numero
FinAlgoritmo
```

En el diagrama de flujo se puede observar el uso de la estructura repetir-hasta (rombo morado) que repite por falso. Además, se utiliza la estructura de selección simple (si simple), para validar el valor de ingreso. Es interesante el uso combinado de las estructuras de selección y repetición como buenas prácticas en programación, observa además cómo el flujo, valor verdadero o falso se ubican en las estructuras, por ejemplo en la estructura de selección simple (rombo azul), el valor por falso está a la izquierda mientras que el valor por verdadero (y el que nos interesa) está a la derecha. En cambio, en la estructura de repetición el valor falso está a la izquierda (regresando hacia arriba) y el valor verdadero está abajo, siguiendo el flujo de arriba-abajo, ver Fig. 2.57.

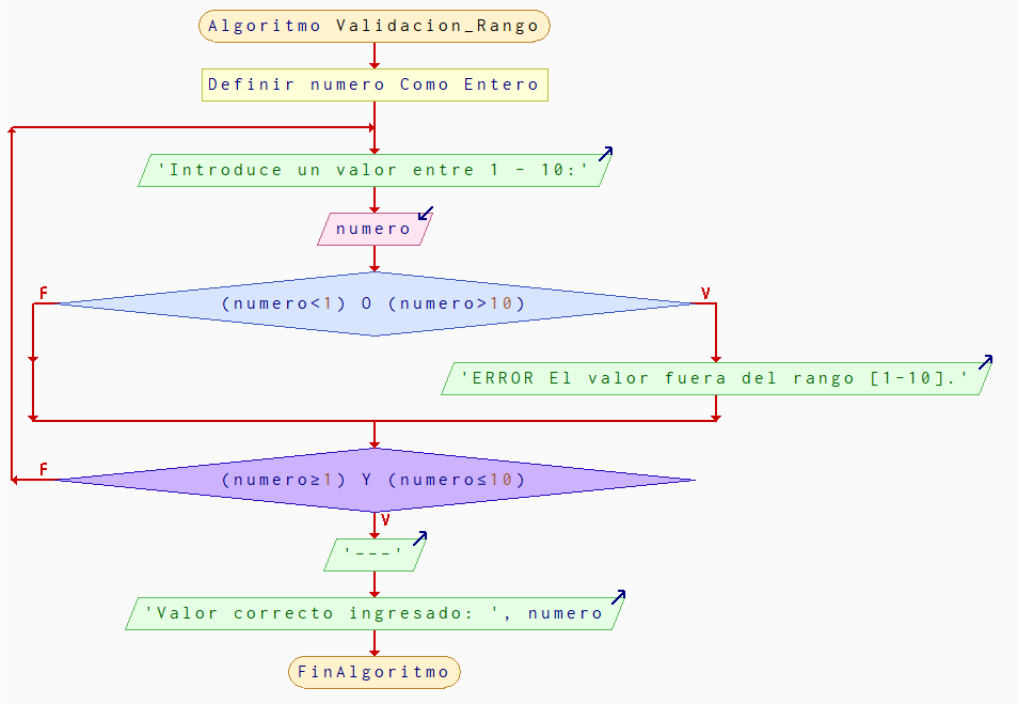


Figura 2.57: Algoritmo que utiliza la estructura mientras con un valor centinela.

Una práctica en programación es el uso de menús. Así como los menús de un local o restaurante, permiten primero leer las opciones existentes y luego escoger una de ellas. Se presenta un algoritmo donde se debe escoger una opción, a partir de un menú de opciones. Para esto el uso de la estructura Repetir-Hasta es de gran utilidad.

Algoritmo 2.16: Pseudocódigo PSeInt: Menú interactivo con Repetir-Hasta

```
1  Algoritmo Menu_Interactivo
2     // Declarar de variables
3     Definir opcion Como Entero
4     // Inicializar la variable opcion para asegurar la entrada inicial
5     // (aunque Repetir lo garantiza)
6     opcion <- 0
7     // Ciclo principal del algoritmo
8     Repetir
9        // 1. Mostrar opciones
10       Escribir "--- LISTA DE OPERACIONES ---"
11       Escribir "1. Sumar dos valores"
12       Escribir "2. Restar dos valores"
13       Escribir "3. Ver datos del programa"
14       Escribir "4. Salir"
15       Escribir "----------------------------"
16       // 2. Leer lo escogido del usuario
17       Escribir "Elige una alternativa (1-4):"
18       Leer opcion
19       // 3. Procesar la alternativa elegida
20       // (Estructura Selectiva Diversa)
21       Segun opcion Hacer
22          1:
23             Escribir "Has elegido Sumar."
24             // Algoritmo para sumar.
25          2:
26             Escribir "Has elegido Restar."
27             // Algoritmo para restar.
28          3:
29             Escribir "Programa de ejemplo usando Repetir-Hasta."
30          4:
31             Escribir "Saliendo del programa. Hasta luego"
32          De Otro Modo:
33             Escribir "Alternativa incorrecta Prueba de nuevo."
34       FinSegun
35       Escribir "" // Salto siguiente para limpiar la vista
36    // La alternativa se estima o analiza al final.
37    // el ciclo termina siempre y cuando
38    // la alternativa sea 4.
39    Hasta Que opcion = 4
40 FinAlgoritmo
```

Se puede observar que las opciones pueden escogerse entre los números del 1 al 4, donde 4 es salir. También es necesario comprender que las líneas 24 y 27, es el lugar apropiado para colocar el subprograma o todo el bloque necesario para otras pequeñas soluciones que, en los lenguajes de programación, pueden llamarse módulos. Así vamos observando el inicio de la modularidad que empieza con esta estructura que permite al programador seguir escalando y extendiendo posibilidades.

En el diagrama de flujo se muestran las figuras geométricas que son apropiadas para representar el problema, por ejemplo se observa cómo la estructura Repetir-Hasta (rombo morado) se encuentra al final, permitiendo que se muestre o presente, al menos una vez, todo el bloque interior y superior al rombo, ver Fig. 2.58.

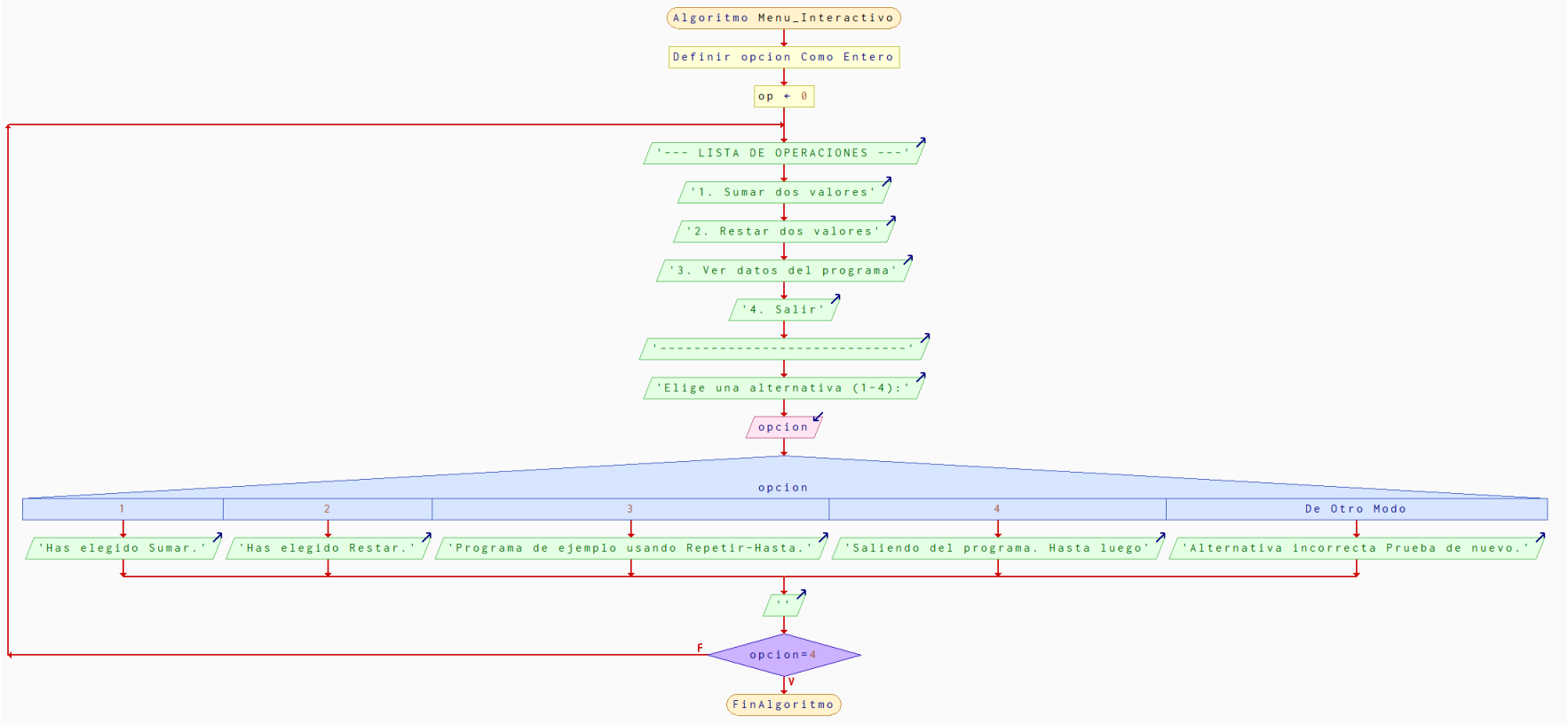


Figura 2.58: Diagrama de flujo que utiliza la estructura mientras con un valor centinela para opciones de menú.

Bucle Para

El bucle Para (o For en otros lenguajes) en pseudocódigo es una estructura de control repetitiva diseñada para ejecutar un bloque de instrucciones un número definido de veces. Es ideal usar esta estructura, cuando sabes de antemano cuántas repeticiones necesitas realizar. Se basa en una variable de control o contador que maneja el ciclo de la repetición. Además, es posible determinar el paso o el incremento (o decremento) adecuado.

Este ejemplo utiliza el bucle Para, para calcular e imprimir la tabla de multiplicar del 5, visualizando el proceso de multiplicación, desde el 1 hasta el 10.

Algoritmo 2.17: Pseudocódigo PSeInt: Tabla de multiplicar con el ciclo Para

```
Algoritmo TablaDelCinco
   // Definimos las variables a utilizar
   Definir num, resultado, i Como Entero;
   // El valor cuya tabla queremos calcular
   num <- 5
   Escribir "--- Tabla de Multiplicar del ", num, " ---"
   // Bucle Para:
   // i es el contador, inicia en 1, y va a 10, con paso de 1 (por defecto)
   Para i <- 1 Hasta 10 Hacer
      // Calculamos el resultado cuando itera
      resultado <- num * i
      // Imprimimos el proceso y el resultado
      Escribir num, " x ", i, " = ", resultado
   FinPara
FinAlgoritmo
```

A continuación, se muestra el diagrama de flujo que presenta la solución del problema. Es necesario observar que el dibujo de la estructura tiene forma de un círculo con tres áreas, donde se muestra la variable de control i, el valor inicial a la izquierda y el valor final a la derecha. Se sobreentiende que la repetición se la realiza por verdadero, y el bloque que está dentro de las líneas del dibujo circular son las instrucciones que se iteran, ver Fig. 2.59.

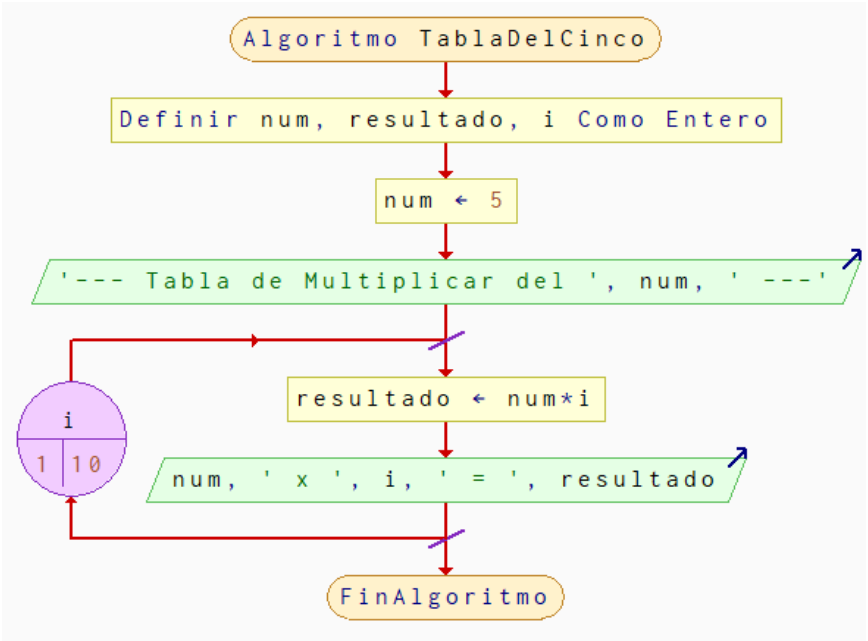


Figura 2.59: Diagrama de flujo que utiliza la estructura Para.

2.2.5. Tipos de variables (Contadores y acumuladores)

Cuando usamos estructuras repetitivas (bucles), siempre necesitamos variables especiales que nos ayuden a controlar el ciclo o a guardar resultados. Hay que poner nombres claros y apropiados, especialmente para compartir el código a otros programadores. Así el algoritmo puede sobrevivir en el tiempo. Entre las variables por su tipo de función encontramos los contadores, acumuladores, sumadores, iteradores de control, incrementadores, decrementadores y las banderas, todos ellos cumpliendo una acción de adicionar valores, en sumas, restas o acumular productos o cocientes con el mismo fin de avanzar o retroceder valores que hacen funcionar al algoritmo, ver Fig. 2.60.

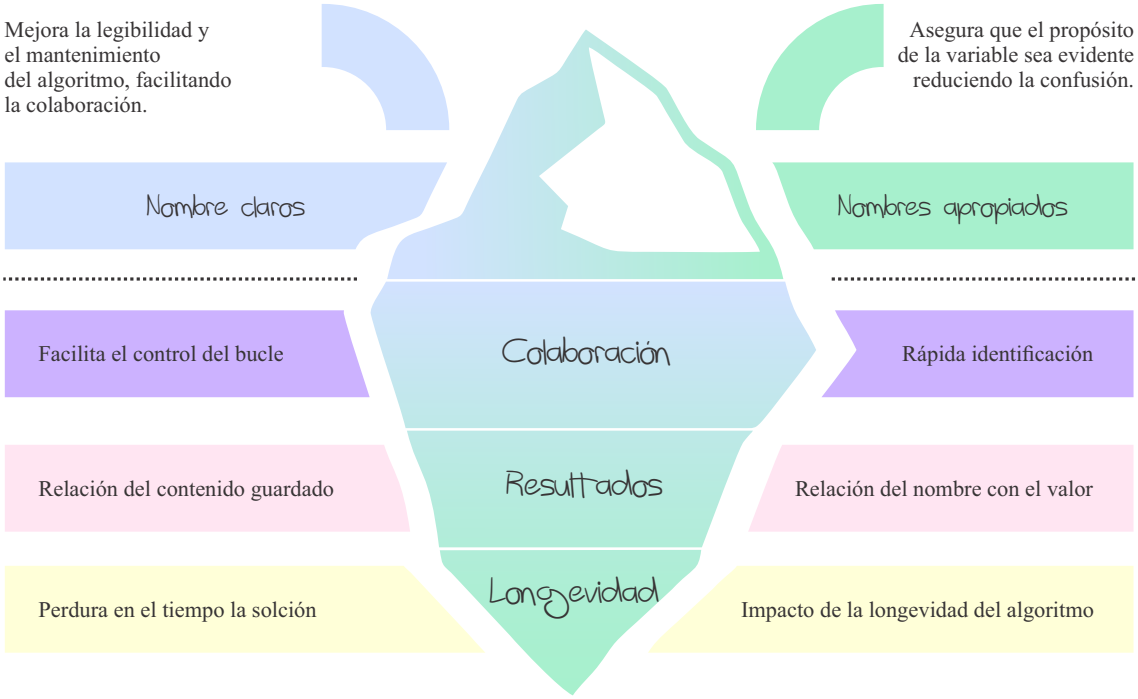


Figura 2.60: Las variables nos ayudan a controlar el ciclo o a guardar resultados.

Los contadores sirven para llevar la cuenta de cuántas veces ha pasado algo. Suelen empezar en 0 o en 1. Y aumentan de 1 en 1 (o de cualquier número). El acumulador (o sumador) sirve para ir sumando valores poco a poco (como una alcancía). Siempre se inicia en 0, cuando se acumulan sumas o restas, mientras se inicializan en 1, cuando se acumulan productos o cocientes.

Un iterador o variable de control de un ciclo, es la variable que usa el bucle (Para) automáticamente (casi siempre se llama i, j, k). Los incrementadores o decrementadores son instrucciones que aumentan o disminuyen el valor de una variable. Estas acciones pueden incrementar un valor de 1 en 1, de otro número o decrementar propiamente. En cambio, una variable bandera es una variable lógica (Verdadero o Falso) que sirve para recordar si algo pasó.

A continuación, se presenta un algoritmo donde se utilizan variables que de acuerdo con su función permiten, acumular, iterar, contar, guardar un valor mayor, ser un valor bandera, que hacen de ellas útiles para la ejecución de una solución o tarea.

Algoritmo 2.18: Pseudocódigo PSeInt: Ejemplo de todas las variables útiles en un algoritmo

```
1 Algoritmo EjemploCompleto
2    Definir numero, suma, mayor Como Entero
3    Definir contador, i Como Entero
4    Definir salir Como Logico
5    suma <- 0 // acumulador
```

```
6     contador <- 0 // contador
7     mayor <- -999999 // a guardar el mayor
8     salir <- Falso // bandera
9     Repetir
10       Escribir "Dame un valor (0 destinado a terminar): "
11       Leer numero
12       Si numero <> 0 Entonces
13          contador <- contador + 1 // contador
14          suma <- suma + numero // acumulador
15          Si numero > mayor Entonces
16             mayor <- numero // guardamos el mayor
17          FinSi
18       Sino
19          salir <- Verdadero // activamos la bandera
20       FinSi
21    Hasta Que salir = Verdadero
22    Escribir "Ingresaste ", contador, " valores"
23    Escribir "La suma total es: ", suma
24    Escribir "El promedio es: ", suma/contador
25    Escribir "El mayor valor fue: ", mayor
26 FinAlgoritmo
```

Las variables de tipo contador y los acumuladores siempre deben inicializarse en un valor conocido (generalmente cero cuando van a incrementar o decrementar, pero en 1 si acumulan productos o cocientes) antes de ingresar a un bucle. Estas variables de acuerdo con su función, su ubicación, su almacenamiento y tipo de valor a guardar son fundamentales para gestionar las repeticiones o resultados lógicos que requiere el algoritmo, ver Tabla 2.7.

Tipos de variables de control de acuerdo a su función

Estas variables son fundamentales para gestionar bucles, totales y estados lógicos.

Tabla 2.7: Clasificación y función de las variables de control.

Nombre	¿Para qué sirve?	Valor inicial	Ejemplo
Contador	Contar cuántas veces ocurre un evento o se ejecuta un bloque.	0 o 1	`conta <- conta + 1`
Acumulador	Sumar o totalizar una serie de valores (montos, notas, etc.).	0 ó 1	`suma <- suma + nota` `acum <- acum * 3`
Iterador (i, j, k)	Variable de control que lleva la cuenta en un ciclo (**Para**).	Lo define el ciclo.	`i toma valores 1, 2, 3...`
Incrementador	Aumentar una variable por una cantidad fija o variable.	-	`edad <- edad + 1`
Decrementador	Disminuir una variable por una cantidad fija o variable.	-	`vidas <- vidas - 1`
Bandera	Recordar si una condición o evento ocurrió (estado Sí o No).	Falso	`ubica <- Verdadero`

El diagrama de flujo permite la representación del algoritmo y el uso importante de las variables, utilizando adecuadamente nombres apropiados y claros para su comprensión, ver Fig. 2.61.

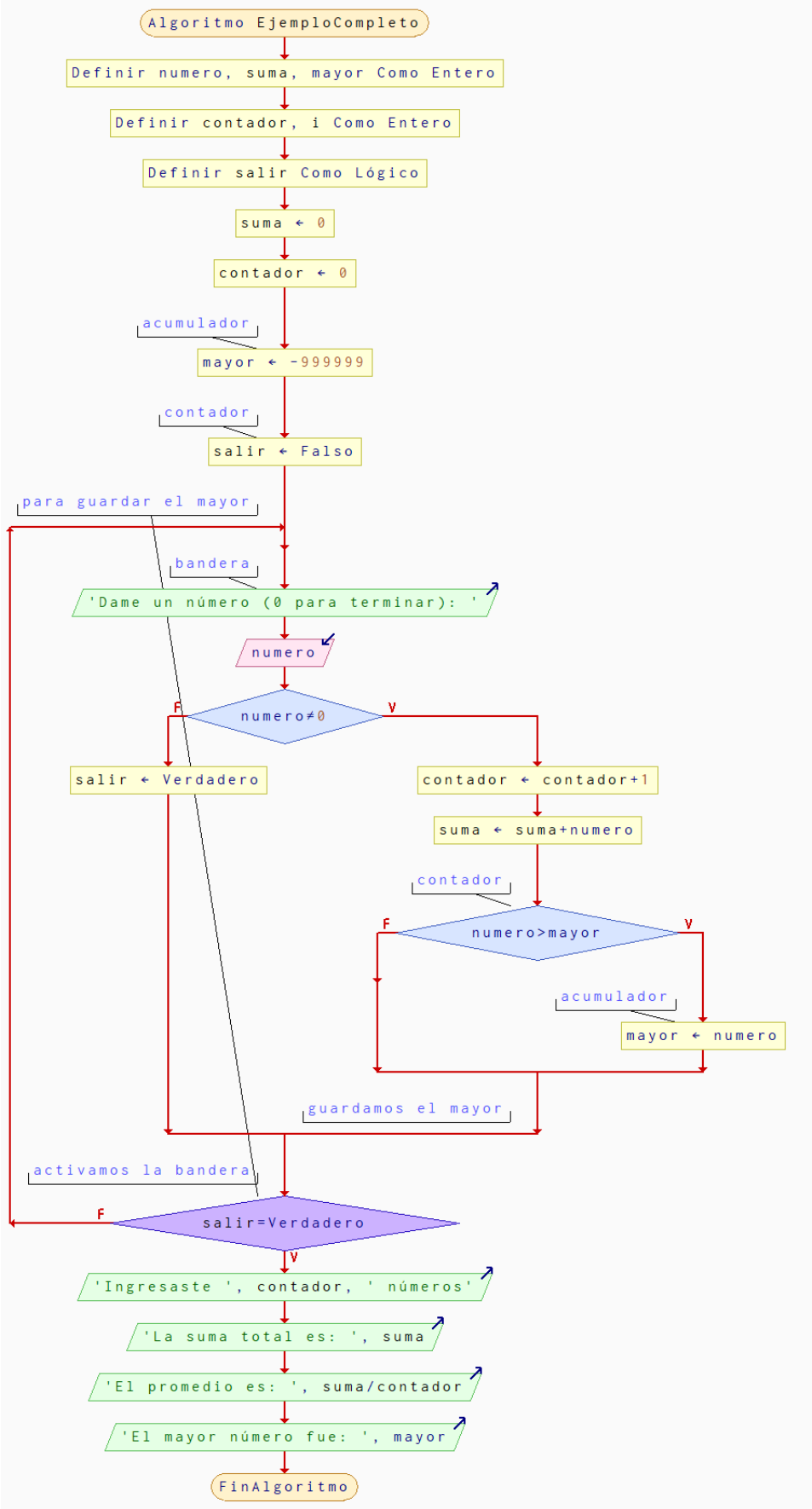


Figura 2.61: Uso de diferentes tipos de variables para controlar el ciclo y guardar resultados.



Laboratorio 3

Capítulo 2

2.3. Ejercicios del Laboratorio 3

* Ejercicio 1

Escribe un algoritmo que nos diga cuál es el volumen de un cono con un radio de la base de 14 y una altura de 26. La fórmula que debes usar es: $V = \frac{1}{3} \cdot \pi \cdot r^2 \cdot h$. Recuerda que el valor (aproximado) de PI es 3.141592. Para este ejercicio utilizaremos los valores fijos proporcionados, asignando al radio con nombre r = 14, altura h = 26 y $\pi \approx 3.141592$.

Algoritmo 2.19: Pseudocódigo PSeInt: Cálculo del volumen de un cono

```
Algoritmo Calcular_Volumen_Cono
   // Declarar variables
   Definir radio, altura Como Entero
   Definir pi, volumen Como Real
   // 1. Asignar valores fijos (Datos del problema)
   radio <- 14
   altura <- 26
   pi <- 3.141592
   // 2. Proceso: V = (1/3) * pi * radio^2 * altura
   // El 1/3 se implementa mediante el cociente por 3 al final.
   // Calcular radio al cuadrado (radio * radio)
   volumen <- pi * (radio * radio) * altura
   // Dividir el resultado por 3 con el fin de obtener el volumen final
   volumen <- volumen / 3
   // 3. Salida de resultados
   Escribir "--- Volumen de un Cono ---"
   Escribir "Radio (r): ", radio
   Escribir "Altura (h): ", altura
   Escribir "El volumen (V) del cono es: ", volumen
FinAlgoritmo
```

* Ejercicio 2

Modifica el algoritmo anterior para que use tres variables, todas de tipo real: radio, altura y volumen. Las dos primeras se inicializarán a 14.5 y 26.79 respectivamente. La tercera obtendrá el resultado de la fórmula.

Algoritmo 2.20: Pseudocódigo PSeInt: Cálculo de volumen de un cono con inicialización de variables

```
Algoritmo Calcular_Volumen_Cono_Modificado
   // Declarar las variables de tipo Real (equivalente a double)
   Definir radio, altura, volumen, PI Como Real
   // 1. Inicializar variables (Datos de entrada fijos)
   radio <- 14.5
   altura <- 26.79
   PI <- 3.141592
   // 2. Proceso: V = (1/3) * PI * radio^2 * altura
   // Calculamos el volumen sin pedir datos al usuario,
   // solo usando las variables inicializadas.
   // Procedimiento: PI * (radio * radio) * altura / 3
   volumen <- (PI * radio * radio * altura) / 3
   // 3. Salida de resultados
   Escribir "--- Volumen de un Cono ---"
   Escribir "Radio (r) inicializado: ", radio
   Escribir "Altura (h) inicializada: ", altura
   Escribir "El volumen (V) del cono es: ", volumen
FinAlgoritmo
```

* Ejercicio 3

Escribe un algoritmo que lea del teclado un número (real) de grados Fahrenheit y lo convierta a Celsius mostrando el resultado en la pantalla. La fórmula para convertir grados Fahrenheit (F) a grados Celsius (C) es:

$$C = (F - 32) \cdot \frac{5}{9}$$

Algoritmo 2.21: Pseudocódigo PSeInt: Conversión de temperatura

```
Algoritmo Fahrenheit_a_Celsius
   // Declarar variables de tipo Real (para manejar decimales)
   Definir fahrenheit, celsius Como Real
   // Lectura de datos
   Escribir "Introduce la temperatura en grados Fahrenheit (F):"
   Leer fahrenheit
   // Proceso: C = (F - 32) * (5/9)
   // Nota: Es fundamental usar 5.0/9.0 o (5/9.0)
   // para asegurar el cociente real
   // y evitar que PSeInt realice un cociente entero (5 div 9 = 0).
   celsius <- (fahrenheit - 32) * (5.0 / 9.0)
   // Salida de resultados
   Escribir "------------------------------------"
   Escribir "La temperatura en grados Fahrenheit es: ", fahrenheit, " F"
   Escribir "Equivale a ", celsius, " C"
FinAlgoritmo
```

* Ejercicio 4

Escribe un algoritmo que lea del teclado un tiempo transcurrido en segundos y muestre en la pantalla las horas, los minutos y los segundos equivalentes.

Algoritmo 2.22: Pseudocódigo PSeInt: Segundos a horas y minutos

```
Algoritmo Conversor_Tiempo
   // Declarar variables.
   Definir total_segundos, segundos, minutos, horas Como Entero
   // Lectura de datos
   Escribir "Introduce el tiempo transcurrido en segundos:"
   Leer total_segundos
   // 1. Encontrar Horas. Hay 3600 segundos en una hora (60 * 60).
   // El cociente nos da la cantidad de horas completas.
   horas <- total_segundos / 3600
   // 2. Minutos, Segundos restantes
   // El operador (%) nos da los segundos sobrantes luego de las horas.
   segundos_restantes_despues_horas <- total_segundos MOD 3600
   // 3. Minutos. Dividimos los segundos restantes entre 60
   // luego obtenemos los minutos completos.
   minutos <- segundos_restantes_despues_horas / 60
   // 4. Segundos Finales
   // El operador del cociente anterior nos da los segundos sobrantes
   // luego de los minutos.
   segundos <- segundos_restantes_despues_horas MOD 60
   // Salida de resultados
   Escribir "------------------------------------"
   Escribir "Tiempo original en segundos: ", total_segundos
   Escribir "Tiempo equivalente: ", horas, " horas, " Sin Saltar
   Escribir "minutos, " minutos y ", segundos, " segundos."
FinAlgoritmo
```

∗ Ejercicio 5

Escriba un programa que calcule la nota final de la asignatura (sobre 100 puntos) a partir de las calificaciones obtenidas en los dos ciclos de evaluación (Interciclo y Final). El programa deberá solicitar al usuario las cuatro calificaciones: Aprovechamiento del Interciclo (Máx. 40 puntos). Examen del Interciclo (Máx. 10 puntos). Aprovechamiento Final (Máx. 40 puntos). Examen Final (Máx. 10 puntos). La Nota Total Final se obtendrá sumando las puntuaciones de los cuatro componentes.

Algoritmo 2.23: Pseudocódigo PSeInt: Cálculo de nota final)

```
Algoritmo Calcular_Nota_Final_100
   // Declarar variables.
   Definir ap_interciclo, ex_interciclo, ap_final, ex_final Como Real
   Definir nota_total Como Real
   // --- 1. Lectura de Calificaciones ---
   Escribir "--- INGRESO DE CALIFICACIONES (100 Puntos) ---"
   // Ciclo Interciclo (50 puntos)
   Escribir "1. Ingrese Nota de Aprovechamiento del INTERCICLO (40 ptos):"
   Leer ap_interciclo
   Escribir "2. Ingrese Nota de Examen del INTERCICLO (10 ptos):"
   Leer ex_interciclo
   // Ciclo Final (50 puntos)
   Escribir "3. Ingrese Nota de Aprovechamiento FINAL (40 ptos):"
   Leer ap_final
   Escribir "4. Ingrese Nota de Examen FINAL (10 ptos):"
   Leer ex_final
   // --- 2. Proceso: Nota Total Final ---
   // Nota Total = (AP Interciclo + EX Interciclo) + (AP Final + EX Final)
   nota_total <- ap_interciclo + ex_interciclo + ap_final + ex_final
   // --- 3. Salida de resultados ---
   Escribir "-----------------------------------"
   Escribir "RESUMEN DE NOTAS OBTENIDAS:"
   Escribir "Total Interciclo (50): ", ap_interciclo + ex_interciclo
   Escribir "Total Final (50): ", ap_final + ex_final
   Escribir "NOTA FINAL TOTAL: ", nota_total, " / 100"
FinAlgoritmo
```

∗ Ejercicio 6

Escriba un algoritmo que evalúe la siguiente expresión en términos de una sola línea (usa los paréntesis que sean estrictamente necesarios):

$$\frac{4}{3 \cdot (r+34)} - 9 \cdot (a + b \cdot c) + \frac{3 + d \cdot (2+a)}{a + b \cdot d}$$

Algoritmo 2.24: Pseudocódigo PSeInt: Evaluación de expresión matemática compleja

```
Algoritmo Evaluar_Expresion
   // Declarar variables para evaluar con decimales
   Definir r, a, b, c, d, resultado, t1, t2, t3 Como Real
   // Asignar valores de ejemplo para probar
   // (Estos valores son arbitrarios, se pueden cambiar)
   r <- 5
   a <- 2
   b <- 3
   c <- 4
   d <- 1
   // Proceso: Evaluar en una sola lista
```

```
    // 4 / (3*(r+34)) - 9*(a+b*c) + (3+d*(2+a))/(a+b*d)
    t1 <- 4 / (3 * (r + 34))
    t2 <- - (9 * (a + b * c))
    t3 <- (3 + d * (2 + a)) / (a + b * d)
    resultado <- t1 + t2 + t3
    // Salida de resultados
    Escribir "--- EVALUAR ---"
    Escribir "El resultado es: ", resultado
FinAlgoritmo
```

* Ejercicio 7

Escriba un programa que calcule el área de un triángulo utilizando la Ley del Seno. El programa debe solicitar al usuario la longitud de dos lados (Lado1 y Lado2) y el ángulo incluido (α) entre ellos, medido en grados. La fórmula a implementar es:

$$\text{Área} = \frac{1}{2} \cdot \text{Lado1} \cdot \text{Lado2} \cdot \sin(\alpha_{\text{rad}})$$

Nota Importante: El programa debe convertir el ángulo ingresado de grados a radianes antes de usar la función seno, usando la fórmula:

$$\alpha_{\text{rad}} = \alpha_{\text{grados}} \cdot \frac{\pi}{180}$$

Algoritmo 2.25: Pseudocódigo PSeInt: Cálculo del área de triángulo por Ley del Seno

```
Algoritmo Area_Triangulo_Ley_Seno
    // Declarar variables.
    Definir lado1, lado2, angulo_grados, angulo_radianes Como Real
    Definir area, Pi Como Real
    // 1. Asignar constante Pi
    Pi <- 3.141592
    // 2. Lectura de datos
    Escribir "--- Hallar valor con Ley del Seno ---"
    Escribir "Ingrese la longitud del primer lado (Lado1):"
    Leer lado1
    Escribir "Ingrese la longitud del segundo lado (Lado2):"
    Leer lado2
    Escribir "Ingrese el valor en grados entre estos dos lados:"
    Leer angulo_grados
    // 3. Proceso: Convertir grados a radianes
    // angulo_radianes = angulo_grados * (Pi / 180)
    angulo_radianes <- angulo_grados * (Pi / 180.0)
    // 4. Proceso: Calcular
    // Area = 0.5 * lado1 * lado2 * sen(angulo_radianes)
    area <- 0.5 * lado1 * lado2 * sen(angulo_radianes)
    // 5. Salida de resultados
    Escribir "-----------------------------------------"
    Escribir "Convertido a radianes: ", angulo_radianes
    Escribir "La superficie es: ", area
FinAlgoritmo
```

* Ejercicio 8

Escribe un algoritmo que pida al usuario el valor de dos variables reales x e y, y a continuación muestre el resultado de aplicarles la siguiente fórmula. Revisa en el programa de PSeInt todas aquellas expresiones válidas y útiles para desarrollar el ejercicio, por ejemplo, en la ayuda de PseInt, en funciones predefinidas, se indica que la raíz puede ser RC( ) o Raiz( ). (Declare constantes para los valores fijos):

$$\text{f(x,y)} = \sqrt{1{,}531^{(x+y)} + \frac{e^x - e^y \cdot (\sin(x) - \tan(y))}{\ln(y) \cdot 3{,}141592^x}}$$

Algoritmo 2.26: Pseudocodigo PSeInt: Evaluacion de formula con constantes (Final)

```
Algoritmo Evaluar_Formula_XY_V2
      // Declarar constantes (valores fijos)
      Definir BASE_POTENCIA Como Real; BASE_POTENCIA <- 1.531
      Definir PII Como Real; PII <- 3.141592
   Definir x, ye, resultado, numerador, denominador Como Real
      // 1. Lectura de datos
      Escribir "Introduce el valor de la variable x:"
      Leer x
      Escribir "Introduce el valor de la variable ye:"
      Leer ye
      // 2. Procesamiento (Numerador entre Denominador)
      // Numerador: e^x - e^ye * (sen(x) - tan(ye))
      // Nota: sen() como tan() esperan radianes.
      numerador <- exp(x) - exp(ye) * (sen(x) - tan(ye))
      // Denominador: ln(ye) * PII^x
      // Nota: ln() es el logaritmo natural.
      // La potencia usa el circunflejo ^
      denominador <- ln(ye) * (PII ^ x)
      // 3. Calcular el valor radical (f(x,ye))
      // El denominador no sea cero antes del cociente
      Si denominador <> 0 Entonces
            // Calculo del valor interno del radicando
            resultado <- (BASE_POTENCIA^(x+ye)) + (numerador/denominador)
            // Obtener el radical con rc(z) (radicando)
            Si resultado >= 0 Entonces
                  Escribir "El resultado de f(x,ye) es: ", rc(resultado)
            Sino
                  Escribir "ERROR: El valor radical es negativo."
            FinSi
      Sino
            Escribir "ERROR: Cociente cero."
      FinSi
FinAlgoritmo
```



Laboratorio 4

Capítulo 2

2.4. Ejercicios del Laboratorio 4

* Ejercicio 1

Realiza un algoritmo que muestre solo el siguiente mensaje: Hola a todos.

Algoritmo 2.27: Pseudocodigo PSeInt: Mensaje de saludo simple

```
Algoritmo Mensaje_Hola
      // Proceso: mostrar el texto requerido
      Escribir "Hola a todos."
FinAlgoritmo
```

* Ejercicio 2

Realiza un algoritmo que, dado un lado, obtenga y muestre el área del cuadrado. Para efectuar con potencias, en algoritmos se utiliza el símbolo $**$ o el circunfijo ^, se muestran a continuación las fórmulas a utilizar: $area = lado^2$ o utilizar $area = lado * lado$.

Algoritmo 2.28: Pseudocodigo PSeInt: Cálculo del área de un cuadrado

```
Algoritmo Area_Cuadrado
      // Definar variables
      Definir lado, area Como Real
      // 1. Lectura de datos
      Escribir "Introduce la medida del lado del cuadrado:"
      Leer lado
      // 2. Procesamiento
      // Se usa el circunflejo ^ para la potencia (lado^2)
      area <- lado ^ 2
      // 3. Salida de resultado
      Escribir "La superficie del cuadrado con lado ", lado, " es: ", area
FinAlgoritmo
```

* Ejercicio 3

Realiza un algoritmo que, dado el valor de un producto, determine, el subtotal a pagar, que depende del valor del producto por unidad y la cantidad de productos a comprar. Luego determine el valor del IVA 12 % del subtotal y sume dicha cantidad con el subtotal para hallar el total a pagar (revise una factura de compras que tenga IVA para su comprensión).

Algoritmo 2.29: Pseudocodigo PSeInt: Computo de una compra con impuesto al valor agregado

```
Algoritmo Proceso_Factura
      // Declarar constantes
      Definir IVA_TASA Como Real; IVA_TASA <- 0.12
      // Representa el 12 por ciento
      // Declaracion de variables
      Definir valor_unitario, cantidad, subtotal Como Real
      Definir iva_computado, total_a_pagar Como Real
      // 1. Lectura de datos
      Escribir "Ingrese el valor unitario del producto:"
      Leer valor_unitario
      Escribir "Ingrese la cantidad de productos a comprar:"
      Leer cantidad
      // 2. Proceso
      // Determinar el subtotal (valor * cantidad)
      subtotal <- valor_unitario * cantidad
      // Determinar el valor del IVA (subtotal * tasa)
      iva_computado <- subtotal * IVA_TASA
      // Determinar el total a pagar (subtotal + IVA)
```

```
19        total_a_pagar <- subtotal + iva_computado
20        // 3. Salida de resultados
21        Escribir "--- Detalle de la factura ---"
22        Escribir "Subtotal a pagar: ", subtotal
23        Escribir "Valor del IVA (12 porciento): ", iva_computado
24        Escribir "Total a pagar: ", total_a_pagar
25  FinAlgoritmo
```

* Ejercicio 4

Realiza un algoritmo que obtenga y muestre en pantalla el valor de y a partir de la ecuación:

$$y = 3 * x^2 + 7x - 15$$

Algoritmo 2.30: Pseudocodigo PSeInt: Computo del valor de la variable

```
1   Algoritmo Obtener_Valor_YE
2         // Declarar variables
3         Definir x, ye Como Real
4         // 1. Lectura de datos
5         // Se requiere el valor de x para calcular ye
6         Escribir "Ingrese el valor de la variable x:"
7         Leer x
8         // 2. Proceso
9         // Aplicar: ye = 3*x^2 + 7x - 15
10        // La potencia se indica con el circunflejo ^
11        ye <- (3 * (x ^ 2)) + (7 * x) - 15
12        // 3. Salida de resultado
13        Escribir "El valor de ye para x = ", x, " es: ", ye
14  FinAlgoritmo
```

* Ejercicio 5

Realiza un algoritmo que obtenga y muestre el área del círculo y el perímetro de la circunferencia: $area = \pi r^2$, $perimetro = 2\pi r$. La salida será de la siguiente forma: El radio ingresado es ....., el área es ....., el perímetro es ..... Para efectuar potencias, en algoritmos se utiliza el símbolo de intercalación ^denominado caret, sombrero o circunfijo.

Algoritmo 2.31: Pseudocodigo PSeInt: Cómputo del área y perímetro de un círculo

```
1   Algoritmo Area_y_Perimetro_Circulo
2         // Declarar constantes
3         Definir PII Como Real; PII <- 3.141592
4         Definir radio, area, perimetro Como Real
5         // 1. Lectura de datos
6         // Se solicita el valor del radio
7         Escribir "Ingrese el valor del radio de la circunferencia:"
8         Leer radio
9         // 2. Proceso
10        // Aplicar area = PII * radio^2
11        area <- PII * (radio ^ 2)
12        // Aplicar perimetro: perimetro = 2 * PII * radio
13        perimetro <- 2 * PII * radio
14        // 3. Salida de resultado
15        // Mostrar el resultado en el formato solicitado
16        Escribir "El radio ingresado es ", radio Sin Saltar
17        Escribir ", el area es ", area, ", el perimetro es ", perimetro
18  FinAlgoritmo
```

* Ejercicio 6

Realiza un algoritmo que al ingresar dos números, obtenga y muestre la suma de ambos. La salida debe mostrarse de la siguiente manera: Los números ingresados son .... y ...., la suma es ....

Algoritmo 2.32: Pseudocodigo PSeInt: Suma de dos números

```
Algoritmo Suma_Dos_Numeros
      // Declarar variables de proceso
      Definir num1, num2, suma Como Real
      // 1. Lectura de datos
      Escribir "Ingrese el primer valor:"
      Leer num1
      Escribir "Ingrese el segundo valor:"
      Leer num2
      // 2. Proceso
      // Aplicar la suma
      suma <- num1 + num2
      // 3. Salida de resultado
      // Mostrar el resultado en el formato solicitado
      Escribir "Los valores ingresados son ", num1 Sin Saltar
      Escribir " y ", num2, ", la suma es ", suma
FinAlgoritmo
```

* Ejercicio 7

Realiza un algoritmo que al ingresar dos números, obtener y mostrar la resta de ambos. La salida debe mostrarse de la siguiente manera: Los números ingresados son ..... y ....., la resta es .....

Algoritmo 2.33: Pseudocodigo PSeInt: Resta de dos números

```
Algoritmo Resta_Dos_Numeros
      // Declarar variables de proceso
      Definir num1, num2, diferencia Como Real
      // 1. Lectura de datos
      Escribir "Ingrese el primer numero (minuendo):"
      Leer num1
      Escribir "Ingrese el segundo numero (sustraendo):"
      Leer num2
      // 2. Proceso de resta
      diferencia <- num1 - num2
      // 3. Salida de resultado
      // Mostrar el resultado en el formato solicitado
      Escribir "Los valores ingresados son ", num1, " y " Sin Saltar
      Escribir num2, ", la resta es ", diferencia
FinAlgoritmo
```

* Ejercicio 8

Realiza un algoritmo que al ingresar dos números, obtener y mostrar la división de ambos. La salida debe mostrarse de la siguiente manera: Los números ingresados son ..... y ....., la división es ..... (explique qué ocurriría si se ingresan los números en orden inverso).

Algoritmo 2.34: Pseudocodigo PSeInt: Cociente de dos números

```
Algoritmo Division_Dos_Numeros
      // Declarar variables de proceso
      Definir num1, num2, cociente Como Real
      // 1. Lectura de datos
      Escribir "Ingrese el primer valor (dividendo):"
      Leer num1
```

```
      Escribir "Ingrese el segundo valor (divisor):"
      Leer num2
      // 2. Proceso
      // El cociente solo es posible si el divisor no es cero
      Si num2 <> 0 Entonces
            cociente <- num1 / num2
            // 3. Salida de resultado
            // Mostrar el resultado en el formato solicitado
            Escribir "Los valores ingresados son ", num1 Sin Saltar
            Escribir " y ", num2, ", la division es ", cociente
      Sino
            // Mensaje de error al ser el divisor cero
            Escribir "ERROR: No es posible dividir por cero."
      FinSi
FinAlgoritmo
```

La división es una operación no conmutativa. Esto significa que el orden en que se ingresan los números afecta directamente el resultado. Caso A (Orden Ingresado): Se calcula $num1/num2$. Caso B (Orden Inverso): Si se invierte el orden de los números en la operación, se calcularía $num2/num1$. Si el usuario ingresa los números 10 y 5, ver Tabla 2.8.

Tabla 2.8: Ejemplos de operaciones de división

Orden de Ingreso	Operación	Resultado
Normal (10, 5)	`10 / 5`	2
Inverso (5, 10)	`5 / 10`	0.5

El cociente resultante sería diferente. La única excepción a esta diferencia ocurriría si $num1$ y $num2$ fueran iguales (por ejemplo, $10/10 = 1$) o si el resultado de la división original fuera -1 (por ejemplo, $-5/5 = -1$ y $5/-5 = -1$). Además, si el primer número ($num1$) fuera cero, y se invirtiera el orden, podría ocurrir un error de división por cero si $num2$ no fuera cero. Por ejemplo: $0/5 = 0$, $5/0 =$ Error, (División por cero).

* Ejercicio 9

Realiza un algoritmo que obtenga y muestre el área e hipotenusa de un triángulo rectángulo. Ingresar b (base), a (lado), $area = (b \cdot a)/2$, e $hipotenusa = \sqrt{(a^2 + b^2)}$. Utilice el símbolo ^, para obtener en bloques de proceso en algoritmos $H^2 = (a^2 + b^2)$, donde $H = \sqrt{(H^2)}$. También hay funciones para elevar potencia que se trabajarán posteriormente. La salida será de la siguiente forma: El lado ingresado es ...., la base ingresada es, el área es ...., la hipotenusa es .....

Algoritmo 2.35: Pseudocodigo PSeInt: Cómputo del área e hipotenusa del triángulo rectángulo

```
Algoritmo Area_e_Hipotenusa_Triangulo
      // Declarar variables
      Definir lado_a, base, area, hipotenusa Como Real
      Escribir "Ingrese el valor del lado a (altura):"
      Leer lado_a
      Escribir "Ingrese el valor de la base b:"
      Leer base
      area <- (base * lado_a) / 2
      hipotenusa <- ( (lado_a ^ 2) + (base ^ 2) ) ^ (1/2)
      Escribir "El lado ingresado es ", lado_a Sin Saltar
      Escribir ", la base ingresada es ", base Sin Saltar
      Escribir ", el area es ", area, ", la hipotenusa es ", hipotenusa
FinAlgoritmo
```

* Ejercicio 10

Realiza un algoritmo que permita ingresar dos números, obtener y mostrar la suma de ambos, la multiplicación de ambos y la división de ambos. La salida debe mostrar de forma concatenada de la siguiente manera: Los números ingresados son .... y ...., la suma es ...., la multiplicación es .... la división es ....

Algoritmo 2.36: Pseudocodigo PSeInt: Operaciones combinadas

```
Algoritmo Operaciones_Basicas
	// Declarar variables de proceso
	Definir num1, num2, suma, producto, cociente Como Real
	// 1. Lectura de datos
	Escribir "Ingrese el primer valor:"
	Leer num1
	Escribir "Ingrese el segundo valor:"
	Leer num2
	// 2. Proceso
	suma <- num1 + num2
	producto <- num1 * num2
	Si num2 <> 0 Entonces
		cociente <- num1 / num2
		Escribir "Los valores ingresados son ", num1 Sin Saltar
		Escribir " y ", num2, ", la suma es ", suma Sin Saltar
		Escribir ", la multiplicacion es ", producto Sin Saltar
		Escribir ", la division es ", cociente
	Sino
		Escribir "ERROR. El segundo valor no puede ser cero."
		Escribir "Los valores ingresados son ", num1 Sin Saltar
		Escribir " y ", num2, ", la suma es ", suma Sin Saltar
		Escribir ", el producto es ", producto Sin Saltar
		Escribir ", el cociente es: INDETERMINADO"
	FinSi
FinAlgoritmo
```

* Ejercicio 11

Realiza un algoritmo que, dada una resistencia de 4 ohmios y el voltaje de 2 voltios y, usando las fórmulas $V = I \cdot R$, donde (V) es voltaje, I es corriente en amperios y resistencia R es ohmios, se calcule y muestre el valor de la corriente, $I = V/R$. Posteriormente, calcular y mostrar la potencia $P = RI^2$, o $P = VI$. Finalmente, mostrar en una sola salida, tanto los datos ingresados, resistencia, voltaje como corriente y la potencia.

Algoritmo 2.37: Pseudocodigo PSeInt: Ley de Ohm (Voltaje y Potencia)

```
Algoritmo Calculo_Electrico
	// Declarar variables
	Definir resistencia, corriente, voltaje, potencia Como Real
	Escribir "Ingrese el valor de la resistencia (R) en Ohmios:"
	Leer resistencia
	Escribir "Ingrese el valor del voltaje (V) en Voltios:"
	Leer voltaje
	corriente = voltaje / resistencia
	potencia = voltaje * corriente
	Escribir "La resistencia (R) es ", resistencia, " Ohmios."
	Escribir "La corriente (I) es ", corriente, " Amperios."
	Escribir "El voltaje (V) es ", voltaje, " Voltios"
	Escribir ", y la Potencia (P) es ", potencia, " Watts."
FinAlgoritmo
```

∗ Ejercicio 12

Realiza un algoritmo que obtenga y muestre el área y perímetro de un rectángulo. Ingresar b (base) y h (altura). Muestre el resultado concatenando las variables de salida y las variables de entradas, $area = b \cdot a$ y $perimetro = 2 \cdot (b + a)$.

Algoritmo 2.38: Pseudocodigo PSeInt: Computo del área y perímetro de un rectángulo

```
Algoritmo Area_y_Perimetro_Rectangulo
      // Declarar variables de proceso
      Definir base, altura, area, perimetro Como Real
      Escribir "Ingrese el valor de la base (b):"
      Leer base
      Escribir "Ingrese el valor de la altura (h):"
      Leer altura
      area <- base * altura
      perimetro <- 2 * (base + altura)
      Escribir "La base ingresada es ", base Sin Saltar
      Escribir ", la altura es ", altura Sin Saltar
      Escribir ", el area es ", area Sin Saltar
      Escribir " y el perimetro es ", perimetro
FinAlgoritmo
```

∗ Ejercicio 13

Realiza un algoritmo que obtenga y muestre el área y perímetro de un rombo. Ingresar D, d, a (lado), $area = 4 \cdot a$, o $perimetro = (D \cdot d)/2$. Muestre el resultado concatenando las variables de salida y las variables de entradas.

Algoritmo 2.39: Pseudocodigo PSeInt: Cómputo del área y perímetro de un rombo

```
Algoritmo Area_y_Perimetro_Rombo
      // Declarar variables de proceso
      Definir diagonal_mayor, diagonal_menor, lado_a Como Real
   Definir area, perimetro Como Real
      Escribir "Ingrese el valor de la Diagonal Mayor (D):"
      Leer diagonal_mayor
      Escribir "Ingrese el valor de la Diagonal Menor (d):"
      Leer diagonal_menor
      Escribir "Ingrese el valor del lado (a):"
      Leer lado_a
      area <- (diagonal_mayor * diagonal_menor) / 2
      perimetro <- 4 * lado_a
      Escribir "La Diagonal Mayor (D) es ", diagonal_mayor Sin Saltar
      Escribir ", la Diagonal Menor (d1) es ", diagonal_menor Sin Saltar
      Escribir ", el lado (a) es ", lado_a Sin Saltar
      Escribir ", el area es ", area Sin Saltar
      Escribir " y el perimetro es ", perimetro
FinAlgoritmo
```

∗ Ejercicio 14

Realiza un algoritmo que obtenga y muestre el área y perímetro de un paralelogramo. Ingresar b, h, a, y las fórmulas $area = b \cdot h$ y $perimetro = 2 \cdot (a + b)$. Muestre el resultado concatenando las variables de salidas y las variables de entradas.

Algoritmo 2.40: Pseudocodigo PSeInt: Cómputo de área y perímetro de un paralelogramo

```
Algoritmo Area_y_Perimetro_Paralelogramo
      // Declarar variables de proceso
      Definir base, altura, lado_a, area, perimetro Como Real
```

```
        // 1. Lectura de datos
        // Ingreso de valores, base, la altura y el lado adyacente
        Escribir "Ingrese el valor de la base (b):"
        Leer base
        Escribir "Ingrese el valor de la altura (h):"
        Leer altura
        Escribir "Ingrese el valor del lado adyacente (a):"
        Leer lado_a
        // 2. Proceso
        area <- base * altura
        perimetro <- 2 * (base + lado_a)
        // 3. Salida de resultado
        Escribir "La base (b) es ", base Sin Saltar
        Escribir ", la altura (h) es ", altura Sin Saltar
        Escribir ", el lado (a) es ", lado_a Sin Saltar
        Escribir ", el area es ", area Sin Saltar
        Escribir " y el perimetro es ", perimetro
FinAlgoritmo
```

* Ejercicio 15

Elabore un algoritmo para calcular el promedio final de la materia de Programación. Tome como referencia los porcentajes de evaluación proporcionados por el docente de la materia. La suma de los puntos máximos es: $10 + 10 + 10 + 20 = 50$ puntos. El promedio final se calcula sobre la base de esos 50 puntos. Para obtener el promedio final sobre 10 (o 100 %, dependiendo de cómo se interprete el "promedio final"), primero sumaremos los puntajes obtenidos y luego aplicaremos la conversión necesaria.

Algoritmo 2.41: Pseudocodigo PSeInt: Calculo del promedio final de Programación

```
Algoritmo Calculo_Promedio_Programacion
        // Declarar variables
        Definir PUNTAJE_MAXIMO Como Real; PUNTAJE_MAXIMO <- 50
        Definir colab, experi, autonomo, examen Como Real
        Definir total_obtenido, promedio_final Como Real
        // 1. Lectura de datos
        // Pedir al usuario los puntajes obtenidos en cada actividad.
        Escribir "Ingrese puntaje de Actividad Colaborativa (Max 10):"
        Leer colab
        Escribir "Ingrese puntaje de Actividad Experimental (Max 10):"
        Leer experi
        Escribir "Ingrese puntaje de Actividad Autonomo (Max 10):"
        Leer autonomo
        Escribir "Ingrese puntaje del Examen (Max 20):"
        Leer examen
        // 2. Proceso
        // A. Sumar todos los puntajes obtenidos
        total_obtenido <- colab + experi + autonomo + examen
        // B. Calcular el promedio final sobre una base de 10 puntos
        // Formula: (Total Obtenido / 50) * 10
        promedio_final <- (total_obtenido / PUNTAJE_MAXIMO) * 10
        // 3. Salida de resultado
        Escribir "El total obtenido es: ", total_obtenido Sin Saltar
        Escribir " de ", PUNTAJE_MAXIMO, " puntos."
        Escribir "El promedio es: ", promedio_final Sin Saltar
        Escribir " / 10 puntos."
FinAlgoritmo
```



Laboratorio 5

Capítulo 2

2.5. Ejercicios del Laboratorio 5

* Ejercicio 1

Realizar un algoritmo que, ingresando una edad, determine si es mayor o menor de edad.

Algoritmo 2.42: Pseudocodigo PSeInt: Determinación de Mayor o Menor de Edad

```
Algoritmo Determinar_Edad
      // Declarar variables
      Definir EDAD_LEGAL Como Entero; EDAD_LEGAL <- 18
      Definir edad Como Entero
      // 1. Lectura de datos
      Escribir "Ingrese la edad de la persona:"
      Leer edad
      // 2. Proceso
      Si edad >= EDAD_LEGAL Entonces
            Escribir "La persona tiene ", edad
            Escribir " anios. Es MAYOR de edad."
      Sino
            // En cualquier otro caso, es menor de edad
            Escribir "La persona tiene ", edad
            Escribir " anios. Es MENOR de edad."
      FinSi
FinAlgoritmo
```

* Ejercicio 2

Realizar un algoritmo que ingrese un nombre, ingrese una edad, determine si es mayor o menor de edad. Muestre los datos concatenados. Ej.: Juan con 19 años es mayor de edad. Operación: Edad >= 18.

Algoritmo 2.43: Pseudocodigo PSeInt: Determinación de Mayor o Menor de Edad con salida concatenada

```
Algoritmo Determinar_Edad_Concatenado
      // Declarar variables
      Definir EDAD_LEGAL Como Entero; EDAD_LEGAL <- 18
      Definir nombre Como Caracter
      Definir edad Como Entero
      // 1. Lectura de datos
      Escribir "Ingrese el nombre de la persona:"
      Leer nombre
      Escribir "Ingrese la edad de la persona:"
      Leer edad
      // 2. Proceso
      Si edad >= EDAD_LEGAL Entonces
            Escribir nombre, " con ", edad Sin Saltar
            Escribir " anios es mayor de edad."
      Sino
            Escribir nombre, " con ", edad Sin Saltar
            Escribir " anios es menor de edad."
      FinSi
FinAlgoritmo
```

∗ Ejercicio 3

Realizar un algoritmo que ingrese un valor real X, luego, determine que pertenece al intervalo x <= 22.

Algoritmo 2.44: Pseudocodigo PSeInt: Determinación de pertenencia a un rango simple

```
Algoritmo Verificar_Rango_Menor_Igual
	// Declarar el limite superior del rango
	Definir LIMITE_SUPERIOR Como Real; LIMITE_SUPERIOR <- 22
	Definir X Como Real
	// 1. Lectura de datos
	Escribir "Ingrese un valor real para X:"
	Leer X
	// 2. Proceso
	// Condicion: X <= 22
	Si X <= LIMITE_SUPERIOR Entonces
		Escribir "El valor X = ", X, " SI pertenece al rango."
	Sino
		Escribir "El valor X = ", X, " NO pertenece al rango."
	FinSi
FinAlgoritmo
```

∗ Ejercicio 4

Realizar un algoritmo que ingrese un valor real X, determine que pertenece al intervalo x <22

Algoritmo 2.45: Pseudocodigo PSeInt: Determinación de pertenencia a un rango menor estricto

```
Algoritmo Verificar_Rango_Menor_22
	// Declarar el limite superior del rango (exclusivo)
	Definir LIMITE_SUPERIOR Como Real; LIMITE_SUPERIOR <- 22
	// Definir la variable de entrada
	Definir X Como Real
	// 1. Lectura de datos
	Escribir "Ingrese un valor real para X:"
	Leer X
	// 2. Proceso
	Si X < LIMITE_SUPERIOR Entonces
		Escribir "El valor X = ", X
		Escribir " SI pertenece al rango (es menor a 22)."
	Sino
		// En este caso, X es mayor o igual a 22
		Escribir "El valor X = ", X
		Escribir " NO pertenece al rango (es mayor o igual a 22)."
	FinSi
FinAlgoritmo
```

∗ Ejercicio 5

Realizar un algoritmo que ingrese un valor real X, determine que pertenece al intervalo x >-13

Algoritmo 2.46: Pseudocodigo PSeInt: Determinación de pertenencia a un rango mayor estricto

```
Algoritmo Verificar_Rango_Mayor_Menos13
	// Declarar el limite inferior del rango (exclusivo)
	Definir LIMITE_INFERIOR Como Real; LIMITE_INFERIOR <- -13
	Definir X Como Real
	// 1. Lectura de datos
	Escribir "Ingrese un valor real para X:"
	Leer X
	// 2. Proceso
```

```
      Si X > LIMITE_INFERIOR Entonces
            Escribir "El valor X = ", X
      Escribir " SI pertenece al rango (es mayor a -13)."
      Sino
            Escribir "El valor X = ", X
            Escribir " NO pertenece al rango (es menor o igual a -13)."
      FinSi
FinAlgoritmo
```

* Ejercicio 6

Realizar un algoritmo que ingrese un valor real X, luego determine que pertenece al intervalo x >= -13

Algoritmo 2.47: Pseudocodigo PSeInt: Determinación de pertenencia a un rango mayor o igual

```
Algoritmo Verificar_Rango_Mayor_Igual_Menos13
      // Declarar
      Definir LIMITE_INFERIOR Como Real; LIMITE_INFERIOR <- -13
      Definir X Como Real
      // 1. Lectura de datos
      Escribir "Ingrese un valor real para X:"
      Leer X
      // 2. Proceso
      Si X >= LIMITE_INFERIOR Entonces
         Escribir "El valor X = ", X
         Escribir " SI pertenece al rango (es mayor o igual a -13)."
      Sino
            // En este caso, X es estrictamente menor a -13
         Escribir "El valor X = ", X
         Escribir " NO pertenece al rango (es menor estricto a -13)."
      FinSi
FinAlgoritmo
```

* Ejercicio 7

Realizar un algoritmo que ingrese un valor real X, determine que pertenece a UNO DE LOS DOS intervalos. Y luego, determine a cuál de los intervalos pertenece, (x >= 18) ∪ (x <11). Hay que recordar que la unión es la expresión OR.

Algoritmo 2.48: Pseudocodigo PSeInt: Determinación de pertenencia a una unión de intervalos (OR)

```
Algoritmo Verificar_Union_Intervalos
      // Declarar los limites de los intervalos
      Definir LIMITE_SUPERIOR Como Real; LIMITE_SUPERIOR <- 18
      Definir LIMITE_INFERIOR Como Real; LIMITE_INFERIOR <- 11
      Definir X Como Real
      // 1. Lectura de datos
      Escribir "Ingrese un valor real para X:"
      Leer X
      // 2. Proceso
      Si X >= LIMITE_SUPERIOR Entonces
            Escribir "El valor X = ", X
            Escribir " SI pertenece al intervalo [18, infinito)."
      Sino Si X < LIMITE_INFERIOR Entonces
            Escribir "El valor X = ", X
            Escribir " SI pertenece al intervalo (-infinito, 11)."
      Sino
            Escribir "El valor X = ", X
            Escribir " NO pertenece a la union de intervalos" Sin Saltar
```

```
            Escribir ", esta en el rango [11, 18)."
      FinSi
FinAlgoritmo
```

* Ejercicio 8

Realizar un algoritmo que ingrese un valor real X, determine si pertenece al intervalo: x >= -10 ∩ x <5. Hay que recordar que la intersección es la expresión AND.

Algoritmo 2.49: Pseudocodigo PSeInt: Determinación de pertenencia a una intersección de intervalos (AND)

```
Algoritmo Verificar_Interseccion_Intervalo
      // Declarar
      Definir LIMITE_INFERIOR Como Real; LIMITE_INFERIOR <- -10
      Definir LIMITE_SUPERIOR Como Real; LIMITE_SUPERIOR <- 5
      Definir X Como Real
      // 1. Lectura de datos
      Escribir "Ingrese un valor real para X:"
      Leer X
      // 2. Proceso
      Si (X >= LIMITE_INFERIOR) Y (X < LIMITE_SUPERIOR) Entonces
            Escribir "El valor X = ", X
            Escribir " SI pertenece al intervalo [-10, 5)."
      Sino
            Escribir "El valor X = ", X
            Escribir " NO pertenece al intervalo [-10, 5)."
      FinSi
FinAlgoritmo
```

* Ejercicio 9

Realizar un algoritmo que ingrese un valor real X, determine si pertenece al intervalo: x <13 ∩ x >= 9. Hay que recordar que la intersección es la expresión AND.

Algoritmo 2.50: Pseudocodigo PSeInt: Determinación de pertenencia a una intersección (AND) de intervalos

```
Algoritmo Verificar_Interseccion_9_13
      // Declarar intervalo cerrado [9, 13)
      Definir LIMITE_INFERIOR Como Real; LIMITE_INFERIOR <- 9
      Definir LIMITE_SUPERIOR Como Real; LIMITE_SUPERIOR <- 13
      Definir X Como Real
      // 1. Lectura de datos
      Escribir "Ingrese un valor real para X:"
      Leer X
      // 2. Proceso
      Si (X >= LIMITE_INFERIOR) Y (X < LIMITE_SUPERIOR) Entonces
            Escribir "El valor X = ", X Sin Saltar
            Escribir " SI pertenece al intervalo [9, 13) (interseccion)."
      Sino
            Escribir "El valor X = ", X Sin Saltar
            Escribir " NO pertenece al intervalo [9, 13)."
      FinSi
FinAlgoritmo
```

∗ Ejercicio 10

Realizar un algoritmo que ingrese un valor real X, determine que pertenece a UNO DE LOS DOS intervalos. Y luego, determine a cuál de los intervalos pertenece. x <0 ∪ x >= 3. Hay que recordar que la unión es la expresión OR.

Algoritmo 2.51: Pseudocodigo PSeInt: Determinación de pertenencia a una unión de intervalos (OR)

```
Algoritmo Verificar_Union_Intervalos_0_3
      // Declarar loslos intervalos
      Definir LIMITE_SUPERIOR Como Real; LIMITE_SUPERIOR <- 3
      Definir LIMITE_INFERIOR Como Real; LIMITE_INFERIOR <- 0
      Definir X Como Real
      // 1. Lectura de datos
      Escribir "Ingrese un valor X:"
      Leer X
      // 2. Proceso
      // Evaluar: Intervalo (-infinito, 0)
      Si X < LIMITE_INFERIOR Entonces
            Escribir "El valor X = ", X
            Escribir " SI pertenece al intervalo (-infinito, 0)."
      // Evaluar: Intervalo [3, infinito)
      Sino Si X >= LIMITE_SUPERIOR Entonces
            Escribir "El valor X = ", X
            Escribir " SI pertenece al intervalo [3, infinito)."
      // Si no cumple ninguna de las anteriores,
      // se encuentra en el intervalo central [0, 3)
      Sino
            Escribir "El valor X = ", X
      Escribir " NO pertenece a la union de intervalos, " Sin Saltar
      Escribir "se encuentra en el rango [0, 3)."
      FinSi
FinAlgoritmo
```

∗ Ejercicio 11

Realice un algoritmo que permita el ingreso de un valor de temperatura (el programa debe indicar qué debe ingresar / o qué desea el usuario ingresar, grados Kelvin, Fahrenheit o Centígrados) luego devuelva su valor correspondiente en los dos grados faltantes.

Algoritmo 2.52: Pseudocodigo PSeInt: Conversor de unidades de temperatura

```
Algoritmo Conversor_Temperatura
      // Declarar variables de proceso
      Definir temp_original, temp_C, temp_F, temp_K Como Real
      Definir opcion Como Entero
      // 1. Lectura de datos
      Escribir "--- Conversor de Temperatura ---"
      Escribir "Ingrese la unidad de temperatura que desea convertir:"
      Escribir "1: Grados Centigrados (Celsius)"
      Escribir "2: Grados Fahrenheit"
      Escribir "3: Grados Kelvin"
      Leer opcion
      Escribir "Ingrese el valor de la temperatura:"
      Leer temp_original
      // 2. Proceso
      Segun opcion Hacer
            // Caso 1: Entrada en Centigrados (C)
            1:
```

```
                    temp_C <- temp_original
                    // Computo a Fahrenheit: F = C * 1.8 + 32
                    temp_F <- (temp_C * 1.8) + 32
                    // Computo a Kelvin: K = C + 273.15
                    temp_K <- temp_C + 273.15
              // Caso 2: Entrada en Fahrenheit (F)
              2:
                    temp_F <- temp_original
                    // Computo a Centigrados: C = (F - 32) / 1.8
                    temp_C <- (temp_F - 32) / 1.8
                    // Computo a Kelvin: K = C + 273.15
                    temp_K <- temp_C + 273.15
              // Caso 3: Entrada en Kelvin (K)
              3:
                    temp_K <- temp_original
                    // Computo a Centigrados: C = K - 273.15
                    temp_C <- temp_K - 273.15
                    // Computo a Fahrenheit: F = C * 1.8 + 32
                    temp_F <- (temp_C * 1.8) + 32
              De Otro Modo:
                    Escribir "ERROR: valor incorrecto."
        FinSegun
        // 3. Salida de resultado
        // Solo mostrar resultados si la opcion fue correcta
        Si (opcion >= 1) Y (opcion <= 3) Entonces
              Escribir "--- Resultado de la conversion ---"
              Escribir "Temperatura en Centigrados: ", temp_C, " C"
              Escribir "Temperatura en Fahrenheit: ", temp_F, " F"
              Escribir "Temperatura en Kelvin: ", temp_K, " K"
        FinSi
FinAlgoritmo
```

* Ejercicio 12

Se solicita un número por teclado y verificar que el número se encuentre entre el rango, mayor a 1000 (sin incluir) y 1999 (incluido), (utilice la tabla de verdad AND y OR para su análisis).

Análisis: El rango solicitado es $X \in (1000, 1999]$. Esto significa que el número X debe cumplir dos condiciones simultáneamente: Condición 1 (Límite Inferior): X debe ser mayor a 1000 ($X > 1000$). Condición 2 (Límite Superior): X debe ser menor o igual a 1999 ($X \leq 1999$). Para que el número pertenezca al rango, la Condición 1 y la Condición 2 deben ser Verdaderas.

El operador AND (Y) garantiza que solo cuando ambas condiciones son ciertas, el resultado final es cierto:

Tabla 2.9: Tabla de verdad (Operador AND para rango)

Condición 1 (X>1000)	**Condición 2** (X≤1999)	**Resultado** (Pertenece al Rango)
Falso	Falso	Falso
Falso	Verdadero	Falso
Verdadero	Falso	Falso
Verdadero	Verdadero	Verdadero

Algoritmo 2.53: Pseudocodigo PSeInt: Verificación de rango con AND

```
Algoritmo Verificar_Rango_Mil
      // Declarar los limites del rango (1000, 1999]
```

```
3       Definir LIMITE_INF Como Entero; LIMITE_INF <- 1000
4       Definir LIMITE_SUP Como Entero; LIMITE_SUP <- 1999
5       Definir numero Como Entero
6       // 1. Lectura de datos
7       Escribir "Ingrese un valor entero:"
8       Leer numero
9       // 2. Proceso
10      // Condicionar: (numero > 1000) and (numero <= 1999)
11      Si (numero > LIMITE_INF) Y (numero <= LIMITE_SUP) Entonces
12            Escribir "El numero ", numero
13            Escribir " SI se encuentra entre 1000" Sin Saltar
14            Escribir "(sin incluir) y 1999 (incluido)."
15      Sino
16            Escribir "El numero ", numero
17            Escribir " NO se encuentra en el rango especificado."
18      FinSi
19  FinAlgoritmo
```

* Ejercicio 13

13. Se solicita un número por teclado y verificar que el número se encuentre entre el rango, menor a 1000 (incluido) o mayor 1999 (sin incluir), utilice la tabla de verdad AND y OR para su análisis.

Análisis: El rango solicitado es $X \in (-\infty, 1000] \cup (1999, \infty)$. Esto significa que el número X debe cumplir al menos una de las dos condiciones para pertenecer al conjunto:Condición 1 (Límite Superior): X debe ser menor o igual a 1000 ($X \leq 1000$). Condición 2 (Límite Inferior): X debe ser mayor a 1999 ($X > 1999$). Para que el número pertenezca al rango, la Condición 1 o la Condición 2 deben ser Verdaderas.

El operador OR (O) garantiza que si al menos una de las condiciones es cierta, el resultado final es cierto:

Tabla 2.10: Tabla de Verdad (Operador OR para rangos disjuntos)

Condición 1 (X≤1000)	**Condición 2** (X>1999)	**Resultado** (Pertenece al Rango)
Falso	Falso	Falso
Falso	Verdadero	Verdadero
Verdadero	Falso	Verdadero
Verdadero	Verdadero	Verdadero

Algoritmo 2.54: Pseudocodigo PSeInt: Verificación de rango con OR

```
1   Algoritmo Verificar_Rango_Disjunto
2       // Declarar los limites del rango disjunto
3       Definir LIMITE_SUP_IN Como Entero
4       LIMITE_SUP_IN <- 1000
5       Definir LIMITE_INF_EX Como Entero
6       LIMITE_INF_EX <- 1999
7       Definir numero Como Entero
8       // 1. Lectura de datos
9       Escribir "Ingrese un valor:"
10      Leer numero
11      // 2. Proceso
12      // Condicionar: (numero <= 1000) or (numero > 1999)
13      // Se usa el operador logico O para verificar
14      // si pertenece a la union
```

```
15        Si (numero <= LIMITE_SUP_IN) O (numero > LIMITE_INF_EX) Entonces
16              Escribir "El valor ", numero
17              Escribir " SI pertenece al rango disjunto " Sin Saltar
18              Escribir "(menor o igual a 1000 O mayor a 1999)."
19        Sino
20              // El valor que no pertenece se encuentra
21        // en el rango central: (1000, 1999]
22              Escribir "El valor ", numero
23              Escribir " NO pertenece al rango disjunto " Sin Saltar
24              Escribir "esta entre 1000 y 1999."
25        FinSi
26  FinAlgoritmo
```

* Ejercicio 14

Elabore un algoritmo que permita el ingreso de un valor de tres cifras. Luego determine si el número es un valor Armstrong de tres cifras. Para lograrlo, el número ingresado debe descomponerse en unidad, decena, centena. Luego cada uno de ellos se eleva al cubo (por ser de tres cifras) y se suman, para reconstruir el número. Si dicho número reconstruido y el valor ingresado son iguales, entonces es un número Armstrong.

Análisis: El algoritmo debe determinar si un número de tres cifras es un número de Armstrong. La clave está en la descomposición usando los operadores MOD (módulo o residuo) y TRUNC (truncar o parte entera). Un Número Armstrong de tres cifras es aquel cuya suma de los cubos de sus dígitos es igual al número original. Por ejemplo, 153 es Armstrong porque $1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153$.

Algoritmo 2.55: Pseudocodigo PSeInt: Verificacion de valor Armstrong

```
1   Algoritmo Verificar_Armstrong
2         // Declarar variables de proceso
3         Definir num_entrada, unidad, decena, centena Como Entero
4         Definir resultado_cubos Como Entero
5         // 1. Lectura de datos
6         Escribir "Ingrese un numero de tres cifras:"
7         Leer num_entrada
8         // 2. Proceso
9         // La descomposicion se basa en la aritmetica modular (MOD)
10        // La Unidad: residuo de dividir por 10
11        unidad <- num_entrada MOD 10
12        // La Decena: Truncar la division por 10, luego residuo por 10
13        decena <- TRUNC(num_entrada / 10) MOD 10
14        // La Centena: Truncar la division por 100
15        centena <- TRUNC(num_entrada / 100)
16        // 3. Proceso: Reconstruir el valor sumando los cubos
17        // Puede ser u*u*u + d*d*d + c*c*c
18        resultado_cubos <- (unidad ^ 3) + (decena ^ 3) + (centena ^ 3)
19        // 4. Verificar y salir
20        // Si el valor de entrada es igual al resultado
21        // de la suma de los cubos
22        Si num_entrada == resultado_cubos Entonces
23              Escribir num_entrada, " SI es un valor ARMSTRONG."
24        SiNo
25              Escribir num_entrada, " NO es un valor ARMSTRONG."
26        Fin Si
27        // Se muestra el resultado
28        Escribir "Resultado de la suma de cubos es: ", resultado_cubos
29  FinAlgoritmo
```

* Ejercicio 15

Elabore un algoritmo que permita recorrer todos los números de tres cifras. Luego determine todos los valores Armstrong de tres cifras. Para lograrlo, el número ingresado debe descomponerse en unidad, decena, centena. Luego cada uno de ellos se eleva al cubo (por ser de tres cifras) y se suman, para reconstruir el número. Si dicho número reconstruido y el valor recorrido son iguales, entonces es un número Armstrong.

Algoritmo 2.56: Pseudocodigo PSeInt: Búsqueda de números Armstrong de tres cifras

```
Algoritmo Encontrar_Armstrongs
	// Declarar variables de proceso
	// x es el contador que va de 100 a 999
	Definir x, num_actual, u, d, c Como Entero
	Definir resultado_cubos Como Entero
	Escribir "--- Valores ARMSTRONG de 3 Cifras ---"
	// 1. Proceso
	// Recorrer todos los valores desde el 100 hasta el 999
	Para x = 100 Hasta 999 Con Paso 1 Hacer
		num_actual <- x // 'num_actual' toma el valor a verificar
		// A. Descomponer el valor
		u <- num_actual MOD 10
		d <- TRUNC(num_actual / 10) MOD 10
		c <- TRUNC(num_actual / 100)
		// B. Reconstruir el valor sumando los cubos
		resultado_cubos <- (u*u*u) + (d*d*d) + (c*c*c)
		// C. Verificar y Salir
		// Si el valor actual es igual al resultado
		// de la suma de los cubos
		Si num_actual == resultado_cubos Entonces
			// 2. Salida de resultado (solo si es Armstrong)
			Escribir num_actual, " SI es un numero ARMSTRONG."
		Fin Si
	Fin Para
	Escribir "--- Proceso finalizado ---"
FinAlgoritmo
```

* Ejercicio 16

Elabore un algoritmo que permita el ingreso de un valor de cuatro cifras. Luego determine si el número es un valor Armstrong de cuatro cifras. Para lograrlo, el número ingresado debe descomponerse en unidad, decena, centena y unidad de mil (m). Luego cada uno de ellos se eleva a la cuarta (por ser de cuatro cifras) y se suman, para reconstruir el número. Si dicho número reconstruido y el valor ingresado son iguales, entonces es un número Armstrong.

Algoritmo 2.57: Pseudocodigo PSeInt: Verificacion de número Armstrong de cuatro cifras

```
Algoritmo Verificar_Armstrong_Cuatro
	// Declarar variables de proceso
	Definir num_entrada, u, d, c, m Como Entero
	// u:unidad, d:decena, c:centena, m:mil
	Definir resultado_potencias Como Entero
	// 1. Lectura de datos
	Escribir "Ingrese un valor de cuatro cifras (entre 1000 y 9999):"
	Leer num_entrada
	// 2. Proceso: Descomponer el valor en sus cifras
	// A. Obtener la Unidad (u): residuo de dividir por 10
	u <- num_entrada MOD 10
	// B. Obtener la Decena (d): Truncar (/10), luego residuo por 10
```

```
      d <- TRUNC(num_entrada / 10) MOD 10
      // C. Obtener la Centena (c): Truncar (/100), luego residuo por 10
      c <- TRUNC(num_entrada / 100) MOD 10
      // D. Obtener la Unidad de mil (m): Truncar la division por 1000
      m <- TRUNC(num_entrada / 1000)
      // 3. Proceso: Reconstruir el valor sumando las cuartas potencias
      // (potencia ^ 4)
      resultado_potencias <- (u ^ 4) + (d ^ 4) + (c ^ 4) + (m ^ 4)
      // 4. Verificar y Salir
      // Valor de entrada es igual al resultado de suma y potencias
      Si num_entrada == resultado_potencias Entonces
            Escribir num_entrada
      Escribir " SI es un numero ARMSTRONG de cuatro cifras."
      SiNo
            Escribir num_entrada
      Escribir " NO es un numero ARMSTRONG de cuatro cifras."
      Fin Si
      // Opcional: Mostrar el resultado del computo
      Escribir "El resultado es: ", resultado_potencias
FinAlgoritmo
```

* Ejercicio 17

Elabore un algoritmo que permita recorrer todos los números de cuatro cifras. Luego determine todos los valores Armstrong de cuatro cifras. Para lograrlo, el número ingresado debe descomponerse en unidad, decena, centena y unidades de mil. Luego cada uno de ellos se eleva a la cuarta (por ser de cuatro cifras) y se suman, para reconstruir el número. Si dicho número reconstruido y el valor recorrido son iguales, entonces es un número Armstrong.

Algoritmo 2.58: Pseudocodigo PSeInt: Búsqueda de números Armstrong de cuatro cifras

```
Algoritmo Encontrar_Armstrongs_Cuatro
      // Declarar variables de proceso
      // x es el contador que va de 1000 a 9999
      Definir x, num_actual, u, d, c, m Como Entero
   // u, d, c, m son los digitos
      Definir resultado_potencias Como Entero
      Escribir "--- Busqueda de valores ARMSTRONG de 4 Cifras ---"
      // 1. Proceso (Uso de estructura Para)
      // Recorrer todos los valores desde el 1000 hasta el 9999
      Para x = 1000 Hasta 9999 Con Paso 1 Hacer
            num_actual <- x // 'num_actual' toma el valor a verificar
            // A. Descomponer el valor en sus digitos
            // u: Unidad, d: Decena, c: Centena, m: Unidad de Mil
            u <- num_actual MOD 10
            d <- TRUNC(num_actual / 10) MOD 10
            c <- TRUNC(num_actual / 100) MOD 10
            m <- TRUNC(num_actual / 1000)
            // B. Reconstruir el valor sumando las cuartas potencias
            resultado_potencias <- (u^4) + (d^4) + (c^4) + (m^4)
            // C. Verificar y Mostrar
            Si num_actual == resultado_potencias Entonces
                  // 2. Salida de resultado (solo si es Armstrong)
                  Escribir num_actual, " SI es un numero ARMSTRONG."
            Fin Si
      Fin Para
      Escribir "--- Proceso finalizado ---"
FinAlgoritmo
```

∗ Ejercicio 18

Elabore un algoritmo que permita recorrer todos los números de cinco cifras. Luego determine todos los valores Armstrong de cinco cifras. Para lograrlo, el número ingresado debe descomponerse en unidad, decena, centena y unidades de mil, decena de miles. Luego cada uno de ellos se eleva a la quinta (por ser de cinco cifras) y se suman, para reconstruir el número. Si dicho número reconstruido y el valor recorrido son iguales, entonces es un número Armstrong.

Algoritmo 2.59: Pseudocodigo PSeInt: Búsqueda de números Armstrong de cinco cifras

```
Algoritmo Encontrar_Armstrongs_Cinco
      // Declarar variables de proceso
      // x es el contador que va de 10000 a 99999
      Definir x, num_actual, u, d, c, m, dm Como Entero
      // dm: decena de mil
      Definir resultado_potencias Como Entero
      Escribir "--- Valores ARMSTRONG de 5 Cifras ---"
      // 1. Proceso (Uso de estructura Para)
      // Recorrer todos los valores desde el 10000 hasta el 99999
      Para x = 10000 Hasta 99999 Con Paso 1 Hacer
            num_actual <- x // 'num_actual' toma el valor a verificar
            // A. Descomponer el valor en sus digitos
            // u: Unidad, d: Decena, c: Centena,
            // m: Mil, dm: Decena de Mil
            u <- num_actual MOD 10
            d <- TRUNC(num_actual / 10) MOD 10
            c <- TRUNC(num_actual / 100) MOD 10
            m <- TRUNC(num_actual / 1000) MOD 10
            dm <- TRUNC(num_actual / 10000)
            // B. Reconstruir el valor
            // sumando las quintas potencias (digito ^ 5)
            resultado_potencias <- (u^5)+(d^5)+(c^5)+(m^5)+(dm^5)
            // C. Verificar y Mostrar
            // Si el valor actual es igual al resultado
            // de la suma de las potencias
            Si num_actual == resultado_potencias Entonces
                  // 2. Salida de resultado (solo si es Armstrong)
                  Escribir num_actual, " SI es un numero ARMSTRONG."
            Fin Si
      Fin Para
      Escribir "--- Proceso finalizado ---"
FinAlgoritmo
```



Laboratorio 6

Capítulo 2

2.6. Ejercicios del Laboratorio 6

∗ Ejercicio 1

Diseñar un algoritmo que imprima la serie de números: 3 7 11 15 19 23 27 31 35 39 (usando el bucle Para)

Algoritmo 2.60: Pseudocodigo PSeInt: Imprimir serie numérica con Para (Incremento de 4)

```
Algoritmo Imprimir_Serie
	// Declarar variables de proceso
	// 'numero' es quien almacena el valor actual de la serie
	Definir numero Como Entero
	Escribir "--- Serie de valores: 3, 7, 11, ..., 39 ---"
	// Recorrer los valores de 3 hasta 39 con un incremento de 4
	Para numero = 3 Hasta 39 Con Paso 4 Hacer
		Escribir Sin Saltar numero, " "
	Fin Para
FinAlgoritmo
```

∗ Ejercicio 2

Diseñar un algoritmo que imprima la serie de números: 3 7 11 15 19 23 27 31 35 39 (usando el bucle Mientras)

Algoritmo 2.61: Pseudocodigo PSeInt: Imprimir serie numérica con Mientras (Incremento de 4)

```
Algoritmo Imprimir_Serie_Mientras
	// Declarar variables de proceso
	Definir numero Como Entero
	Escribir "--- Serie de valores: 3, 7, 11, ..., 39 ---"
	// 1. Inicializar el primer valor de la serie
	numero <- 3
	// 2. Permanece en el bucle siempre que el valor no exceda 39
	Mientras numero <= 39 Hacer
		Escribir Sin Saltar numero, " "
		numero <- numero + 4
	Fin Mientras
FinAlgoritmo
```

∗ Ejercicio 3

Diseñar un algoritmo que imprima la serie de números: 3 7 11 15 19 23 27 31 35 39 (usando el bucle Repetir-Hasta)

Algoritmo 2.62: Pseudocodigo PSeInt: Imprimir serie numérica con Repetir-Hasta (Incremento de 4)

```
Algoritmo Imprimir_Serie_Repetir
	// Declarar variables de proceso
	Definir numero Como Entero
	Escribir "--- Serie de valores: 3, 7, 11, ..., 39 ---"
	// 1. Inicializar el primer valor de la serie
	numero <- 3
	// 2. Se detiene cuando es > 39
	Repetir
		Escribir Sin Saltar numero, " "
		numero <- numero + 4
	// 3. Condicion de Salida
	Hasta Que numero > 39
FinAlgoritmo
```

∗ Ejercicio 4

Diseñar un algoritmo que imprima la siguiente serie de números de forma descendente: 3 7 11 15 19 23 27 31 35 39, se mostrarían así: 39 35 31 27 23 19 15 11 7 3, usando el bucle Para.

Algoritmo 2.63: Pseudocodigo PSeInt: Imprimir serie numérica descendente con Para (Decremento de 4)

```
Algoritmo Imprimir_Serie_Descendente
	// 'numero' es quien almacena el valor actual de la serie
	Definir numero Como Entero
	Escribir "--- Serie de valores descendente: 39, 35, ..., 3 ---"
	// Recorrer los valores de 39 hasta 3 con un decremento de -4
	// El 'Paso' es un valor negativo en el orden descendente
	Para numero = 39 Hasta 3 Con Paso -4 Hacer
		Escribir Sin Saltar numero, " "
	Fin Para
FinAlgoritmo
```

∗ Ejercicio 5

Diseñar un algoritmo que imprima la siguiente serie de números de forma descendente: 3 7 11 15 19 23 27 31 35 39, se mostrarían así: 39 35 31 27 23 19 15 11 7 3, usando el bucle Mientras.

Algoritmo 2.64: Pseudocodigo PSeInt: Imprimir serie numérica descendente con Mientras (Decremento de 4)

```
Algoritmo Imprimir_Serie_Descendente_Mientras
	Definir numero Como Entero
	// 'numero' es quien almacena el valor actual
	Escribir "--- Serie de valores descendente: 39, 35, ..., 3 ---"
	// 1. Inicializar el primer valor de la serie
	numero <- 39
	// 2. Permanece en el bucle siempre que
	// el valor sea mayor o igual a 3
	Mientras numero >= 3 Hacer
		Escribir Sin Saltar numero, " "
		// Decrementar el valor para el siguiente termino
		numero <- numero - 4
	Fin Mientras
FinAlgoritmo
```

∗ Ejercicio 6

Diseñar un algoritmo que imprima la siguiente serie de números de forma descendente: 3 7 11 15 19 23 27 31 35 39, se mostrarían así: 39 35 31 27 23 19 15 11 7 3, usando el bucle Repetir-Hasta.

Algoritmo 2.65: Pseudocodigo PSeInt: Imprimir serie numérica descendente con Repetir-Hasta (Decremento de 4)

```
Algoritmo Imprimir_Serie_Descendente_Repetir
	Definir numero Como Entero
	// 'numero' es quien almacena el valor actual
	Escribir "--- Serie de valores descendente: 39, 35, ..., 3 ---"
	// 1. Inicializar el primer valor de la serie
	numero <- 39
	// 2. Se detiene cuando el valor de 'numero' es menor que 3
	Repetir
		Escribir Sin Saltar numero, " "
		// Decrementar el valor para el siguiente termino
		numero <- numero - 4
	// 3. Condicion de Salida
	Hasta Que numero < 3
FinAlgoritmo
```

∗ Ejercicio 7

Diseñar un algoritmo que imprima la serie de números 8 11 14 17 20 23 26 29 Mostrar la salida indicando en la parte izquierda el número de iteración ejemplo: número 1 = 8, número 2 = 11 (Primera forma).

Algoritmo 2.66: Pseudocodigo PSeInt: Imprimir serie numérica (Incremento de 3) con Contador

```
Algoritmo Imprimir_Serie_Contador
      // Declarar variables de proceso
      // 'termino' almacena el valor actual de la serie (8, 11, ...)
      Definir termino Como Entero
      // 'iteracion' almacena el numero de la posicion (1, 2, ...)
      Definir iteracion Como Entero
      Escribir "--- Serie de valores: 8, 11, ..., 29 ---"
      // 1. Inicializar el contador de iteraciones
      iteracion <- 1
      // 2. Proceso
      // Recorrer los valores de 8 hasta 29 con un incremento de 3
      Para termino = 8 Hasta 29 Con Paso 3 Hacer
            Escribir "Numero ", iteracion, " = ", termino
            iteracion <- iteracion + 1
      Fin Para
FinAlgoritmo
```

∗ Ejercicio 8

Diseñar un algoritmo que imprima la serie de números 8 11 14 17 20 23 26 29 Mostrar la salida indicando en la parte izquierda el número de iteración ejemplo: número 1 = 8, número 2 = 11 (Segunda forma).

Algoritmo 2.67: Pseudocodigo PSeInt: Imprimir serie numérica con Contador (Usando Mientras)

```
Algoritmo Imprimir_Serie_Contador_Mientras
      // Declarar variables de proceso
      Definir termino, iteracion Como Entero
      Escribir "--- Serie de valores: 8, 11, ..., 29 ---"
      // 1. Inicializar variables
      termino <- 8
      iteracion <- 1
      // 2. Permanece en el bucle mientras el valor no exceda 29
      Mientras termino <= 29 Hacer
            Escribir "Numero ", iteracion, " = ", termino
            // Actualizar el valor y la iteracion
            termino <- termino + 3
            iteracion <- iteracion + 1
      Fin Mientras
      Escribir "--- Proceso finalizado ---"
FinAlgoritmo
```

∗ Ejercicio 9

Diseñar un algoritmo que muestre la serie de números: 2 7 12 17 22 27 32. Mostrar la salida indicando en la parte izquierda la cantidad de repetición.

Algoritmo 2.68: Pseudocodigo PSeInt: Imprimir serie numérica (Incremento de 5) con Contador

```
Algoritmo Imprimir_Serie_Contador_5
      // Declarar variables de proceso
      // 'termino' almacena el valor actual de la serie (2, 7, ...)
      Definir termino Como Entero
```

```
      // 'iteracion' almacena el numero de la repeticion (1, 2, ...)
      Definir iteracion Como Entero
      Escribir "--- Serie de valores: 2, 7, 12, ..., 32 ---"
      // 1. Inicializar el contador de repeticiones
      iteracion <- 1
      // 2. Proceso (Usando el bucle Para para generar la serie)
      // Recorrer los valores de 2 hasta 32 con un incremento de 5
      Para termino = 2 Hasta 32 Con Paso 5 Hacer
            // A. Salida de resultado
            Escribir "Repeticion ", iteracion, " = ", termino
            // B. Incrementar el contador
            iteracion <- iteracion + 1
      Fin Para
FinAlgoritmo
```

* Ejercicio 10

Diseñar un algoritmo que dado un número ingresado por teclado, determine la tabla de multiplicar de dicho número desde 0 hasta 12.

Algoritmo 2.69: Pseudocodigo PSeInt: Generar tabla de multiplicar

```
Algoritmo Tabla_Multiplicar
      // Declarar variables de proceso
      Definir num_tabla, contador, resultado Como Entero
      Escribir "--- Generar Tabla de Multiplicar (0 a 12) ---"
      // 1. Lectura de datos
      Escribir "Ingrese el numero del cual desea la tabla:"
      Leer num_tabla
      // 2. Proceso (Uso de estructura Para)
      // Recorrer los valores del multiplicador desde 0 hasta 12
      Para contador = 0 Hasta 12 Con Paso 1 Hacer
            // A. Calcular el resultado
            resultado <- num_tabla * contador
            // B. Salida de resultado: Mostrar la operacion y el resultado
            Escribir num_tabla, " x ", contador, " = ", resultado
      Fin Para
FinAlgoritmo
```

* Ejercicio 11

La empresa eléctrica desea calcular el consumo generado por sus abonados. Por cada uno de ellos se ingresará el código del abonado y el consumo de electricidad en Kw/h. Se pide: a) Ingresar abonados o clientes hasta que se ingrese una marca de fin de datos, en la cual el código del abonado sea – 99. b) Mostrar el listado de los abonados, el consumo en Kw/h, y también mostrar el consumo en dólares, pero teniendo en cuenta que, si el consumo ingresado es mayor a 125 Kw/h cobrarlo a 6 centavos, en otro caso cobrarlo a 4 centavos. c) Mostrar el número de abonados ingresados. d) Mostrar el consumo de Kw/h de todos los abonados. e) Mostrar el total de cobro que ha realizado la empresa eléctrica.

Algoritmo 2.70: Pseudocodigo PSeInt: Cálculo de consumo eléctrico y cobro

```
Algoritmo Consumo_Electrico
      // Declarar variables de control y acumuladores
      Definir codigo_abonado, num_abonados Como Entero
      Definir consumo_kwh, cobro_abonado Como Real
   Definir total_cobro, total_kwh Como Real
      // Declarar constantes para tarifas
      Definir TARIFA_ALTA Como Real
```

```
8         TARIFA_ALTA <- 0.06 // $0.06 por kWh
9         Definir TARIFA_BAJA Como Real
10        TARIFA_BAJA <- 0.04 // $0.04 por kWh
11        Definir LIMITE_TARIFA Como Real; LIMITE_TARIFA <- 125
12        // 1. Inicializar contadores y acumuladores
13        num_abonados <- 0
14        total_cobro <- 0.0
15        total_kwh <- 0.0
16        Escribir "--- INICIO DE INGRESO DE ABONADOS ---"
17        Escribir "Ingrese el codigo del abonado (-99 para finalizar):"
18        Leer codigo_abonado
19        // 2. Proceso (Uso de estructura Mientras para ingreso de datos)
20        // Permanecer en el bucle mientras
21        // el codigo NO sea la marca de fin (-99)
22        Mientras codigo_abonado <> -99 Hacer
23              // A. Leer los datos del abonado actual
24              Escribir "Ingrese el consumo de electricidad en Kw/h:"
25              Leer consumo_kwh
26              // B. Determinar la tarifa
27              // y Calcular el cobro (Usando Si-Sino)
28              Si consumo_kwh > LIMITE_TARIFA Entonces
29                    // Consumo mayor a 125 kWh -> Tarifa alta ($0.06)
30                    cobro_abonado <- consumo_kwh * TARIFA_ALTA
31              SiNo
32                    // Consumo menor o igual a
33              // 125 kWh -> Tarifa baja ($0.04)
34                    cobro_abonado <- consumo_kwh * TARIFA_BAJA
35              Fin Si
36              // C. Acumular y contar
37              num_abonados <- num_abonados + 1
38              // Contar el abonado ingresado
39              total_cobro <- total_cobro + cobro_abonado
40              // Acumular el cobro total
41              total_kwh <- total_kwh + consumo_kwh
42              // Acumular el consumo total
43              // D. Mostrar el listado del abonado procesado (Punto b)
44              Escribir "-----------------------------------"
45              Escribir "Abonado: ", codigo_abonado
46              Escribir "Consumo (Kw/h): ", consumo_kwh
47              Escribir "Cobro (Dolares): $", cobro_abonado
48              Escribir "-----------------------------------"
49              // E. Leer el siguiente codigo para continuar o salir
50              Escribir "Ingrese el codigo del siguiente abonado" Sin Saltar
51           Escribir "(-99 para finalizar):"
52              Leer codigo_abonado
53        Fin Mientras
54        // 3. Salida de resultados finales (Puntos c, d, e)
55        Escribir "--- RESUMEN DE LA EMPRESA ELECTRICA ---"
56        Escribir "c) Numero de abonados ingresados: ", num_abonados
57        Escribir "d) Consumo total de Kw/h de todos los abonados: "
58        Escribir total_kwh, " Kw/h"
59        Escribir "e) Total de cobro realizado por la empresa: $"
60        Escribir total_cobro
61  FinAlgoritmo
```

∗ Ejercicio 12

Diseñar un algoritmo que presente la serie de Fibonacci, permitiendo solicitar al usuario un valor que representa la cantidad límite n. Por ejemplo, si se ingresa 10, se mostrará: 0 1 1 2 3 5 8 13 21 34. Para esto las variables iniciales serán a = -1, b = 1, con la fórmula c = a + b, n repetidamente.

Algoritmo 2.71: Pseudocodigo PSeInt: Generar Serie de Fibonacci hasta N terminos

```
Algoritmo Serie_Fibonacci
	// Declarar variables
	Definir a, b, c, n, contador Como Entero
	Escribir "--- Generar Serie de Fibonacci ---"
	// 1. Lectura de datos
	Escribir "Ingrese la cantidad limite de terminos (N):"
	Leer n
	// 2. Inicializar los valores base: a=-1, b=1
	a <- -1
	b <- 1
	// 3. Imprimir la cabecera
	Escribir "Serie de Fibonacci (", n, " terminos):"
	// 4. Proceso
	// Recorrer el bucle N veces para generar los N terminos
	Para contador = 1 Hasta n Con Paso 1 Hacer
		// A. Calcular el siguiente termino de la serie
		c <- a + b
		// B. Imprimir el termino actual
		Escribir Sin Saltar c, " "
		// C. Actualizar los valores base
		a <- b // El valor antiguo de b pasa a ser el nuevo a
		b <- c // El nuevo termino c pasa a ser el nuevo b
	Fin Para
FinAlgoritmo
```

∗ Ejercicio 13

Diseñar un algoritmo que presente la serie de Ulam, para esto, se solicita al usuario un número n. Y la generación de la serie empieza analizando el número, si es par se lo divide para 2 y si es impar se multiplica por 3 y luego se suma 1. Todo repetidamente hasta llegar a 1.

Análisis: La serie sigue dos reglas condicionales: Si N es par: $N \leftarrow N/2$. Si N es impar: $N \leftarrow 3 \times N + 1$

Algoritmo 2.72: Pseudocodigo PSeInt: Generar serie de Ulam o Collatz

```
Algoritmo Serie_Ulam
	// Declarar variables
	Definir n, contador Como Entero
	// n es el numero actual en la serie
	Escribir "--- Generar Serie de Ulam o Collatz ---"
	// 1. Lectura de datos
	Escribir "Ingrese un numero entero positivo (n)"
	Escribir " para iniciar la serie:"
	Leer n
	// 2. Imprimir el valor inicial
	Escribir "Serie generada: "
	Escribir Sin Saltar n, " "
	// 3. Proceso (Uso de estructura Mientras)
	// Permanecer en el bucle mientras
	// el valor actual sea mayor que 1
```

```
16        Mientras n > 1 Hacer
17              // A. Verificar la paridad (Uso de Si-Sino)
18              Si n MOD 2 == 0 Entonces
19                    // El valor es par: Dividir para 2
20                    n <- n / 2
21              SiNo
22                    // El valor es impar: Multiplicar por 3 y sumar 1
23                    n <- (3 * n) + 1
24              Fin Si
25              // B. Imprimir el nuevo termino
26              Escribir Sin Saltar n, " "
27        Fin Mientras
28  FinAlgoritmo
```

∗ Ejercicio 14

Diseñar un algoritmo que presente la serie de números primos. Para esto el usuario ingresa un valor n que representa la cantidad de números a generar. Un número primo es aquel que es divisible para sí mismo y para la unidad.

Algoritmo 2.73: Pseudocodigo PSeInt: Generar N números Primos

```
1   Algoritmo Generar_Primos
2         // Declarar variables
3         Definir n_limite, contador_primos, numero_actual Como Entero
4         Definir divisor, residuo, iter Como Entero
5         Definir es_primo Como Logico
6         Escribir "--- Generar N Numeros Primos ---"
7         // 1. Lectura de datos
8         Escribir "Ingrese la cantidad (N) de numeros primos a generar:"
9         Leer n_limite
10        // 2. Inicializar contadores y el numero a evaluar
11        contador_primos <- 0
12        numero_actual <- 1 // Empezar a buscar desde 1 o 2
13        Escribir "Los primeros ", n_limite, " numeros primos son:"
14        // 3. Proceso (Uso de estructura para contar primos)
15        // Permanecer en el bucle mientras
16        // no se haya alcanzado el limite N
17        Mientras contador_primos < n_limite Hacer
18              // A. Incrementar el numero a evaluar
19              numero_actual <- numero_actual + 1
20              // B. Verificar Primalidad: Asumir que el numero es primo
21              es_primo <- Verdadero
22              // C. Los numeros menores o iguales a 1 no son primos
23              Si numero_actual <= 1 Entonces
24                    es_primo <- Falso
25              SiNo
26                    // D. Bucle anidado para verificar divisores
27                    // (desde 2 hasta raiz de N)
28                    // La verificacion solo necesita llegar hasta
29                    // la raiz cuadrada (o hasta numero_actual - 1
30                    // si no se usa la raiz)
31                    iter = TRUNC(RAIZ(numero_actual))
32                 Para divisor = 2 Hasta iter Con Paso 1 Hacer
33                          residuo <- numero_actual MOD divisor
34                          // Si el residuo es 0, el numero tiene
35                          // otro divisor (no es primo)
36                          Si residuo == 0 Entonces
```

```
                                es_primo <- Falso
                                Romper
                          Fin Si
                    Fin Para
              Fin Si
              // E. Si es primo, contabilizar e imprimir
              Si es_primo Entonces
                    contador_primos <- contador_primos + 1
                    Escribir Sin Saltar numero_actual, " "
              Fin Si
        Fin Mientras
FinAlgoritmo
```

* Ejercicio 15

Elaborar un algoritmo para verificar si un conjunto de cuatro números enteros ingresados por el usuario cumple con la propiedad de ser un Conjunto Diofántico con una constante fija de 1.

Análisis: Un conjunto $\{a, b, c, d\}$ es Diofántico si, para cualquier par distinto (x, y) del conjunto, se cumple:

$$x \cdot y + 1 = k^2$$

donde k es un entero.

Algoritmo 2.74: Pseudocodigo PSeInt: Verificación de conjunto Diofántico

```
Algoritmo Verificar_Conjunto_Diofantico
      // Declarar funcion para verificar
      // si un numero es cuadrado perfecto
      Funcion es_cuadrado_perfecto(num_entrada)
            Definir raiz Como Real
            Definir es_perfecto Como Logico
            // Si el numero es negativo,
            // no es un cuadrado perfecto real
            Si num_entrada < 0 Entonces
                  es_perfecto <- Falso
            SiNo
                  // 1. Calcular la raiz cuadrada
                  raiz <- RAIZ(num_entrada)
                  // 2. Verificar si la parte truncada
                  // es igual al valor real
                  // Si son iguales, la raiz es entera
                  // y es un cuadrado perfecto
                  Si TRUNC(raiz) == raiz Entonces
                        es_perfecto <- Verdadero
                  SiNo
                        es_perfecto <- Falso
                  Fin Si
            Fin Si
            Retornar es_perfecto
      Fin Funcion
      // -----------------------------------------------------
      // Declarar variables principales
      Definir a, b, c, d Como Entero
      Definir producto_1 Como Entero
      Definir es_diofantico Como Logico
      Escribir "--- Conjunto Diofantico (Constante +1) ---"
      // 1. Lectura de datos: Ingresar los cuatro numeros
      Escribir "Ingrese el primer numero (a):"
```

```
34        Leer a
35        Escribir "Ingrese el segundo numero (b):"
36        Leer b
37        Escribir "Ingrese el tercer numero (c):"
38        Leer c
39        Escribir "Ingrese el cuarto numero (d):"
40        Leer d
41        // 2. Proceso: Inicializar la bandera de verificacion
42        es_diofantico <- Verdadero
43        // 3. Evaluar las 6 combinaciones de pares (Uso del operador Y)
44        // Par 1: a * b + 1
45        producto_mas_uno <- a * b + 1
46        es_diofantico <- es_diofantico Y es_cuadrado_perfecto(producto_1)
47        // Par 2: a * c + 1
48        producto_mas_uno <- a * c + 1
49        es_diofantico <- es_diofantico Y es_cuadrado_perfecto(producto_1)
50        // Par 3: a * d + 1
51        producto_mas_uno <- a * d + 1
52        es_diofantico <- es_diofantico Y es_cuadrado_perfecto(producto_1)
53        // Par 4: b * c + 1
54        producto_mas_uno <- b * c + 1
55        es_diofantico <- es_diofantico Y es_cuadrado_perfecto(producto_1)
56        // Par 5: b * d + 1
57        producto_mas_uno <- b * d + 1
58        es_diofantico <- es_diofantico Y es_cuadrado_perfecto(producto_1)
59        // Par 6: c * d + 1
60        producto_mas_uno <- c * d + 1
61        es_diofantico <- es_diofantico Y es_cuadrado_perfecto(producto_1)
62        // 4. Salida de resultados
63        Escribir "--- Resultado de la Verificacion ---"
64        Si es_diofantico Entonces
65              Escribir "El conjunto {", a, ", ", b, ", ", c, ", ", d, "}"
66           Escribir "SI es un Conjunto Diofantico."
67        SiNo
68              Escribir "El conjunto {", a, ", ", b, ", ", c, ", ", d, "}"
69           Escribir "NO es un Conjunto Diofantico."
70        Fin Si
71  FinAlgoritmo
```

* Ejercicio 16

Diseñar un algoritmo que le permita al usuario ingresar un valor entero N, el cual representa la cantidad de términos que desea generar de la sucesión de números cuadrados perfectos (1, 4, 9, 16, 25, ...) por medio de la suma acumulativa de los números impares consecutivos.

Algoritmo 2.75: Pseudocodigo PSeInt: Serie de sumas de impares (Números cuadrados)

```
1   Algoritmo Generar_Cuadrados_Impares
2         // Declarar variables de proceso
3         Definir n_limite, contador Como Entero
4         Definir impar_actual, suma_acumulada Como Entero
5         Escribir "--- Generar Numeros Cuadrados Perfectos ---"
6         // 1. Lectura de datos
7         Escribir "Ingrese la cantidad (N) de numeros cuadrados a generar:"
8         Leer n_limite
9         // 2. Inicializar acumuladores y generadores
10        suma_acumulada <- 0 // Acumular la suma de impares
11        impar_actual <- 1 // El primer impar a sumar
```

```
      Escribir "Resultados de la suma:"
      // 3. Proceso (Uso de estructura Para para generar N terminos)
      // Recorrer el bucle N veces
      Para contador = 1 Hasta n_limite Con Paso 1 Hacer
            // A. Sumar el impar actual al acumulador
         // (genera el numero cuadrado)
            suma_acumulada <- suma_acumulada + impar_actual
            // B. Salida de resultado
            Escribir "Suma de los primeros ", contador Sin Saltar
      Escribir " impares = ", suma_acumulada, " (", contador, "^2)"
            // C. Actualizar el impar para la siguiente iteracion
         // (incremento de 2)
            impar_actual <- impar_actual + 2
      Fin Para
FinAlgoritmo
```

* Ejercicio 17

Elaborar un algoritmo que solicite al usuario dos valores enteros, uno que será la base y otro que será el exponente. Luego genere la potencia del número.

Algoritmo 2.76: Pseudocodigo PSeInt: Cálculo de Potencia

```
Algoritmo Calcular_Potencia
      // Declarar variables de proceso
      Definir base, exponente, contador Como Entero
      Definir resultado Como Real
      // Usar Real para manejar posibles resultados grandes
      Escribir "--- Calculo de la Potencia (Base ^ Exponente) ---"
      // 1. Lectura de datos
      Escribir "Ingrese la base (valor entero):"
      Leer base
      Escribir "Ingrese el exponente (valor entero):"
      Leer exponente
      // 2. Inicializar el resultado a 1
      // La multiplicacion debe comenzar en 1
      resultado <- 1
      // 3. Proceso (Simulacion de la potencia mediante un bucle Para)
      // Multiplicar la 'base' por si misma tantas veces
      // como indique el 'exponente'
      // Manejar caso especial: Exponente negativo
      Si exponente < 0 Entonces
            // Para manejar potencias negativas
            // Usamos el valor absoluto del exponente
            Para contador = 1 Hasta ABS(exponente) Con Paso 1 Hacer
                  resultado <- resultado * base
            Fin Para
            // Invertir el resultado
            resultado <- 1 / resultado
      // Manejar caso especial: Exponente cero
      Sino Si exponente == 0 Entonces
            // Cualquier numero elevado a 0 es 1 (si base no es 0)
            Si base == 0 Entonces
                  Escribir "ERROR: Indeterminacion (0^0)."
                  resultado <- 0 // O asignar valor de error
            SiNo
                  resultado <- 1
            Fin Si
```

```
36        // Manejar caso estandar: Exponente positivo
37        Sino
38              Para contador = 1 Hasta exponente Con Paso 1 Hacer
39                    resultado <- resultado * base
40              Fin Para
41        Fin Si
42        // 4. Salida de resultado
43        Si (exponente <> 0) O (base <> 0) Entonces
44              Escribir "El resultado de ", base, " elevado a ",
45              Escribir "exponente, " es: ", resultado
46        Fin Si
47        Escribir "--- Proceso finalizado ---"
48  FinAlgoritmo
```

∗ Ejercicio 18

Elaborar un algoritmo que solicite al usuario un valor entero. Luego genere el factorial del número.

Algoritmo 2.77: Pseudocodigo PSeInt: Cálculo del Factorial (N!)

```
1   Algoritmo Calcular_Factorial
2         // Declarar variables de proceso
3         Definir n, contador Como Entero
4         Definir factorial Como Real
5         // Usar Real para manejar factoriales grandes
6         Escribir "--- Factorial (N!) ---"
7         // 1. Lectura de datos
8         Escribir "Ingrese un numero entero positivo (N):"
9         Leer n
10        // 2. Inicializar el factorial en 1
11        // El factorial de 0 es 1, y la multiplicacion debe comenzar en 1
12        factorial <- 1
13        // 3. Proceso y Verificacion (Usando Si y Para)
14        Si n < 0 Entonces
15              Escribir "ERROR: El factorial solo esta definido"
16              Escribir "para numeros positivos o cero."
17        SiNo Si n == 0 Entonces
18              // Caso base: 0! = 1
19              factorial <- 1
20        SiNo
21              // Recorrer el bucle desde 1 hasta N para multiplicar
22              Para contador = 1 Hasta n Con Paso 1 Hacer
23                    // Multiplicar el acumulador por el valor del contador
24                    factorial <- factorial * contador
25              Fin Para
26        Fin Si
27        // 4. Salida de resultado
28        Si n >= 0 Entonces
29              Escribir "El factorial de ", n, " (", n, "!) es: ", factorial
30        Fin Si
31        Escribir "--- Proceso finalizado ---"
32  FinAlgoritmo
```



Al-Khwārizmī, Ada Lovelace y Alan Turing

¡Buen trabajo!

Finalización del capítulo 2

DOI: 10.17163/abyaups.167.3

Capítulo 3
Lenguaje de Programación

Resultado de aprendizaje

Desarrolla aplicaciones utilizando el lenguaje de programación.

Indicador de logro

Identifica la sintaxis y semántica en un lenguaje de programación.
Diseña e implementa programas en un lenguaje de programación.

Un lenguaje de Programación es un conjunto de símbolos y códigos usados para orientar el flujo de una solución con base en las estructuras de control propias, durante el desarrollo o implementación de algoritmos. Conocer cómo funciona un lenguaje de Programación en el computador es importante para mejorar procesos reales que hagan una diferenciación competitiva en todo ámbito. En los últimos años, los lenguajes de programación han permitido mejores desarrollos (aplicaciones de escritorio, aplicativos en la web, aplicación en dispositivos), aumentado su potencia y flexibilidad para llevar a cabo tareas complejas que la innovación exige. El índice de la Comunidad de Programación TIOBE[1] es un indicador de la popularidad de los lenguajes de programación, ver Fig. 3.1.

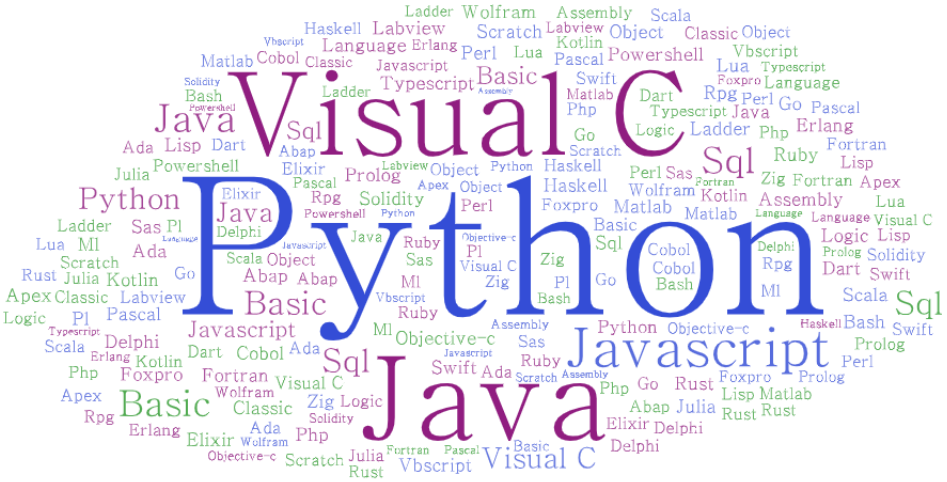


Figura 3.1: Los lenguajes de programación del índice TIOBE expresados en una nube de palabras.

[1] TIOBE Quality Indicator Quality Model

3.1. Fundamentos a los lenguajes de desarrollo

Hoy en día muchos dispositivos y máquinas requieren de un lenguaje de programación para cumplir con sus funciones. Si se conoce como poder utilizarlos adecuadamente, estos permitirán alcanzar los objetivos en menos tiempo y poder alcanzar mayor productividad. Así, un lenguaje de programación, es el conjunto de instrucciones a través del cual los humanos programan e interactúan con las computadoras. Un lenguaje de programación nos permite establecer una forma de comunicarnos con las máquinas a través de instrucciones escritas en una sintaxis que representa el algoritmo y la computadora lo entiende e interpreta en lenguaje de máquina.

Los programadores utilizan un lenguaje de programación para implementar una idea a partir de un algoritmo, la pragmática. El lenguaje de programación permite al computador procesar de forma rápida y eficientemente grandes y complejas cantidades de información, la semántica. Por ejemplo, si a una persona se le da una lista de números aleatorios que van de uno a diez mil y se le pide que los coloque en orden ascendente, es probable que tome una cantidad considerable de tiempo e incluya algunos errores, mientras que si le das la misma instrucción a una computadora utilizando un lenguaje de programación, podrás obtener la respuesta en unos cuantos segundos y sin errores.

El uso de un lenguaje de programación, requiere entonces de la comprensión desde tres perspectivas. Desde la sintaxis: que es el conjunto de símbolos y reglas para formar sentencias y así su escritura. Desde la semántica: que son las reglas que permiten transformar las sentencias en instrucciones lógicas. Desde la pragmática: que son aquellas construcciones particulares del lenguaje que van estableciendo un estilo de programar. Así como en nuestro lenguaje natural, las letras forman palabras y estas forman oraciones, luego un estilo de escribir y redactar. En los lenguajes de programación, los caracteres forman sentencias que en su conjunto forman las instrucciones, luego un programador va desarrollando un pensamiento algorítmico para establecer buenas prácticas al programar, ver Fig. 3.2.

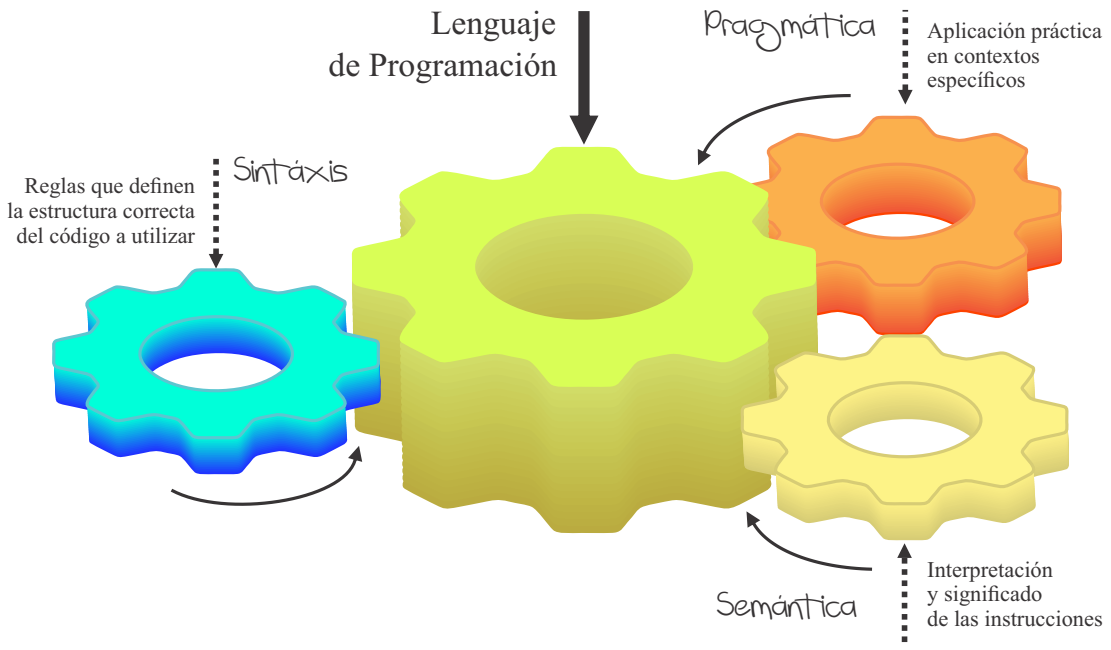


Figura 3.2: Características de un lenguaje de programación.

3.1.1. Introducción a los lenguajes de desarrollo

Los lenguajes de desarrollo, también conocidos como lenguajes de programación, son herramientas necesarias en el mundo de la computación, de los sistemas tecnológicos, de la informática en general. Estos lenguajes se basan en un conjunto de conceptos que definen la utilidad de elementos como las variables y sus estructuras (simples y compuestas), los operadores (matemáticos o aritméticos, lógicos y relacionales), las estructuras de control (secuenciales, selectivas o condicionales y, repetitivas o bucles) y funciones (subrutinas), que forman la base de cualquier programa, aplicativo o desarrollo.

Definición general

Un lenguaje de programación se define como un conjunto de instrucciones y reglas sintácticas que permiten a los humanos comunicarse con las computadoras para resolver problemas, tareas o cálculos, y automatizarlas.

Sin estos elementos característicos, sería complejo crear software correcto y eficiente. Por ejemplo, las variables compuestas permiten almacenar datos, mientras que los operadores aritméticos y lógicos realizan cálculos matemáticos, razonables y coherentes respectivamente, lo que facilita la manipulación datos a información en tiempo real, ver Fig. 3.3.

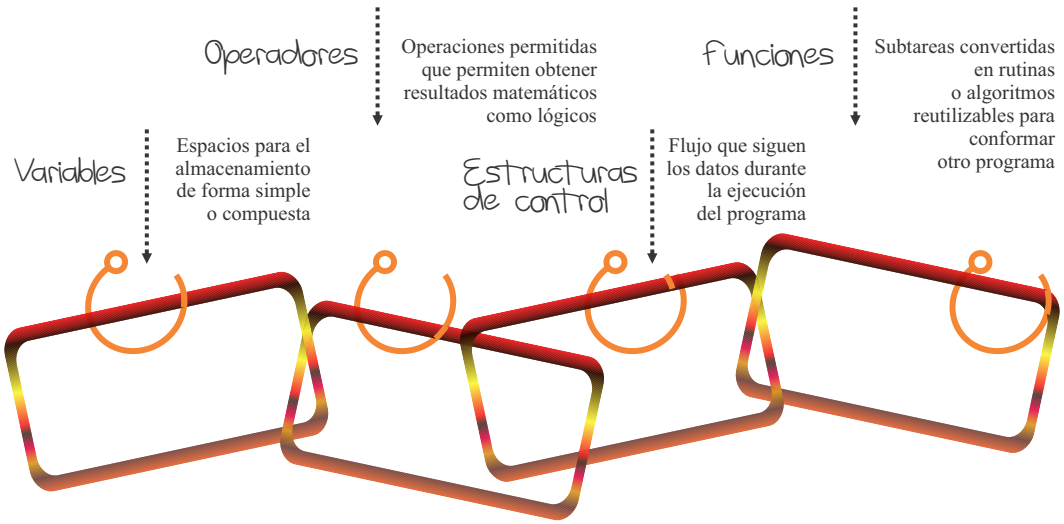


Figura 3.3: Elementos fundamentales de un lenguaje de programación.

Otro aspecto importante es la comprensión de la lógica de programación, que implica descomponer problemas complejos en otros más simples, así como realizar pasos simples y secuenciales, es decir "divide y vencerás". Esto se relaciona con el paradigma de la programación estructurada, de Edsger Dijkstra [Dijkstra, 1974], quien resalta la importancia de evitar código desorganizado [Randell, 2003]. Además, hay que tener en cuenta conceptos como la depuración (identificación y corrección de errores) y la optimización de código, que aseguran que la solución algorítmica sea funcional y eficiente en términos de respuesta (tiempo) y elementos computacionales (recursos del sistema) [First et al., 2025]. Estos aspectos aplican a todos los lenguajes, independientemente de su complejidad o propósito, formando la base para cualquier profesional, desarrollador, principiante o experto [Jäger et al., 2025].

Es así como los lenguajes de desarrollo, surgieron en la década de 1950 con la llegada de las primeras computadoras digitales [Baker, 1972]. El lenguaje FORTRAN, desarrollado por IBM en 1957 [Van Snyder, 2007; Backus, 1978], fue uno de los pioneros, diseñado específicamente para

cálculos científicos y matemáticos. Posteriormente, lenguajes como COBOL (para aplicaciones empresariales) [Sammet, 1978] y BASIC (para educación) ampliaron su disponibilidad [Newell et al., 1957]. Hoy en día, el uso de un lenguaje de programación es indispensable en campos como el desarrollo web, la inteligencia artificial, la investigación espacial y los videojuegos [Carver et al., 2025]. Su evolución ha sido impulsada por la necesidad de mayor abstracción, permitiendo a los programadores enfocarse en la lógica en lugar de en los detalles del hardware [Von Neumann & Kurzweil, 2012].

3.1.2. Clasificación de los lenguajes de desarrollo

Los lenguajes de desarrollo se clasifican dependiendo de diferentes criterios como el nivel de abstracción, el paradigma de programación o el propósito específico. En cuanto al nivel de abstracción, se dividen en lenguajes de bajo nivel (como el ensamblador, cercanos al código máquina) y de alto nivel (como Python, C o Java, más cercanos al lenguaje humano). Los de bajo nivel ofrecen mayor control sobre el hardware, pero son más difíciles de aprender, mientras que los de alto nivel priorizan la productividad y la legibilidad, y su nivel de aprendizaje es adecuado, ver Fig. 3.4.

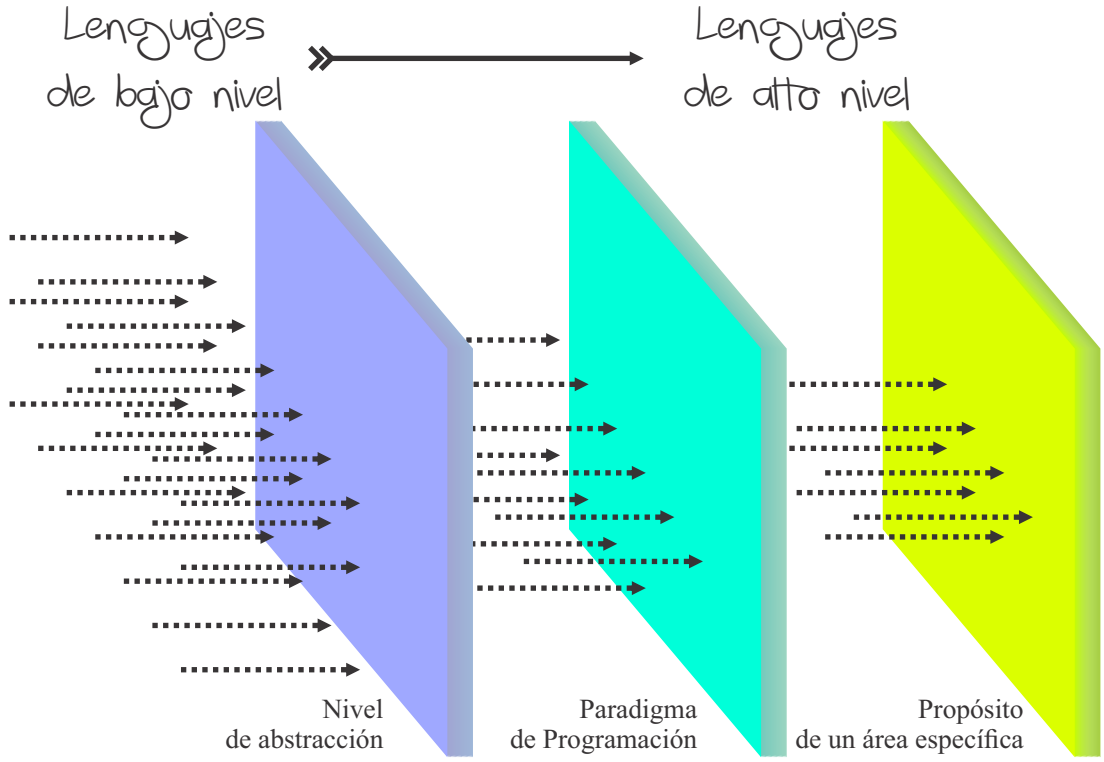


Figura 3.4: Clasificación de los lenguajes de programación.

Cuando se habla de la clasificación de los lenguajes de programación por paradigmas, se hace referencia a los lenguajes con características procedimentales (como el lenguaje C, enfocados en procedimientos y funciones), los lenguajes orientados a objetos - POO - (como Java, que usan clases y objetos para modelar el mundo real), funcionales (como Haskell, basados en funciones matemáticas puras) y declarativos (como SQL - *Structured Query Language* - o lenguaje de consulta estructurado, donde se describe el resultado deseado en lugar de los pasos).

Además, existen lenguajes multiparadigma, como Python, que combinan varios enfoques. Gracias a esta diversidad es necesario saber elegir el lenguaje adecuado según el proyecto a desarrollar, desde aplicaciones móviles hasta sistemas embebidos.

Los paradigmas de programación son enfoques o estilos fundamentales para estructurar y resolver problemas en el código. Cada uno enfatiza diferentes conceptos, como procedimientos, objetos, funciones puras o reglas lógicas. A continuación, se presentan los principales paradigmas con ejemplos de lenguajes representativos. Muchos lenguajes modernos son multiparadigma, lo que significa que soportan varios enfoques simultáneamente.

El paradigma *Imperativo o Procedimental*, se centra en describir cómo realizar tareas mediante secuencias de comandos que cambian el estado del programa. Ejemplos clásicos incluyen C, Pascal, Fortran y BASIC. Estos lenguajes usan procedimientos o funciones para organizar el código.

El paradigma *Orientado a Objetos - OOP*, modela el mundo real mediante objetos que encapsulan datos y comportamientos. Los ejemplos más destacados son Java, C++, Python, C# y Ruby. Conceptos clave como herencia, polimorfismo y encapsulamiento son centrales aquí.

El paradigma *Funcional* trata la computación como evaluación de funciones matemáticas, evitando estados mutables y efectos secundarios para favorecer su naturaleza. Ejemplos puros incluyen Haskell y Lisp (en su variante Scheme). Otros con soporte fuerte son Scala, Erlang, F# y Clojure. Lenguajes como Python y JavaScript también incorporan elementos funcionales.

El paradigma *Declarativo* se enfoca en describir qué se quiere lograr, no cómo hacerlo paso a paso. Incluye subparadigmas como el lógico (basado en hechos y reglas) con ejemplos como Prolog; y el de bases de datos con SQL. Otros declarativos son HTML (HyperText Markup Language) o CSS (Cascading Style Sheets), para estructurar y estilar web, o lenguajes como Mercury.

Otros paradigmas, como el paradigma *concurrente o paralelo* destaca en lenguajes como Go (goroutines) o Rust (propiedad y borrowing para seguridad). Asimismo, el paradigma *basado en eventos* es común en JavaScript (para web asíncrona). Lenguajes multiparadigma como Python, JavaScript, Scala o Kotlin combinan varios de estos enfoques, ofreciendo flexibilidad para diferentes tipos de proyectos, ver Fig. 3.5.

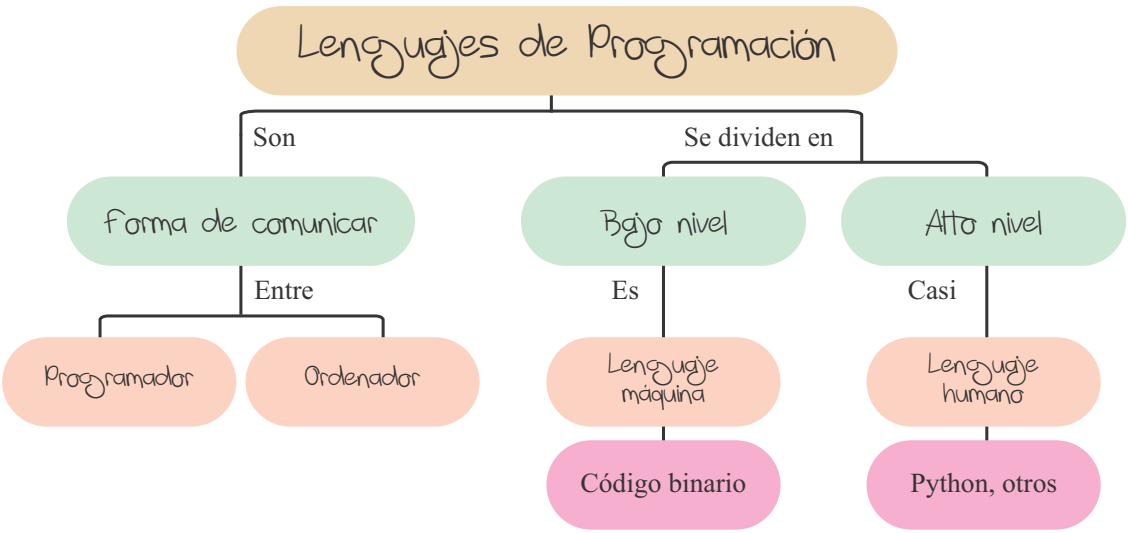


Figura 3.5: Clasificación de los lenguajes de programación.

3.2. Introducción a Python

Con el paso del tiempo, se han visto en crecimiento un gran número de lenguajes de desarrollo y con ello una secuencia de herramientas para los desarrolladores. Pocos lenguajes han logrado alcanzar dos ámbitos a nivel mundial, ser una herramienta de estudio para estudiantes y educadores y, ser el motor principal del desarrollo tecnológico para las corporaciones.

Python, creado a finales de la década de 1980 por el programador holandés Guido van Rossum, representa una singularidad particular, un lenguaje que prioriza la elegancia y la simplicidad humana por encima de la eficiencia ordinaria de la máquina.

El inicio de Python se sitúa en las vacaciones de Navidad de 1989 (según registros en Amazon[2]). Lo que comenzó como un proyecto personal de "pasatiempo" para dar continuidad al lenguaje ABC en el Centrum Wiskunde & Informatica (CWI), terminó por redefinir los estándares de la industria. Desde su publicación oficial en 1991, Python introdujo conceptos que hoy damos por hecho, como el manejo de excepciones y las clases con herencia, pero lo hizo con una sintaxis tan limpia que parecía pseudocódigo en inglés.

El crecimiento del lenguaje no fue repentino, sino organizado. Su primera gran transformación ocurrió con Python 2.0 en el año 2000, cuando se introdujeron características como la recolección de basura y el soporte para Unicode. Sin embargo, el verdadero punto de cambio de dirección fue su filosofía de código abierto. Al permitir que una comunidad global contribuyera a sus bibliotecas, Python dejó de ser una herramienta de una comunidad especializada para sistemas operativos y se convirtió en un ecosistema modular.

Uno de los acontecimientos más críticos en su historia fue el lanzamiento de Python 3.0 en 2008. Esta versión "limpió" el lenguaje de redundancias, pero no era compatible con las versiones anteriores (Python 2.0). Durante una década, la comunidad estuvo dividida, pero la apuesta por la modernización resultó ser la correcta. Hoy, gracias a esa evolución, Python es la columna vertebral de la revolución de la Inteligencia Artificial. Bibliotecas como NumPy, Pandas y TensorFlow han encontrado en Python el "adhesivo" perfecto para unir algoritmos complejos con interfaces fáciles de usar.

3.2.1. Tipos de datos y variables

En el centro de todo lenguaje de programación reside la capacidad de representar, tratar y procesar información. En Python, esta capacidad es una extensión de su filosofía de diseño. A través de un sistema de variables flexibles y una jerarquía de tipos de datos rica y coherente, Python permite a los desarrolladores pasar de la abstracción de una idea a la concreción del código con una fricción mínima.

En Python, las variables son nombres que se vinculan a objetos mediante el operador de asignación igual a (=). No es necesario declarar el tipo de una variable de antemano, ya que Python es un lenguaje de tipado dinámico (el tipo se determina en tiempo de ejecución según el objeto asignado). Por ejemplo, al escribir x = 5, la variable x se refiere a un objeto de tipo entero (declarativamente como int), y si luego asignamos x = "hola", ahora se refiere a una cadena de caracteres (declarativamente como str). Esta flexibilidad permite cambiar el tipo de una variable en cualquier momento. Las asignaciones múltiples, como a, b = 0, 1, son comunes y evalúan el lado derecho primero antes de vincular las variables.

Los tipos de datos básicos en Python incluyen números y texto. Entre los numéricos destacan los enteros (int), que representan números sin decimales y tienen precisión arbitraria (por ejemplo, 50 o, 2**100 o 2^{100}, para potencias grandes); los flotantes (declarativamente como float), para

[2] Historia de Python en Amazon

números con decimales (como 5.0 o 1.6); y operaciones como la división (/) que siempre devuelve un número real (float), o la división entera (//).

Python maneja automáticamente conversiones en operaciones mixtas, convirtiendo enteros a flotantes cuando es necesario. Las cadenas de texto (str) son secuencias inmutables de caracteres, encerradas entre comillas simples o dobles, y soportan concatenación con +, repetición con *, indexación y rebanado (slicing). Otro aspecto fundamental es la inmutabilidad de ciertos tipos. Las cadenas, números y tuplas son inmutables: una vez creados, no se pueden modificar directamente. Intentar cambiar un elemento de una cadena, como s[0] = "x", genera un error.

En cambio, operaciones como la concatenación crean un nuevo objeto. Esto contrasta con tipos mutables, donde el objeto puede cambiar in situ. Entre los tipos de datos estructurados más importantes se encuentran las listas (list), que son secuencias mutables y ordenadas de elementos heterogéneos. Se definen con corchetes [1, "dos", 3.0] y permiten métodos como append(), extend(), pop() o sort() para modificar su contenido. Las listas son ideales para colecciones que cambian durante la ejecución, como pilas o colas.

Las tuplas (tuple) son similares a las listas, pero inmutables, definidas con paréntesis (1, 2, 3). Se usan para datos fijos o como claves en diccionarios. Los conjuntos (set), creados con llaves 1, 2, 3 o la función set(), almacenan elementos únicos sin orden, y son útiles para operaciones matemáticas como unión o intersección. Finalmente, los diccionarios (dict) son colecciones mutables de pares clave-valor, con llaves nombre: "Ana", edad: 25, donde las claves deben ser inmutables.

Python trata todo como objetos, lo que implica que las variables son referencias a estos objetos. Al asignar una variable a otra se crea un alias (referencias compartidas), por lo que modificar un objeto mutable afecta a todas las referencias. Esta característica, combinada con la diversidad de tipos built-in, hace que Python sea potente y expresivo para manejar datos de manera eficiente.

3.2.2. Operadores

Los operadores en Python actúan como un puente entre la información almacenada y los resultados deseados. Son los componentes o símbolos que permiten que el lenguaje deje de ser una simple lista de valores para convertirse en un sistema lógico capaz de tomar decisiones, realizar cálculos científicos y manipular estructuras de datos masivas.

Los operadores en Python son símbolos especiales que realizan operaciones aritméticas y lógicas, sobre variables y valores, permitiendo manipular datos de manera eficiente. Python ofrece una amplia variedad de operadores clasificados en categorías como aritméticos, de comparación, lógicos, de asignación, bitwise, de identidad y de membresía. Estos operadores son fundamentales para escribir expresiones que forman la base de cualquier programa, desde cálculos simples hasta lógica compleja. Su uso correcto mejora la legibilidad y el rendimiento del código, y Python prioriza la claridad con una precedencia definida (por ejemplo, los aritméticos tienen mayor prioridad que los lógicos).

Los operadores aritméticos son los más básicos y se utilizan para realizar operaciones matemáticas. Incluyen suma (+), resta (-), multiplicación (*), división (/ que siempre devuelve un real, float), división entera (//), módulo (% para el resto) y potenciación (**). Por ejemplo, 5 ** 2, 2^5, resulta en 25, y 10 % 3 da 1. Estos operadores funcionan con números (int y float) y, en algunos casos, con otros tipos como cadenas (para concatenación con + o repetición con *).

Los operadores de comparación evalúan relaciones entre valores y devuelven un booleano (True o False). Son: igualdad (==), desigualdad (!=), mayor (>), menor (<), mayor o igual (>=) y menor o igual (<=). Se usan comúnmente en estructuras de control como (si) en inglés if. Python permite encadenar comparaciones, como $a < b < c$, lo que equivale a $a < b$ and $b < c$, facilitando expresiones más naturales y legibles.

Los operadores lógicos combinan expresiones booleanas: and (verdadero si ambos son verda-

deros), or (verdadero si al menos uno es verdadero) y not (invierte el valor). Estos son esenciales para decisiones complejas. Además, los operadores bitwise operan a nivel de bits: && (AND), || (OR), ^ (XOR), ∼ (NOT), ≪ (desplazamiento izquierdo) y ≫ (desplazamiento derecho). Son útiles en optimizaciones de bajo nivel, como manipulación de flags o compresión de datos.

Otros operadores importantes incluyen los de asignación, como = para asignar valores, y versiones augmentadas como +=, -=, *=, que combinan operación y asignación (por ejemplo, x += 5 es equivalente a x = x + 5). Los operadores de identidad (is y is not) verifican si dos variables referencian el mismo objeto en memoria, diferente de == que compara valores. Finalmente, los de membresía (in y not in) comprueban si un elemento está en una secuencia como listas o diccionarios. La combinación de estos operadores, junto con la precedencia y asociatividad definida en Python, permite crear expresiones fuertes y concisas. Dominar el uso de estas expresiones es importante para avanzar en la programación, desde *scripts* (códigos precompiladores o funcionales) simples hasta aplicaciones avanzadas en datos o *machine learning*.

3.2.3. Estructuras secuenciales

La estructura secuencial se basa en el principio de "paso a paso". En Python, esto se traduce en la ejecución de líneas de código que el intérprete lee de manera descendente. Aunque parezca un concepto trivial, es donde se establecen las reglas de dependencia de un programa. Para procesar una variable, primero debe ser declarada; para mostrar un resultado, primero debe ser calculado. Esta dependencia jerárquica es lo que permite que una computadora, una máquina de estados finitos, transforme electricidad en información útil.

El diseño de una secuencia eficaz requiere claridad de pensamiento. El programador debe actuar como un coreógrafo que decide qué acción precede a la otra para evitar errores de referencia o lógica. Por ejemplo, en Python, la interacción con el usuario a través de input() y la entrega de resultados mediante print() son los "sujetalibros" de una secuencia. Entre ellos ocurre la magia del procesamiento, donde los operadores y las variables que hemos analizado previamente se combinan para transformar la realidad cruda de los datos en conocimiento estructurado, ver Fig. 3.6.

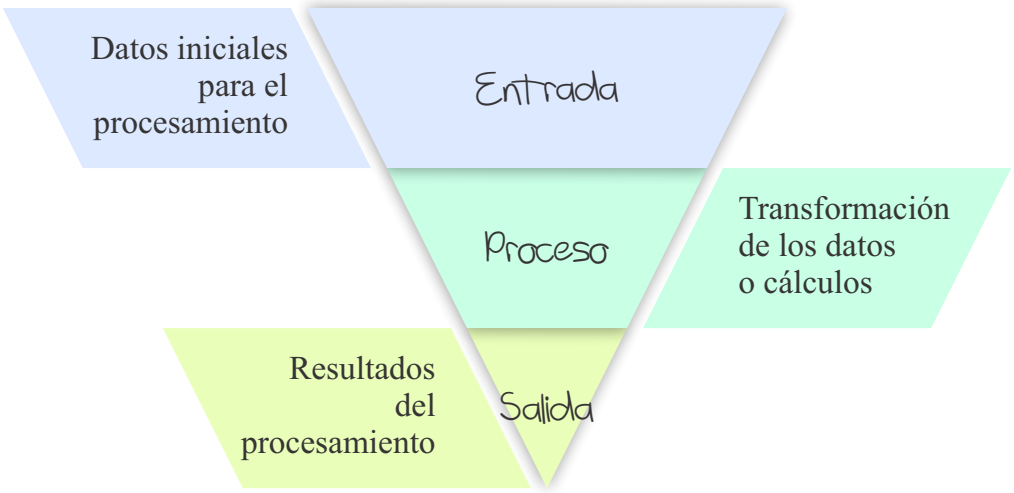


Figura 3.6: Programación secuencial.

Además, la linealidad proporciona una ventaja crítica: la facilidad de depuración (debugging). Al no haber saltos lógicos ni ramificaciones, el flujo de datos es transparente. Esta simplicidad es la que permite a los principiantes construir sus primeras herramientas —como convertidores de

moneda o calculadoras de área— y a los expertos construir los cimientos de sistemas críticos que requieren una ejecución predecible y segura.

Las estructuras secuenciales representan el fundamento más básico del control de flujo en la programación. Consisten en la ejecución lineal y ordenada de instrucciones, una tras otra, sin desviaciones ni repeticiones condicionales. En este paradigma, el programa sigue un camino único desde el inicio hasta el fin, procesando cada sentencia en el orden en que aparece en el código. Esta estructura es ideal para tareas simples como cálculos directos, entrada y salida de datos, o procesos que no requieren decisiones o bucles. En términos conceptuales, la estructura secuencial se basa en tres pasos principales: entrada de datos (obtener información del usuario o fuentes externas), procesamiento (realizar operaciones con esos datos) y salida (mostrar resultados). Por ejemplo, un programa que calcula el área de un rectángulo recibe la base y la altura, multiplica ambos valores y muestra el resultado. Esta simplicidad facilita la comprensión y depuración del código, especialmente para principiantes. En Python, las estructuras secuenciales se implementan de forma natural, ya que el intérprete ejecuta las líneas de código de arriba hacia abajo.

Algoritmo 3.1: Ejemplo básico de Entrada y Salida en Python

```
# 0. Comentar, autor: Joe Llerena, fecha: 30-12-2025
# 1. Solicitar el nombre (input siempre devuelve una cadena de texto)
nombre = input("Ingrese su nombre: ")
# 2. Solicitar la edad y convertirla a un valor entero (int)
edad = int(input("Ingrese su edad: "))
# 3. Mostrar el saludo usando una "f-string"
print(f"Hola {nombre}, tienes {edad} anios.")
# 3. Mostrar otra alternativa
print("Hola", nombre, "tienes", edad, "anios.")
# 4. Mensaje final de bienvenida
print("Bienvenido al programa.")
```

El resultado alcanzado, del programa anterior, se muestra en la imagen siguiente donde se presentan las líneas de código, ver Fig. 3.7.

```
Ingrese su nombre: Joe
Ingrese su edad: 12
Hola Joe, tienes 12 anios.
Hola Joe tienes 12 anios.
```

Figura 3.7: Programación secuencial.

3.2.4. Estructuras de control condicionales

Las estructuras condicionales se representan visualmente mediante diagramas de flujo, donde un rombo simboliza la decisión (condición), con ramas “Sí” y “No” que llevan a procesos rectangulares. Esto facilita el diseño algorítmico antes de codificar.

En Python, la estructura condicional se rige por el principio de la elegancia. El uso de la palabra clave *if* seguida de una expresión lógica es casi idéntico al razonamiento humano: “Si ocurre esto, haz aquello”. Sin embargo, la sofisticación de Python brilla en su manejo de la ambigüedad mediante el *elif* y el *else*. Estas herramientas permiten al programador cubrir todo el espectro de posibilidades, asegurando que el programa nunca se encuentre en un estado de incertidumbre.

Un aspecto revolucionario en la sintaxis de Python es la obligatoriedad de la indentación. En otros lenguajes, un programador podría escribir condicionales desordenados que funcionan, pero son imposibles de leer. En Python, la estructura visual del código refleja directamente su estructura

lógica. Esto reduce drásticamente los errores de "lógica anidada", donde un desarrollador podría perderse en múltiples niveles de decisiones.

Además, las condicionales modernas en Python han evolucionado para manejar la complejidad con mayor eficacia y simplicidad. Con la introducción de características como las expresiones condicionales (el operador ternario) y, más recientemente, el *Structural Pattern Matching* (match-case), el lenguaje demuestra que la toma de decisiones puede ser tan potente como una base de datos y tan clara como un esquema. Estas estructuras no solo controlan el flujo; definen la robustez del sistema, permitiendo que el software maneje excepciones y casos de borde que, en una estructura secuencial pura, causarían el colapso del programa, ver Fig. 3.8.

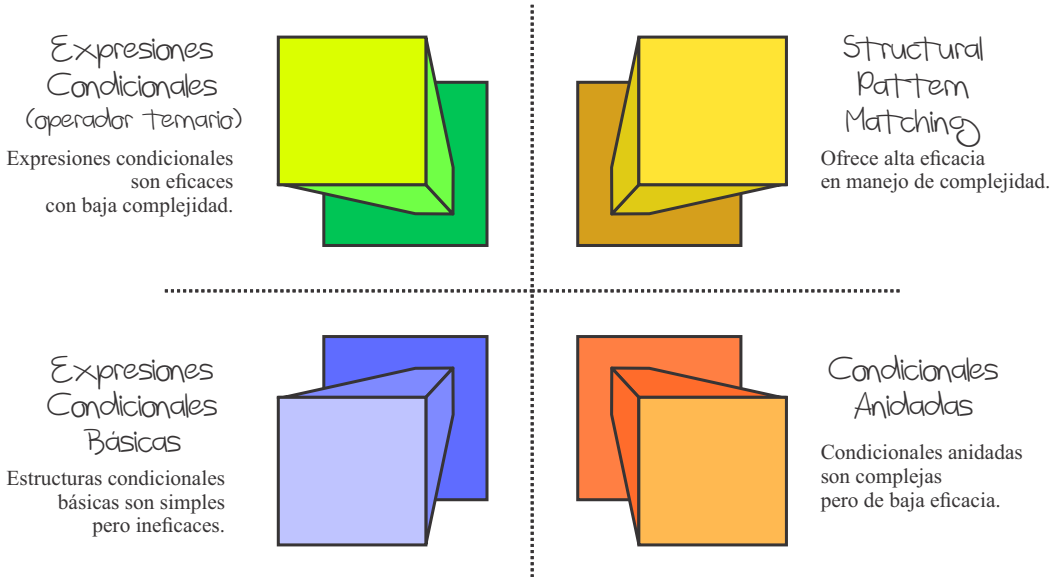


Figura 3.8: Estructuras de control.

3.2.5. Estructuras de control repetitivas

Las estructuras de control repetitivas, también conocidas como bucles o ciclos, permiten ejecutar un bloque de código múltiples veces mientras se cumpla una condición o para un número determinado de iteraciones. Estas estructuras son el tercer pilar fundamental del control de flujo, junto con las secuenciales y condicionales, y son indispensables para automatizar tareas repetitivas, procesar colecciones de datos o implementar algoritmos eficientes. Sin ellas, tareas como sumar elementos de una lista o validar entradas hasta que sean correctas requerirían código redundante y poco mantenible.

Existen tres tipos principales de bucles: el bucle *while (mientras)*, que evalúa la condición antes de ejecutar el bloque y repite mientras sea verdadera; el bucle *for (para)*, ideal para iterar sobre secuencias conocidas como rangos o listas; y el bucle *do-while (hacer-mientras)*, que ejecuta el bloque al menos una vez y luego verifica la condición (aunque no está nativo en Python, se puede emular con un while).

La iteración en Python destaca por su elegancia sintáctica y su versatilidad. El bucle *while* representa la persistencia; es la herramienta del control continuo, fundamental en sistemas que deben mantenerse activos hasta que ocurre un evento específico, como un servidor esperando peticiones o un videojuego procesando la física en cada fotograma. Su riesgo es el "bucle infinito", un error lógico donde la condición nunca se vuelve falsa, pero bajo un diseño cuidadoso, es la base de la estabilidad de muchos sistemas operativos.

Por otro lado, el bucle *for* en Python redefine la manera de interactuar con los datos. Al adoptar un enfoque basado en iteradores, Python permite al programador decir: "Para cada elemento en esta colección, haz esto". Esta abstracción elimina la necesidad de gestionar manualmente contadores o índices complejos (comunes en lenguajes como C o Java), reduciendo así la probabilidad de errores de tipo "fuera de rango". La combinación de *for* con la función range() o con estructuras como listas, convierte a Python en el lenguaje predilecto para la ciencia de datos, donde el procesamiento de filas y columnas es constante.

Finalmente, la capacidad de estas estructuras para anidarse —un bucle dentro de otro— permite resolver problemas multidimensionales, como procesar imágenes (píxel por píxel) o matrices matemáticas. Aunque la potencia es inmensa, la filosofía de Python nos recuerda que "lo simple es mejor que lo complejo", instando a los desarrolladores a utilizar estas herramientas de forma eficiente para evitar el desperdicio de recursos computacionales.

3.2.6. Funciones

Las funciones son uno de los conceptos más importantes en la programación, ya que permiten modularizar el código, reutilizarlo y organizar programas complejos en bloques manejables. Una función es un bloque de código reutilizable que realiza una tarea específica, recibe entradas (parámetros), procesa datos y opcionalmente devuelve un resultado.

Este enfoque sigue el principio *DRY (Don't Repeat Yourself)*, reduciendo la duplicación y facilitando el mantenimiento. Además, las funciones mejoran la legibilidad, permiten depuración independiente y promueven el diseño estructurado o modular. En Python, las funciones se definen con la palabra clave def, seguida del nombre, paréntesis con parámetros opcionales y dos puntos. El bloque de código se indenta, y se usa return para devolver valores.

Definir una función en Python mediante la palabra clave def es un acto de creación de herramientas personalizadas. Una función bien diseñada actúa como una "caja negra": recibe entradas, opera sobre ellas y devuelve un resultado. Esta capacidad de aislamiento es fundamental; permite que equipos de desarrolladores trabajen en diferentes partes de un sistema sin interferir entre sí. Al segmentar un problema complejo en pequeñas funciones manejables, la tarea de programar se vuelve una labor de ensamblaje de piezas lógicas.

La elegancia de Python se manifiesta especialmente en cómo trata a las funciones: son "objetos de primera clase". Esto significa que pueden ser pasadas como argumentos a otras funciones, asignadas a variables o devueltas como resultados. Esta característica permite paradigmas avanzados como la programación funcional, facilitando la creación de código más limpio y expresivo. Además, el uso de parámetros con valores predeterminados permite que las funciones sean flexibles, adaptándose a casos simples y complejos sin necesidad de duplicar el código.

Otro aspecto vital es la documentación intrínseca. A través de los docstrings (cadenas de texto descriptivas), Python fomenta que la función explique su propósito por sí misma. Esto, sumado a la legibilidad natural del lenguaje, convierte a las funciones en la mejor forma de comunicación entre programadores. Un código compuesto por funciones bien nombradas se lee como un manual de instrucciones, claro, reduciendo el tiempo de mantenimiento y facilitando la detección de errores.

3.2.7. Arreglos o vectores

El concepto de arreglo nace de la necesidad de orden y velocidad. En un arreglo tradicional, los datos se almacenan "hombro con hombro" en la memoria del sistema. Esta disposición no es trivial; permite que la CPU prediga y acceda a los datos con una latencia mínima. En Python, aunque las listas ofrecen una flexibilidad asombrosa, es a través de herramientas como NumPy donde el concepto de vector alcanza su máximo potencial. Un vector no es solo una lista de números;

es una entidad matemática sobre la que se pueden aplicar transformaciones complejas en un solo paso.

La importancia de los vectores reside en la "vectorización". En un programa convencional, para sumar dos listas de mil números, un programador usaría un bucle que recorre cada posición, una por una. Con vectores, la instrucción se envía al procesador como una única operación masiva. Esta distinción es la que permite que aplicaciones modernas, desde el reconocimiento de rostros hasta las predicciones climáticas, funcionen en tiempo real. Los arreglos transforman el procesamiento de datos de una tarea secuencial agotadora en un flujo paralelo de alta velocidad.

Además, los arreglos multidimensionales (matrices) expanden esta lógica a planos y volúmenes, permitiendo representar desde imágenes digitales (donde cada píxel es un elemento de un arreglo) hasta modelos físicos del universo. La rigidez de los arreglos —el hecho de que normalmente todos sus elementos deben ser del mismo tipo— es, paradójicamente, su mayor fortaleza, ya que elimina la ambigüedad y permite una optimización que las estructuras de datos más flexibles no pueden alcanzar.

Los arreglos (arrays) son estructuras de datos fundamentales que almacenan una colección fija de elementos del mismo tipo en posiciones contiguas de memoria. Esto permite acceso directo a cualquier elemento mediante su índice, con una complejidad de $O(1)$. Los arreglos son estáticos: su tamaño se define al crearse y no puede cambiarse fácilmente. Se clasifican en unidimensionales (como vectores), bidimensionales (matrices) y multidimensionales. Su ventaja principal es la eficiencia en acceso aleatorio, pero insertar o eliminar elementos en medio requiere desplazar otros, lo que es costoso ($O(n)$). En muchos lenguajes como C o Java, los arreglos son de tamaño fijo. Por ejemplo, en Python las listas (*list*) actúan como arreglos dinámicos: pueden crecer o reducirse, pero internamente usan un mecanismo de sobre asignación (over-allocation) para amortizar el costo de redimensionamiento. Los vectores, en contextos de estructuras de datos, suelen referirse a arreglos unidimensionales o a implementaciones dinámicas como std::vector en C++, que crecen automáticamente duplicando su capacidad cuando se llenan. En matemáticas y física, un vector es una entidad con magnitud y dirección, representada frecuentemente como un arreglo de componentes, ver Fig. 3.9.

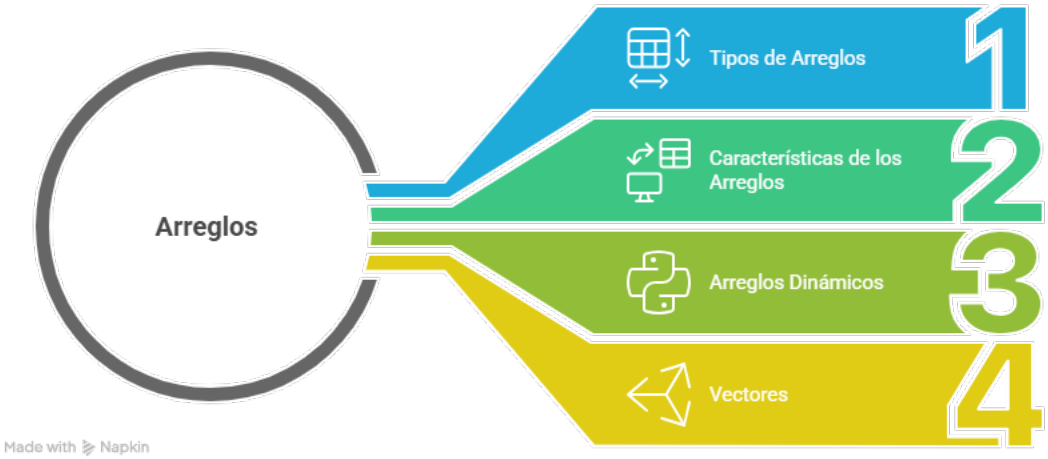


Figura 3.9: Imagen creada por la inteligencia artificial Napkin.ai, en la imagen se muestran los elementos importantes de los arreglos.

En programación científica, bibliotecas como NumPy en Python usan arreglos (ndarray) optimizados para operaciones vectorizadas, evitando bucles explícitos y aprovechando código compilado para mayor velocidad. En Python, las listas son heterogéneas y flexibles, pero menos eficientes para cálculos numéricos. NumPy introduce arreglos homogéneos (todos del mismo tipo),

con soporte para broadcasting y funciones universales (ufuncs), lo que los hace ideales para datos científicos, machine learning y procesamiento de imágenes.

3.2.8. Matrices

Una matriz es, en esencia, una colección de vectores organizados armónicamente. En Python, esta organización permite pasar del pensamiento lineal al pensamiento espacial. La belleza de trabajar con matrices radica en la capacidad de manipular grandes conjuntos de datos como una sola entidad. Por ejemplo, al aplicar un filtro a una fotografía, no estamos modificando cada píxel individualmente mediante instrucciones aisladas; estamos aplicando una operación matemática a una matriz completa. Esta capacidad de síntesis es lo que permite que el software moderno procese video en alta definición o simule fluidos en un entorno virtual. En el ámbito del aprendizaje automático (*Machine Learning*), las matrices son el lenguaje de las neuronas artificiales, ver Fig. 3.10.

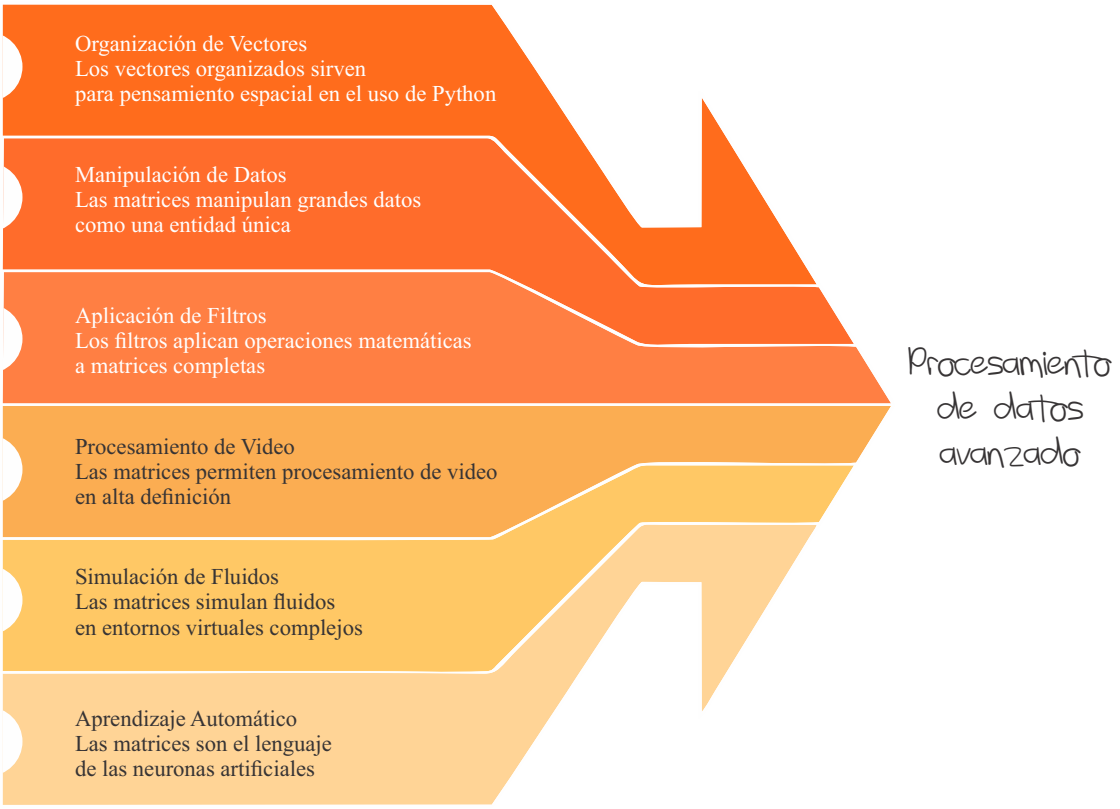


Figura 3.10: Matrices en Python.

Los pesos de una red neuronal se almacenan y procesan en forma de matrices, y el aprendizaje no es otra cosa que el ajuste constante de los valores dentro de esas grillas. Python, gracias a su sintaxis limpia, permite que estas operaciones —que matemáticamente pueden ser abrumadoras— se expresen de forma casi poética, permitiendo que el desarrollador se concentre en la arquitectura del modelo y no en la aritmética subyacente. Sin embargo, el manejo de matrices también impone un desafío de eficiencia. Debido a que el tamaño de los datos crece de forma cuadrática (una matriz de 1000×1000 ya contiene un millón de elementos), el programador de Python debe aprender a utilizar la memoria de forma inteligente. Aquí es donde la historia de Python y su evolución se unen: el lenguaje ha desarrollado interfaces que permiten que estas estructuras residan en la memoria de manera optimizada, permitiendo cálculos que serían imposibles de gestionar manualmente.

Las matrices son estructuras de datos bidimensionales que extienden el concepto de arreglos unidimensionales (vectores) para organizar elementos en filas y columnas, formando una rejilla rectangular. Cada elemento se accede mediante dos índices: uno para la fila y otro para la columna (por ejemplo, matriz[i][j]). Las matrices son homogéneas (generalmente del mismo tipo de dato) y de tamaño fijo en muchos lenguajes, aunque en Python se pueden implementar de forma flexible. Son fundamentales en problemas que involucran datos tabulares, como gráficos, imágenes digitales, ecuaciones lineales o algoritmos de procesamiento. En términos de representación en memoria, las matrices se almacenan de forma contigua, usualmente en orden row-major (fila por fila, común en C y Python) o column-major (columna por columna, como en Fortran). Esto afecta la eficiencia de acceso: recorrer por filas es más rápido en row-major debido a la localidad de caché. Por ejemplo, una matriz 3x3 ocupa 9 posiciones consecutivas en memoria, permitiendo cálculos rápidos de direcciones.

3.2.9. Gráficas en Python

Las gráficas o visualizaciones de datos en Python son herramientas esenciales para explorar, analizar y comunicar información de manera efectiva. Convertir datos numéricos en representaciones visuales como líneas, barras o dispersión facilita la identificación de patrones, tendencias y anomalías que serían difíciles de detectar en tablas crudas. Python destaca en este ámbito gracias a su ecosistema rico en bibliotecas, desde las básicas hasta las avanzadas para visualizaciones interactivas y estadísticas. La biblioteca más fundamental es Matplotlib, que sirve como base para muchas otras. Es altamente personalizable y permite crear gráficos estáticos de calidad pública.

La potencia de Python en la visualización radica en su versatilidad. El proceso comienza con la librería Matplotlib, que estableció el estándar de "lo que puedes imaginar, lo puedes programar". Sin embargo, la evolución hacia librerías como Seaborn refleja un cambio de paradigma: ya no solo buscamos representar datos, sino entender sus correlaciones. Una gráfica de dispersión (*scatter plot*) puede revelar en segundos una relación entre dos variables que a una computadora le tomaría miles de ciclos procesar lógicamente.

Además, la visualización en Python ha trascendido el papel estático. Con la llegada de herramientas interactivas, las gráficas se han convertido en lienzos dinámicos. El usuario ya no es un espectador pasivo; puede hacer zoom, filtrar categorías y explorar dimensiones temporales en tiempo real. Esta interactividad es lo que permite que el "análisis exploratorio de datos" (EDA) sea una conversación entre el humano y la máquina, donde la gráfica responde preguntas a medida que surgen.

Por último, el aspecto estético en Python tiene una función pragmática. El uso correcto del color (mapas de calor o *heatmaps*) y la jerarquía visual no son meros adornos; son herramientas para evitar el sesgo y la mala interpretación. Una gráfica bien construida en Python respeta la "integridad de los datos", asegurando que la escala y la proporción cuenten la verdad matemática sin distorsiones, facilitando la toma de decisiones basada en evidencia.



Laboratorio 7

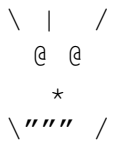
Capítulo 3

3.3. Ejercicios del Laboratorio 7

Efectuar la solución del problema usando el lenguaje Python y utilice como variables sus nombres y apellidos, ejemplo si usted se llama Joe Frand Llerena Izquierdo, use esa información en las variables como “area”, “perimetro” o “suma” por “Joe”, “Frand” o “Llerena”.

* Ejercicio 1

Escriba un programa que muestre la siguiente figura:



Algoritmo 3.2: Programa para mostrar figura usando nombres como variables

```
# Definicion de la figura usando nombres y apellidos como variables
# Se usa doble backslash (\\) para que Python
# lo interprete como un caracter literal
Joe = " \\ | /"
Frand = " @ @ "
Llerena = " * "
Izquierdo = " \\\"\"\"/"

# Imprimir contenido de las variables para formar la figura
print(Joe)
print(Frand)
print(Llerena)
print(Izquierdo)
```

* Ejercicio 2

Escriba un programa que siendo a=“Hola”, b=“mundo”, c=87 y d=2.33145, armar y mostrar las siguientes cadenas: “Hola mundo” (usando a y b), “-Hola-mundo-” (usando a y b), “El resultado es: 87” (usando c), “El resultado es: 87 min (5220 seg)” (usando c ambas veces), “La temperatura es: 2.3”

Algoritmo 3.3: Manipulación de cadenas y formateo en Python

```
# Asignar valores a variables con nombres y apellidos
Joe = "Hola" # Corresponde a 'a'
Frand = "mundo" # Corresponde a 'b'
Llerena = 87 # Corresponde a 'c'
Izquierdo = 2.33145 # Corresponde a 'd'

# 1. "Hola mundo"
print(f"{Joe} {Frand}")
# 2. "-Hola-mundo-"
print(f"-{Joe}-{Frand}-")
# 3. "El resultado es: 87"
print(f"El resultado es: {Llerena}")
# 4. "El resultado es: 87 min (5220 seg)"
# Realizamos el calculo de segundos (87 * 60) dentro de la misma llave
print(f"El resultado es: {Llerena} min ({Llerena * 60} seg)")
# 5. "La temperatura es: 2.3"
# Utilizamos :.1f para formatear el valor a un solo decimal
print(f"La temperatura es: {Izquierdo:.1f}")
```

* Ejercicio 3

Hacer un programa que le pida una cadena al usuario (usando la función “input”) y la imprima a pantalla al revés. Por ejemplo, para el texto “Esto es así”, debería mostrar: “ísa se otsE”.

Algoritmo 3.4: Invertir una cadena de texto en Python

```
# 1. Solicitar la cadena al usuario
# usando el primer nombre como variable
Joe = input("Ingrese el texto que desea invertir: ")
# 2. Invertir la cadena usando slicing [::-1]
# y guardarla en el segundo nombre
# El formato [inicio:fin:paso] con paso -1 recorre la cadena
# del final hacia adelante
Frand = Joe[::-1]
# 3. Mostrar el resultado
print(f"El texto invertido es: {Frand}")
```

* Ejercicio 4

Hacer un programa que le pida un número al usuario (usando la función “input”) y muestre ese número menos dos, más dos, multiplicado por dos, dividido por dos, dividido por dos de forma entera, y elevado a la potencia de dos. Por ejemplo, para el número 7, debería mostrar: 5 9 14 3 3.5 49

Algoritmo 3.5: Operadores aritmeticos básicos en Python

```
# 1. Solicitar el valor al usuario
# (usando Joe como variable de entrada)
Joe = float(input("Ingrese un valor: "))

# 2. Realizar las operaciones solicitadas
# usando los nombres como variables
Frand = Joe - 2 # Resta
Llerena = Joe + 2 # Suma
Izquierdo = Joe * 2 # Producto
Joe_Frand = Joe // 2 # Cociente entero (Ej: 7 // 2 = 3)
Llerena_Izquierdo = Joe / 2 # Cociente real (Ej: 7 / 2 = 3.5)
Joe_Llerena = Joe ** 2 # Potencia al cuadrado (Ej: 7 ** 2 = 49)

# 3. Mostrar los resultados siguiendo el orden del ejemplo
print(Frand)
print(Llerena)
print(Izquierdo)
print(Joe_Frand)
print(Llerena_Izquierdo)
print(Joe_Llerena)
```

* Ejercicio 5

Pedirle un número al usuario, elevarlo al cuadrado, y mostrar los dígitos al revés y separados por espacio. Por ejemplo, si el usuario ingresa 17, la salida tiene que ser “9 8 2”.

Algoritmo 3.6: Elevar al cuadrado e invertir dígitos con espacios

```
# 1. Pedir valor y convertirlo a entero
Joe = int(input("Ingrese un valor: "))

# 2. Elevar al cuadrado
Frand = Joe ** 2

```

```
# 3. Convertir el resultado a texto y darle la vuelta (slicing [::-1])
# Ejemplo: 289 -> "982"
Llerena = str(Frand)[::-1]

# 4. Separar los digitos con un espacio usando .join()
# Izquierdo tomara cada caracter de Llerena
# y le pone un espacio en medio
Izquierdo = " ".join(Llerena)

# 5. Mostrar el resultado final
print(Izquierdo)
```

* Ejercicio 6

Pedirle tres números (que pueden ser con decimales) al usuario, base y altura de un rectángulo, y radio de un círculo. Calcular perímetro y área de cada figura.

Algoritmo 3.7: Cálculo de Área y Perímetro para Rectángulo y Círculo

```
import math

# 1. Solicitar datos al usuario (convertidos a float para decimales)
Joe = float(input("Ingrese la base del rectangulo: "))
Frand = float(input("Ingrese la altura del rectangulo: "))
Llerena = float(input("Ingrese el radio del circulo: "))

# 2. Calculos para el RECTANGULO
# Izquierdo almacenara el area, Joe_Frand el perimetro
Izquierdo = Joe * Frand
Joe_Frand = 2 * (Joe + Frand)

# 3. Calculos para el CIRCULO
# Llerena_Izquierdo almacenara el area, Joe_Llerena
# el perimetro (longitud)
Llerena_Izquierdo = math.pi * (Llerena ** 2)
Joe_Llerena = 2 * math.pi * Llerena

# 4. Mostrar resultados
print("\n--- RESULTADOS DEL RECTANGULO ---")
print(f"Area: {Izquierdo:.2f}")
print(f"Perimetro: {Joe_Frand:.2f}")

print("\n--- RESULTADOS DEL CIRCULO ---")
print(f"Area: {Llerena_Izquierdo:.2f}")
print(f"Perimetro (Circunferencia): {Joe_L_lerena:.2f}")
```

* Ejercicio 7

Pedirle un número con muchos decimales al usuario y mostrarlo redondeado a 3 dígitos decimales, y como entero. ¿Qué tipo de redondeo usó? ¿Por qué?

Algoritmo 3.8: Redondeo y Truncamiento de valores en Python

```
# 1. Solicitar el valor con muchos decimales
Joe = float(input("Un valor con muchos decimales (ej: 15.67891): "))

# 2. Redondear a 3 decimales usando la funcion round()
Frand = round(Joe, 3)

```

```
# 3. Convertir a entero (esto provoca truncar)
Llerena = int(Joe)

# 4. Mostrar los resultados
print(f"Valor original: {Joe}")
print(f"Redondeado a 3 decimales: {Frand}")
print(f"Como entero (usando int): {Llerena}")
```

Al ejecutar este código, Python aplica dos comportamientos distintos, el redondeo si se usó para los decimales. Y el redondeo si se usó para el entero.

Para la variable Frand (decimales), usamos la función round(valor, 3). El tipo de redondeo es al más cercano. Python utiliza un estándar llamado Banker's Rounding (Redondeo de banquero) o redondeo al par más cercano en casos de empate exacto (cuando termina en .5). Para la mayoría de los casos cotidianos, si el cuarto decimal es 5 o superior, sube el tercero; si es menor a 5, lo mantiene. Esto se hace para reducir el sesgo acumulativo en cálculos estadísticos.

Para la variable Llerena (entero), al usar int(), no se realiza un redondeo matemático. El tipo de redondeo es truncamiento (Hacia cero). La función int() simplemente "corta" o elimina toda la parte decimal sin importar si el número era .1 o .99. Por ejemplo, int(15.999) resultará en 15. Es la forma más rápida de obtener la parte entera de un número en programación.

A veces, debido a cómo las computadoras almacenan números de punto flotante en binario, podrías ver resultados inesperados (como 2.675 redondeándose a 2.67 en lugar de 2.68). Si necesitas precisión financiera exacta, en el futuro podrías usar la librería decimal.

* Ejercicio 8

Sumar 0.3 diez veces. Por otro lado, multiplicar 0.3 por diez. ¿El resultado es el mismo? ¿Por qué? ¿Qué tipo de dato debe usar?.

Algoritmo 3.9: Precisión de punto flotante en sumas y multiplicaciones

```
# 1. Sumar 0.3 diez veces usando un bucle
Joe = 0.0
for i in range(10):
    Joe = Joe + 0.3

# 2. Multiplicar 0.3 por diez
Frand = 0.3 * 10

# 3. Mostrar resultados con alta exactitud
print(f"Resultado de la suma (10 veces 0.3): {Joe:.20f}")
print(f"Resultado de la multiplicacion (0.3 * 10): {Frand:.20f}")

# 4. Comparar directamente
Llerena = (Joe == Frand)
print(f"Son exactamente iguales?: {Llerena}")
```

A la pregunta si ¿el resultado es el mismo?, se responde No exactamente. Si observamos la salida con muchos decimales, se nota que la suma suele dar algo como 2.9999999999999996 o similar, mientras que la multiplicación tiende a ser más cercana a 3.0 (aunque también podría variar según el procesador). Para Python, en una comparación estricta (==), el resultado será False. Esto sucede por la representación binaria de los números de punto flotante (estándar IEEE 754). Los computadores no guardan los números como base 10 (decimal), sino como base 2 (binario). El número 0.3 es un número periódico infinito en binario (algo así como intentar escribir 1/3 en decimal: 0.3333...). Al truncar ese número infinito para que quepa en la memoria, se introduce un error minúsculo. Al sumar el número diez veces, ese error se acumula, mientras que al multiplicar,

el error se procesa de forma distinta en el hardware. Para cálculos científicos o generales donde un error de 10^{-16} no importa, se usa float (el estándar por defecto).Si necesitas precisión exacta (como en aplicaciones financieras o contables donde cada centavo cuenta), debes usar el tipo de dato Decimal de la librería decimal. Para una exactitud usamos:

```
from decimal import Decimal
Joe = Decimal('0.3') + Decimal('0.3') + ... # Esto si daria 3.0 exacto
```

* Ejercicio 9

Pedirle una frase al usuario e imprima, en orden, qué consonantes se utilizaron. Por ejemplo, si se ingresó “la rana rene”, mostrar “lnr”.

Algoritmo 3.10: Extraer y ordenar consonantes únicas de una frase

```
# 1. Pedir la frase al usuario
Joe = input("Ingrese una frase: ").lower()

# 2. Definir que NO es una consonante (vocales, espacios y signos)
Frand = "aeiou "

# 3. Filtrar las consonantes unicas
# Llerena sera un conjunto (set) para evitar duplicados automaticamente
Llerena = set()

for caracter in Joe:
    # Verificamos si es una letra y si no esta en nuestra lista
    # de "no consonantes"
    if caracter.isalpha() and caracter not in Frand:
        Llerena.add(caracter)

# 4. Ordenar las consonantes alfabeticamente para que coincida
# con el ejemplo
# Izquierdo sera la lista ordenada unida en una sola cadena
Izquierdo = "".join(sorted(Llerena))

# 5. Mostrar el resultado
print(f"Consonantes utilizadas: {Izquierdo}")
```

* Ejercicio 10

Mostrar los números potencia de 2 menores a 10000 de la siguiente manera: 0001 0002 0004 0008 ... 8192

Algoritmo 3.11: Potencias de 2 menores a 10000 con formato de 4 dígitos

```
# 1. Inicializar la variable con la primera potencia (2^0 = 1)
Joe = 1

# 2. Definir el limite maximo
Frand = 10000

# 3. Ciclo para generar y mostrar las potencias
while Joe < Frand:
    # El formato :04d indica:
    # 0 -> rellenar con ceros
    # 4 -> ancho total de 4 caracteres
    # d -> valor entero (decimal)
```

```
    print(f"{Joe:04d}")

    # 4. Calcular la siguiente potencia de 2
    Joe = Joe * 2
```

* Ejercicio 11

Pedir una frase al usuario y mostrar en mayúscula la inicial de cada palabra. Ejemplo, si el usuario ingresa “No somos nada”, mostrar NSN.

Algoritmo 3.12: Extraer e imprimir iniciales en mayúscula (Acrónimo)

```
# 1. Pedir la frase al usuario
Joe = input("Ingrese una frase: ")

# 2. Dividir la frase en una lista de palabras
# (usando el espacio como separador)
Frand = Joe.split()

# 3. Extraer la primera letra de cada palabra,
# pasarla a mayuscula y unirlas
# Recorremos cada 'palabra' en la lista 'Frand'
Llerena = "".join([palabra[0].upper() for palabra in Frand])

# 4. Mostrar el resultado final
print(f"Resultado: {Llerena}")
```

* Ejercicio 12

Hacer un programa que le pida una cadena al usuario, y arme un triángulo creciente y decreciente con ese texto. Por ejemplo, para el texto “klop”, el resultado sería:

```
k
kl
klo
klop
klo
kl
k
```

Algoritmo 3.13: Triángulo creciente y decreciente con texto

```
# 1. Pedir la cadena al usuario
Joe = input("Ingrese una palabra o frase: ")

# 2. Obtener la longitud de la cadena
Frand = len(Joe)

# 3. Parte Creciente: recorremos desde 1 hasta la longitud total
print("--- Resultado ---")
for Llerena in range(1, Frand + 1):
    # Mostramos desde el inicio hasta la posicion Llerena
    print(Joe[:Llerena])

# 4. Parte Decreciente: recorremos desde la longitud-1 hasta 1
for Izquierdo in range(Frand - 1, 0, -1):
```

```
    # Mostramos desde el inicio hasta la posicion Izquierdo
    print(Joe[:Izquierdo])
```

* Ejercicio 13

Escribir un programa que le pregunte un número al usuario. Si el número es 5, que muestre ¡"Suerte!"; si el número es mayor a 10, que muestre "Grande!"; para los otros casos que muestre "Sin suerte, :(".

Algoritmo 3.14: Uso de condicionales anidados en Python

```
# 1. Solicitar el valor al usuario
# Usamos float para que funcione tanto con enteros como con decimales
Joe = float(input("Ingrese un valor: "))

# 2. Estructura de control para evaluar las condiciones
if Joe == 5:
    # Condicion de igualdad exacta
    Frand = "Suerte!"
elif Joe > 10:
    # Condicion de mayor que
    Frand = "Grande!"
else:
    # Caso por defecto (cualquier otro valor)
    Frand = "Sin suerte, :("

# 3. Mostrar el resultado almacenado en la variable
print(Frand)
```

* Ejercicio 14

Escribir un programa que le pregunte un número al usuario. Si el número es 5, que muestre ¡"Suerte!"; si el número es mayor a 10, que le vuelva a pedir el número al usuario; para los otros casos que muestre "Sin suerte, :(".

Algoritmo 3.15: Uso de bucles y condicionales combinados

```
# 1. Inicializamos la variable de control
Joe = float(input("Ingrese un valor: "))

# 2. Bucle: Mientras el valor sea mayor a 10, se volvera a pedir
while Joe > 10:
    print("Grande! (Intentalo de nuevo)")
    Joe = float(input("Ingrese un valor: "))

# 3. Una vez que el valor NO es mayor a 10, evaluamos los casos finales
if Joe == 5:
    Frand = "Suerte!"
else:
    Frand = "Sin suerte, :("

# 4. Mostrar el resultado final
print(Frand)
```

* Ejercicio 15

Hacer una función que recibe un número y contesta "par" o "impar" en función de si el número es par o no.

Algoritmo 3.16: Función para determinar si un número es par o impar

```
# 1. Definir la funcion usando nombres como variables
def Joe(Frand):
    # El operador % calcula el residuo de la division entre 2
    if Frand % 2 == 0:
        return "par"
    else:
        return "impar"

# 2. Ejemplo de uso (Pedir valor al usuario)
Llerena = int(input("Ingrese un valor para verificar: "))

# 3. Llamada a la funcion y almacenamiento del resultado
Izquierdo = Joe(Llerena)

# 4. Mostrar el resultado
print(f"El valor {Llerena} es {Izquierdo}.")
```

* Ejercicio 16

Hacer una función que recibe dos números y devuelve “mayor” (si el primer número es mayor que el segundo), “menor”, o “iguales”.

Algoritmo 3.17: Función para comparar dos números

```
# 1. Definicion de la funcion con nombres y apellidos
def Joe(Frand, Llerena):
    if Frand > Llerena:
        return "mayor"
    elif Frand < Llerena:
        return "menor"
    else:
        return "iguales"

# 2. Solicitar datos al usuario para probar la funcion
# Usamos Izquierdo y Joe_Frand como variables de entrada
Izquierdo = float(input("Ingrese el primer valor: "))
Joe_Frand = float(input("Ingrese el segundo valor: "))

# 3. Llamada a la funcion y almacenamiento del resultado
resultado = Joe(Izquierdo, Joe_Frand)

# 4. Mostrar el veredicto
print(f"El primer valor es {resultado} que el segundo.")
```

* Ejercicio 17

Hacer un programa que genere un número entero al azar (módulo random) entre 0 y 1000, y le vaya pidiendo al usuario que ingrese números enteros para adivinarlo. Si el usuario ingresa un número menor que el objetivo, muestra “¡Es más alto!”; si el usuario ingresa uno mayor, muestra “¡Es más bajo!”; si el usuario acierta, muestra “¡Viva Python!”, y termina. Si el usuario no acertó en 7 intentos, muestre “¡Alpiste perdiste! Booo” y termine.

Algoritmo 3.18: Juego de adivinar número con límite de intentos

```
import random

```

```
# 1. Generar valor al azar y configurar intentos
# Joe sera el valor secreto, Frand el contador de intentos
Joe = random.randint(0, 1000)
Frand = 0
Llerena = False # Variable de control para saber si acerto
print("Piensa un valor entre 0 y 1000.", end=" ")
print("Tienes 7 intentos para adivinarlo.")
# 2. Bucle principal (maximo 7 intentos)
while Frand < 7:
    # Izquierdo es el valor que ingresa el usuario
    Izquierdo = int(input(f"Intento {Frand + 1} - Ingresa tu valor: "))
    Frand += 1 # Aumentamos el contador de intentos
    if Izquierdo == Joe:
        print("Viva Python!")
        Llerena = True
        break # Sale del bucle inmediatamente
    elif Izquierdo < Joe:
        print("Es mas alto!")
    else:
        print("Es mas bajo!")
# 3. Mensaje final si agoto los intentos y no acerto
if not Llerena:
    print(f"Alpiste perdiste! Booo. El valor era {Joe}")
```

* Ejercicio 18

Hacer una función que reciba un texto y devuelva el mismo texto, pero con cada palabra invertida. Por ejemplo, llamándola con “Esto es una prueba”, debe devolver “otsE se anu abeurp”.

Algoritmo 3.19: Función para invertir cada palabra de un texto

```
# 1. Definir la funcion con nombres y apellidos
def Joe(Frand):
    # Separamos el texto en una lista de palabras
    Llerena = Frand.split()
    # Invertimos cada palabra individualmente
    # y las guardamos en una lista nueva
    # Usamos una "list comprehension" para mayor eficiencia
    Izquierdo = [palabra[::-1] for palabra in Llerena]
    # Unimos las palabras invertidas con un espacio de por medio
    return " ".join(Izquierdo)
# 2. Ejemplo de uso del programa
texto_usuario = "Esto es una prueba"
resultado = Joe(texto_usuario)
# 3. Mostrar el resultado final
print(f"Original: {texto_usuario}")
print(f"Resultado: {resultado}")
```

* Ejercicio 19

Hacer una función que reciba dos palabras y que imprima línea por línea las primeras, segundas, etc. letras de ambas palabras. Por ejemplo, llamándola con "Holaz "mundo", el resultado sería:

```
H m
o u
```

```
l n
a d
  o
```

Algoritmo 3.20: Imprimir letras de dos palabras fila por fila

```python
from itertools import zip_longest

# 1. Definicion de la funcion
def Joe(Frand, Llerena):
    # zip_longest empareja las letras indice por indice.
    # fillvalue=' ' asegura que si una palabra es mas corta,
    # se use un espacio.
    for letra1, letra2 in zip_longest(Frand, Llerena, fillvalue=' '):
        print(f"{letra1} {letra2}")
# 2. Ejemplo de uso con las palabras del enunciado
Izquierdo = "Hola"
Joe_Frand = "mundo"
# 3. Llamada a la funcion
Joe(Izquierdo, Joe_Frand)
```

* Ejercicio 20

Armar una función que reciba una tupla y devuelva si la tupla está ordenada (de menor a mayor) o no.

Algoritmo 3.21: Función para verificar si una tupla esta ordenada

```python
# 1. Definicion de la funcion
def Joe(Frand):
    # sorted(Frand) crea una lista ordenada de menor a mayor
    # tuple(...) la convierte de nuevo a tupla para comparar
    Llerena = tuple(sorted(Frand))

    # Comparamos si la original es igual a la version ordenada
    if Frand == Llerena:
        return True
    else:
        return False
# 2. Ejemplos de uso
Izquierdo = (1, 5, 8, 10)
Joe_Frand = (10, 5, 8, 1)

print(f"Esta ordenada {Izquierdo}?: {Joe(Izquierdo)}")
print(f"Esta ordenada {Joe_Frand}?: {Joe(Joe_Frand)}")
```

Si la tupla es muy grande, existe una forma de hacerlo sin crear una copia ordenada, comparando cada elemento con el siguiente:

```python
def Joe_eficiente(Frand):
    # Verifica si cada elemento es menor o igual al siguiente
    return all(Frand[i] <= Frand[i+1] for i in range(len(Frand)-1))
```

* Ejercicio 21

Escriba un algoritmo que permita hallar la suma de los N primeros números enteros.

Algoritmo 3.22: Suma de los N primeros números enteros

```
# 1. Pedir el valor de N al usuario
Joe = int(input("Ingrese la cantidad de valores a sumar (N): "))
# --- METODO 1: Usando un bucle (Iterativo) ---
Frand = 0
for i in range(1, Joe + 1):
    Frand += i
# --- METODO 2: Usando la formula de Gauss (Matematico) ---
# Formula: S = (n * (n + 1)) / 2
Llerena = (Joe * (Joe + 1)) // 2
# 3. Mostrar resultados
print(f"Suma calculada con bucle: {Frand}")
print(f"Suma calculada con formula: {Llerena}")
```

Otra versión es:

Algoritmo 3.23: Suma de los N primeros números enteros en otra versión

```
N = input("Ingresar el valor: ")
suma = N
i = 0
print ("Suma")
for i in range(N):
 suma = suma + i;
print ("La suma es:", suma)
```

* Ejercicio 22

Escriba un algoritmo que permita hallar cualquier tabla de multiplicar.

Algoritmo 3.24: Generador de tablas de multiplicar

```
# 1. Pedir que tabla desea y hasta donde
Joe = int(input("Tabla de multiplicar desea ver?: "))
Frand = int(input("Hasta que valor desea multiplicar?: "))
print(f"\n--- Tabla del {Joe} ---")
# 2. Bucle para generar la tabla
# Llerena sera el multiplicador que cambia en cada vuelta
for Llerena in range(1, Frand + 1):
    # Izquierdo almacenara el resultado de la operacion
    Izquierdo = Joe * Llerena
    print(f"{Joe} x {Llerena} = {Izquierdo}")
```

Otra versión es:

Algoritmo 3.25: Generador de tablas de multiplicar en otra versión

```
N = input("Ingresar el valor de la tabla de multiplicar: ")
for i in range(1,13):
    print (" %d * %d= %d" % (N,i,N*i))
```

* Ejercicio 23

Escriba un algoritmo que permita hallar si es primo o no.

Algoritmo 3.26: Algoritmo para verificar números primos

```
# 1. Solicitar el valor al usuario
Joe = int(input("Ingrese un valor para verificar si es primo: "))
# 2. Variable de control (bandera)
```

```
# Llerena empezara siendo True y cambiara si encontramos un divisor
Llerena = True
# 3. Validacion inicial (valores menores a 2 no son primos)
if Joe < 2:
   Llerena = False
else:
   # 4. Probar divisores desde 2 hasta la raiz cuadrada
   # del valor (mas eficiente)
   # Frand sera nuestro divisor de prueba
   for Frand in range(2, int(Joe**0.5) + 1):
      if Joe % Frand == 0:
         Llerena = False # Encontramos un divisor, no es primo
         break # Salimos del bucle para ahorrar tiempo
# 5. Mostrar el resultado final usando Izquierdo
if Llerena:
   Izquierdo = "es primo"
else:
   Izquierdo = "no es primo"
print(f"El valor {Joe} {Izquierdo}.")
```

Otra versión es:

Algoritmo 3.27: Algoritmo para verificar números primos en otra versión

```
a=0
n=int(input("Ingrese valor: "))
for i in range(1,n+1):
 if(n % i==0):
  a=a+1
if(a!=2):
 print("No es primo")
else:
 print("si es primo")
```

* Ejercicio 24

Se tiene un arreglo de diez números, solicite al usuario un número a encontrar, el programa debe buscar el valor e indicar si lo encontró o no lo encontró y finaliza mostrando el número (si lo encontró) en la posición y ubicación en el arreglo (vector).

Algoritmo 3.28: Búsqueda secuencial en un arreglo de 10 números

```
# 1. Definir el arreglo de diez valores (pueden ser fijos o aleatorios)
# Joe sera nuestro vector/arreglo
Joe = [15, 22, 45, 8, 99, 54, 31, 7, 12, 60]
# 2. Solicitar al usuario el valor a buscar
Frand = int(input("Ingrese el valor a encontrar en el arreglo: "))
# 3. Variables para el control de la busqueda
Llerena = False # Bandera para saber si se encontro
Izquierdo = -1 # Variable para guardar la posicion (indice)
# 4. Proceso de buscar
for i in range(len(Joe)):
   if Joe[i] == Frand:
      Llerena = True
      Izquierdo = i
      break # Detener la busqueda al encontrar la primera coincidencia
# 5. Mostrar resultados finales
if Llerena:
```

```
    # Mostramos la posicion (indice)
    # y la ubicacion humana (posicion + 1)
    print(f"Lo encontro! El valor {Frand}")
    print(f"ubicado en la posicion {Izquierdo}")
    print("Lugar real en el arreglo:")
    print(f"elemento ubicado en {Izquierdo + 1}")
else:
    print("No lo encontro.")
```

* Ejercicio 25

Realice la sumatoria de los números del 1 al 10, visualizando los números de dos formas, ascendente y descendentemente (de forma horizontal), al final de la línea de números muestre la suma total.

Algoritmo 3.29: Sumatoria con visualizacion ascendente y descendente

```
# 1. Inicializar la variable para la suma total
Joe = 0
# 2. Visualizacion Ascendente (1 al 10)
print("Orden Ascendente:", end=" ")
for Frand in range(1, 11):
    print(Frand, end=" ")
    Joe += Frand # Acumulamos la suma en cada paso
print(f"| Suma Total: {Joe}")
# 3. Visualizacion Descendente (10 al 1)
# Usamos range(inicio, fin, paso) con paso -1 para ir hacia atras
print("Orden Descendente:", end=" ")
for Llerena in range(10, 0, -1):
    print(Llerena, end=" ")
# Izquierdo almacenara el mensaje final
Izquierdo = f"| Suma Total: {Joe}"
print(Izquierdo)
```

Otra versión es:

Algoritmo 3.30: Sumatoria dinámica con visualización dual

```
# 1. Pedir el limite al usuario
Joe = int(input("Valor que desea realizar la sumatoria: "))
Frand = 0
# 2. Generar secuencia ascendente y calcular suma
print(f"\nAscendente (1 a {Joe}):", end=" ")
for Llerena in range(1, Joe + 1):
    print(Llerena, end=" ")
    Frand += Llerena
print(f"| Total: {Frand}")
# 3. Generar secuencia descendente
print(f"Descendente ({Joe} a 1):", end=" ")
for Izquierdo in range(Joe, 0, -1):
    print(Izquierdo, end=" ")
print(f"| Total: {Frand}")
```



Laboratorio 8

Capítulo 3

3.4. Ejercicios del Laboratorio 8

* Ejercicio 1

Elabore un programa que muestre los números del 1 al 10 de forma ascendente.

Algoritmo 3.31: Contador ascendente del 1 al 10 con nuevas variables

```
# 1. Definir el limite superior (queremos llegar al 10)
Natalia = 11

# 2. Bucle para recorrer la secuencia
# Giselle toma los valores cada vez que itera
print("Valores del 1 al 10:")
for Giselle in range(1, Natalia):
    # Llerena recibe el valor actual y Ayala se encarga de mostrarlo
    Llerena = Giselle
    Ayala = Llerena
    print(Ayala)
```

* Ejercicio 2

Elabore un programa que muestre los números del 1 al 10 de forma descendente.

Algoritmo 3.32: Contador descendente del 10 al 1

```
# 1. Definir los puntos de inicio y fin
Natalia = 10 # Inicio de la cuenta
Giselle = 0 # Fin de la cuenta (se detiene en el 1)

# 2. Bucle para recorrer la secuencia hacia atras
print("Valores del 10 al 1:")
for Llerena in range(Natalia, Giselle, -1):
    # Ayala toma el valor actual para imprimirlo
    Ayala = Llerena
    print(Ayala)
```

Usando una estructura ***While*** se tiene:

Algoritmo 3.33: Cuenta regresiva del 10 al 1 usando bucle while

```
# 1. Inicializamos la variable con el valor de partida
Natalia = 10

# 2. El bucle se ejecuta mientras Natalia sea mayor o igual a 1
print("Cuenta regresiva con while:")
while Natalia >= 1:
    # Giselle guarda el valor actual
    Giselle = Natalia

    # Llerena y Ayala se encargan de la visualizacion
    Llerena = Giselle
    Ayala = Llerena
    print(Ayala)

    # 3. IMPORTANTE: Restamos 1 a Natalia en cada vuelta
    # Si no hacemos esto, el bucle seria infinito
    Natalia = Natalia - 1
```

Usando una estructura ***Do - While*** se tiene:

Algoritmo 3.34: Simulación de un bucle Hacer-Mientras del 10 al 1

```
# 1. Inicializamos la variable inicial
Natalia = 10

print("Simulando Hacer-Mientras (descendente):")

# 2. Iniciamos un bucle infinito
while True:
    # --- CUERPO DEL BUCLE (Se ejecuta al menos una vez) ---
    Giselle = Natalia
    Llerena = Giselle
    Ayala = Llerena
    print(Ayala)

    # Restamos para la siguiente vuelta
    # Natalia = Natalia - 1 , o
    Natalia -= 1

    # --- CONDICION DE SALIDA (Al final del bloque) ---
    # Si ya bajamos de 1, rompemos el bucle
    if Natalia < 1:
        break
```

* Ejercicio 3

Desarrolle un programa que dado un número determine la sumatoria del 1 hasta ese número.

Algoritmo 3.35: Sumatoria acumulada desde 1 hasta N

```
# 1. Natalia es el valor del final ingresado por el usuario
Natalia = int(input("Ingrese un valor para calcular la sumatoria: "))

# 2. Giselle procede como el acumulador (donde guardamos la suma)
Giselle = 0

# 3. Llerena es nuestra variable de control en el bucle
# Usamos Natalia + 1 para que incluya el valor ingresado
for Llerena in range(1, Natalia + 1):
    Giselle = Giselle + Llerena

# 4. Ayala guarda el mensaje del resultado final
Ayala = f"La sumatoria de 1 hasta {Natalia} es: {Giselle}"
print(Ayala)
```

* Ejercicio 4

Determine la secuencia de Ulam para un número n determinado o ingresado por teclado.

Algoritmo 3.36: Generación de la secuencia de Ulam o Collatz

```
# 1. Natalia es el valor inicial ingresado por el usuario
Natalia = int(input("Ingrese un valor para la secuencia de Ulam: "))

print(f"Secuencia para {Natalia}:")

# 2. El proceso continua mientras el valor sea distinto de 1
while Natalia != 1:
    # Mostramos el valor actual seguido de una flecha
    print(Natalia, end=" -> ")

```

```
    # 3. Aplicamos la logica de Ulam
    # Giselle nos ayuda a verificar si es par o impar
    Giselle = Natalia % 2

    if Giselle == 0:
        # Si es par, se divide por 2
        Llerena = Natalia // 2
    else:
        # Si es impar, se multiplica por 3 y suma 1
        Llerena = (Natalia * 3) + 1

    # Actualizamos Natalia con el nuevo valor calculado en Llerena
    Natalia = Llerena

# 4. Ahora, Ayala muestra el valor final (que siempre sera 1)
Ayala = Natalia
print(Ayala)
```

* Ejercicio 5

Determine la secuencia de Fibonacci para un número n determinado o ingresado por teclado.

Algoritmo 3.37: Generador de la serie de Fibonacci

```
# 1. Natalia define la cantidad de la secuencia queremos ver
Natalia = int(input("Ingrese la cantidad de terminos de Fibonacci: "))

# 2. Inicializamos los dos primeros valores
Giselle = 0
Llerena = 1

print(f"Secuencia de Fibonacci ({Natalia} terminos):")

# 3. Bucle para calcular y mostrar la serie
for i in range(Natalia):
    # Ayala muestra el valor actual de la secuencia
    print(Giselle, end=" ")

    # Calculamos el siguiente valor sumando los dos anteriores
    Ayala = Giselle + Llerena

    # Desplazamos los valores para la siguiente vuelta
    Giselle = Llerena
    Llerena = Ayala
```

* Ejercicio 6

Determine el número factorial para un número n determinado o ingresado por teclado.

Algoritmo 3.38: Cálculo del factorial de un número N

```
# 1. Natalia es el valor que ingresa el usuario
Natalia = int(input("Ingrese un valor para calcular su factorial: "))

# 2. Giselle se inicializa en 1 (no en 0, porque es un producto)
Giselle = 1

# Validar: el factorial de 0 es 1 y no existe para negativos
if Natalia < 0:
    print("El factorial no existe para valores negativos.")
```

```
elif Natalia == 0:
   print("El factorial de 0 es 1.")
else:
   # 3. Multiplicamos desde 1 hasta el valor Natalia
   for Llerena in range(1, Natalia + 1):
      Giselle = Giselle * Llerena

   # 4. Ayala almacena el mensaje final
   Ayala = f"El factorial de {Natalia} es: {Giselle}"
   print(Ayala)
```

* Ejercicio 7

Determine la potencia para un número n determinado o ingresado por teclado.

Algoritmo 3.39: Cálculo de potencia mediante bucles

```
# 1. Solicitar base y exponente
Natalia = int(input("Ingrese la base: "))
Giselle = int(input("Ingrese el exponente: "))

# 2. Llerena es nuestro acumulador.
# Inicializamos en 1 porque es una multiplicacion.
Llerena = 1

# 3. El bucle se repite tantas veces
# como indique el exponente (Giselle)
for i in range(Giselle):
   Llerena = Llerena * Natalia

# 4. Ayala guarda el resultado final
Ayala = f"El resultado de {Natalia} elevado"
print(Ayala)
print(f"a la {Giselle} es: {Llerena}")
```

* Ejercicio 8

Elabore un menú para los cuatro anteriores ejercicios.

Algoritmo 3.40: Menú interactivo para ejercicios matemáticos

```
# Natalia procede como la variable de opcion del menu
Natalia = -1

while Natalia != 0:
   print("\n--- MENU DE EJERCICIOS ---")
   print("1. Secuencia de Ulam")
   print("2. Secuencia de Fibonacci")
   print("3. Factorial de un valor")
   print("4. Potencia de un valor")
   print("0. Salir")

   Natalia = int(input("\nSeleccione una alternativa: "))

   if Natalia == 1:
      # --- EJERCICIO: ULAM ---
      Giselle = int(input("Ingrese valor para Ulam: "))
      while Giselle != 1:
         print(Giselle, end=" -> ")
         if Giselle % 2 == 0:
```

```
            Giselle //= 2
         else:
            Giselle = (Giselle * 3) + 1
      print(1)

   elif Natalia == 2:
      # --- EJERCICIO: FIBONACCI ---
      Llerena = int(input("Cantidad de valores que desea?: "))
      Giselle, Ayala = 0, 1
      for i in range(Llerena):
         print(Giselle, end=" ")
         Giselle, Ayala = Ayala, Giselle + Ayala
      print()

   elif Natalia == 3:
      # --- EJERCICIO: FACTORIAL ---
      Llerena = int(input("Ingrese un valor para factorial: "))
      Giselle = 1
      for i in range(1, Llerena + 1):
         Giselle *= i
      print(f"Resultado: {Giselle}")

   elif Natalia == 4:
      # --- EJERCICIO: POTENCIA ---
      Llerena = int(input("Ingrese la base: "))
      Ayala = int(input("Ingrese el exponente: "))
      Giselle = 1
      for i in range(Ayala):
         Giselle *= Llerena
      print(f"Resultado: {Giselle}")

   elif Natalia == 0:
      print("Saliendo del programa...")
   else:
      print("Alternativa no valida, intente de nuevo.")
```

* Ejercicio 9

Elabore un programa que encuentre el resultado de las Series de Taylor: aproximación de e^x.

Algoritmo 3.41: Aproximación de e^x usando Series de Taylor

```
import math

# 1. Natalia es el valor de x (exponente)
Natalia = float(input("Ingrese el valor de x para e^x: "))
# 2. Giselle es la cantidad de valores (a mayor valor, mas preciso)
Giselle = int(input("Ingrese el valor de cantidad para la serie: "))

# 3. Llerena acumula la suma de la serie
Llerena = 0.0

for n in range(Giselle):
   # Ayala calcula el factorial de n de forma sencilla
   Ayala = math.factorial(n)

   # Aplicamos la formula del termino n: (x^n) / n!
   termino = (Natalia**n) / Ayala
```

```
    Llerena += termino

print(f"\nResultado aproximado con Taylor: {Llerena}")
print(f"Resultado real (math.exp): {math.exp(Natalia)}")
```

Ahora elabore un programa que encuentre el resultado de las Series de Taylor: aproximación de Seno de x.

Algoritmo 3.42: Aproximación de Seno(x) usando Taylor

```
import math

# 1. Natalia es el angulo en grados (luego lo pasamos a radianes)
Ayala = float(input("Ingrese el valor en grados: "))
Natalia = math.radians(Ayala) # Taylor requiere radianes

# 2. Giselle define la precision (cantidad de terminos)
Giselle = int(input("Ingrese la cantidad de valores: "))

# 3. Llerena acumula el valor del Seno
Llerena = 0.0

for n in range(Giselle):
    # Calculamos el exponente y factorial impar: (2n + 1)
    expo = 2 * n + 1

    # Calculamos el termino con signo alternado
    termino = ((-1)**n) * (Natalia**expo) / math.factorial(expo)

    Llerena += termino

print(f"\nResultado con Taylor: {Llerena}")
print(f"Resultado real (math.sin): {math.sin(Natalia)}")
```

Ahora elabore un programa que encuentre el resultado de las Series de Taylor: aproximación de Coseno de x.

Algoritmo 3.43: Aproximacion de Coseno(x) usando Taylor

```
import math

# 1. Ayala recibe el angulo y Natalia lo convierte a radianes
Ayala = float(input("Ingrese el angulo en grados: "))
Natalia = math.radians(Ayala)

# 2. Giselle define la cantidad de iteraciones para la precision
Giselle = int(input("Ingrese el valor de terminos (ej. 10): "))

# 3. Llerena es el acumulador del resultado
Llerena = 0.0

for n in range(Giselle):
    # En el coseno usamos exponentes pares: 2 * n
    expo_par = 2 * n

    # Calculamos el termino actual con signo alternado
    # El primer termino siempre es 1 ya que x^0 / 0! = 1
    termino = ((-1)**n) * (Natalia**expo_par) / math.factorial(expo_par)
```

```
    Llerena += termino

print(f"\nResultado aproximado del Coseno: {Llerena}")
print(f"Resultado real (math.cos): {math.cos(Natalia)}")
```

* Ejercicio 10

El método de multiplicación rusa consiste en multiplicar sucesivamente por 2 el multiplicando y dividir por 2 el multiplicador hasta que el multiplicador tome el valor 1. Luego, se suman todos los multiplicandos correspondientes a los multiplicadores impares. Dicha suma es el producto de los dos números. La siguiente tabla muestra el cálculo realizado para multiplicar 37 por 12, cuyo resultado final es 12 + 48 + 384 = 444. Desarrolle un programa que reciba como entrada el multiplicador y el multiplicando, y entregue como resultado el producto de ambos, calculado mediante el método de multiplicación rusa.

Algoritmo 3.44: Método de Multiplicación Rusa

```
# 1. Solicitar los datos de entrada
# Natalia sera el multiplicador (el que dividimos)
Natalia = int(input("Ingrese el multiplicador: "))
# Giselle sera el multiplicando (el que duplicamos)
Giselle = int(input("Ingrese el multiplicando: "))

# 2. Llerena guardara la suma de los resultados parciales
Llerena = 0

print(f"\nProceso de multiplicacion para {Natalia} x {Giselle}:")
print("-" * 30)
print("Multiplicador | Multiplicando | Es Impar?")

# 3. El ciclo continua hasta que el multiplicador (Natalia) llegue a 1
while Natalia >= 1:
    # Verificamos si el multiplicador actual es impar
    # Ayala nos sirve para evaluar la paridad
    Ayala = Natalia % 2

    if Ayala != 0:
        # Si es impar, sumamos el multiplicando actual a la suma total
        Llerena = Llerena + Giselle
        es_impar = "SI (Sumar)"
    else:
        es_impar = "NO"

    # Mostramos la fila actual de la tabla
    print(f"{Natalia:<13} | {Giselle:<13} | {es_impar}")

    # 4. Actualizamos los valores:
    # Natalia se divide por 2 (division entera)
    Natalia = Natalia // 2
    # Giselle se duplica
    Giselle = Giselle * 2

# 5. Resultado final
print("-" * 30)
print(f"El producto total es: {Llerena}")
```



Laboratorio 9

Capítulo 3

3.5. Ejercicios del Laboratorio 9

* Ejercicio 1

Elabore un programa que determine si un número es primo o no.

Algoritmo 3.45: Determinación de números primos

```
# 1. Anthony es el valor que ingresa el usuario
Anthony = int(input("Ingrese un valor para verificar si es primo: "))

# 2. Santiago es como una bandera (booleano)
# Asumimos que es primo hasta que se demuestre lo contrario
Santiago = True

# 3. Llerena es nuestro contador para probar divisores
if Anthony < 2:
   Santiago = False # Los valores menores a 2 no son primos
else:
   for Llerena in range(2, int(Anthony**0.5) + 1):
      # Ayala comprueba el residuo de la division
      Ayala = Anthony % Llerena

      if Ayala == 0:
         Santiago = False
         break # Si encontramos un divisor, dejamos de buscar

# 4. Mostrar el resultado final
if Santiago:
   print(f"El valor {Anthony} es PRIMO.")
else:
   print(f"El valor {Anthony} NO ES PRIMO.")
```

* Ejercicio 2

Elabore un programa que muestre los números del 100 al 200 en bloques de cinco en cinco, utilice las estructuras repetitivas for, while y do-while para demostrar que las tres realizan el mismo resultado. Usando el bucle for:

Algoritmo 3.46: Secuencia con bucle for

```
print("--- Bucle FOR (100 al 200 de 5 en 5) ---")
# Anthony define el inicio, Santiago el fin, Llerena el paso
Anthony = 100
Santiago = 201 # Usamos 201 para incluir el 200
Llerena = 5
for Ayala in range(Anthony, Santiago, Llerena):
   print(Ayala, end=" ")
print("\n")
```

Usando el bucle while:

Algoritmo 3.47: Secuencia con bucle while

```
print("--- Bucle WHILE ---")
Anthony = 100 # Inicializacion
while Anthony <= 200: # Condicion
   Santiago = Anthony
   print(Santiago, end=" ")
   Anthony += 5 # Incremento manual de 5 en 5
```

```
print("\n")
```

Usando el bucle do-while:

Algoritmo 3.48: Secuencia simulando do-while

```
print("--- Bucle DO-WHILE (Simulado) ---")
Anthony = 100
while True:
    Llerena = Anthony
    print(Llerena, end=" ")

    Anthony += 5

    # Ayala evalua la condicion de salida al final
    if Anthony > 200:
        break
print("\n")
```

Ahora con la secuencia de números correlativos de cinco en cinco:

Algoritmo 3.49: Números correlativos con saltos de línea cada 5 unidades

```
# --- VERSION CON BUCLE FOR ---
print("Valores del 100 al 200 (Bloques de 5):")
# Anthony define el inicio y Santiago el fin
Anthony = 100
Santiago = 200
# Llerena serve para contar cuantos valores
# llevamos impresos en la linea actual
Llerena = 0
for Ayala in range(Anthony, Santiago + 1):
    print(Ayala, end=" ")
    Llerena += 1

    # Si Llerena llega a 5, imprimimos
    # un salto de linea y reiniciamos el contador
    if Llerena == 5:
        print() # Salto de linea
        Llerena = 0

# --- VERSION CON BUCLE WHILE ---
print("\nRepitiendo con WHILE:")
Anthony = 100
Llerena = 0
while Anthony <= 200:
    print(Anthony, end=" ")
    Llerena += 1

    if Llerena == 5:
        print()
        Llerena = 0

    Anthony += 1
```

∗ Ejercicio 3

Elabore un programa que genere un número aleatorio.

Algoritmo 3.50: Generación de números aleatorios

```
import random

# 1. Anthony definira el rango inferior
Anthony = 1
# 2. Santiago definira el rango superior
Santiago = 100
# 3. Llerena genera el valor aleatorio entero entre Anthony y Santiago
Llerena = random.randint(Anthony, Santiago)
# 4. Ayala guarda el mensaje para mostrar el resultado
Ayala = f"El valor aleatorio generado es: {Llerena}"

print(Ayala)
```

∗ Ejercicio 4

Elabore un programa que redondee “hacia arriba” y “hacia abajo”.

Algoritmo 3.51: Redondeo hacia arriba y hacia abajo

```
import math

# 1. Anthony recibe el valor decimal por teclado
Anthony = float(input("Ingrese un valor decimal (ejemplo 7.4): "))
# 2. Santiago realiza el redondeo hacia arriba (Techo)
# El resultado sera el entero mas cercano mayor o igual al valor
Santiago = math.ceil(Anthony)
# 3. Llerena realiza el redondeo hacia abajo (Piso)
# El resultado sera el entero mas cercano menor o igual al valor
Llerena = math.floor(Anthony)
# 4. Ayala muestra los resultados finales
Ayala = f"Valor original: {Anthony}\nRedondeo arriba: {Santiago}"
print(Ayala)
print(f"\nRedondeo abajo: {Llerena}")
```

∗ Ejercicio 5

Elabore un programa que muestre los números pares comprendidos entre 10 y 20 inclusive.

Algoritmo 3.52: Números pares entre 10 y 20 inclusive

```
# 1. Anthony y Santiago definen el rango de busqueda
Anthony = 10
Santiago = 20
print(f"Valores pares entre {Anthony} y {Santiago}:")
# 2. Llerena recorre los valores en el rango
# Usamos Santiago + 1 para que el 20 sea incluido
for Llerena in range(Anthony, Santiago + 1):

    # 3. Ayala verifica si el valor es par
    # Un valor es par si el residuo de dividirlo por 2 es 0
    Ayala = Llerena % 2

    if Ayala == 0:
        print(Llerena)
```

∗ Ejercicio 6

Elabore un programa que permita ingresar mi nombre.

Algoritmo 3.53: Programa para captura de nombre

```
# 1. Anthony se encarga de recibir el dato desde el teclado
Anthony = input("Por favor, ingrese su nombre: ")
# 2. Santiago puede usarse para transformar el texto (ej. ponerlo en mayusculas)
Santiago = Anthony.upper()
# 3. Llerena prepara el mensaje de bienvenida
# personalizado
Llerena = "Bienvenido al sistema,"
# 4. Ayala une todo y muestra el resultado final en pantalla
Ayala = f"{Llerena} {Santiago}."
print(Ayala)
```

* Ejercicio 7

Elabore un programa que cuente la cantidad de vocales de una palabra.

Algoritmo 3.54: Contador de vocales en una palabra

```
# 1. Anthony recibe la palabra ingresada por el usuario
Anthony = input("Ingrese una palabra o frase: ")
# 2. Llerena define cuales son las letras que queremos buscar
Llerena = "aeiouAEIOU"
# 3. Santiago funciona como nuestro acumulador (empieza en 0)
Santiago = 0
# 4. El bucle recorre cada letra de la palabra (Ayala)
for Ayala in Anthony:
    if Ayala in Llerena:
        Santiago += 1

# Mostrar el resultado final
print(f"La palabra '{Anthony}' tiene {Santiago} vocales.")
```

* Ejercicio 8

Elabore un programa que cuente la cantidad de espacios vacíos de una frase.

Algoritmo 3.55: Contador de espacios en blanco

```
# 1. Anthony captura la frase completa
Anthony = input("Ingrese una frase: ")
# 2. Santiago actuara como el contador de espacios
Santiago = 0
# 3. El bucle recorre cada caracter de la frase
for Ayala in Anthony:
    # 4. Llerena verifica si el caracter actual es un espacio
    if Ayala == " ":
        Santiago += 1

print(f"La frase ingresada tiene {Santiago} espacios vacios.")
```

* Ejercicio 9

Elabore un programa que valide el número de cédula ecuatoriana.

Algoritmo 3.56: Validador de cedula ecuatoriana

```
# 1. Anthony recibe el valor de cedula
Anthony = input("Ingrese su valor de cedula (10 digitos): ")
# Validacion basica de longitud
if len(Anthony) == 10 and Anthony.isdigit():
    # 2. El decimo digito es el verificador
    verificador_real = int(Anthony[9])
```

```
    Santiago = 0

    # 3. Iteramos por los primeros 9 digitos
    for i in range(9):
        Ayala = int(Anthony[i])

        # Multiplicamos posiciones impares por 2 y pares por 1
        if (i + 1) % 2 != 0:
            Ayala = Ayala * 2
            # Si el resultado es mayor a 9, restamos 9
            if Ayala > 9:
                Ayala -= 9

        Santiago += Ayala

    # 4. Calculamos el digito verificador esperado (Llerena)
    decena_superior = ((Santiago // 10) + 1) * 10
    if Santiago % 10 == 0:
        Llerena = 0
    else:
        Llerena = decena_superior - Santiago

    # Comprobacion final
    if Llerena == verificador_real:
        print("La cedula es VALIDA.")
    else:
        print("La cedula es INCORRECTA.")
else:
    print("Entrada no valida. Debe tener 10 digitos numericos.")
```

* Ejercicio 10

Implemente el siguiente juego: el programa seleccionará un número aleatorio entre 0 y 100 y el jugador debe acertarlo. En cada intento el jugador propondrá una solución y se le informará si el número a acertar es menor o mayor que el propuesto. El juego termina cuando se acierte la cifra o haya realizado un máximo de 12 intentos en cuyo caso se le mostrará al jugador la calificación obtenida según la siguiente tabla: Número de intentos por categoría, 1-3 Suertudo, 4-6 Genio, 7 No está mal, 8 Se puede mejorar, >= 9 Qué pasa amigo.

Algoritmo 3.57: Juego de Adivina el valor con calificación

```
import random

# 1. Anthony genera el valor aleatorio secreto entre 0 y 100
Anthony = random.randint(0, 100)
# 2. Llerena sera nuestro contador de intentos
Llerena = 0
# 3. Santiago guardara el intento del usuario (inicializado fuera del rango)
Santiago = -1
print("Bienvenido al juego! Tienes un maximo de 12 intentos.")
# El juego sigue mientras no acierte y no supere los 12 intentos
while Santiago != Anthony and Llerena < 12:
    Llerena += 1
    Santiago = int(input(f"Intento {Llerena} - Ingrese un valor: "))

    if Santiago < Anthony:
        print("El valor secreto es MAYOR.")
```

```
    elif Santiago > Anthony:
        print("El valor secreto es MENOR.")
    else:
        print(f"FELICIDADES! Acertaste en el intento {Llerena}.")
# 4. Al terminar el juego, Ayala determina la categoria segun Llerena
if Santiago == Anthony:
    if 1 <= Llerena <= 3:
        Ayala = "Suertudo"
    elif 4 <= Llerena <= 6:
        Ayala = "Genio"
    elif Llerena == 7:
        Ayala = "No esta mal"
    elif Llerena == 8:
        Ayala = "Se puede mejorar"
    else:
        Ayala = "Que pasa amigo"

    print(f"Tu calificacion es: {Ayala}")
else:
    print(f"Lo siento, agotaste tus 12 intentos. El valor era {Anthony}.")
    print("Calificacion: Que pasa amigo")
```

∗ **Ejercicio 11**

Elabora un programa para el juego adivine el número. El usuario ingresa un número y si el valor ingresado es mayor, el programa debe enviar un mensaje indicando al usuario: “Ingrese un valor más bajo”.

Algoritmo 3.58: Juego de Adivina el valor con mensajes de guia

```
import random

# 1. Anthony genera el valor aleatorio entre 0 y 100
Anthony = random.randint(0, 100)
Llerena = 0
Santiago = -1
print("Bienvenido! Intenta adivinar el valor secreto entre 0 y 100.")
# El juego dura maximo 12 intentos (Llerena)
while Santiago != Anthony and Llerena < 12:
    Llerena += 1
    Santiago = int(input(f"\nIntento {Llerena}: Ingrese su valor: "))
    # 2. Logica de guia para el usuario
    if Santiago > Anthony:
        print("Ingrese un valor mas bajo.")
    elif Santiago < Anthony:
        print("Ingrese un valor mas alto.")
    else:
        print(f"Excelente! Acertaste en {Llerena} intentos.")
# 3. Asignacion de categoria en Ayala tras terminar el bucle
if Santiago == Anthony:
    if 1 <= Llerena <= 3:
        Ayala = "Suertudo"
    elif 4 <= Llerena <= 6:
        Ayala = "Genio"
    elif Llerena == 7:
        Ayala = "No esta mal"
    elif Llerena == 8:
        Ayala = "Se puede mejorar"
```

```
    else:
        Ayala = "Que pasa amigo"

    print(f"Tu calificacion es: {Ayala}")
else:
    print(f"\nSe agotaron los intentos. El valor era {Anthony}.")
    print("Calificacion: Que pasa amigo")
```

* Ejercicio 12

Elabora un programa para generar una tabla de bingo para elegir entre 75 números y lo muestra en pantalla una tabla, con las columnas de B, I, N G y O.

Algoritmo 3.59: Generador de Tabla de Bingo 75 valores

```
import random

# 1. Anthony almacena los valores de nuestra tabla por columnas
Anthony = {
    'B': range(1, 16),
    'I': range(16, 31),
    'N': range(31, 46),
    'G': range(46, 61),
    'O': range(61, 76)
}
# 2. Creamos la matriz de la tabla seleccionando
# 5 valores al azar por letra
tabla = {}
for columna, Santiago in Anthony.items():
    # Llerena selecciona 5 valores sin repetir de cada rango
    Llerena = random.sample(Santiago, 5)
    tabla[columna] = Llerena
# 3. Ponemos el espacio libre en el centro de la columna 'N'
tabla['N'][2] = "FREE"
# 4. Ayala se encarga de mostrar la tabla formateada en pantalla
print(" B  I  N  G  O")
print("-" * 30)

for i in range(5):
    linea = ""
    for columna in ['B', 'I', 'N', 'G', 'O']:
        valor = tabla[columna][i]
        # Formateamos para que cada columna ocupe 6 espacios
        linea += f"{str(valor):<6}"
    print(linea)
```



Laboratorio 10

Capítulo 3

3.6. Ejercicios del Laboratorio 10

* Ejercicio 1

Ingrese una cantidad que represente dólares, muestre a continuación el valor correspondiente en euros. (1 dólar estadounidense = 0,897070169 euros).

Algoritmo 3.60: Conversor de Dólares a Euros

```
# 1. Raquel captura la cantidad en dolares (tipo flotante)
Raquel = float(input("Ingrese un valor en dolares (USD): "))
# 2. Ayala define la tasa de cambio oficial segun el ejercicio
Ayala = 0.897070169
# 3. Carabajo realiza el calculo de la conversion
Carabajo = Raquel * Ayala
# 4. Mostramos el resultado formateado a dos decimales
print(f"\n--- Resultado de la conversion ---")
print(f"{Raquel} dolares equivalen a {Carabajo:.2f} euros.")
```

* Ejercicio 2

Ingrese tres valores de calificación sobre diez (notas de una materia), luego calcule su promedio.

Algoritmo 3.61: Cálculo de promedio de calificaciones

```
# 1. Raquel y Ayala capturan las primeras dos notas
Raquel = float(input("Ingrese la primera nota (sobre 10): "))
Ayala = float(input("Ingrese la segunda nota (sobre 10): "))
# 2. Reutilizamos Carabajo para la tercera nota temporalmente
nota_tres = float(input("Ingrese la tercera nota (sobre 10): "))
# 3. Carabajo calcula el promedio final
# Sumamos las tres notas y dividimos para 3
Carabajo = (Raquel + Ayala + nota_tres) / 3
# 4. Mostramos el resultado
print(f"\nEl promedio final de la materia es: {Carabajo:.2f}")
# Verificacion de estado
if Carabajo >= 7:
   print("Estado: Aprobado")
else:
   print("Estado: Reprobado")
```

* Ejercicio 3

Ingrese dos valores enteros, correspondientes a edades de personas, luego muestre en pantalla cuál de las dos es mayor.

Algoritmo 3.62: Comparación de dos edades

```
# 1. Raquel y Ayala capturan las edades
# (usamos int para valores enteros)
Raquel = int(input("Ingrese la edad de la primera persona: "))
Ayala = int(input("Ingrese la edad de la segunda persona: "))
# 2. Carabajo determinara el resultado mediante comparaciones logicas
if Raquel > Ayala:
   Carabajo = f"La primera persona ({Raquel} anos)"
   Carabajo1 = " es mayor que la segunda."
elif Ayala > Raquel:
   Carabajo = f"La segunda persona ({Ayala} anos)"
   Carabajo1 =" es mayor que la primera."
else:
```

```
    Carabajo = "Ambas personas tienen la misma edad."
# 3. Mostrar el resultado final almacenado en Carabajo
print(f"\nResultado: {Carabajo}{Carabajo1}")
```

* Ejercicio 4

Ingrese un número del uno al siete, luego muestra su correspondiente día.

Algoritmo 3.63: Conversor de número a día de la semana

```
# 1. Raquel recibe el numero del 1 al 7
Raquel = int(input("Ingrese un numero del 1 al 7: "))
# 2. Ayala contiene la coleccion de dias (usamos una lista)
# Nota: La posicion 0 es lunes, la 1 es martes, etc.
Ayala = ["Lunes", "Martes", "Miercoles",
         "Jueves", "Viernes", "Sabado", "Domingo"]
# 3. Carabajo valida si el numero esta en el rango correcto
if 1 <= Raquel <= 7:
    # Restamos 1 a Raquel porque las listas en Python empiezan desde 0
    Carabajo = f"El dia correspondiente es: {Ayala[Raquel - 1]}"
else:
    Carabajo = "Error: El valor debe estar entre 1 y 7."
# 4. Mostrar el resultado
print(Carabajo)
```

* Ejercicio 5

Muestra los números del 25 al 58 utilizando el lazo repetitivo “mientras” o “while”.

Algoritmo 3.64: Bucle While del 25 al 58

```
# 1. Raquel define el punto de partida
Raquel = 25
# 2. Ayala define el punto de parada
Ayala = 58
print(f"Iniciando conteo desde {Raquel} hasta {Ayala}:")
# 3. El lazo "mientras" se ejecuta mientras Raquel
# sea menor o igual a Ayala
while Raquel <= Ayala:
    # 4. Carabajo muestra el valor actual
    Carabajo = Raquel
    print(Carabajo, end=" ")
    # IMPORTANTE: Incrementamos Raquel en 1 para evitar
    # un bucle infinito
    Raquel += 1
```

* Ejercicio 6

Muestra los números del 74 al 96 utilizando el lazo repetitivo “hacer mientras o hasta que” o “do while o repeat until”.

Algoritmo 3.65: Emulación de Do-While del 74 al 96

```
# 1. Raquel define el inicio del conteo
Raquel = 74
# 2. Ayala define el limite final
Ayala = 96
print(f"Conteo (estilo Do-While) desde {Raquel} hasta {Ayala}:")
# 3. Iniciamos el bucle infinito controlado
while True:
    # Carabajo toma el valor actual e imprimimos
    Carabajo = Raquel
```

```
    print(Carabajo, end=" ")
    # Incrementamos el contador
    Raquel += 1
    # 4. Condicion de salida (el "Hasta que" o "Until")
    # Si Raquel supera el limite, rompemos el ciclo
    if Raquel > Ayala:
        break
```

* Ejercicio 7

Muestra los números del 11 al 99 en bloques de diez en diez utilizando el lazo repetitivo “para” o “for”.

Algoritmo 3.66: Números del 11 al 99 en bloques de 10

```
# 1. Raquel define el rango del 11 al 99
# (usamos 100 para incluir el 99)
Raquel = range(11, 100)
# 2. Carabajo sirve para contar cuantos numeros van en la fila actual
Carabajo = 0
print("Valores en bloques de diez:")
# 3. Ayala recorre cada valor dentro de Raquel
for Ayala in Raquel:
    print(Ayala, end=" ")
    Carabajo += 1
    # 4. Cuando Carabajo llega a 10, hacemos un salto de linea
    if Carabajo == 10:
        print() # Salto de linea
        Carabajo = 0
```

* Ejercicio 8

Ingrese una cantidad de x números del uno (1) al noventa y nueve (99), su programa finaliza cuando el usuario ingresa –888. A continuación, se mostrará en pantalla cuántos números son: impares, pares y primos.

Algoritmo 3.67: Clasificador de números

```
# 1. Ayala almacena los contadores en una lista pares, impares, primos
Ayala = [0, 0, 0]
print("Ingrese valores del 1 al 99 (Use -888 para salir):")
while True:
    # 2. Raquel recibe la entrada del usuario
    Raquel = int(input("Ingrese un numero: "))
    # Condicion de salida
    if Raquel == -888:
        break
    # Validacion de rango
    if 1 <= Raquel <= 99:
        # A. Clasificacion Par/Impar
        if Raquel % 2 == 0:
            Ayala[0] += 1 # Es par
        else:
            Ayala[1] += 1 # Es impar
        # B. Clasificacion de Primos (Carabajo)
        if Raquel > 1:
            es_primo = True
            for Carabajo in range(2, int(Raquel**0.5) + 1):
                if Raquel % Carabajo == 0:
                    es_primo = False
```

```
                break
            if es_primo:
                Ayala[2] += 1
    else:
        print("Numero fuera de rango (1-99). Intente de nuevo.")
# 3. Mostrar resultados finales
print("\n--- RESUMEN DE CONTEO ---")
print(f"Numeros Pares: {Ayala[0]}")
print(f"Numeros Impares: {Ayala[1]}")
print(f"Numeros Primos: {Ayala[2]}")
```

* Ejercicio 9

Realizar la sumatoria de los números del 20 al 30.

Algoritmo 3.68: Sumatoria de números del 20 al 30

```
# 1. Definimos el rango de inicio y fin
Raquel = 20
Ayala = 30
# 2. Carabajo sirve para guardar la suma acumulada (empieza en 0)
Carabajo = 0
# 3. Utilizamos un bucle for para recorrer el rango
# Usamos Ayala + 1 para que el numero 30 sea incluido
for i in range(Raquel, Ayala + 1):
    Carabajo += i
# 4. Mostramos el resultado de la sumatoria
print(f"La sumatoria de los numeros del {Raquel} al {Ayala}", end="")
print(f" es: {Carabajo}")
```

* Ejercicio 10

Realiza la simulación del lanzamiento de dos dados. Recuerda que son aleatorios el valor a obtener y solo puede generarse los números del 1 al 6.

Algoritmo 3.69: Simulación de lanzamiento de dos dados

```
import random
# 1. Raquel simula el lanzamiento del primer dado (rango 1 a 6)
Raquel = random.randint(1, 6)
# 2. Ayala simula el lanzamiento del segundo dado
Ayala = random.randint(1, 6)
# 3. Carabajo calcula la suma de ambos resultados
Carabajo = Raquel + Ayala
# 4. Mostrar los resultados en pantalla
print("--- Resultado del Lanzamiento ---")
print(f"Dado 1: {Raquel}")
print(f"Dado 2: {Ayala}")
print(f"Suma total: {Carabajo}")
# Verificacion especial (si ambos dados son iguales)
if Raquel == Ayala:
    print("Genial! Sacaste un par.")
```

* Ejercicio 11

Emula un juego del lanzamiento de dos dados, si suman 7 tres veces gana el jugador.

Algoritmo 3.70: Juego: Tres veces Siete

```
import random
# 1. Carabajo funciona como nuestro contador de victorias (inicia en 0)
Carabajo = 0
```

```
intentos = 0
print("Iniciando el juego... El objetivo es sumar 7 tres veces.")
# 2. El bucle continua hasta que Carabajo llegue a 3
while Carabajo < 3:
    intentos += 1
    # Simulamos el lanzamiento de los dos dados
    Raquel = random.randint(1, 6)
    Ayala = random.randint(1, 6)
    suma = Raquel + Ayala
    print(f"Lanzamiento {intentos}: [{Raquel}] + [{Ayala}] = {suma}")
    # 3. Verificamos si la suma es 7
    if suma == 7:
        Carabajo += 1
        print(f"--> Llevas {Carabajo} vez/veces sumando siete!")
# 4. Al salir del bucle, el jugador ha ganado
print(f"\nFELICIDADES! Ganaste el juego en {intentos} lanzamientos.")
```

* Ejercicio 12

Una versión mejorada del juego anterior, para saber cuántas veces debe jugarse el juego hasta ganar 3 veces.

Algoritmo 3.71: Simulador de intentos hasta ganar

```
import random
# 1. Carabajo cuenta cuantas veces hemos sumado 7 (necesitamos 3)
Carabajo = 0
# 2. Variable para contar el total de jugadas realizadas
total_jugadas = 0
print("Contando lanzamientos hasta obtener tres '7'...")
# El bucle se repite hasta que Carabajo sea igual a 3
while Carabajo < 3:
    total_jugadas += 1
    # 3. Raquel y Ayala simulan los dados en cada jugada
    Raquel = random.randint(1, 6)
    Ayala = random.randint(1, 6)
    # Verificamos si la suma de esta jugada es 7
    if (Raquel + Ayala) == 7:
        Carabajo += 1
        print(f"Jugada {total_jugadas}: [{Raquel}+{Ayala}] ", end="")
        print(f"= 7 (Victoria {Carabajo}/3)")
# 4. Resultado final de la simulacion
print("-" * 40)
print(f"EL JUEGO HA TERMINADO.")
print(f"Un total de {total_jugadas} jugadas para ganar 3 veces.")
```



Laboratorio 11

Capítulo 3

3.7. Ejercicios del Laboratorio 11

Ejercicio 1

A continuación, tienes bloques de códigos que representan la secuencia de creación de archivos. Tu misión, si decides aceptar, es hacer funcionar todo el código.

Archivo: ejercicios.py

Algoritmo 3.72: Proyecto de misión parte 1

```
from math import *
print(" / ")
print(" / ")
print(" / ")
print("/ ")
##############################
character_name = "Joe"
character_age = "35"
age = 45
isMale = True
is_male = False
print(" Hola soy "+character_name+" Llerena")
print(" y tengo "+character_age)
print("Su alumno es "+str(is_male))
print("Su hermano tiene "+str(age)+" anios")
##############################
# Usando strings
print (" Universidad \n Politecnica \n Salesiana")
frase = "Esto es un string o cadena de caracteres "
print (frase)
print (frase.index("d"))
print (frase.replace("es","mira a"))
##############################
# visualizar numeros
my_num = -4
print(str(my_num))
print (abs(my_num))
print (str(abs(my_num)) + " es mi numero preferido")
print(pow(3, 2))
my_num = 4
print (round(3.4))
print (floor(my_num))
print (sqrt(my_num))
# get input de users
name = input("Ingrese su nombre ")
age = input("Ingrese su edad ")
print ("Su nombre es " + name +" y su edad es "+ age)
num1 = input("Ingrese un primer numero ")
num2 = input("Ingrese un segundo numero ")
result = num1 + num2 # no es correcto
result1 = int(num1) + int(num2) # es correcto
print (result)
print (result1)
```

Archivo: principal1.py

Algoritmo 3.73: Proyecto de misión parte 2

```
# forma 1
# import funciones
# funciones.menu()
# forma 2
from funciones import menu, metodo1, factorial
menu()
```

Archivo: funciones.py

Algoritmo 3.74: Proyecto de misión parte 2

```
def menu():
    while True:
        opcion = input("Ingrese una opcion: ")
        if opcion==1:
            print ("metodo 1")
            metodo1()
        elif opcion==2:
            print ("metodo 2")
            n = raw_input("Ingrese numero: ")
            r = factorial(n)
            print ("resultado es "+str(r))
        elif opcion==3:
            print ("metodo 1")
        elif opcion==4:
            print ("metodo 1")
        elif opcion==5:
            print ("metodo 1")
        elif opcion==6:
            print ("metodo 1")
        elif opcion==7:
            print ("Salir")
            break
def metodo1():
    nombre = raw_input("Ingrese su nombre: ")
    veces = input("Ingrese veces a repetir: ")
    for i in range (veces):
        print (str(i+1) + " "+ nombre)
def factorial(a):
    fac = 1
    for i in range(1, int(a)+1):
        fac = fac * (i)
    return fac
```

Ejercicio 2

El código que se presenta a continuación, debe ser complementado con las líneas de código adecuadas además de corregir aquellas líneas que no corresponden a la sencuencia lógica. Agrega e importa aquellas librerías necesarias para que el algoritmos se ejecute sin error.

Algoritmo 3.75: Nueva Misión a completar el siguiente código

```
def rellenar_matriz(filas, columnas, matriz):
  for f in range(filas):
    matriz.append([])
    for c in range(columnas):
     print("digite el valor de la posicion fila: ",(f+1),end="")
     print("columna: ", (c+1))
     matriz[f].append(int(input()))
  print (matriz)
def sumar_matriz (matriz, filasu, columnas, filas):
sumafila=0
for x in range (filas):
   sumafila =sumafila+ matriz [filasu][x]
   print("la suma es: " , sumafila)
opciones =["1", "2", "3", "4", "5", "6"]
while True:
  print('''
Selecione una opcion(1, 2, 3, 4, 5, 6):
1. Rellenar matriz
2. Suma Fila
3. Suma columna
4. Suma Diagonal principal
5. Suma Diagonal Inversa
6. Salir
''')
opcion=input("Introduce una alternativa: ")
if not (opcion in opciones):
print("No selecciono ninguna opcion valida")
input ("Pulse para continuar")
continue
if (opcion =="1"):
try:
     filas=int(input("ingrese la cantidad de filas"))
     columnas=int(input("ingrese la cantidad de columnas"))
     matriz=[]
    rellenar_matriz(filas, columnas, matriz)
    rellenado= True
except:
    input = ("Pulse para escoger")
    continue
if (opcion =="2"):
try:
     if rellenado:
             filasu=int(input("Digite la fila a Sumar"))
             if (filasu >= 0 and filasu <=filas):
sumar_matriz(matriz, filasu, columnas, filas)
print("Ya se lleno la matriz")
else:
print ("Digite una Fila en el valor correcto")
except:
    print("falta no se rellena")
```



Laboratorio 12

Capítulo 3

3.8. Ejercicios del Laboratorio 12

* Ejercicio 1

Lea una matriz cuadrada. Descomponga la matriz en tres matrices: submatriz debajo de la diagonal, submatriz diagonal, y submatriz sobre la diagonal. Verifique que la suma de las tres matrices coincide con la matriz original. Muestre las matrices obtenidas.

Algoritmo 3.76: Descomposición de Matriz Cuadrada

```
import numpy as np
# 1. Definir el tamano de la matriz (Llerena)
Llerena = int(input("Ingrese el tamano de la matriz cuadrada (n): "))
# 2. Leer la matriz original (Raquel)
print(f"Ingrese los elementos de la matriz {Llerena}x{Llerena}:")
Raquel = []
for i in range(Llerena):
    fila = [float(x) for x in input(f"Fila {i+1}: ").split()]
    Raquel.append(fila)
Raquel = np.array(Raquel)
# 3. Inicializar las tres matrices con ceros
Joe = np.zeros((Llerena, Llerena)) # Debajo
Anthony = np.zeros((Llerena, Llerena)) # Diagonal
Natalia = np.zeros((Llerena, Llerena)) # Sobre
# 4. Descomposicion logica
for i in range(Llerena):
    for j in range(Llerena):
        if i > j:
            Joe[i][j] = Raquel[i][j] # i > j: Debajo de la diagonal
        elif i == j:
            Anthony[i][j] = Raquel[i][j] # i == j: Diagonal principal
        else:
            Natalia[i][j] = Raquel[i][j] # i < j: Sobre la diagonal
# 5. Verificacion de la suma (Ayala)
Ayala = Joe + Anthony + Natalia
# 6. Mostrar resultados
print("\n--- MATRIZ ORIGINAL (Raquel) ---")
print(Raquel)
print("\n--- SUBMATRIZ DEBAJO DE LA DIAGONAL (Joe) ---")
print(Joe)
print("\n--- SUBMATRIZ DIAGONAL (Anthony) ---")
print(Anthony)
print("\n--- SUBMATRIZ SOBRE LA DIAGONAL (Natalia) ---")
print(Natalia)
if np.array_equal(Ayala, Raquel):
    print("\nVERIFICACION: La suma coincide perfectamente")
    print("\ncon la matriz original.")
```

* Ejercicio 2

Lea una matriz n x m. Para cada fila, muestre el producto de los elementos cuyo valor es un número par.

Algoritmo 3.77: Producto de elementos pares por fila

```
import numpy as np
# 1. Joe y Anthony definen las dimensiones de la matriz
Joe = int(input("Ingrese el numero de filas (n): "))
Anthony = int(input("Ingrese el numero de columnas (m): "))
```

```
# 2. Raquel almacena la matriz cargada por el usuario
print(f"Ingrese los elementos de la matriz {Joe}x{Anthony}:")
Raquel = []
for i in range(Joe):
    fila = [int(x) for x in input(f"Fila {i+1}: ").split()]
    Raquel.append(fila)
# 3. Procesamiento por fila
print("\n--- Resultados del producto de pares por fila ---")
for i in range(Joe):
    Natalia = 1 # Reiniciamos el producto para la nueva fila
    Ayala = False # Bandera para saber si encontramos al menos un par
    for j in range(Anthony):
        Llerena = Raquel[i][j]

        # Verificamos si Llerena es un numero par
        if Llerena % 2 == 0:
            Natalia *= Llerena
            Ayala = True
    # 4. Mostrar el resultado de la fila actual
    if Ayala:
        print(f"Fila {i+1}: El producto de los pares es {Natalia}")
    else:
        print(f"Fila {i+1}: No se encontraron")
        print("numeros pares (Producto: 0)")
```

Ejercicio 3

Lea una matriz cuadrada. Muestre la suma de los elementos que no están en las dos diagonales principales.

Algoritmo 3.78: Suma de elementos fuera de las diagonales

```
import numpy as np
# 1. Joe define el tamano de la matriz cuadrada
Joe = int(input("Ingrese el tamano de la matriz cuadrada (n): "))
# 2. Raquel almacena la matriz
print(f"Ingrese los elementos de la matriz {Joe}x{Joe}:")
Raquel = []
for i in range(Joe):
    fila = [float(x) for x in input(f"Fila {i+1}: ").split()]
    Raquel.append(fila)
# 3. Anthony inicializa la suma en 0
Anthony = 0
print("\nElementos que no estan en las diagonales:")
# 4. Recorremos la matriz con Natalia (i) y Llerena (j)
for Natalia in range(Joe):
    for Llerena in range(Joe):
        Ayala = Raquel[Natalia][Llerena]
        # Condicion para DIAGONAL PRINCIPAL: Natalia == Llerena
        # Condicion para DIAGONAL SECUNDARIA:
        # Natalia + Llerena == Joe - 1
        if not (Natalia == Llerena or Natalia + Llerena == Joe - 1):
            Anthony += Ayala
            print(f"[{Ayala}]", end=" ")
# 5. Resultado final
print("\n\nLa suma de los elementos")
print(f"\n\nfuera de las diagonales es: {Anthony}")
```

Ejercicio 4

Un cuadrado semi-mágico es una matriz cuadrada conteniendo números tales que la suma, de las dos diagonales principales, producen el mismo resultado. Ejemplo. Un cuadrado semi-mágico de 4 filas y 4 columnas:

```
8 1 6 7
6 5 7 3
4 3 2 1
2 8 9 4
```

Escriba un programa que coloque números enteros aleatorios de una cifra en una matriz de 4 filas y 4 columnas. Repita el ciclo hasta que la matriz sea un cuadrado semi-mágico. El programa debe mostrar la matriz resultante y la cantidad de intentos que realizó el programa hasta llenar la matriz con éxito.

Algoritmo 3.79: Simulador del Cuadrado Semi-Magico 4x4

```
import random
# 1. Joe contara todos los intentos realizados
Joe = 0
Ayala = False
print("Buscando un cuadrado semi-magico...")
print("... Esto puede tardar unos segundos.")
# 2. El ciclo se repite hasta que Ayala sea True
while not Ayala:
    Joe += 1
    # Generamos una matriz 4x4 con numeros aleatorios de una cifra (0-9)
    Raquel = []
    for i in range(4):
        fila = [random.randint(0, 9) for _ in range(4)]
        Raquel.append(fila)
    # 3. Calcular suma de la diagonal principal (Anthony)
    Anthony = 0
    for i in range(4):
        Anthony += Raquel[i][i]
    # 4. Calcular suma de la diagonal secundaria (Natalia)
    Natalia = 0
    for i in range(4):
        Natalia += Raquel[i][3 - i]
    # 5. Verificar condicion de cuadrado semi-magico
    if Anthony == Natalia:
        Ayala = True
# 6. Mostrar resultados finales
print("\n--- CUADRADO ENCONTRADO! ---")
for fila in Raquel:
    print(fila)
print(f"\nSuma de las diagonales: {Anthony}")
print(f"Cantidad de intentos realizados: {Joe}")
```

Ejercicio 5

Escriba un programa que coloque números aleatorios de una cifra en los cuatro bordes de una matriz. Después, rellene los elementos del interior de la matriz con números aleatorios de una cifra, tales que cada uno sea menor o igual al promedio de todos los elementos en los bordes. Ejemplo. Matriz inicial de 4 x 4

```
8 1 6 7
6 0 0 3
4 0 0 1
2 8 9 4
```

El promedio es 4.91

Ejemplo. Matriz rellena

```
8 1 6 7
6 3 4 3
4 2 3 1
2 8 9 4
```

Al inicio no interesan los valores que se asignan a los elementos interiores, pues serán sustituidos. El programa debe mostrar la matriz resultante y la cantidad de intentos que realizó el programa hasta llenar la matriz con éxito.

Algoritmo 3.80: Relleno de matriz basado en promedio de bordes

```
import random
import math
# 1. Crear matriz vacia 4x4
Raquel = [[0 for _ in range(4)] for _ in range(4)]
Joe = 0
Natalia = 0
Llerena = 0 # Contador de intentos totales
# 2. Llenar los cuatro bordes con numeros aleatorios de una cifra (0-9)
for i in range(4):
   for j in range(4):
      # Condicion de borde: primera/ultima
      # fila o primera/ultima columna
      if i == 0 or i == 3 or j == 0 or j == 3:
         Raquel[i][j] = random.randint(0, 9)
         Joe += Raquel[i][j]
         Natalia += 1
# 3. Calcular el promedio de los bordes (Anthony)
Anthony = Joe / Natalia
# 4. Rellenar el interior (posiciones [1,1], [1,2], [2,1], [2,2])
# Repetimos intentos hasta que cada numero cumpla la condicion
for i in range(1, 3):
   for j in range(1, 3):
      exito = False
      while not exito:
         Llerena += 1 # Cada generacion cuenta como un intento
         Ayala = random.randint(0, 9)
         if Ayala <= Anthony:
            Raquel[i][j] = Ayala
            exito = True
# 5. Mostrar resultados
```

```
print(f"Promedio de los bordes: {Anthony:.2f}")
print(f"Intentos totales realizados: {Llerena}")
print("\n--- Matriz Resultante ---")
for fila in Raquel:
    print(fila)
```

Ejercicio 6

Escriba una función b = diagonales(a) que reciba una matriz nxn e intercambie los elementos de la diagonal principal con los elementos de la otra diagonal. Ejemplo. Matriz inicial de 4 x 4

```
3 2 7 9
6 5 3 7
8 8 1 6
3 5 9 2
```

Ejemplo. Matriz modificada

```
9 2 7 3
6 3 5 7
8 1 8 6
2 5 9 3
```

En la ventana interactiva, genere una matriz cuadrada con números aleatorios de una cifra. Llame a la función y verifique si el resultado es correcto.

Algoritmo 3.81: Intercambio de diagonales en matriz cuadrada

```
import random
def diagonales(Raquel):
    # Obtenemos el tamanio de la matriz (Anthony)
    Anthony = len(Raquel)
    # Copiamos la matriz para no modificar la original directamente
    Joe = [fila[:] for fila in Raquel]
    # Recorremos la matriz con Natalia (i)
    for Natalia in range(Anthony):
        # La diagonal principal es [Natalia][Natalia]
        # La diagonal secundaria es [Natalia][Anthony - 1 - Natalia]
        # Intercambio de valores
        Joe[Natalia][Natalia], Joe[Natalia][Anthony - 1 - Natalia] = \
        Joe[Natalia][Anthony - 1 - Natalia], Joe[Natalia][Natalia]
    return Joe

# --- Ventana Interactiva ---
# 1. Definir tamano y generar matriz aleatoria (Llerena)
Anthony = 4
Llerena = [[random.randint(0, 9) for _ in range(Anthony)]
           for _ in range(Anthony)]
print("--- Matriz Inicial (Llerena) ---")
for fila in Llerena:
    print(fila)
# 2. Llamar a la funcion y guardar en Ayala
Ayala = diagonales(Llerena)
print("\n--- Matriz Modificada (Ayala) ---")
for fila in Ayala:
    print(fila)
```

✳ Ejercicio 7

Escriba una función que reciba una matriz. La función debe entregar un vector con la cantidad de elementos pares que contiene cada columna de la matriz: Ejemplo. Entra la siguiente información:

```
3 4 5
6 1 8
8 6 3
7 8 7
```

sale [2, 3, 1]

Algoritmo 3.82: Conteo de pares por columna

```
def contar_pares_columnas(Raquel):
    # 1. Obtener dimensiones: Anthony (filas) y Natalia (columnas)
    Anthony = len(Raquel)
    Natalia = len(Raquel[0])
    # 2. Joe sera el vector donde guardaremos los resultados
    Joe = []
    # 3. Recorremos primero las columnas (Natalia)
    for j in range(Natalia):
        Llerena = 0 # Reiniciamos el contador de pares para cada columna
        # 4. Recorremos las filas para la columna actual
        for i in range(Anthony):
            Ayala = Raquel[i][j]
            # Verificamos si Ayala es par
            if Ayala % 2 == 0:
                Llerena += 1
        # Agregamos el resultado de la columna al vector Joe
        Joe.append(Llerena)
    return Joe
# --- Ejemplo de uso ---
matriz_ejemplo = [
    [2, 4, 1],
    [6, 8, 3],
    [1, 2, 5],
    [4, 6, 7]
]
resultado = contar_pares_columnas(matriz_ejemplo)
print(f"Vector de pares por columna: {resultado}")
```

✳ Ejercicio 8

Escriba un programa que lea una matriz, llame a la función creada y determine cuál es la columna con la mayor cantidad de números pares.

Algoritmo 3.83: Identificar columna con más pares

```
def contar_pares_columnas(matriz):
    filas = len(matriz)
    columnas = len(matriz[0])
    vector_pares = []
    for j in range(columnas):
        conteo = 0
        for i in range(filas):
            if matriz[i][j] % 2 == 0:
                conteo += 1
```

```
        vector_pares.append(conteo)
    return vector_pares
# --- Programa Principal ---
# 1. Leer dimensiones de la matriz
Anthony = int(input("Ingrese numero de filas (n): "))
Natalia = int(input("Ingrese numero de columnas (m): "))
# 2. Cargar la matriz Raquel
Raquel = []
print(f"Ingrese los elementos de la matriz {Anthony}x{Natalia}:")
for i in range(Anthony):
    fila = [int(x) for x in input(f"Fila {i+1}: ").split()]
    Raquel.append(fila)
# 3. Llamar a la funcion y obtener el vector Joe
Joe = contar_pares_columnas(Raquel)
# 4. Determinar la columna con mayor cantidad de pares
Llerena = max(Joe) # Cantidad maxima de pares
Ayala = Joe.index(Llerena)
# Indice de la primera columna que tiene ese maximo
# 5. Mostrar resultados
print(f"\nConteo de pares por columna: {Joe}")
print(f"La columna con mas pares es la columna {Ayala + 1}")
print(f"(con {Llerena} pares).")
```

* Ejercicio 9

Escriba una función con nombre suma(n) que entregue la suma de las cifras de un número dado n. Con esta función escriba un programa que genere 10 números aleatorios entre 1 y 100 y encuentre cuál de ellos tiene la mayor suma de sus cifras.

Algoritmo 3.84: Suma de cifras y búsqueda de máximo

```
import random
def suma(n):
    # La funcion recibe un numero n y suma sus digitos
    # Ayala actua como el acumulador de la suma
    Ayala = 0
    for cifra in str(abs(n)):
        Ayala += int(cifra)
    return Ayala
# --- Programa Principal ---
# 1. Generar 10 numeros aleatorios entre 1 y 100 en la lista Joe
Joe = [random.randint(1, 100) for _ in range(10)]
# 2. Inicializar Anthony (el numero) y Natalia (su suma)
# con el primer elemento
Anthony = Joe[0]
Natalia = suma(Joe[0])
print("Listado de numeros y sus sumas:")
# 3. Recorrer cada Raquel en la lista Joe
for Raquel in Joe:
    Llerena = suma(Raquel)
    print(f"Numero: {Raquel:3} | Suma de cifras: {Llerena}")
    # 4. Determinar si es la mayor suma encontrada hasta ahora
    if Llerena > Natalia:
        Natalia = Llerena
        Anthony = Raquel
# 5. Mostrar el resultado final
print("-" * 35)
print(f"El numero con mayor suma de cifras es: {Anthony}")
```

```
print(f"La suma obtenida fue: {Natalia}")
```

* Ejercicio 10

Escriba una función llamado cuadrado(n) que determine si el cuadrado de un número natural n dado, es igual a la suma de los primeros n números impares. Ej. 62 = 1+3+5+7+9+11 Escriba un programa de prueba que ingrese un dato desde el teclado, use la función y muestre el resultado en la pantalla.

Algoritmo 3.85: Verificación de propiedad de números impares

```
def cuadrado(n):
    # 1. Joe calcula el cuadrado del numero
    Joe = n ** 2
    # 2. Anthony acumulara la suma de los primeros n impares
    Anthony = 0
    # Llerena sera el primer numero impar
    Llerena = 1
    # 3. Sumamos los primeros n impares usando un bucle Natalia
    for Natalia in range(n):
        Anthony += Llerena
        # Avanzamos al siguiente numero impar
        Llerena += 2
    # 4. Ayala determina si la propiedad se cumple
    Ayala = (Joe == Anthony)
    # Devolvemos el resultado y el valor de la suma para informar
    return Ayala, Joe, Anthony
# --- Programa de Prueba ---
# Raquel recibe el dato desde el teclado
Raquel = int(input("Ingrese un numero natural (n): "))
if Raquel > 0:
    # Llamada a la funcion
    es_igual, valor_cuadrado, valor_suma = cuadrado(Raquel)
    # Mostrar resultados
    print(f"\nAnalisis para n = {Raquel}:")
    print(f"El cuadrado (n^2) es: {valor_cuadrado}")
    print(f"La suma de los primeros {Raquel} impares es: {valor_suma}")
    if es_igual:
        print("Resultado: La propiedad se cumple!")
    else:
        print("Resultado: La propiedad no se cumple.")
else:
    print("Por favor, ingrese un numero natural mayor a 0.")
```

* Ejercicio 11

Escriba una función secuencia1(n) que entregue el n-ésimo término de la siguiente secuencia, en la cual cada término, a partir del tercero, se obtiene sumando los dos anteriores: 1, 1, 2, 3, 5, 8, 13, 21, ... Escriba un programa de prueba que ingrese un dato desde el teclado, use la función y muestre el resultado en la pantalla.

Algoritmo 3.86: Cálculo del n-esimo término de Fibonacci

```
def secuencia1(n):
    # Casos base: si n es 1 o 2, el resultado es 1
    if n <= 2:
        return 1
    # Inicializamos los dos primeros terminos
    Natalia = 1 # n-2
```

```
    Anthony = 1 # n-1
    Joe = 0
    # Calculamos desde el tercer termino hasta n
    for Llerena in range(3, n + 1):
        Joe = Anthony + Natalia
        # Desplazamos los valores para la siguiente iteracion
        Natalia = Anthony
        Anthony = Joe
    return Joe
# --- Programa de Prueba ---
# Raquel recibe la posicion deseada desde el teclado
Raquel = int(input("Ingrese la posicion (n) de la secuencia: "))
if Raquel > 0:
    # Ayala guarda el resultado de llamar a la funcion
    Ayala = secuencia1(Raquel)

    print(f"\nEl termino en la posicion {Raquel} es: {Ayala}")
    # Ejemplo visual de la secuencia para n=8
    if Raquel == 8:
        print("Secuencia: 1, 1, 2, 3, 5, 8, 13, 21")
else:
    print("Por favor, ingrese un numero entero positivo.")
```

* Ejercicio 12

Escriba una función secuencia2(n) que entregue el n-ésimo término de la siguiente secuencia, en la cual cada término, a partir del cuarto, se obtiene sumando los tres anteriores: 1, 1, 1, 3, 5, 9, 17, 31, 57... Escriba un programa de prueba que ingrese un dato desde el teclado, use la función y muestre el resultado en la pantalla.

Algoritmo 3.87: Calculo del n-esimo termino de Tribonacci

```
def secuencia2(n):
    # Casos base: si n es 1, 2 o 3, el resultado es 1
    if n <= 3:
        return 1
    # Inicializamos los tres primeros terminos
    Llerena = 1 # n-3
    Natalia = 1 # n-2
    Anthony = 1 # n-1
    Joe = 0
    # Calculamos desde el cuarto termino hasta n
    for i in range(4, n + 1):
        Joe = Anthony + Natalia + Llerena
        # Desplazamos los valores para la siguiente iteracion
        Llerena = Natalia
        Natalia = Anthony
        Anthony = Joe
    return Joe

# --- Programa de Prueba ---
# Raquel recibe la posicion deseada desde el teclado
Raquel = int(input("Ingrese (n) para la secuencia Tribonacci: "))
if Raquel > 0:
    # Ayala guarda el resultado de llamar a la funcion
    Ayala = secuencia2(Raquel)

    print(f"\nEl termino en la posicion {Raquel} es: {Ayala}")
```

```
else:
    print("Por favor, ingrese un numero entero positivo.")
```

Ejercicio 13

Escriba una función simetrico(x) que reciba un entero y determine si es simétrico, es decir, si los dígitos opuestos alrededor del centro son iguales. Escriba un programa de prueba que genere números aleatorios entre 1 y 10000 hasta obtener un número que sea simétrico.

Algoritmo 3.88: Búsqueda aleatoria de números simétricos

```
import random
def simetrico(x):
    # Convertimos el numero a texto para comparar extremos
    # Anthony guardara la cadena de texto
    Anthony = str(x)
    # Comparamos la cadena con su version invertida
    return Anthony == Anthony[::-1]

# --- Programa de Prueba ---
# Natalia contara cuantos numeros generamos hasta tener exito
Natalia = 0
Joe = False
print("Buscando un numero simetrico entre 1 y 10000...")
while not Joe:
    Natalia += 1
    # Generamos un numero aleatorio (Raquel)
    Raquel = random.randint(1, 10000)
    # Verificamos si es simetrico llamando a la funcion
    Joe = simetrico(Raquel)
    # Opcional: imprimir intentos para ver el progreso
    if Natalia % 10 == 0 or Joe:
        print(f"Intento {Natalia}: {Raquel} ->")
        print(f"{'Simetrico!' if Joe else 'No'}")
# --- Resultado Final ---
print("-" * 40)
print(f"Bien! El numero simetrico encontrado es: {Raquel}")
print(f"Se realizaron {Natalia} intentos.")
```

Esta siguiente versión utiliza aritmética pura (divisiones y residuos) para “voltear” el número sin depender de herramientas de texto. Es la forma clásica en la que se enseña lógica de programación y matemáticas computacionales.

Algoritmo 3.89: Simetría numérica mediante lógica matemática

```
import random
def simetrico(x):
    # Caso para numeros de una sola cifra
    if x < 10:
        return True
    Joe = x
    Anthony = 0
    # 1. Construimos el numero invertido matematicamente
    while Joe > 0:
        # Extraemos el ultimo digito (Llerena)
        Llerena = Joe % 10
        # Lo agregamos a Anthony desplazando los anteriores una posicion
        Anthony = (Anthony * 10) + Llerena
        # Eliminamos el ultimo digito de Joe
```

```
        Joe = Joe // 10
    # 2. Si el original es igual al invertido, es simetrico
    return x == Anthony

# --- Programa de Prueba ---
Natalia = 0
Ayala = False
print("Buscando numero simetrico (Version Matematica)...")
while not Ayala:
    Natalia += 1
    Raquel = random.randint(1, 10000)
    # Verificamos la simetria
    Ayala = simetrico(Raquel)
    if Ayala:
        print(f"\nENCONTRADO!")
        print(f"Numero: {Raquel}")
        print(f"Intentos: {Natalia}")
```

* Ejercicio 14

Escriba una función final(n) que entregue como resultado la cantidad de veces que debe lanzarse un dado hasta que salga un número n dado como parámetro. Escriba un programa de prueba que ingrese un dato desde el teclado, use la función y muestre el resultado en la pantalla.

Algoritmo 3.90: Simulación de lanzamientos hasta obtener n

```
import random
def final(Raquel):
    # 1. Joe contara cuantas veces lanzamos el dado
    Joe = 0
    Ayala = False
    # 2. El bucle sigue hasta que el dado sea igual a Raquel
    while not Ayala:
        Joe += 1
        # Anthony simula el lanzamiento del dado (1 a 6)
        Anthony = random.randint(1, 6)
        # Verificamos si logramos el objetivo
        if Anthony == Raquel:
            Ayala = True
    return Joe

# --- Programa de Prueba ---
# Natalia recibe el numero deseado por el usuario
Natalia = int(input("Ingrese el numero que desea obtener (1-6): "))
# Validacion de entrada
if 1 <= Natalia <= 6:
    # Llerena recibe la cantidad de intentos desde la funcion
    Llerena = final(Natalia)
    print("-" * 45)
    print(f"Para obtener el numero {Natalia}, el dado")
    print(f"tuvo que lanzarse {Llerena} veces.")
    print("-" * 45)
else:
    print("Error: El numero debe estar entre 1 y 6.")
```

* Ejercicio 15

Escriba una función conteo(x) que determine la cantidad de términos que deben sumarse de la serie: 1*2*3 + 2*3*4 + 3*4*5 + 4*5*6 + ... hasta que la suma exceda a un valor x dado. Escriba

un programa de prueba que genere un número aleatorio para x entre 1 y 1000, use la función y muestre el resultado en la pantalla.

Algoritmo 3.91: Cálculo de términos necesarios para exceder x

```
import random
def conteo(Raquel):
    # 1. Inicializamos variables
    Joe = 0 # Suma acumulada
    Natalia = 0 # Cantidad de terminos
    Llerena = 1 # Base del termino (i)
    # 2. Bucle hasta que la suma Joe exceda el limite Raquel
    while Joe <= Raquel:
        # Calculamos el termino Anthony: i * (i+1) * (i+2)
        Anthony = Llerena * (Llerena + 1) * (Llerena + 2)
        # Sumamos al acumulador y aumentamos contadores
        Joe += Anthony
        Natalia += 1
        Llerena += 1
    return Natalia, Joe

# --- Programa de Prueba ---
# 3. Ayala genera un numero aleatorio entre 1 y 1000
Ayala = random.randint(1, 1000)
# 4. Llamamos a la funcion
total_terminos, suma_final = conteo(Ayala)
# 5. Mostrar resultados
print(f"Valor limite generado (x): {Ayala}")
print("-" * 45)
print(f"Cantidad de terminos necesarios: {total_terminos}")
print(f"Suma final alcanzada: {suma_final}")
print("-" * 45)
```

Ejercicio 16

Realiza un programa que almacene en una variable a la lista obtenida con range(1,4) y, a continuación, la modifique para que cada componente sea igual al cuadrado del componente original. El programa mostrará la lista resultante por pantalla.

Algoritmo 3.92: Modificación de elementos de una lista al cuadrado

```
# 1. Almacenamos en Raquel la lista obtenida con range(1, 4)
# Nota: range(1, 4) genera [1, 2, 3]
Raquel = list(range(1, 4))
print(f"Lista original: {Raquel}")
# 2. Modificamos cada componente por su cuadrado
# Usamos Joe como indice para recorrer la posicion 0, 1 y 2
for Joe in range(len(Raquel)):
    # Anthony guarda el valor original de la posicion actual
    Anthony = Raquel[Joe]
    # Natalia calcula el cuadrado
    Natalia = Anthony ** 2
    # Actualizamos la componente original en Raquel
    Raquel[Joe] = Natalia
# 3. Mostramos la lista resultante
print(f"Lista modificada (cuadrados): {Raquel}")
```

En esta versión siguiente, la técnica de List Comprehension (comprensión de listas) es una de las características más potentes y elegantes de Python. Permite crear o transformar listas en una

sola línea de código, siendo más eficiente que un bucle for tradicional.

Algoritmo 3.93: Transformación de lista usando List Comprehension

```
# 1. Creamos la lista inicial en Raquel
Raquel = list(range(1, 4))
# 2. Aplicamos List Comprehension para calcular los cuadrados
# Anthony = [expresion for elemento in lista]
Anthony = [Joe ** 2 for Joe in Raquel]
# 3. Mostrar resultados
print(f"Lista original (Raquel): {Raquel}")
print(f"Lista resultante (Anthony): {Anthony}")
```

* Ejercicio 17

Realiza un programa que almacene en una lista obtenida con range(1, n), donde n es un entero que se pide al usuario y modifique dicha lista para que cada componente sea igual al cuadrado del componente original. El programa mostrará la lista resultante por pantalla.

Algoritmo 3.94: Cuadrados de una lista de tamaño n

```
# 1. Raquel recibe el limite superior n desde el teclado
Raquel = int(input("Ingrese el valor de n para crear la lista: "))
# 2. Joe almacena la lista inicial.
# Nota: range(1, Raquel) genera numeros desde 1 hasta n-1
Joe = list(range(1, Raquel))
# 3. Anthony usa List Comprehension para calcular los cuadrados
Anthony = [Natalia ** 2 for Natalia in Joe]
# 4. Mostrar los resultados
print(f"\nLista original (Joe): {Joe}")
print(f"Lista de cuadrados (Anthony): {Anthony}")
```

* Ejercicio 18

Realiza un programa que, dada una lista a cualquiera, sustituya cualquier elemento negativo por cero.

Algoritmo 3.95: Sustitución de negativos por cero

```
# 1. Definimos una lista Raquel con valores variados
Raquel = [10, -5, 3, -1, 0, 8, -22, 15]
# 2. Usamos List Comprehension con una condicion (if-else)
# Sintaxis: [valor_si_true if condicion else valor_si_false
# for elemento in lista]
Joe = [Anthony if Anthony >= 0 else 0 for Anthony in Raquel]
# 3. Mostrar resultados
print(f"Lista original (Raquel): {Raquel}")
print(f"Lista modificada (Joe): {Joe}")
```

* Ejercicio 19

Diseña un programa que construya una lista con los n primeros números primos (ojo: no los primos entre 1 y n, sino los n primeros números primos). ¿Necesitas usar append? ¿Puedes reservar en primer lugar un vector con n celdas nulas y asignarle a cada una de ellas uno de los números primos?

Algoritmo 3.96: Lista de los n primeros primos sin usar append

```
def es_primo(num):
    if num < 2: return False
    for i in range(2, int(num**0.5) + 1):
        if num % i == 0:
```

```
            return False
    return True

# --- Programa Principal ---
# 1. Raquel es la cantidad de primos solicitada
Raquel = int(input("Cuantos numeros primos deseas obtener?: "))
# 2. Reservamos un vector con n celdas nulas (None o 0)
# Esto evita el uso de .append()
Joe = [None] * Raquel
Anthony = 0 # Contador de primos encontrados
Natalia = 2 # Primer numero a evaluar
# 3. Bucle hasta llenar las celdas de Joe
while Anthony < Raquel:
    # Ayala verifica si Natalia es primo
    Ayala = es_primo(Natalia)
    if Ayala:
        # Asignamos directamente a la celda usando el indice Llerena
        Llerena = Anthony
        Joe[Llerena] = Natalia
        Anthony += 1
    # Pasamos al siguiente numero para evaluar
    Natalia += 1
# 4. Mostrar el resultado
print(f"\nLos {Raquel} primeros numeros primos son:")
print(Joe)
```

Ejercicio 20

Diseña un programa que elimine de una lista todos los elementos de índice par y muestre por pantalla el resultado. (Ejemplo: si trabaja con la lista [1, 2, 1, 5, 0, 3], esta pasará a ser [2, 5, 3]).

Algoritmo 3.97: Eliminación de índices pares

```
# 1. Definimos la lista original Raquel
Raquel = [1, 2, 1, 5, 0, 3]
Llerena = len(Raquel)
# METODO 1: Usando Slicing (Rebanado)
# Sintaxis: lista[inicio:fin:paso]
# Empezamos en 1, hasta el final, saltando de 2 en 2
Joe = Raquel[1::2]
print(f"Lista original: {Raquel}")
print(f"Resultado (solo indices impares): {Joe}")
# METODO 2: Usando List Comprehension (para entender la logica)
# Solo incluimos a Natalia si su indice Anthony es impar
Joe_v2 = [Raquel[Anthony] for Anthony in range(Llerena)
       if Anthony % 2 != 0]
# Comprobacion: [2, 5, 3]
```

Nota adicional: El operador is en Python es muy especial porque no compara si dos valores son iguales (como lo hace ==), sino si dos objetos son exactamente el mismo en la memoria (es decir, si comparten la misma dirección de memoria).

Una versión que utiliza is para identificar qué elementos de la lista original no son los mismos que los elementos de índice par que queremos descartar.

Algoritmo 3.98: Uso de is para filtrar elementos

```
# 1. Lista original Raquel
Raquel = [1, 2, 1, 5, 0, 3]
# 2. Creamos una lista con los elementos que queremos ELIMINAR (indices pares)
```

```
# Estos son: Raquel[0], Raquel[2], Raquel[4] -> [1, 1, 0]
eliminados = Raquel[0::2]
# 3. Usamos List Comprehension con 'is not'
# Nota: Esto solo funciona de forma segura si los objetos son unicos
# o si comparamos las posiciones de memoria.
Joe = []
for Anthony in range(len(Raquel)):
    Natalia = Raquel[Anthony]
    # Si el indice Anthony es impar, lo agregamos a Joe
    if not (Anthony % 2 == 0):
        Joe.append(Natalia)
print(f"Lista resultante: {Joe}")
# --- Ejemplo tecnico de 'is' ---
# Si quisieras saber si dos listas son la misma:
a = [1, 2, 3]
b = a
c = [1, 2, 3]
print(f"a is b: {a is b}") # True (mismo objeto)
print(f"a is c: {a is c}") # False (mismo valor, distinto objeto)
```

Se usa == (Igualdad) si es para compruebar si dos variables tienen el mismo valor (ej. si ambos son 5). Y se usa is (Identidad) para comprobar si dos variables son el mismo "recipiente" en la memoria del computador. Es decir, se usa is principalmente para comparar con None (ej. if x is None) o para verificar si dos variables apuntan a la misma lista u objeto complejo.

Caso 1: En el ejercicio donde reservamos un vector con n celdas nulas (Joe = [None] * Raquel), el uso de is es la mejor práctica de la industria.

Algoritmo 3.99: Uso recomendado de is con None

```
# En el ejercicio de los n primeros primos:
Joe = [None] * Raquel
# Para verificar si una celda sigue vacia:
if Joe[0] is None:
    print("La primera posicion aun no tiene un primo")
```

None es un objeto único (singleton) en Python. Solo existe un None en toda la memoria del programa, por lo que is None es más rápido y seguro que == None.

Caso 2: Para verificar si dos listas son la misma (Identidad).

En los ejercicios de modificar listas, is es útil si necesitas saber si estás trabajando sobre la lista original o sobre una copia.

Algoritmo 3.100: Verificar si es el mismo objeto lista

```
Raquel = [1, 2, 3]
Joe = Raquel # Joe es un alias (misma direccion)
Anthony = Raquel[:] # Anthony es una copia (distinta direccion)
print(Joe is Raquel) # True: Son el mismo objeto
print(Anthony is Raquel)
# False: Tienen los mismos valores pero son distintos objetos
```

Casos donde NO es recomendable el uso de is (Es un peligro) De los ejercicios anteriores, en aquellos, donde el uso de is podría fallar son en el de Suma de cifras / Simetría: Si comparas Anthony is Raquel para ver si los números son iguales, podrías obtener resultados inconsistentes. Python "pre-crea" los números del -5 al 256 en memoria, pero para números más grandes (como 10,000), cada uno es un objeto nuevo. El otro ejercicio es en Índices pares: Como discutimos antes, si comparas valores numéricos con is, podrías terminar eliminando elementos que no querías solo porque "se parecen" en memoria.



Guido van Rossum, creador de Python

¡Muy buen trabajo!

Finalización del capítulo 3

DOI: 10.17163/abyaups.167.4

Capítulo 4
Los recursos multimedia y su importancia

Resultado de aprendizaje

Comprende el mundo de la Programación mediante experiencias ejemplares.

Indicador de logro

Identifica autores relevantes en el ámbito de la Programación.
Utiliza los recursos existentes en el medio con diferentes formatos.

Todo buen trabajo empieza con una buena motivación [Mendoza Solórzano, 2024; Llerena-Izquierdo, 2022; Quishpi Caldas, 2025]. Por eso es importante, adecuado y esencial tener herramientas, recursos e instrumentos que permitan lograr la motivación requerida [Llerena Izquierdo & Ayala Carabajo, 2018; Ayala Carabajo, 2009; Puga Paredes, 2023]. En este primer recurso se presenta un video motivador, que representa esa primera idea que alimenta la mecha (trabajo a realizar) para encender una de tantas llamas (tu propósito) al empezar el curso introductorio [Llerena-Izquierdo & Ayala-Carabajo, 2018; Ayala Carabajo, 2011; Montero Castillo, 2025], ver Tabla 4.1.

Video de Bienvenida

Tabla 4.1: Video de bienvenida al contenido del libro

	Título	Intervienen	Misión	Valores
Al inicio de todo trabajo: Bienvenida al contenido del libro				
. clic	Inicio	Estudiantes	Motivar a un buen inicio de clases	Motivación
Sitio: https://youtu.be/VmArGOLynhE?si=hsOvytZHrReQccBk				

Los videos permiten desarrollar estrategias de pronta intervención sin necesidad de la presencialidad, es un recurso compartido [Valverde Rivera, 2023; Tumbaco-Loor & Llerena-Izquierdo, 2024] y manteniendo un cuidado de cada uno de los capítulos o unidades de estudio[1], [Carabajo, 2023; Ayala, 2007; Bravo Quezada, 2025; Santamaria Chico, 2024], ver Tabla 4.2.

[1] Recursos en GitHub

Canales de videos por capítulos del libro

Tabla 4.2: Canal de videos para las guías de aprendizaje por capítulos y sus laboratorios

Capítulo 1 - Guía de aprendizaje 1: Laboratorios 1 y 2	
	Sitio: https://www.youtube.com/playlist?list=PLqcsZoFzFKH2N_P-yfYRyoBS7eE2DC5FW
Capítulo 2 - Guía de aprendizaje 2: Laboratorios 3, 4, 5 y 6	
	Sitio: https://www.youtube.com/playlist?list=PLqcsZoFzFKH2CDXRk00EC86q5V45HsFPC
Capítulo 3 - Guía de aprendizaje 3: Laboratorios 7, 8, 9, 10, 11 y 12	
	Sitio: https://www.youtube.com/playlist?list=PLqcsZoFzFKH34—WJ0e01ljqgOWyQpFu

En el contexto de la programación, los recursos multimedia juegan un rol importante al transformar conceptos abstractos en experiencias profesionales y accesibles, fomentando el aprendizaje técnico y a la vez inspirando de forma creativa a quienes deseen aprender a programar [Turbay Gusqui, 2025; Llerena-Izquierdo & Ayala-Carabajo, 2021c]. Estos recursos, que incluyen videos, imágenes, infografías, códigos, guías, podcasts y simulaciones interactivas, permiten comprender el mundo de la programación mediante experiencias ejemplares, alineándose con el objetivo de explorar esta disciplina de manera inmersiva y motivadora [Ayala-Carabajo & Llerena-Izquierdo, 2024; Puma-Rodriguez et al., 2025].

La importancia de los recursos multimedia radica en su capacidad para democratizar el conocimiento en programación, haciendo que ideas complejas sean más digeribles y atractivas [Muñoz Bermeo, 2024; Ayala-Carabajo & Llerena-Izquierdo, 2023a]. Por ejemplo, un video explicativo o de realidad aumentada puede ilustrar el funcionamiento de un algoritmo de manera visual [Quimis Taines, 2024], mientras que una infografía resume la evolución de los lenguajes de programación en un formato compacto, llamativo y atractivo [Llerena & Ayala-Carabajo, 2020; Palma Núñez, 2024]. Esto es especialmente valioso en un campo como la programación, donde la abstracción matemática y lógica puede desmotivar a principiantes en el ámbito educativo [Villao González & López Zambrano, 2024]. Según diversas fuentes históricas, pioneros como Ada Lovelace, considerada la primera programadora por su trabajo en la máquina analítica de Charles Babbage en el siglo XIX, han sido revividos a través de documentales y animaciones que destacan su contribución a los algoritmos modernos [Llerena-Izquierdo & Ayala-Carabajo, 2022; Zambrano Peñafiel, 2024]. Estos productos multimedia no solo educan, sino que inspiran, mostrando cómo la programación ha configurado el mundo digital actual, desde la inteligencia artificial hasta las redes sociales [Rendón De La Torre, 2024; Pino Gonzalez & Navarrete Bonilla, 2024].

Para identificar autores relevantes en el ámbito de la programación, es esencial recurrir a recursos multimedia que capturen sus contribuciones. Alan Turing, el padre de la computación teórica, es un ejemplo importante: sus ideas sobre máquinas universales se exploran en videos educativos y películas como “The Imitation Game”, que ilustran su rol en la Segunda Guerra Mundial y la IA. Grace Hopper, inventora del primer compilador y quien popularizó el término “bug”, aparece en entrevistas archivadas y TED-like talks que motivan a mujeres en STEM [Garcia-Silva et al., 2025]. Dennis Ritchie, creador del lenguaje C, es homenajeado en tutoriales visuales que muestran cómo su trabajo influyó en sistemas operativos como Unix. Guido van Rossum, el “benevolent dictator for life” de Python, comparte su filosofía en conferencias grabadas, enfatizando la simplicidad y legibilidad del código. Finalmente, Donald Knuth, conocido como el “Yoda de la programación” por su obra “The Art of Computer Programming”, es accesible a través de lecciones en video que desglosan algoritmos complejos. Estos autores representan puntos de referencia en la historia, y los multimedia permiten identificar sus contribuciones de forma dinámica [Ayala Carabajo, 2016; Llerena-Izquierdo & Ayala-Carabajo, 2021b; López-Chila et al., 2023; Zapata Martínez & Llerena-Izquierdo, 2023].

Utilizando recursos existentes en diferentes formatos, se enriquece la comprensión de la programación [Llerena-Izquierdo & Ayala-Carabajo, 2021a]. Videos motivadores, como charlas TED

sobre innovación en código o documentales sobre la historia de la informática, proporcionan narrativas emocionales que conectan teoría con práctica [Ayala-Carabajo, 2017].

Imágenes e infografías, por su parte, ofrecen resúmenes visuales de conceptos importantes, como diagramas de flujo algorítmico o timelines de lenguajes. Podcasts, como "Software Engineering Daily", permiten escuchar a expertos en movimiento, mientras que simuladores interactivos (e.g., en plataformas como Codecademy) ofrecen experiencias prácticas [Ayala-Carabajo & Llerena-Izquierdo, 2023b]. En entornos educativos, estos formatos combinados facilitan el logro de indicadores como la identificación de autores, al presentar sus biografías en videos ilustrados o galerías fotográficas [Quiñónez Martínez, 2023; Caicedo Chávez, 2023; Arteaga Sotomayor, 2023].

Para comprender el mundo de la programación mediante experiencias ejemplares, es fundamental reconocer la huella de los autores que han formado nuestra visión contemporánea [Atiaja Balseca, 2023]. Figuras como Guido van Rossum, creador de Python, defendieron desde el inicio que la legibilidad del código es una forma de arte visual que facilita la comprensión humana. Esta visión se complementa con el legado de pioneros como Seymour Papert, quien mediante el lenguaje Logo demostró que el uso de recursos visuales y táctiles podía transformar el aprendizaje en una aventura intuitiva. En la actualidad, esta tradición continúa a través de visionarios como Hadi y Ali Partovi, quienes han revolucionado el medio al democratizar el acceso al conocimiento mediante formatos de video de alta producción que narran la lógica computacional de una manera casi cinematográfica [Cardona Figueroa, 2022; Anton Yagual, 2022].

La utilización estratégica de los recursos existentes en el medio permite que el proceso de aprendizaje trascienda el formato de texto plano y se convierta en una experiencia sensorial completa [Valverde-Macias & Llerena-Izquierdo, 2022]. Cuando un estudiante interactúa con un video motivador, recibe esa mecha inicial que alimenta el trabajo arduo que la disciplina requiere [Llerena et al., 2021]. Al diversificar los formatos, el programador en formación logra identificar patrones y estructuras que en un manual estático resultarían opacas [Álava Morán, 2021]. Por ejemplo, los diagramas de flujo dinámicos y las simulaciones en tiempo real actúan como un lenguaje universal que conecta la teoría con la práctica inmediata [Rugel Lucín, 2021]. Esta combinación de medios auditivos y visuales no solo facilita la retención de información, sino que humaniza la tecnología al presentarla como un lienzo donde la creatividad y la técnica convergen para resolver problemas reales del entorno [Andrade Medina, 2021; Barros & Izquierdo, 2015].

4.1. Herramientas en el internet para el aprendizaje

La evolución del aprendizaje en línea ha sido impulsada por autores y plataformas que entendieron que el conocimiento debe ser dinámico y colaborativo [Isanoa-Sinche & Llerena-Izquierdo, 2023]. Figuras como Salman Khan, a través de Khan Academy, demostraron que los recursos multimedia pueden personalizar la educación, permitiendo que cada estudiante avance a su propio ritmo [Pilapaxi Cunalata & Llerena Izquierdo, 2025]. En el ámbito específico de la programación, sitios como GitHub han redefinido el concepto de herramienta de aprendizaje, convirtiéndose en bibliotecas vivas donde se puede estudiar el código de expertos de todo el mundo. Al explorar estos repositorios, el aprendiz no solo consume información, sino que participa en una experiencia ejemplar de desarrollo real, observando cómo se estructuran los grandes proyectos de software que sostienen la infraestructura digital contemporánea [Zerega-Prado & Llerena-Izquierdo, 2022; Acero Carrión, 2022; Guzmán García & López Idrovo, 2023; Burgos Cárdenas, 2023].

La diversidad de formatos disponibles en el medio digital permite que el aprendizaje sea multidimensional y adaptable a diferentes estilos cognitivos. Mientras que las plataformas de cursos masivos ofrecen una estructura secuencial y certificada, las comunidades de preguntas y respuestas

como Stack Overflow funcionan como un recurso de resolución inmediata de problemas, fomentando una cultura de ayuda mutua. Asimismo, el uso de editores de código en la nube y entornos de ejecución integrados en el navegador permite que cualquier persona, independientemente de la potencia de su hardware, pueda practicar y experimentar de forma instantánea [Palma Núñez, 2024]. Estas herramientas eliminan las barreras técnicas iniciales, asegurando que la chispa de la idea original no se apague ante la complejidad de la configuración de sistemas, sino que se transforme rápidamente en un trabajo funcional y creativo [Llerena-Izquierdo, 2023], ver Tabla 4.3.

Sitios web de aplicaciones relevantes por capítulos del libro

Tabla 4.3: Sitios de Internet para el aprendizaje por capítulos

Capítulo 1		
	Logisim	Sitio: https://sourceforge.net/projects/circuit/
	Logic Calculator	Sitio: https://sourceforge.net/projects/logiccalculator/
Capítulo 2		
	Code.org	Sitio: https://code.org/en-US/hour-of-code/starwars
	PSeInt	Sitio: https://pseint.sourceforge.net/
	Raptor	Sitio: https://raptor.martincarlisle.com/
Capítulo 3		
	Google Colab	Sitio: https://colab.research.google.com/
	Visual Studio Code	Sitio: https://code.visualstudio.com/

Internet ha revolucionado el acceso al conocimiento, convirtiéndose en un vasto ecosistema de herramientas gratuitas y de pago que facilitan el aprendizaje autónomo de la programación. Estas plataformas ofrecen cursos estructurados, ejercicios interactivos, videos tutoriales y entornos de codificación en línea, permitiendo a cualquier persona, desde principiantes hasta expertos, avanzar a su ritmo [Haugtvedt & Abata, 2021; Llerena-Izquierdo & Ayala-Carabajo, 2018].

La diversidad de recursos motiva el aprendizaje continuo, alineándose con la idea de que una buena motivación es el inicio de todo gran trabajo [Mendoza Solórzano, 2024; Valverde Rivera, 2023; Bouman et al., 2013]. En el ámbito de la programación, estas herramientas no solo enseñan sintaxis y algoritmos, sino que fomentan la práctica real y la resolución de problemas, esenciales para dominar lenguajes como Python [Nguyen et al., 2024; Llerena-Izquierdo & Ayala-Carabajo, 2018]. Entre las plataformas más destacadas para cursos interactivos se encuentran Codecademy, freeCodeCamp, Coursera, edX y Khan Academy. Codecademy ofrece lecciones gamificadas con retroalimentación inmediata, ideal para principiantes en Python o web development. freeCodeCamp proporciona certificaciones gratuitas a través de proyectos reales, enfocándose en desarrollo full-stack. Coursera y edX colaboran con universidades como Harvard (CS50) o MIT, ofreciendo MOOCs de alta calidad. Khan Academy, por su parte, explica conceptos básicos de manera visual y accesible [Llerena Izquierdo, 2020; Cuevas et al., 2025].

YouTube es otra herramienta indispensable, con canales como Traversy Media, CS50 (de Harvard), freeCodeCamp.org y The Net Ninja que ofrecen tutoriales gratuitos de horas de duración. Estos videos combinan explicación teórica con proyectos prácticos, motivando a los aprendices mediante ejemplos reales [Llerena-Izquierdo & Ayala-Carabajo, 2021a]. Para práctica interactiva, plataformas como Replit, LeetCode, HackerRank y Codewars permiten codificar en el navegador sin instalaciones. Replit es un IDE colaborativo en línea, perfecto para proyectos rápidos. LeetCode y HackerRank se centran en problemas algorítmicos para preparación de entrevistas técnicas. Otras herramientas complementarias incluyen documentación oficial (como docs.python.org), Stack Overflow para resolver dudas y GitHub para explorar código abierto. En conjunto, estos recursos forman un entorno rico y motivador [Llerena-Izquierdo & Atiaja-Balseca, 2021; Llerena-Izquierdo & Zamora-Galindo, 2021].

4.2. Recursos multimedia al aula

Llevar los recursos multimedia al aula implica seguir el ejemplo de autores que han entendido la pedagogía como un proceso de construcción activa. Seymour Papert sostenía que el entorno de aprendizaje debe ser un espacio donde el estudiante pueda "pensar sobre su propio pensamiento", y el uso de formatos visuales y auditivos facilita precisamente este proceso de metacognición [Essel et al., 2025]. En el aula moderna, la utilización de pantallas interactivas, simuladores de lógica y videos explicativos permite que el docente deje de ser la única fuente de conocimiento para convertirse en un guía que ayuda a encender la chispa de la curiosidad [Tang et al., 2025]. Al observar cómo expertos y autores relevantes resuelven problemas reales a través de tutoriales o demostraciones en vivo, los estudiantes acceden a experiencias ejemplares que sirven como modelo para su propio crecimiento profesional [Llerena-Izquierdo et al., 2024a].

La riqueza de formatos que ofrece el medio multimedia permite que el aula se adapte a la diversidad de ritmos y estilos de aprendizaje presentes en un grupo de estudiantes. Mientras que algunos encuentran su mecha en la visualización de infografías dinámicas que desglosan la arquitectura de un algoritmo, otros logran encender su llama a través de entornos de programación gamificados que proponen retos constantes [Li et al., 2025a; Pinna et al., 2019]. La integración de estos recursos facilita que el trabajo a realizar sea percibido como un desafío creativo y no como una tarea monótona [Peng et al., 2025; Magadan-Diaz & Rivas-Garcia I, 2022]. Además, el uso de herramientas multimedia colaborativas dentro del salón de clases fomenta que los alumnos utilicen los formatos digitales no solo para consumir contenido, sino para producirlo, transformando el aula en un centro de innovación donde se comparten soluciones y se debate la lógica detrás de cada línea de código Zhang et al. [2025].

Los recursos multimedia en el aula representan una transformación profunda en la educación contemporánea, integrando elementos como videos, imágenes, animaciones, audio y herramientas interactivas para enriquecer el proceso de enseñanza-aprendizaje [Llerena-Izquierdo & Cedeño-Gonzabay, 2020; Valladares et al., 2023]. Estos recursos combinan texto, sonido y visuales para captar la atención de los estudiantes, fomentando una motivación inicial que, como bien se sabe, es el punto de partida de todo buen trabajo. En un entorno donde los alumnos crecen rodeados de tecnología digital, el uso de multimedia no solo actualiza las metodologías tradicionales, sino que las hace más inclusivas y efectivas, adaptándose a diferentes estilos de aprendizaje [Llerena-Izquierdo, 2024; Gonzalez-Morga et al., 2024]. La importancia de estos recursos radica en su capacidad para aumentar la motivación y el compromiso de estudio. Videos educativos, simulaciones interactivas o presentaciones animadas convierten conceptos abstractos en experiencias concretas, estimulando sentidos múltiples y mejorando la retención de información [Smirani & Yamani, 2024]. Por ejemplo, un video puede servir como un agente motivador, esa "chispa" que enciende el interés por un tema, mientras que infografías o animaciones ayudan a visualizar procesos complejos, como ciclos biológicos o eventos históricos [Reche et al., 2022; Korenova et al., 2024]. Entre los recursos más comunes se encuentran las pizarras digitales interactivas (PDI), proyectores para videos educativos, plataformas como Kahoot o Genially para actividades gamificadas, y aplicaciones en tablets para colaboración en tiempo real [Zabala-Vargas et al., 2019]. Estas herramientas promueven la interacción, permiten feedback inmediato y facilitan la educación inclusiva, atendiendo a necesidades diversas como discapacidades visuales o auditivas mediante subtítulos o audio descriptivo [Carrion Candel et al., 2021].

Los beneficios son amplios: mayor motivación estudiantil, mejora en la comprensión y retención, desarrollo de competencias digitales y fomento de la creatividad. Estudios destacan que el uso de multimedia incrementa la participación activa y reduce el abandono, especialmente en contextos de aprendizaje híbrido o remoto [Magadan-Diaz & Rivas-Garcia I, 2022], ver Tabla 4.4.

Videos en YouTube

Tabla 4.4: Datos principales de los videos sugeridos en cada capítulo del libro

Capítulo	Título	Intervienen	Misión	Valores
Primero: Lógica Matemática				
. clic	George Boole's Legacy	George Boole (Matemático)	Reducir la lógica a un sistema algebraico con dos valores: Verdadero y Falso	Fundamentos lógicos, Precisión lógica
Sitio: https://youtu.be/S9YboDEDzP0?si=cmjWB9Z19eHIHNY8				
. clic	George Boole	George Boole, Claude Shannon, Ingenieros de Intel	Formalizar la lógica en un sistema algebraico para diseñar circuitos	Pensamiento Estructural y Creatividad
Sitio: https://youtu.be/js1Hlr3q–U?si=KwwplGmJ9kfYn7UZ				
. clic	Misión recate	Un astronauta y un equipo de científicos	Subsistir y encontrar una forma de transmitir (individual); luego, lograr un rescate imposible (colectivo)	Pensamiento algorítmico, resolución de problemas y autosuficiencia técnica
Sitio: https://youtu.be/XvB58bCVfng?si=Q0easKTioflehuAp				
Segundo: Algoritmos				
. clic	Talentos Ocultos	Grupo de científicas, mujeres matemáticas afro-americanas	Calcular el lanzamiento a la órbita del astronauta John Glenn y garantizar su regreso	Adaptabilidad, autodeterminación y perseverancia
Sitio: https://youtu.be/xFALsPCeJvE?si=KGfqMtyDiAnGZYIe				
. clic	CÓDIGO: Depurando la brecha de género	Mujeres en STEM (Documental)	Eliminar las barreras que limitan la participación femenina en la tecnología	Inclusión, diversidad, ética, mentoría y liderazgo.
Sitio: https://youtu.be/rQKRw6tWsb8?si=5p0Gksc4xWGoI7tv				
Tercero: Lógica de Programación				
. clic	Descifrando Enigma	Alan Turing	Descifrar el código Enigma nazi, construyendo una máquina electromecánica	Pensamiento Lógico, intuición
Sitio: https://youtu.be/zXKHGaRxQaE?si=zsjOnFvlhKvpIctu				
. clic	La red social	Mark Zuckerberg	Crear una plataforma social que revoluciona la comunicación	Impacto rápido, escalabilidad y prototipado
Sitio: https://youtu.be/K1JWiaIY-Xg?si=hAbnPI-kDPM2edps				
. clic	Piratas de Silicon Valley	Steve Jobs, Steve Wozniak, Bill Gates	Liderar la revolución de la computadora personal	Visión, aplicación práctica y competencia
Sitio: https://youtu.be/K1JWiaIY-Xg?si=hAbnPI-kDPM2edps				

El uso de videos documentales y películas en el aula representa una de las estrategias multimedia más poderosas para encender esa chispa motivacional inicial que todo buen trabajo requiere [Zhu et al., 2025; Kurucz et al., 2025]. Estos recursos no solo entretienen, sino que contextualizan conceptos abstractos, humanizan figuras históricas y generan debates profundos, transformando la clase en una experiencia inmersiva [Mejia-Manzano et al., 2024; Fan et al., 2025]. En el aprendizaje de la programación, por ejemplo, un documental o película puede ilustrar el impacto real de la computación en la sociedad, motivando a los estudiantes a ver el código no como mera sintaxis, sino como herramienta para cambiar el mundo [Llerena-Izquierdo, 2023; Díaz et al., 2025], ver Tabla 4.5.

Documentales

Tabla 4.5: Documentales para el aprendizaje por capítulos

Capítulo 1	
Lo and behold	https://youtu.be/VNjr729oqPc?si=bkPYYcfrwCWA9Z3R
Godfather of AI (1)	https://youtu.be/qrvK_KuIeJk?si=XDr7iopEbLfxunco
Godfather of AI (2)	https://youtu.be/5KmopXwjXik?si=No2ocT2oXKVUKfl4
Risks of AI	https://youtu.be/qe9QSCF-d88?si=8yXOFb10f3hD124y
AI2027	https://youtu.be/1UufaK3pQMg?si=-SHrDwsoCEfGzH66
Aprendemos Juntos	https://www.youtube.com/@AprendemosJuntos
60 Minutes	https://www.youtube.com/@60minutes
BBC World Service	https://www.youtube.com/@BBCWorldService
Capítulo 2	
Revolution OS	https://youtu.be/jfYBeWHc3GE?si=c9KYfdSpY4ZC1YQO
AlphaGo	https://youtu.be/8tq1C8spV_g?si=uZfyco2Gig-vT5GD
Data workers	https://youtu.be/hw7kVyxaPD0?si=-1z45GKtZ4i4oYSv
Generative AI	https://youtu.be/B2zCWJBnfuE?si=VHN8jS_JG83fVzm8
Bloomberg	https://www.youtube.com/@bloomberg_espanol
Bloomberg Originals	https://www.youtube.com/@business
DW Español	https://www.youtube.com/@dwespanol
DW Tecnología	https://www.youtube.com/@DWTecnologiaymundodigital
Capítulo 3	
Snowden	https://youtu.be/cMLik5V3VOI?si=hgfv7xHSZebEdIPU
Citizenfour	https://youtu.be/6SUTBAWvuvk?si=ZqOC0OEUmYjLzbXx
El Círculo	https://youtu.be/Z4wu6J6DLLM?si=CI96CdfIL6K58vk_
Experto en IA	https://youtu.be/CYvBatB7vq8?si=qWzcEYXvluJMrghP
Nada es privado	https://youtu.be/HVHKYXJq7qo?si=UoCaPM4OgexpmslJ
The Social Dilemma	https://youtu.be/uaaC57tcci0?si=b7bnVAIy09VvZcMA
Google DeepMind	https://www.youtube.com/@googledeepmind
The Diary Of A CEO	https://www.youtube.com/@TheDiaryOfACEO
JLA Speakers	https://www.youtube.com/@JLASpeakers

Los videos documentales ofrecen rigor factual y profundidad educativa. Series como “The Codebreaker” o documentales sobre la historia de internet permiten explorar la evolución de la programación de manera cronológica y visual, destacando desafíos éticos y técnicos [Beege et al., 2024]. En el aula, se usan para introducir temas como algoritmos o ciberseguridad, seguidos de discusiones que refuerzan la comprensión [Garcia-Silva et al., 2025]. Por otro lado, las películas (incluso de ficción) aportan narrativas emocionales que hacen los contenidos memorables [Diaz et al., 2024]. Ejemplos destacados en programación incluyen “The Imitation Game” (sobre Alan Turing y el descifrado Enigma), que motiva discusiones sobre lógica computacional y ética en IA; “The Social Network” (sobre la creación de Facebook), ideal para temas de emprendimiento digital; o “Hidden Figures” (sobre mujeres matemáticas en NASA), que inspira inclusión en STEM

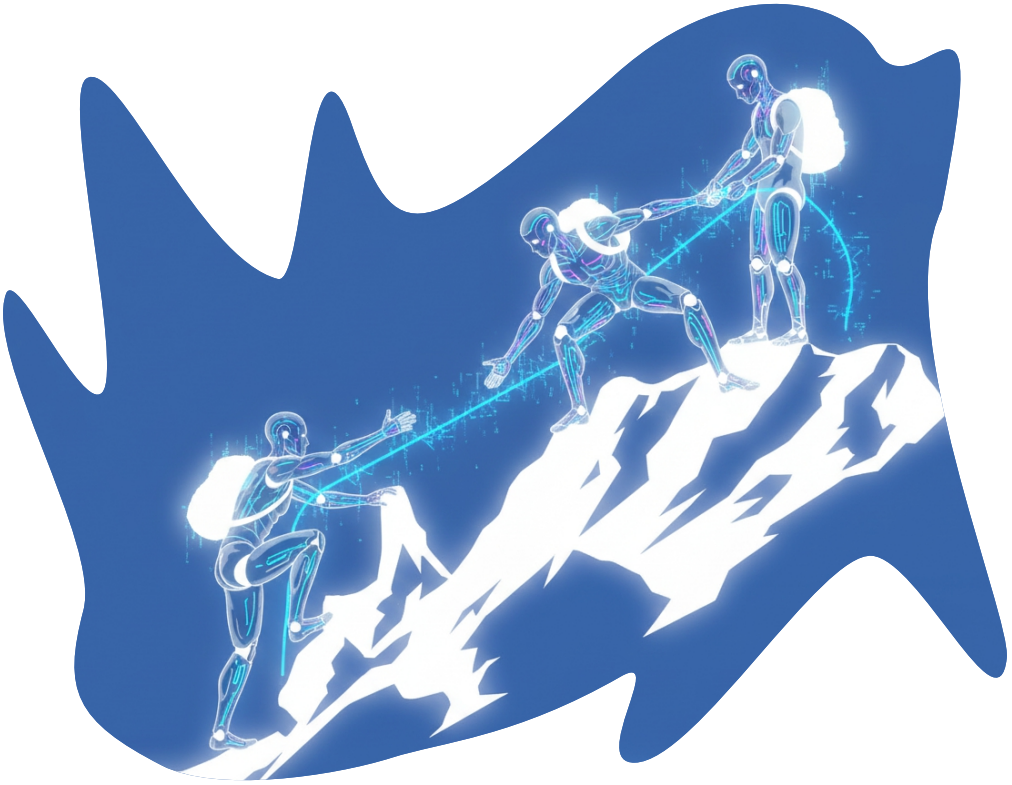
[Buniel et al., 2025].

El uso efectivo implica seleccionar contenidos adecuados a la edad y objetivos, complementar con actividades como guías de visión, debates o proyectos relacionados (e.g., codificar un cifrador simple tras ver Turing). Beneficios incluyen mayor retención, empatía histórica y motivación intrínseca [Yolanda Burgos-Lopez et al., 2024].

La utilización de documentales en el proceso educativo permite a los estudiantes identificar a los autores relevantes y los acontecimientos que definieron la computación moderna de una manera vívida [Cook et al., 2024]. Obras como The Code o Revolution OS presentan a figuras fundamentales como Linus Torvalds y Richard Stallman, permitiendo que el aprendiz comprenda la filosofía del código abierto no como un concepto teórico, sino como una lucha por la libertad del conocimiento [Chen et al., 2023]. Estos recursos multimedia ofrecen experiencias ejemplares al mostrar los fracasos, las iteraciones y la resiliencia necesaria para crear sistemas operativos que hoy sostienen la web global [Magadan-Diaz & Rivas-Garcia I, 2022]. Al observar estas narrativas, la mecha del trabajo a realizar se alimenta de una comprensión más profunda sobre la responsabilidad que conlleva escribir software [Tsakalerou et al., 2024].

Por otro lado, el cine de ficción basado en hechos reales, como las representaciones de la vida de Alan Turing en The Imitation Game o la creación de redes sociales en The Social Network, permite utilizar el formato cinematográfico para discutir dilemas éticos en el aula [Pinna et al., 2019]. Estas películas sirven como instrumentos para analizar las consecuencias del uso de datos, la privacidad y el poder de los algoritmos [Nikitina et al., 2023]. Al integrar estos recursos en el medio educativo, se fomenta una visión crítica que va más allá de la sintaxis [Reche et al., 2022]. El estudiante deja de ser un operario de teclado para convertirse en un analista que comprende cómo una sola idea puede encender llamas que afecten a millones de personas [Zabala-Vargas et al., 2019]. El cine, por tanto, actúa como un simulador de escenarios sociales que prepara al futuro programador para los retos morales de su carrera [Selco & Habbak, 2021].

El uso de videos documentales y películas en la enseñanza de la programación cierra el círculo de la formación integral [Smirani & Yamani, 2024]. Estos recursos logran que el propósito del estudiante se ancle en una realidad histórica y social, proporcionando una perspectiva que los manuales técnicos no pueden ofrecer [Valladares et al., 2023; Korenova et al., 2024]. Al diversificar los formatos y aprovechar la potencia narrativa del cine, se garantiza que la llama del conocimiento no solo sea técnica, sino también humanista [Gonzalez-Morga et al., 2024]. En última instancia, ver la historia de la programación a través de la gran pantalla permite que el aprendizaje sea una experiencia ejemplar, inspirando a la próxima generación de desarrolladores a escribir código que no solo sea eficiente, sino también ético y transformador [Andrews & Boklage, 2023].



¡El trabajo en equipo, aligera la carga!

¡Muy buen trabajo!

Finalización del capítulo 4

Bibliografía

Abbate, J., Narasimhan, S., Odeh, S., Bedwei, F. N., Musse, S. R. & Rieser, V. (2023). 'I wrote my first piece of code at seven': women share highs and lows in computer science for Ada Lovelace Day. *Nature*, 622(7982), 238–241. https://doi.org/10.1038/d41586-023-03146-8

Abo Khamis, M., Ngo, H. Q., Pichler, R., Suciu, D. & Wang, Y. R. (2024). Convergence of datalog over (pre-) semirings. *J. ACM*, 71(2). https://doi.org/10.1145/3643027

Acero Carrión, R. F. (2022). Modelo esquema para el análisis y tratamiento de la información en empresas ecuatorianas de comercio exterior mediante el uso de big data. B.S. thesis, Universidad Politécnica Salesiana. https://dspace.ups.edu.ec/handle/123456789/23637

Agbo, F. J., Oyelere, S. S., Suhonen, J. & Adewumi, S. (2019). A systematic review of computational thinking approach for programming education in higher education institutions. In *Proceedings of the 19th Koli calling international conference on computing education research*, (pp. 1–10). https://doi.org/10.1145/3364510.3364521

Aguayo-Torres, M.-C., Olea, L., Berlanga, R. & Godoy, R. (2024). Attracting Girls to Technology Through Public-Private Partnership: The Hedy Lamarr Chair at The University of Malaga. In *2024 ITU Kaleidoscope: Innovation and Digital Transformation for a Sustainable World (ITU K)*, (pp. 1–8). IEEE. https://doi.org/10.23919/ITUK62727.2024.10772815

Aiken, H. H. & Hopper, G. M. (1946). The automatic sequence controlled calculator — I. *Electrical Engineering*, 65(8-9), 384–391. https://doi.org/10.1109/EE.1946.6434251

Akitaya, H. A., Biniaz, A., Demaine, E. D., Kleist, L., Stock, F. & Tóth, C. D. (2025). Minimum plane bichromatic spanning trees. *ACM Trans. Algorithms*, 21(4). https://doi.org/10.1145/3747591

Álava Morán, N. S. (2021). Metodologías y técnicas analíticas de aprendizaje en la educación superior: un mapeo sistemático. https://dspace.ups.edu.ec/handle/123456789/20737

Altaher, M. & Ferchichi, A. (2018). Algothink: An algorithmic computational thinking approach. In *2018 JCCO Joint International Conference on ICT in Education and Training, International Conference on Computing in Arabic, and International Conference on Geocomputing (JCCO: TICET-ICCA-GECO)*, (pp. 1–7). https://doi.org/10.1109/ICCA-TICET.2018.8726205

Anami, B. S., Angadi, S. A. & Manvi, S. S. (2010). *Computer Concepts and C Programming:: Holistic Approach To Learning C*. PHI Learning Pvt. Ltd.

Andrade Medina, A. V. (2021). Gestión Informática Educativa: Un mapeo sistemático. https://dspace.ups.edu.ec/handle/123456789/20841

Andrews, M. E. & Boklage, A. (2023). Supporting inclusivity in STEM makerspaces through critical theory: A systematic review. *Journal of Engineering Education*. https://doi.org/10.1002/jee.20546

Angluin, D. (2016). Ada Byron Lovelace. *Complexities: Women in Mathematics*, (pp. 60–67). https://doi.org/10.2307/j.ctt1dr35cc.22

Anton Yagual, J. P. (2022). *Herramientas E-Learning para el Refuerzo Académico durante la Pandemia Del Covid-19 Utilizando Servicios Educativos en Línea*. B.S. thesis, Universidad Politécnica Salesiana. https://dspace.ups.edu.ec/handle/123456789/23696

Apt, K. R. & Wallace, M. (2006). *Constraint logic programming using ECLiPSe*. Cambridge University Press.

Arteaga Sotomayor, J. E. (2023). Efectividad del uso de la herramienta cuestionario en los vle´ s como herramienta de evaluación para profesores. B.S. thesis, Universidad Politécnica Salesiana. https://dspace.ups.edu.ec/handle/123456789/25049

Ashok, K. (2010). *C Programming: Test Your Skills*. Pearson Education India.

Assadi, S., Farach-Colton, M. & Kuszmaul, W. (2025). Tight bounds for monotone minimal perfect hashing. *ACM Trans. Algorithms*, 21(4). https://doi.org/10.1145/3677608

Atiaja Balseca, L. E. (2023). *Uso de la analítica del aprendizaje de los estudiantes para minimizar la pérdida escolar en las diferentes modalidades de estudio*. B.S. thesis, Universidad Politécnica Salesiana. https://dspace.ups.edu.ec/handle/123456789/25199

Austin, M. & Chancogne, D. (1999). *Introduction to engineering programming: in C, Matlab and Java*. John Wiley & Sons.

Avila, C., Bacca, J. & Rodriguez-Rojas, I. (2024). Women in STEAM Through the Lens of an Immersive-Based Storytelling Design. *Communications in Computer and Information Science*, 2209 CCIS, 245–258. https://doi.org/10.1007/978-3-031-75236-0_18

Ayala, R. (2007). Introducción a la concepción y aplicación del Método Fenomenológico-Hermenéutico de M. Van Manen. *Universitat de Barcelona.*

Ayala Carabajo, R. (2009). La esperanza pedagógica en la experiencia vivida de los educadores: un estudio fenomenológico-hermenéutico. *Tomos I y II. 4ESIS DOCTORAL INÁDITA. Departamento de Pedagogía Sistemática y Social, Universidad Autónoma de Barcelona-España.*

Ayala Carabajo, R. (2011). Pedagogical hope: A fresh and deep glance to educational experience from van Manen's approach. *Revista Espanola de Pedagogia*, 69(248), 119–143.

Ayala Carabajo, R. (2016). Formación de investigadores de las ciencias sociales y humanas en el enfoque fenomenológico hermenéutico (De van manen) en el contexto hispanoamericano. *Educacion XX1*, 19(2), 359–381. https://doi.org/10.5944/educXX1.13945

Ayala-Carabajo, R. (2017). *Retorno a lo esencial: Fenomenología hermenéutica aplicada desde el enfoque de Max Van Manen*. Caligrama.

Ayala-Carabajo, R. & Llerena-Izquierdo, J. (2023a). Modelo preventivo para minimizar estudiantes en riesgo académico para las asignaturas del primer año universitario. *CODES*, 5. https://doi.org/10.15443/codes1917

Ayala-Carabajo, R. & Llerena-Izquierdo, J. (2023b). Proyectos de investigación y publicaciones de estudiantes de grado. Caso de éxito en universidad ecuatoriana. *CODES*, 5. https://doi.org/10.15443/codes1906

Ayala-Carabajo, R. & Llerena-Izquierdo, J. (2024). Bibliometric Review on a Hybrid Learning Model with VLEs from a Higher Education Context. In J. P. Salgado-Guerrero, H. R. Vega-Carrillo, G. García-Fernández, & V. Robles-Bykbaev (Eds.), *Systems, Smart Technologies and Innovation for Society*, (pp. 148–157). Cham, Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-52090-7_15

Bachman, L. F. (2024). What does language testing have to offer? 1. In *The Writings of Lyle F. Bachman*, (pp. 367–399). Routledge.

Backus, J. (1978). The history of fortran i, ii, and iii. *SIGPLAN Not.*, 13(8), 165–180. https://doi.org/10.1145/960118.808380

Baker, F. T. (1972). System quality through structured programming. In *Proceedings of the December 5-7, 1972, Fall Joint Computer Conference, Part I*, AFIPS '72 (Fall, part I), (pp. 339–343), New York, NY, USA. Association for Computing Machinery. https://doi.org/10.1145/1479992.1480037

Balcan, M.-F., Dick, T., Sandholm, T. & Vitercik, E. (2024). Learning to branch: Generalization guarantees and limits of data-independent discretization. *J. ACM*, 71(2). https://doi.org/10.1145/3637840

Bantyukov, S., Merkulov, V., Biziuk, I. & Bantyukova, S. (2020). *Computer Science: Fundamentals of Algorithmization of Basic Computational Processes. Tutorial Student's Book*. Kharkiv, Ukraine: Ukrainian State University of Railway Transport.

Barros, E. J. C. & Izquierdo, J. F. L. (2015). El aprendizaje cooperativo en la preparación de una clase y el rendimiento de los estudiantes. In *II Congreso Salesiano de Ciencia, Tecnología e Innovación para la Sociedad (CITIS 2015)*

Basili, V. R. (1997). Evolving and packaging reading technologies. *Journal of Systems and Software*, 38(1), 3–12. https://doi.org/10.1016/S0164-1212(97)00065-4

Basili, V. R. & Selby, R. W. (1987). Comparing the Effectiveness of Software Testing Strategies. *IEEE Transactions on Software Engineering*, SE-13(12), 1278–1296. https://doi.org/10.1109/TSE.1987.232881

Basili, V. R. & Selby, R. W. (1991). Paradigms for experimentation and empirical studies in software engineering. *Reliability Engineering & System Safety*, 32(1), 171–191. https://doi.org/10.1016/0951-8320(91)90053-A

Bayle, J. (2013). *C programming for Arduino*. Packt Publishing Ltd.

Beege, M., Hug, C. & Nerb, J. (2024). AI in STEM education: The relationship between teacher perceptions and ChatGPT use. *Elsevier B.V.*, 16. https://doi.org/10.1016/j.chbr.2024.100494

Beelen, P. & Neiger, V. (2025). Faster list decoding of ag codes. *IEEE Transactions on Information Theory*, 71(5), 3397–3408. https://doi.org/10.1109/TIT.2025.3550750

Bender, M., Conway, A., Farach-Colton, M., Kuszmaul, W. & Tagliavini, G. (2025). Tiny pointers. *ACM Trans. Algorithms*, 21(4). https://doi.org/10.1145/3700594

Beyer, K. W. (2012). *Grace Hopper and the invention of the information age*. Mit Press.

Bhattacharya, B., Golin, M. J., Higashikawa, Y., Kameda, T. & Katoh, N. (2025). Improved algorithms for optimal k sink location on path networks. *Theoretical Computer Science*, 1039, 115190. https://doi.org/10.1016/j.tcs.2025.115190

Blackburn, H. (2024). Life After Lovelace: A Select Bibliography of Women in Computer Science. *Collection Management*, 49(3), 103–125. https://doi.org/10.1080/01462679.2024.2362135

Böhm, C. & Jacopini, G. (1966). Flow diagrams, turing machines and languages with only two formation rules. *Commun. ACM*, 9(5), 366–371. https://doi.org/10.1145/355592.365646

Bosworth, C. E. (1964). Some new manuscripts of al-khwarizmi's mafatih al-'ulum. *Journal of Semitic Studies*, 9(2), 341–345. https://doi.org/10.1093/jss/9.2.341

Bouman, K. L. (2020). Portrait of a black hole: Here's how the event horizon telescope team pieced together a now-famous image. *IEEE Spectrum*, 57(2), 22–29. https://doi.org/10.1109/MSPEC.2020.8976898

Bouman, K. L., Abdollahian, G., Boutin, M. & Delp, E. J. (2010). A low complexity method for detection of text area in natural images. In *2010 IEEE International Conference on Acoustics, Speech and Signal Processing*, (pp. 1050–1053). https://doi.org/10.1109/ICASSP.2010.5495331

Bouman, K. L., Johnson, M. D., Dalca, A. V., Chael, A. A., Roelofs, F., Doeleman, S. S. & Freeman, W. T. (2018). Reconstructing Video of Time-Varying Sources From Radio Interferometric Measurements. *IEEE Transactions on Computational Imaging*, 4(4), 512–527. https://doi.org/10.1109/TCI.2018.2838452

Bouman, K. L., Johnson, M. D., Zoran, D., Fish, V. L., Doeleman, S. S. & Freeman, W. T. (2016). Computational Imaging for VLBI Image Reconstruction. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, (pp. 913–922). https://doi.org/10.1109/CVPR.2016.105

Bouman, K. L., Xiao, B., Battaglia, P. & Freeman, W. T. (2013). Estimating the Material Properties of Fabric from Video. In *2013 IEEE International Conference on Computer Vision*, (pp. 1984–1991). https://doi.org/10.1109/ICCV.2013.455

Bouman, K. L., Ye, V., Yedidia, A. B., Durand, F., Wornell, G. W., Torralba, A. & Freeman, W. T. (2017). Turning Corners into Cameras: Principles and Methods. In *2017 IEEE International Conference on Computer Vision (ICCV)*, (pp. 2289–2297). https://doi.org/10.1109/ICCV.2017.249

Bravo Quezada, J. H. (2025). *Microlearning como estrategia de apoyo al modelo de aprendizaje de las instituciones educativas de enseñanza media*. PhD thesis, Universidad Politécnica Salesiana. https://dspace.ups.edu.ec/handle/123456789/30171

Buitrago Flórez, F., Casallas, R., Hernández, M., Reyes, A., Restrepo, S. & Danies, G. (2017). Changing a Generation's Way of Thinking: Teaching Computational Thinking Through Programming. *Review of Educational Research*, 87(4), 834–860. https://doi.org/10.3102/0034654317710096

Buniel, J. M., Intano, J., Cuartero, O., Grustan, K. J., Sumaoy, R., Reyes, N., Calipayan Jr, J., Arreo, R., Duero, D., Rosil, I., Agustin, S., Diron, T. J., Pingol, R. J., Sapuras, J. V., Miranda, K.,

Julve, J., Josol, M., Mercado, K. R., Latoja, L., Cubillan, J., Fallado, P. A. J., Duran, E. L., Ambray, F. I., Miranda, M., Etchon, F. M., Ramoso, M. M., Rubenial, J. R., Ganancias, F., Orozco, L., Gracia, J., Notado, N., Darao, G. & Cortes, S. (2025). Modeling the influence of AI dependence to research productivity among STEM undergraduate students: case of a state university in the Philippines. *Frontiers in Education*, 10. https://doi.org/10.3389/feduc.2025.1535466

Burgos Cárdenas, M. J. (2023). Modelo computacional de datos para gestión de captaciones bancarias del sector ecuatoriano basado en big data. B.S. thesis, Universidad Politécnica Salesiana. https://dspace.ups.edu.ec/handle/123456789/24169

Burkland, M. (2021). Technology Transfers Into The Defense Industry: Implications For Spintronics In The Future Of National Security. In *2021 IEEE 32nd Magnetic Recording Conference (TMRC)*, (pp. 1–2). IEEE. https://doi.org/10.1109/TMRC53175.2021.9605120

Caicedo Chávez, Á. C. (2023). Revisión de la literatura sobre la aproximación del metaverso en la educación superior. B.S. thesis, Universidad Politécnica Salesiana. https://dspace.ups.edu.ec/handle/123456789/24393

Callaghan, S. A., Darbyshire, T. & Farajnia, S. (2020). Ada Lovelace Day and Celebrating Women in STEM. *Patterns*, 1(7). https://doi.org/10.1016/j.patter.2020.100125

Candler, R. (2020). Beyond Marie Curie: Grace Hopper and the ENIAC Six. *IEEE Potentials*, 39(3), 10–12. https://doi.org/10.1109/MPOT.2019.2892012

Cao, N. & Stepnicka, M. (2024). Upper Boundary Algebra for Modeling the Missing Values Is a Residuated Lattice. In *2024 IEEE International Conference on Fuzzy Systems, Fuzz-IEEE 2024*, IEEE International Fuzzy Systems Conference Proceedings. IEEE. https://doi.org/10.1109/FUZZ-IEEE60900.2024.10612160

Carabajo, R. A. (2023). Pedagogical hope: A fresh and deep glance to educational experience from van manens approach. *Revista Española de Pedagogía*, 69(248), 10.

Cardona Figueroa, E. G. (2022). Impacto de las plataformas virtuales para la educación en tiempos de pandemia. B.S. thesis, Universidad Politécnica Salesiana. https://dspace.ups.edu.ec/handle/123456789/23694

Carrion Candel, E., Perez Agustin, M. & Gimenez De Ory, E. (2021). ICT and gamification experiences with CLIL methodology as innovative resources for the development of competencies in compulsory secondary education. *Digital Education Review*, 1(39), 238–256.

Carver, K., Pacifici, C. & Averbukh, J. (2025). Image Exploration with Jdaviz, the James Webb Space Telescope Data Analysis and Visualization Tool. In *American Astronomical Society Meeting Abstracts #245*, volume 245 of *American Astronomical Society Meeting Abstracts*, (pp. 361.02). https://ui.adsabs.harvard.edu/abs/2025AAS...24536102C

Casciaro, M. & Mammino, L. (2016). *Node. js Design Patterns*. Packt Publishing Ltd.

Case, B. A. B. & Leggett, A. M. (2016). Complexities: Women in mathematics. *Complexities*, (pp. 1–457). https://doi.org/10.2307/j.ctt1dr35cc

Cavendish, R. (2015). Ada Lovelace born in London: The mathematician and pioneering computer programmer was born on December 10th, 1815. *History Today*, 65(12). https://research.ebsco.com/linkprocessor/plink?id=0b0ee74a-22ea-3f6a-bc63-1576dda00474

Chajda, I. & Laenger, H. (2025). Algebraic Structures Formalizing the Logic of Quantum Mechanics Incorporating Time Dimension. *Studia Logica*, 113(1), 163–181. https://doi.org/10.1007/s11225-024-10103-7

Chanana, N. (2024). Inspirational Women Leaders: Diverse Success Stories and Inspirations. In M. Haoucha (Ed.), *Empowering and Advancing Women Leaders and Entrepreneurs*, (pp. 37–59). Hershey, PA, USA, IGI Global Scientific Publishing. https://doi.org/10.4018/979-8-3693-7107-7.ch003

Chapuy, G. & Perarnau, G. (2025). Short synchronizing words for random automata. *ACM Trans. Algorithms*, 21(4). https://doi.org/10.1145/3736722

Charman-Anderson, S. (2015). Ada lovelace: Victorian computing visionary. *Ada User Journal*, 36(1), 35–41. https://www.ada-europe.org/auj/archive/

Charman-Anderson, S. (2020). Ada Lovelace: A Simple Solution to a Lengthy Controversy. *Patterns*, 1(7). https://doi.org/10.1016/j.patter.2020.100118

Chatterjee, A., Kumar, M. & Volk, B. L. (2024). Determinants vs. Algebraic Branching Programs. *computational complexity*, 33(2), 11. https://doi.org/10.1007/s00037-024-00258-z

Chen, C., Hardjo, S., Sonnert, G., Hui, J. & Sadler, P. M. (2023). The role of media in influencing students' STEM career interest. *International Journal of STEM Education*, 10(1), 56. https://doi.org/10.1186/s40594-023-00448-1

Chen, D.-S., Batson, R. G. & Dang, Y. (2011). *Applied integer programming: modeling and solution*. John Wiley & Sons.

Chen, L., Kyng, R., Liu, Y., Peng, R., Probst Gutenberg, M. & Sachdeva, S. (2025). Maximum flow and minimum-cost flow in almost-linear time. *J. ACM*, 72(3). https://doi.org/10.1145/3728631

Chernyshev, V., Rauch, J., Rautenbach, D. & Redina, L. (2025). A faster algorithm for independent cut. *Theoretical Computer Science*, 1054, 115542. https://doi.org/10.1016/j.tcs.2025.115542

Childs, A., Kothari, R., Kovacs-Deak, M., Sundaram, A. & Wang, D. (2025). Quantum divide and conquer. *ACM Transactions on Quantum Computing*, 6(2). https://doi.org/10.1145/3723884

Clark, K. M. (2012). History of mathematics: Illuminating understanding of school mathematics concepts for prospective mathematics teachers. *Educational Studies in Mathematics*, 81(1), 67–84. https://doi.org/10.1007/s10649-011-9361-y

Cohen, L., Forster, Y., Kirst, D., Paiva, B. d. R. & Rahli, V. (2024). Separating Markov's Principles. In *Proceedings of the 39th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2024*. https://doi.org/10.1145/3661814.3662104

Cook, K. L., Ivy, J., Maiorca, C., Tripp, O., Burton, M., Jackson, C., Bush, S. B., Mohr-Schroeder, M., Edelen, D., Schroeder, C. & Roberts, T. (2024). STEM Rocks Research Collective: Building and Sustaining a Collaborative, Equity-Focused Scholarly Community. *Education Sciences*, 14(9). https://doi.org/10.3390/educsci14090947

Cormen, T. H., Leiserson, C. E., Rivest, R. L. & Stein, C. (2022). *Introduction to algorithms*. MIT press

Crilly, T. (2025). The essential George Boole by Desmond MacHale & Yvonne Cohen, pp. 238, £15.99 (paper), ISBN 978-1-78117-840-9, Mercier Press (2024). *The Mathematical Gazette*, 109(575), 364–365. https://doi.org/10.1017/mag.2025.10103

Cuevas, E. H., Zaldivar, D. & Perez, M. (2025). Impact of Programming Languages on Learning Performance. *International Journal of Information and Communication Technology Education (IJICTE)*, 21(1), 1–17. https://doi.org/10.4018/IJICTE.371419

Czarnul, P., Antal, M., Baniata, H., Griebler, D., Kertesz, A., Kessler, C. W., Kouloumpris, A., Kovačić, S., Markus, A., Michael, M. K., Nikolaou, P., Öz, I., Prodan, R. & Rakić, G. (2025). Optimization of resource-aware parallel and distributed computing: a review. *The Journal of Supercomputing*, 81(7), 848. https://doi.org/10.1007/s11227-025-07295-7

Dale, N. B. & Weems, C. (1996). *Introduction to Pascal and structured design*. Jones & Bartlett Learning.

Dale, N. B., Weems, C. & McCormick, J. W. (2000). *Programming and Problem Solving with ADA 95*. Jones & Bartlett Learning.

Dalmau, D. S.-C. (2004). *Core techniques and algorithms in game programming*. New Riders.

Develaki, M. (2019). Methodology and Epistemology of Computer Simulations and Implications for Science Education. *Journal of Science Education and Technology*, 28(4), 353–370. https://doi.org/10.1007/s10956-019-09772-0

Díaz, B., Lynch, C., Delgado, C. & Han, K. (2025). Analysis of two pedagogical approaches to foster discipline integrations in an educational data mining class using communities of practice. *International Journal of STEM Education*, 12(1). https://doi.org/10.1186/s40594-025-00538-2

Diaz, F., Rosua, J. C., Ferrer, G. F., Carmona, R. M. & Narvaez, R. (2024). Assessment of immersive technologies and STEM focus in initial teacher training. *RIED-Revista Iberoamericana de Educación a distancia*, 27(1). https://doi.org/10.5944/ried.27.1.37688

D'Ignazio, C. & Klein, L. (2020). Introduction: Why data science needs feminism. *Data Feminism*. https://data-feminism.mitpress.mit.edu/pub/frfa9szd/release/2

Dijkstra, E. W. (1968a). Letters to the editor: go to statement considered harmful. *Communications of the ACM*, 11(3), 147–148. https://dl.acm.org/doi/pdf/10.1145/362929.362947

Dijkstra, E. W. (1968b). Letters to the editor: go to statement considered harmful. *Commun. ACM*, 11(3), 147–148. https://dl.acm.org/doi/pdf/10.1145/362929.362947

Dijkstra, E. W. (1974). Programming as a discipline of mathematical nature. *The American Mathematical Monthly*, 81(6), 608–612. https://doi.org/10.1080/00029890.1974.11993624

Doorenbos, R. B. (1995). *Production matching for large learning systems*. Carnegie Mellon University.

dos Santos, C. H. M. & de Lima, S. M. L. (2024). XAI-driven antivirus in pattern identification of citadel malware. *Journal of Computational Science*, 82, 102389. https://doi.org/10.1016/j.jocs.2024.102389

Dourish, P. (2016). Algorithms and their others: Algorithmic culture in context. *Big Data & Society*, 3(2), 2053951716665128. https://doi.org/10.1177/2053951716665128

Dunlop, D. M. (1943). Muḥammad b. Mūsā al- Kh wārizmī. *Journal of the Royal Asiatic Society*, 75(3-4), 248–250. https://doi.org/10.1017/S0035869X00098464

Ensmenger, N. (2016). The multiple meanings of a flowchart. *Information & Culture*, (pp. 321–351). https://www.jstor.org/stable/44667617

Eschenburg, J. H. (2020). Algebra from Al Khwarizmi to Galois, 830-1830. *Bulletin of the Iranian Mathematical Society*, 46(3), 613–624. https://doi.org/10.1007/s41980-019-00279-3

Espinal, A., Vieira, C. & Magana, A. J. (2024). Professional development in computational thinking: A systematic literature review. *ACM Transactions on Computing Education*, 24(2), 1–24.

Essel, H. B., Vlachopoulos, D., Nunoo-Mensah, H. & Amankwa, J. O. (2025). Exploring the impact of VoiceBots on multimedia programming education among Ghanaian university students. *British Journal of Educational Technology*, 56(1), 276–295. https://doi.org/10.1111/bjet.13504

Faerna, A. M. (2024). The concept of "warranted assertibilityïn Dewey's naturalistic logic: three misunderstandings. *Teorema*, 43(1), 29–46.

Fahmy-Hooke, J. (2024). Heidegger's Critical Confrontation with the Concept of Truth as Validity. *Open Philosophy*, 7(1). https://doi.org/10.1515/opphil-2024-0054

Fan, G., Liu, D., Zhang, R. & Pan, L. (2025). The impact of AI-assisted pair programming on student motivation, programming anxiety, collaborative learning, and programming performance: a comparative study with traditional pair programming and individual approaches. *International Journal of STEM Education*, 12(1), 16. https://doi.org/10.1186/s40594-025-00537-3

Fauziah, A. H. (2025). Integrating Al-Khawarizmi's Principles in K-Means Algorithm: A Case Study of Household Electricity Access Data in Indonesia. In *Proceedings of International Conference of Islamic Studies (ICONIS 2025)*, (pp. 255–270). Atlantis Press. https://doi.org/10.2991/978-2-38476-464-8_18

Ferlier, L. (2025). Atmospheric footnotes: Ada Lovelace on climate. *Journal of the Royal Society Interface*, 79(1), 59–86. https://doi.org/10.1098/rsnr.2024.0021

Ferry, G. (2015). Ada Lovelace: in search of "a calculus of the nervous system". *The Lancet*, 386(10005), 1731. https://doi.org/10.1016/S0140-6736(15)00686-8

First, E. C., Mishra, I., Gazel, E., Lewis, N. K., Letai, J. & Hanssen, L. (2025). Potential for observing geological diversity from mid-infrared spectra of rocky exoplanets. *Nature Astronomy*, 9(3), 370–379. https://doi.org/10.1038/s41550-024-02412-7

Füegi, J. & Francis, J. (2015). Lovelace & Babbage and the creation of the 1843 'Notes'. *Ada User Journal*, 36(2), 89–98.

Futschek, G. (2006). Algorithmic Thinking: The Key for Understanding Computer Science. In R. T. Mittermeir (Ed.), *The Bridge between Using and Understanding Computers*, (pp. 159–168). Berlin, Heidelberg, Springer Berlin Heidelberg. https://doi.org/10.1007/11915355_15

Gabow, H. N. (2024). Maximum cardinality f-matching in time o(n2/3m). *ACM Trans. Algorithms*, 21(1). https://doi.org/10.1145/3696668

García-Díaz, J., Cornejo-Acosta, J. A. & Trejo-Sánchez, J. A. (2025). A greedy heuristic for graph burning. *Computing*, 107(3), 91. https://doi.org/10.1007/s00607-025-01436-9

Garcia-Silva, E., Perez-Suarez, S., Zavala-Parrales, A., Melendez-Anzures, F. E. & Dominguez, A. (2025). Continuing education of academic women in STEM: perspectives on mentoring and professional roles. *Frontiers in Education*, 10. https://doi.org/10.3389/feduc.2025.1473331

Gerez, S. H. (1999). *Algorithms for VLSI design automation*. John Wiley & Sons.

Gilbreth, F. B. & Gilbreth, L. M. (1921). Process charts: First steps in finding the one best way to do work. *Transactions of the American Society of Mechanical Engineers*, 43, 1029–1043.

Goldstine, H. H. (1977). A brief history of the computer. *Proceedings of the American Philosophical Society*, 121(5), 339–345.

Goldstine, H. H. & Von Neumann, J. (1951). Numerical inverting of matrices of high order. ii. *Proceedings of the American Mathematical Society*, 2(2), 188–202.

Goldstine, H. H., Von Neumann, J. & Von Neumann, J. (1947). Planning and coding of problems for an electronic computing instrument. *Institute for Advanced Study Princeton.*

Gonçalves, B. (2024). Lady Lovelace's Objection: The Turing-Hartree Disputes over the Meaning of Digital Computers, 1946-1951. *IEEE Annals of the History of Computing*, 46(1), 6–18. https://doi.org/10.1109/MAHC.2023.3326607

Gonzalez-Morga, N., Gonzalez-Lorente, C., Martinez-Clares, P. & Perez-Cuso, J. (2024). Flipped Tutoring in Higher Education: Student Satisfaction with an Experience of Educational Innovation. *Revista Electrónica EDUCARE*, 28(1), 1–20. https://doi.org/10.15359/ree.28-1.17280

Griffiths, H. & Farina, A. (2021). History Column: Hedy Lamarr. *IEEE Aerospace and Electronic Systems Magazine*, 36(10), 43–43. https://doi.org/10.1109/MAES.2021.3101776

Grzybowski, A. E., Pawlikowska-Łagód, K. & Lambert, W. C. (2024). A History of Artificial Intelligence. *Clinics in Dermatology*, 42(3), 221–229. https://doi.org/10.1016/j.clindermatol.2023.12.016

Guzmán García, A. A. & López Idrovo, J. M. (2023). Modelo computacional para el seguimiento de la producción en una industria ecuatoriana basado en big data y business intelligence. B.S. thesis, Universidad Politécnica Salesiana. https://dspace.ups.edu.ec/handle/123456789/25913

Hamilton, M. (1994). Preventative software systems. In *Proceedings Eighteenth Annual International Computer Software and Applications Conference (COMPSAC 94)*, (pp. 410–416). IEEE Comput. Soc. Press. https://doi.org/10.1109/CMPSAC.1994.342770

Hamilton, M. & Hackler, R. (1991). Prototyping distributed environments with 001. In *[1991 Proceedings] The Second International Workshop on Rapid System Prototyping*, (pp. 110–111). IEEE Comput. Soc. Press. https://doi.org/10.1109/IWRSP.1991.218618

Hamilton, M. & Hackler, W. (1990). 001: a rapid development approach for rapid prototyping based on a system that supports its own life cycle. In *[1990 Proceedings] The First International Workshop on Rapid System Prototying*, (pp. 46–62). IEEE Comput. Soc. https://doi.org/10.1109/IWRSP.1990.144033

Hamilton, M. H. (1986). Zero-defect software: The elusive goal: It is theoretically possible but difficult to achieve; logic and interface errors are most common, but errors in user intent may also occur. *IEEE Spectrum*, 23(3), 47–53. https://doi.org/10.1109/MSPEC.1986.6371028

Hamilton, M. H. (2018). What the Errors Tell Us. *IEEE Software*, 35(5), 32–37. https://doi.org/10.1109/MS.2018.290110447

Hamilton, M. H. & Hackler, W. R. (2008). Universal Systems Language: Lessons Learned from Apollo. *Computer*, 41(12), 34–43. https://doi.org/10.1109/MC.2008.541

Hartmann, W., Nievergelt, J. & Reichert, R. (2001). Kara, finite state machines, and the case for programming as part of general education. In *Proceedings IEEE Symposia on Human-Centric Computing Languages and Environments (Cat. No.01TH8587)*, (pp. 135–141). https://doi.org/10.1109/HCC.2001.995251

Hatami, P. & Hoza, W. (2024). Paradigms for Unconditional Pseudorandom Generators. *Foundations and Trends® in Theoretical Computer Science*, 16(1-2), 1–210. https://doi.org/10.1561/0400000109

Haugtvedt, E. & Abata, D. L. (2021). Ada Lovelace: First Computer Programmer and Hacker? In *ASEE Annual Conference and Exposition, Conference Proceedings*. American Society for Engineering Education.

Hénin, S. (2015). Augusta Ada Lovelace (1815-1852); Augusta ada lovelace (1815 - 1852). *Mondo Digitale*, 14(57), 276–352.

Hien, A., Aribi, N., Loudni, S., Lebbah, Y., Ouali, A. & Zimmermann, A. (2024). Mining diverse sets of patterns with constraint programming using the pairwise Jaccard similarity relaxation. *Constraints*, 29(1), 80–111. https://doi.org/10.1007/s10601-024-09373-8

Hirobe, T. & Kasai, K. (2025). Recursively extended permutation codes under chebyshev distance. *IEEE Transactions on Information Theory*, 71(11), 8312–8323. https://doi.org/10.1109/TIT.2025.3611022

Hoare, T. (2022). Forty years with edsger. In *Edsger Wybe Dijkstra: His Life, Work, and Legacy*, (pp. 411–422). ACM Books.

Hollings, C. D., Martin, U. & Rice, A. C. (2017a). The early mathematical education of Ada Lovelace. *BSHM Bulletin*, 32(3), 221–234. https://doi.org/10.1080/17498430.2017.1325297

Hollings, C. D., Martin, U. & Rice, A. C. (2017b). The Lovelace–De Morgan mathematical correspondence: A critical re-appraisal. *Historia Mathematica*, 44(3), 202–231. https://doi.org/10.1016/j.hm.2017.04.001

Hopper, G. M. (1952). The education of a computer. In *Proceedings of the 1952 ACM national meeting (Pittsburgh) on - ACM '52*, ACM '52, (pp. 243–249), New York, New York, USA. ACM Press. https://doi.org/10.1145/609784.609818

Hsu, T.-C., Chang, S.-C. & Hung, Y.-T. (2018). How to learn and how to teach computational thinking: Suggestions based on a review of the literature. *Computers & Education*, 126, 296–310. https://doi.org/10.1016/j.compedu.2018.07.004

Hu, Z., Liu, J., Liu, Z., Liu, Y., Xie, Z. & Song, Y. (2025). RMath: A Logic Reasoning-Focused Datasets Toward Mathematical Multistep Reasoning Tasks. In T. Walsh, J. Shah, & Z. Kolter (Eds.), *Thirty-Ninth AAAI Conference On Artificial Intelligence, AAAI-25, Vol 39 No 22*, (pp. 24104–24112). https://doi.org/10.1609/aaai.v39i22.34585

Isanoa-Sinche, M. & Llerena-Izquierdo, J. (2023). Towards a Meaningful Experience on the Use of Digital Educational Resources and Media created at Ardora. In *2023 IEEE Seventh Ecuador Technical Chapters Meeting (ECTM)*, (pp. 1–5). https://doi.org/10.1109/ETCM58927.2023.10308988

Jäger, E., Pearson, D., Petermann, T. & Dietl, G. (2025). Advancing operability and observability in real-time critical nanosatellite missions through a low-resource embedded flight software stack. *CEAS Space Journal*. https://doi.org/10.1007/s12567-025-00686-9

Jiménez Toledo, J. A., Collazos, C. A., Ortega, M. & Ramos, D. X. (2023). Algorithmic thinking and extension of its definition for trainee software developers: A systematic literature mapping. *IEEE Revista Iberoamericana de Tecnologias del Aprendizaje*, 18(4), 331–343. https://doi.org/10.1109/CVPR.2016.105

Jin, Y., Wang, H., Tang, X., Guo, Z., Zhao, Y., Hoefler, T., Liu, T., Liu, X. & Zhai, J. (2025). Leveraging graph analysis to pinpoint root causes of scalability issues for parallel applications. *IEEE Transactions on Parallel and Distributed Systems*, 36(2), 308–325. https://doi.org/10.1109/TPDS.2024.3485789

Karandikar, N. & Warnekar, C. (2025). The Concept of Iterative Functional Algorithms in classical Sanskrit texts on Mathematics. In *2025 1st International Conference on AIML-Applications for Engineering & Technology (ICAET)*, (pp. 1–5). https://doi.org/10.1109/ICAET63349.2025.10932194

Karau, H., Konwinski, A., Wendell, P. & Zaharia, M. (2015). *Learning spark: lightning-fast big data analysis*. .ºReilly Media, Inc."

Kessler, C. W. (2006). Teaching parallel programming early. In *Proc. Workshop on Developing Computer Science Education How Can It Be Done.*

Knight, J. C. & Myers, E. A. (1993). An improved inspection technique. *B T - Communications of the A C M*, 36(11), 50+.

Knuth, D. E. (1997). *The Art of Computer Programming: Fundamental Algorithms, Volume 1*. Addison-Wesley Professional.

Knuth, D. E. (2022). *The Art of Computer Programming, Volume 4B: Combinatorial Algorithms*. Addison-Wesley Professional.

Kolczyk, E. (2008). Algorithm – fundamental concept in preparing informatics teachers. In R. T. Mittermeir & M. M. Sysło (Eds.), *Informatics Education - Supporting Computational Thinking*, (pp. 265–271), Berlin, Heidelberg. Springer Berlin Heidelberg.

Kolmogorov, V. (2025). A simpler and parallelizable $O(\sqrt{\log n})$ -approximation algorithm for sparsest cut. *ACM Trans. Algorithms*, 21(4). https://doi.org/10.1145/3748723

Korenova, L., Krpec, R. & Barot, T. (2024). Digital Technologies in Primary Mathematics Education: Insights from Future Teachers' Portfolios. In F. Moriera (Ed.), *PROCEEDINGS OF THE 23RD EUROPEAN CONFERENCE ON E-LEARNING, ECEL 2024*, volume 23/1 of *Proceedings on the European Conference of e-Learning*, (pp. 197–208), CURTIS FARM, KIDMORE END, NR READING, RG4 9AY, ENGLAND. ACAD CONFERENCES LTD

Kundu, R., Chattopadhyay, S., Nag, S., Navarro, M. A. & Oliva, D. (2024). Prism refraction search: a novel physics-based metaheuristic algorithm. *The Journal of Supercomputing*, 80(8), 10746–10795. https://doi.org/10.1007/s11227-023-05790-3

Kurgalin, S. & Borzunov, S. (2019). *A practical approach to high-performance computing*, volume 206. Springer

Kurucz, E., Spencer-Mueller, E. K., Hands, C., Gudz, N. & Archer, K. (2025). Co-constructing educational innovations for an uncertain future: Design thinking and developmental evaluation in the school reform process. *Evaluation and Program PLANNING*, 111. https://doi.org/10.1016/j.evalprogplan.2025.102559

Kydyraliev, S., Urdaletova, A., Japarova, S. & Burova, E. (2025). *From Diophantus and AL-Khwarizmi to the Modern Time*, (pp. 335–339). Springer Nature Switzerland: Cham. https://doi.org/10.1007/978-3-031-84628-1_27

Kyriazis, G. A. (2025). A Deductive System for Boole's 'The Mathematical Analysis of Logic' and its Application to Hypothetical Deductions. *History and Philosophy of Logic*, (pp. 1–25). https://doi.org/10.1080/01445340.2025.2460968

Lagouvardos, S., Dolby, J., Grech, N., Antoniadis, A. & Smaragdakis, Y. (2020). Static analysis of shape in tensorflow programs. In *34th European Conference on Object-Oriented Programming (ECOOP 2020)*, (pp. 15–1). Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

Landman, M. & Kohn, T. (2024). "something that happens each day students' explanations of what algorithms are. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*, ITiCSE 2024, (pp. 199–205), New York, NY, USA. Association for Computing Machinery. https://doi.org/10.1145/3649217.3653531

Lee, K. D. (2008). *Programming languages: An active learning approach*. springer science & Business Media.

Leverrier, A. & Zémor, G. (2025). Efficient decoding up to a constant fraction of the code length for asymptotically good quantum codes. *ACM Trans. Algorithms*, 21(4). https://doi.org/10.1145/3663763

Levis, A. (2025). Seeing the world through the lens of computational imaging. https://doi.org/10.1145/3703395

Li, C., Chan, E., Denny, P., Luxton-Reilly, A. & Tempero, E. (2019). Towards a framework for teaching debugging. In *Proceedings of the Twenty-First Australasian Computing Education Conference*, (pp. 79–86)

Li, Q., Liu, D., Zu, G., Li, S., Sun, H. & Wang, J. Z. (2025a). Multigranularity Feature Aggregation and Cross-level Boundary Modeling for Temporal Action Detection. *ACM Transactions on Multimedia Computing Communications and Applications*, 21(3). https://doi.org/10.1145/3712598

Li, S., Liu, S., Ma, L., Wan, Y. & Xing, C. (2025b). Encoding of algebraic geometry codes with quasi-linear complexity o(nlogn). *IEEE Transactions on Information Theory*, 71(7), 5013–5026. https://doi.org/10.1109/TIT.2025.3562424

Li, Z., Fu, Y., Tian, M., Li, C., Yu, F. R. & Cheng, N. (2025c). Fedstdn: A federated learning-enabled spatial-temporal prediction model for wireless traffic prediction. *IEEE Transactions on Mobile Computing*, 24(9), 8945–8958. https://doi.org/10.1109/TMC.2025.3559066

Lin, Y.-T., Yeh, M. K.-C. & Hsieh, H.-L. (2021). Teaching computer programming to science majors by modelling. *Computer Applications in Engineering Education*, 29(1), 130–144. https://doi.org/10.1002/cae.22247

Linger, R. C., Mills, H. D. & Witt, B. I. (1979). Structured programming: theory and practice. *Tennessee Research and Creative Exchange (TRACE).*

Liu, J., Poulsen, S., Goodwin, E., Chen, H., Williams, G., Gertner, Y. & Franklin, D. (2024). Teaching algorithm design: A literature review. https://arxiv.org/abs/2405.00832

Liu, J., Poulsen, S., Goodwin, E., Chen, H., Williams, G., Gertner, Y. & Franklin, D. (2025). Teaching algorithm design: A literature review. *ACM Trans. Comput. Educ.*, 25(2). https://doi.org/10.1145/3727987

Llerena, J., Alava-Moran, N. & Zamora-Galindo, J. (2021). Learning analytics for student academic tracking, a comparison between Analytics Graphs and Edwiser Reports. In *2021 Second International Conference on Information Systems and Software Technologies (ICI2ST)*, (pp. 101–107). IEEE. https://doi.org/10.1109/ICI2ST51859.2021.00022

Llerena, J. & Ayala-Carabajo, R. (2020). Significant learning activities (ASA) in the modality of face-to-face studies with integration of virtual educational media in Engineering careers. *2020 XV Conferencia Latinoamericana de Tecnologias de Aprendizaje (LACLO)*, (pp. 1–9). https://doi.org/10.1109/LACLO50806.2020.9381134

Llerena Izquierdo, J. (2020). Codifica en Python. http://dspace.ups.edu.ec/handle/123456789/19346

Llerena-Izquierdo, J. (2022). Virtual Classroom Design Model and Its Relation to Student Motivation and Performance in a Moodle Learning Environment During the Emergency of COVID-19. In Springer (Ed.), *Berrezueta, S., Abad, K. (eds) Doctoral Symposium on Information and Communication Technologies - DSICT. Lecture Notes in Electrical Engineering*, volume 846, (pp. 21–32). Springer, Cham. https://doi.org/10.1007/978-3-030-93718-8_3

Llerena-Izquierdo, J. (2023). Adaptation of the curriculum in relation to student learning outcomes in initial programming courses. In *2023 IEEE World Engineering Education Conference (EDUNINE)*, (pp. 1–6). IEEE. https://doi.org/10.1109/EDUNINE57531.2023.10102894

Llerena Izquierdo, J. (2023a). Guía de aprendizaje de programación. http://dspace.ups.edu.ec/handle/123456789/24037

Llerena Izquierdo, J. (2023b). Idear con Blender. http://dspace.ups.edu.ec/handle/123456789/26536

Llerena-Izquierdo, J. (2024). Microlearning as a Learning Consolidation Strategy for the University Blended Learning Modality in a VLE. In J. P. Salgado-Guerrero, H. R. Vega-Carrillo, G. García-Fernández, & V. Robles-Bykbaev (Eds.), *Systems, Smart Technologies and Innovation for Society*, (pp. 189–198). Cham, Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-51982-6_17

Llerena-Izquierdo, J. & Atiaja-Balseca, L. (2021). Gamification Within the Learning Evaluation Process Using Ardora at the Salesian Polytechnic University (Guayaquil, Ecuador). *Springer, Cham*, (pp. 139–150). https://doi.org/10.1007/978-3-030-71503-8_11

Llerena-Izquierdo, J. & Ayala-Carabajo, R. (2018). The use of video recordings as a resource for the evaluation of learning knowledge. https://dspace.ups.edu.ec/handle/123456789/16318

Llerena-Izquierdo, J. & Ayala-Carabajo, R. (2021a). La acción de acompañamiento salesiano del joven investigador universitario desde un proceso formativo. In *Memorias del IV Congreso de Educación Salesiana: desafíos juveniles para una transformación social*, (pp. 137–147). Ediciones Abya Yala. https://dspace.ups.edu.ec/handle/123456789/20973

Llerena-Izquierdo, J. & Ayala-Carabajo, R. (2021b). Training Methodology for Applied Research in the Graduation Mentoring Process of Engineering Students. In *2021 IEEE URUCON*, (pp. 531–535). IEEE. https://doi.org/10.1109/URUCON53396.2021.9647382

Llerena-Izquierdo, J. & Ayala-Carabajo, R. (2021c). University Teacher Training During the COVID-19 Emergency: The Role of Online Teaching-Learning Tools. *International Conference on Information Technology & Systems*, (pp. 90–99). https://doi.org/10.1007/978-3-030-68418-1_10

Llerena-Izquierdo, J. & Ayala-Carabajo, R. (2022). Inventory of ICTs for learning in engineering for emergency virtual teaching by COVID-19. In *2022 IEEE World Engineering Education Conference (EDUNINE)*, (pp. 1–6). IEEE, IEEE. https://doi.org/10.1109/EDUNINE53672.2022.9782389

Llerena-Izquierdo, J. & Ayala-Carabajo, R. (2025a). Ethics of the use of artificial intelligence in academia and research: The most relevant approaches, challenges and topics. *Informatics*, 12(4). https://doi.org/10.3390/informatics12040111

Llerena-Izquierdo, J. & Ayala-Carabajo, R. (2025b). *Guía práctica para redactar y publicar manuscritos científicos*. Ediciones Abya-Yala. https://abyayala.ups.edu.ec/index.php/abayayala/catalog/book/134

Llerena Izquierdo, J. & Ayala Carabajo, R. (2025). Hacia una perspectiva ética e integral del uso de la inteligencia artificial aplicada a la educación: apuntes para la reflexión. In Ediciones ABYA-YALA (Ed.), *Bioética. Inteligencia artificial, cambio climático y muerte asistida*, (pp. 85–109). Quito, Universidad Politécnica Salesiana. https://dspace.ups.edu.ec/handle/123456789/31497

Llerena-Izquierdo, J. & Cedeño-Gonzabay, L. (2020). Photogrammetry and Augmented Reality to Promote the Religious Cultural Heritage of San Pedro Cathedral in Guayaquil, Ecuador. *Communications in Computer and Information Science*, 1194 CCIS, 593–606. https://doi.org/10.1007/978-3-030-42520-3_47

Llerena-Izquierdo, J., Fiallos-Vargas, I., Portugal-Gorozabel, J. & Veloz-Arce, A. (2024a). Towards an experience in AI-driven development for programming applied to multimedia using Tabnine. In *2024 IEEE URUCON*, (pp. 1–5). IEEE. https://doi.org/10.1109/URUCON63440.2024.10850207

Llerena-Izquierdo, J., Mendez-Reyes, J., Ayala-Carabajo, R. & Andrade-Martinez, C. (2024b). Innovations in Introductory Programming Education: The Role of AI with Google Colab and Gemini. https://doi.org/10.3390/educsci14121330

Llerena-Izquierdo, J. & Zamora-Galindo, J. (2021). Using H5P Services to Enhance the Student Evaluation Process in Programming Courses at the Universidad Politécnica Salesiana (Guayaquil, Ecuador). *Springer, Cham*, (pp. 216–227). https://doi.org/10.1007/978-3-030-68080-0_16

Llerena Izquierdo, J. F. & Ayala Carabajo, R. (2018). El Uso De Grabaciones Por Video Como Recurso De Evaluación De Conocimientos De Aprendizajes. In *4to. Congreso Internacional de Ciencia, Tecnología e Innovación para la Sociedad (CITIS). Memoria Académica*, (pp.47̃), Guayaquil, Ecuador. Universidad Politécnica Salesiana.

López-Chila, R., Zambrano-Parrales, F., Córdova-León, J., Andrade-Calderon, J. & Llerena-Izquierdo, J. (2023). Experiences to work in the professors' cloisters as a work environment that improves teaching practices. In *International Conference on Applied Engineering and Innovative Technologies*, (pp. 395–403). Springer. https://link.springer.com/chapter/10.1007/978-3-031-70760-5_37

Lovelace, A. A. (2015). 1842 notes to the translation of the sketch of the analytical engine. *Ada User Journal*, 36(3), 152–180. https://www.ada-europe.org/auj/archive/

Magadan-Diaz, M. & Rivas-Garcia I, J. (2022). Classroom gamification in online higher education: the use of Kahoot. *Campus Virtuales*, 11(1), 137–152. https://doi.org/10.54988/cv.2022.1.978

Malekian, R., Xing, H. & Maleki, A. (2025). Is speckle noise more challenging to mitigate than additive noise? *IEEE Transactions on Information Theory*, 71(8), 6353–6372. https://doi.org/10.1109/TIT.2025.3578430

Malet, A. & Panza, M. (2006). The origins of algebra: From al-Khwarizmi to Descartes. International workshop held at Barcelona, 27-29 March 2003. *Historia Mathematica*, 33(1 SPEC. ISS.), 1–3. https://doi.org/10.1016/j.hm.2005.09.004

Markstrom, K.-A. (2020). The invention by Hedy Lamarr and George Antheil of frequency-hopping spread-spectrum secret communications. *URSI Radio Science Bulletin*, 2020(372), 62–63. https://doi.org/10.23919/URSIRSB.2020.9240110

Martin, U. (2015). Ada Lovelace Computer Scientist. *ITNOW*, 57(4), 54–55. https://doi.org/10.1093/itnow/bwv109

Marwan, S., Dombe, A. & Price, T. W. (2020). Unproductive help-seeking in programming: What it is and how to address it. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '20, (pp. 54–60), New York, NY, USA. Association for Computing Machinery. https://doi.org/10.1145/3341525.3387394

McDevitt, K. & McGettigan, D. (2024). Grace Hopper Earns IEEE Milestone At Last: A winding journey [WIE From Around the World]. *IEEE Women in Engineering Magazine*, 18(2), 38–40. https://doi.org/10.1109/MWIE.2024.3467868

Mejia-Manzano, L. A., Vazquez-Villegas, P., Diaz-Arenas, I. E., Escalante-Vazquez, E. J. & Membrillo-Hernandez, J. (2024). Disciplinary Competencies Overview of the First Cohorts of Undergraduate Students in the Biotechnology Engineering Program under the Tec 21 Model. *Education Sciences*, 14(1). https://doi.org/10.3390/educsci14010030

Mendoza Solórzano, J. F. (2024). Revisión de la literatura sobre el uso de la inteligencia artificial con enfoque a su aplicación en los videojuegos. B.S. thesis, Universidad Politécnica Salesiana. https://dspace.ups.edu.ec/handle/123456789/28119

Michaeli, T. & Romeike, R. (2019). Improving debugging skills in the classroom: The effects of teaching a systematic debugging process. In *Proceedings of the 14th Workshop in Primary and Secondary Computing Education*, WiPSCE '19, New York, NY, USA. Association for Computing Machinery.

Mills, H. (1976). Software development. *IEEE Transactions on Software Engineering*, SE-2(4), 265–273. https://doi.org/10.1109/TSE.1976.233831

Mills, H. D. (1972). Mathematical foundations for structured programming. *Tennessee Research and Creative Exchange (TRACE).*

Möller, R. D. & Collignon, P. (2023). *Towards a Philosophy of Algorithms as an Element of Mathematics Education*, (pp. 227–237). Springer International Publishing: Cham. https://doi.org/10.1007/978-3-031-35209-6_11

Montero Castillo, G. B. (2025). Seguimiento del aprendizaje mediante la minería de datos educativos. Master's thesis, Universidad Politécnica Salesiana. https://dspace.ups.edu.ec/handle/123456789/30544

Muñoz Bermeo, N. A. (2024). Estado de arte sobre el uso de la realidad virtual y aumentada enfocada en la rehabilitación mental ante los síntomas de ansiedad y estrés. B.S. thesis, Universidad Politécnica Salesiana. https://dspace.ups.edu.ec/handle/123456789/28027

Nabirahni, D. M. A., Evans, B. R. & Persaud, A. (2019). Al-Khwarizmi (algorithm) and the development of algebra. *Mathematics Teaching-Research Journal*, 11(12), 13–17. https://mtrj.commons.gc.cuny.edu/volume-11-n-1-2/

Nacar, F., Cakin, A., Dilek, S., Tosun, S. & Chakrabarty, K. (2024). Neuron grouping and mapping methods for 2D-mesh NoC-based DNN accelerators. *Journal of Parallel and Distributed Computing*, 193, 104949. https://doi.org/10.1016/j.jpdc.2024.104949

Naramore, E., Gerner, J., Le Scouarnec, Y., Stolz, J. & Glass, M. K. (2005). *Beginning PHP5, Apache, and MySQL web development*. John Wiley & Sons.

Nassi, I. & Shneiderman, B. (1973). Flowchart techniques for structured programming. *SIGPLAN Not.*, 8(8), 12–26. https://dl.acm.org/doi/pdf/10.1145/953349.953350

Natale, S. & Henrickson, L. (2024). The Lovelace effect: Perceptions of creativity in machines. *New Media and Society*, 26(4), 1909–1926. https://doi.org/10.1177/14614448221077278

Nazareth, J. L. (2023). *Algorithmics: The Spirit of Computing*, (pp. 21–43). Springer International Publishing: Cham. https://doi.org/10.1007/978-3-031-21762-3_3

Newell, A., Shaw, J. C. & Simon, H. A. (1957). Empirical explorations of the logic theory machine: a case study in heuristic. In *Papers Presented at the February 26-28, 1957, Western Joint Computer Conference: Techniques for Reliability*, IRE-AIEE-ACM '57 (Western), (pp. 218–230), New York, NY, USA. Association for Computing Machinery. https://doi.org/10.1145/1455567.1455605

Newton, S. E. & Maas, A. N. (2025). Ada Lovelace: The First Computer Programmer and Why She Is Important to Nursing Informatics. *CIN - Computers Informatics Nursing*, 43(5). https://doi.org/10.1097/CIN.0000000000001312

Nguyen, H., Kim, C. & Li, F. (2024). Space–time recurrent memory network. *Computer Vision and Image Understanding*, 241, 103943. https://doi.org/10.1016/j.cviu.2024.103943

Nikitina, T., Licznerska, M., Ozolipa-Ozola, I. & Lapina, I. (2023). Individual entrepreneurial orientation: comparison of business and STEM students. *Education and Training*, 65(4, SI), 565–586. https://doi.org/10.1108/ET-07-2021-0256

Oaks, J. A. & Alkhateeb, H. M. (2007). Simplifying equations in Arabic algebra. *Historia Mathematica*, 34(1), 45–61. https://doi.org/10.1016/j.hm.2006.02.006

Oliveira, A. L. & Figueiredo, M. A. T. (2024). Artificial Intelligence: Historical Context and State of the Art. *Law, Governance and Technology Series*, 58, 3–24. https://doi.org/10.1007/978-3-031-41264-6_1

Oomori, Y., Tsukamoto, H., Nagumo, H., Takemura, Y., Iida, K., Monden, A. & Matsumoto, K.-i. (2019). Algorithmic expressions for assessing algorithmic thinking ability of elementary school children. In *2019 IEEE Frontiers in Education Conference (FIE)*, (pp. 1–8). https://doi.org/10.1109/FIE43999.2019.9028486

O'Sullivan, L. (2024). Frege and the Logic of the Historical Proposition. *Journal of the Philosophy of History*, 18(1), 68–93. https://doi.org/10.1163/18722636-12341505

Palma Núñez, J. A. (2024). Análisis sobre el impacto de los navegadores web a partir de las ciencias del comportamiento. B.S. thesis, Universidad Politécnica Salesiana. https://dspace.ups.edu.ec/handle/123456789/27866

Parshall, K. H. (1988). The Art of Algebra from Al-Khwārizmī to Viète: A Study in the Natural Selection of Ideas. *History of Science*, 26(2), 129–164. https://doi.org/10.1177/007327538802600202

Pasquali, P. (2020). History of Medical Photography BT - Photography in Clinical Medicine. In P. Pasquali (Ed.), *Photography in Clinical Medicine*, (pp. 47–72). Cham, Springer International Publishing. https://doi.org/10.1007/978-3-030-24544-3_4

Passi, S. & Jackson, S. (2017). Data vision: Learning to see through algorithmic abstraction. In *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing*, CSCW '17, (pp. 2436–2447), New York, NY, USA. Association for Computing Machinery. https://doi.org/10.1145/2998181.2998331

Peng, C. L., Wang, B. Y., Liu, D. C., Wang, N. N., Hu, R. M. & Gao, X. B. (2025). Masked Attribute Description Embedding for Cloth-Changing Person Re-Identification. *IEEE TRANSACTIONS ON MULTIMEDIA*, 27, 1475–1485. https://doi.org/10.1109/TMM.2024.3521730

Pilapaxi Cunalata, N. & Llerena Izquierdo, J. (2025). Disrupción de las tecnologías en el aula de acuerdo con la teoría unificada de aceptación y uso de la tecnología . *Revista InGenio*, 8(1 SE - Artículos), 97–113. https://doi.org/10.18779/ingenio.v8i1.878

Pinna, G., Mena, J. & Funes, S. (2019). Undergraduate students' perceptions about the use of Kahoot as part of the Flipped Classroom methodology. In M. A. C. Gonzalez, F. J. R. Sedano, C. F. Llamas, & F. J. GarciaPenalvo (Eds.), *TEEM'19: Seventh International Conference on Technological Ecosystems for Enhancing Multiculturality*, (pp. 619–625), 1601 Broadway, 10th Floor, New York, NY, United States. Univ Leon, Robot Grp; Univ Salamanca, Res Grp InterAct & eLearning, ASSOC Computing Machinery. https://doi.org/10.1145/3362789.3362900

Pino Gonzalez, E. A. & Navarrete Bonilla, T. M. (2024). Análisis del uso e impacto de herramientas de inteligencia artificial en el aprendizaje de las ciencias computacionales en la educación superior. B.S. thesis, Universidad Politécnica Salesiana. https://dspace.ups.edu.ec/handle/123456789/29276

Ponder, C. G., McGeer, P. & Ng, A. P. (1988). Are applicative languages inefficient? *ACM SIGPLAN Notices*, 23(6), 135–139.

Puga Paredes, J. L. (2023). *Mapeo sistemático sobre el seguimiento del aprendizaje de estudiantes mediante el uso de la minería de datos educativos*. B.S. thesis, Universidad Politécnica Salesiana. https://dspace.ups.edu.ec/handle/123456789/26641

Puma-Rodriguez, R., Jaime-Carriel, J., Pintado-Cuji, M. & Llerena-Izquierdo, J. (2025). Towards a Learning Experience on the Pre-Assembled Mobile Robot Platform in the First Year of Engineering. In E. M. Inga Ortega, N. García Herranz, V. E. Robles-Bykbaev, & E. Gallego Diaz (Eds.), *Springer Nature Switzerland*, (pp. 3–11). Cham, Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-87065-1_1

Pym, D., Spring, J. M. & O'Hearn, P. (2019). Why separation logic works. *Philosophy & Technology*, 32(3), 483–516.

Qiu, H., Xu, C., Fang, J., Zhang, J., Deng, L., Dai, Z., Ding, Y., Wang, Y., Han, Z., Che, Y. & Liu, J. (2025). Dcsolver: Accelerating sparse iterative solvers via divide-and-conquer on gpus. *ACM Trans. Archit. Code Optim.*, 22(3). https://doi.org/10.1145/3746233

Quimis Taines, V. M. (2024). Estado del arte sobre el uso de la realidad virtual y aumentada enfocada a la medicina. B.S. thesis, Universidad Politécnica Salesiana. https://dspace.ups.edu.ec/handle/123456789/27879

Quiñónez Martínez, J. J. (2023). Factores tecnológicos y sociales para la implementación del metaverso en el contexto educativo ecuatoriano. B.S. thesis, Universidad Politécnica Salesiana. https://dspace.ups.edu.ec/handle/123456789/24404

Quishpi Caldas, N. M. (2025). Aprendizaje significativo a través de la implementación estratégica de laboratorios remotos. Master's thesis, Universidad Politécnica Salesiana. https://dspace.ups.edu.ec/handle/123456789/31096

Rafiq, A. (2024). Ada Lovelace Challenge celebrates women in STEM. *Nuclear Future*, 20(2), 30–33.

Rajaraman, V. (1994). *Computer programming in C*. PHI Learning Pvt. Ltd.

Rajwar, K. & Deep, K. (2025). Structural bias in metaheuristic algorithms: Insights, open problems, and future prospects. *Swarm and Evolutionary Computation*, 92, 101812. https://doi.org/10.1016/j.swevo.2024.101812

Randell (2003). Edsger dijkstra. In *2003 The Ninth IEEE International Workshop on Object-Oriented Real-Time Dependable Systems*, (pp. 3–3). https://doi.org/10.1109/WORDS.2003.1267483

Reas, C. & Fry, B. (2007). *Processing: a programming handbook for visual designers and artists*, volume 6812. Mit Press.

Reche, M. P. C., Navas-Parejo, M. R., Villalba, M. J. S. & Ruiz, M. R. S. (2022). Active methodologies and ICT to prevent bullying. Main study background and contributions. *International Journal of Educational Research and Innovation*, 1(18), 151–169. https://doi.org/10.46661/ijeri.5222

Redondi, G., Cimatti, A., Griggio, A. & Mcmillan, K. L. (2024). Invariant checking for smt-based systems with quantifiers. *ACM Trans. Comput. Logic*, 25(4). https://doi.org/10.1145/3686153

Rendón De La Torre, A. J. (2024). Metodologías agiles en la era de la inteligencia artificial, un camino hacia la innovación: Un mapeo sistemático. B.S. thesis, Universidad Politécnica Salesiana. https://dspace.ups.edu.ec/handle/123456789/29306

Rice, A. C. (2024). An Enchantress of Number? Reassessing the Mathematical Reputation of Ada Lovelace. *Notices of the American Mathematical Society*, 71(3), 374–385. https://doi.org/10.1090/noti2890

Roelens, M. (2025). *Algorithms Before Computers*, (pp. 195–208). Springer Nature Switzerland: Cham. https://doi.org/10.1007/978-3-031-86870-2_13

Rosengren, P. L. (2022). *Famous Women Engineers in History - Book 1: Hedy Lamarr*. IEEE. http://ieeexplore.ieee.org.ecups.idm.oclc.org/document/9717301

Rugel Lucín, J. (2021). *Revisión de literatura sobre el uso de los servicios H5P como estrategia de aprendizaje: Un mapeo sistemático*. B.S. thesis, Universidad Politécnica Salesiana. http://dspace.ups.edu.ec/handle/123456789/20944

Sakdiyah, H., Sujiwo, D. A. C. & Panglipur, I. R. (2025). Analysis of Sudoku Compilation Algorithms: Understanding Number Concepts in Students Special Needs at Extraordinary School. *ETDC: Indonesian Journal of Research and Educational Review*, 4(4 SE - Articles), 1236–1248. https://doi.org/10.51574/ijrer.v4i4.3790

Sammet, J. E. (1978). *The early history of COBOL*, (pp. 199–243). Association for Computing Machinery: New York, NY, USA. https://doi.org/10.1145/800025.1198367

Santamaria Chico, M. A. (2024). Modelo de un cuadro de mando académico para los profesores basado en las calificaciones de las actividades de los alumnos. B.S. thesis, Universidad Politécnica Salesiana. https://dspace.ups.edu.ec/handle/123456789/28724

Sasikumar, M., Shikhare, D. & Prakash, R. P. (2014). *Introduction to parallel processing*. PHI Learning Pvt. Ltd.

Sato, R., Gordon, C., Saito, K., Kawashima, H., Nikuni, T. & Watabe, S. (2025). Two-step quantum search algorithm for solving traveling salesman problems. *IEEE Transactions on Quantum Engineering*, 6, 1–12. https://doi.org/10.1109/TQE.2025.3548706

Schlombs, C. (2021). Women, gender and computing: The social shaping of a technical field from Ada Lovelace's algorithm to Anita Borg's 'systers'. *The Palgrave Handbook of Women and Science since 1660*, (pp. 307–331). https://doi.org/10.1007/978-3-030-78973-2_15

Sefati, S. T., Razavi, S. N. & Salehpour, P. (2025). Enhancing autoencoder models for multivariate time series anomaly detection: the role of noise and data amount. *The Journal of Supercomputing*, 81(4), 559. https://doi.org/10.1007/s11227-025-07044-w

Selby, R. W., Basili, V. R. & Baker, F. T. (1987). Cleanroom Software Development: An Empirical Evaluation. *IEEE Transactions on Software Engineering*, SE-13(9), 1027–1037. https://doi.org/10.1109/TSE.1987.233525

Selco, J. I. & Habbak, M. (2021). STEM Students' Perceptions on Emergency Online Learning during the COVID-19 Pandemic: Challenges and Successes. *Education Sciences*, 11(12). https://doi.org/10.3390/educsci11120799

Silva-Maceda, G., David Arjona-Villicaña, P. & Edgar Castillo-Barrera, F. (2016). More time or better tools? a large-scale retrospective comparison of pedagogical approaches to teach programming. *IEEE Transactions on Education*, 59(4), 274–281. https://doi.org/10.1109/TE.2016.2535207

Singh, S. K. (2021). *Linux yourself: concept and programming*. Chapman and Hall/CRC.

Sinnamon, C. & Tarjan, R. E. (2025). Efficiency of self-adjusting heaps. *ACM Trans. Algorithms*, 21(4). https://doi.org/10.1145/3708989

Smirani, L. & Yamani, H. (2024). Analysing the Impact of Gamification Techniques on Enhancing Learner Engagement, Motivation, and Knowledge Retention: A Structural Equation Modelling Approach. *Electronic Journal of E-Learning*, 22(9), 111–124. https://doi.org/10.34190/ejel.22.9.3563

Solomon, E. (1984). *Games programming*. Cambridge University Press.

Stamouli, I., Doyle, E. & Huggard, M. (2004). Establishing structured support for programming students. In *34th Annual Frontiers in Education, 2004. FIE 2004.*, (pp. F2G–5). https://doi.org/10.1109/FIE.2004.1408612

Stepanov, A. A. & McJones, P. (2009). *Elements of programming*. Addison-Wesley Professional.

Sterling, L. & Shapiro, E. Y. (1994). *The art of Prolog: advanced programming techniques*. MIT press.

Stoddart, B., Dunne, S., Mu, C. & Zeyda, F. (2024). Bunch theory: Axioms, logic, applications and model. *Journal of Logical and Algebraic Methods in Programming*, 140, 100977. https://doi.org/10.1016/j.jlamp.2024.100977

Strachey, C. (2000). Fundamental Concepts in Programming Languages. *Higher-Order and Symbolic Computation*, 13(1), 11–49. https://doi.org/10.1023/A:1010000313106

Strawn, G. O. (2023). Masterminds of Computer Design: Charles Babbage and Ada Lovelace. *IT Professional*, 25(4), 7–10. https://doi.org/10.1109/MITP.2023.3297386

Sun, C., Yang, S. & Becker, B. (2024a). Debugging in Computational Thinking: A Meta-analysis on the Effects of Interventions on Debugging Skills. *Journal of Educational Computing Research*, 62(4), 867–901. https://doi.org/10.1177/07356331241227793

Sun, Y., Luo, L., Guo, D., Liu, L. & Xie, J. (2024b). Kmsharing: The framework and space abstraction for efficient data sharing at the network edge. *IEEE/ACM Transactions on Networking*, 32(6), 5440–5458. https://doi.org/10.1109/TNET.2024.3465844

Tamboukou, M. (2023). 'Ever yours, mathematically': women's letters and the mathematical imagination. *Gender and Education*, 35(8), 742–757. https://doi.org/10.1080/09540253.2023.2265283

Tang, P., Zhou, X., Zhou, Y. & Cheng, R. (2025). How do multimedia and blended learning enhance music elective courses? examining the roles of learning attitudes, styles, and teaching presence. *PLOS ONE*, 20(6). https://doi.org/10.1371/journal.pone.0326037

Tatarchenko, K. (2019). Thinking algorithmically: From cold war computer science to the socialist information culture. *Historical Studies in the Natural Sciences*, 49(2), 194–225. https://doi.org/10.1525/hsns.2019.49.2.194

Tatarchenko, K. (2023). Algorithm's Cradle: Commemorating al-Khwarizmi in the Soviet History of Mathematics and Cold War Computer Science. *Osiris*, 38, 286–304. https://doi.org/10.1086/725145

Terroso Sáenz, F. (2025). Artificial Intelligence for Text Analysis: Models, Applications, and Challenges. In *2nd EUt+ International Conference on Languages (EUtLC2024)*. Universidad Politécnica de Cartagena. https://doi.org/10.31428/10317/18679

Thengvall, B. G., Hall, S. N. & Deskevich, M. P. (2025). Measuring the effectiveness and efficiency of simulation optimization metaheuristic algorithms. *Journal of Heuristics*, 31(1), 12. https://doi.org/10.1007/s10732-025-09549-2

Thomas, L., Ratcliffe, M., Woodbury, J. & Jarman, E. (2002). Learning styles and performance in the introductory programming sequence. In *Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '02, (pp. 33–37), New York, NY, USA. Association for Computing Machinery. https://doi.org/10.1145/563340.563352

Thomas, S. L., Nafus, D. & Sherman, J. (2018). Algorithms as fetish: Faith and possibility in algorithmic work. *Big Data & Society*, 5(1), 2053951717751552. https://doi.org/10.1177/2053951717751552

Tikva, C. & Tambouris, E. (2021). A systematic mapping study on teaching and learning Computational Thinking through programming in higher education. *Thinking Skills and Creativity*, 41, 100849. https://doi.org/10.1016/j.tsc.2021.100849

Tsakalerou, M., Perveen, A., Ayapbergenov, A. & Rysbekova, A. (2024). The role of environment on women's perception about their STEM studies: observations from a Global South country. *Scientific Reports*, 14(1), 228. https://doi.org/10.1038/s41598-023-50571-w

Tumbaco-Loor, D. & Llerena-Izquierdo, J. (2024). Use of Leaderboards to Gamified Drive Student Performance as a Motivational Strategy in VLE Environments. In J. P. Salgado-Guerrero, H. R. Vega-Carrillo, G. García-Fernández, & V. Robles-Bykbaev (Eds.), *Systems, Smart Technologies and Innovation for Society*, (pp. 179–188), Cham. Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-51982-6_16

Turbay Gusqui, M. A. (2025). Impacto del Big Data en la evaluación y optimización del rendimiento académico. B.S. thesis, Universidad Politécnica Salesiana. https://dspace.ups.edu.ec/handle/123456789/29957

Vagianou, E. (2006). Program working storage: a beginner's model. In *Proceedings of the 6th Baltic Sea Conference on Computing Education Research: Koli Calling 2006*, Baltic Sea '06, (pp. 69–76), New York, NY, USA. Association for Computing Machinery. https://doi.org/10.1145/1315803.1315816

Valladares, V. O., Montes, L. H. & Arbalti, F. Z. (2023). Game as a Learning Tool in Higher Education. *Revista Electrónica de Investigación Educativa*, 25. https://doi.org/10.24320/redie.2023.25.e28.4952

Valverde-Macias, A. & Llerena-Izquierdo, J. (2022). Google Classroom as a Mobile and Blended Learning Strategy for Salesian Groups Training. In Á. Rocha, P. C. López-López, & J. P. Salgado-Guerrero (Eds.), *Communication, Smart Technologies and Innovation for Society*, (pp. 97–106), Singapore. Springer Singapore. https://doi.org/10.1007/978-981-16-4126-8_10

Valverde Rivera, F. J. (2023). Modelo de seguimiento por videovigilancia pública en red para la seguridad ciudadana mediante Dynamic Domain Name System. B.S. thesis, Universidad Politécnica Salesiana. https://dspace.ups.edu.ec/handle/123456789/25624

Van Roy, P. & Haridi, S. (2004). *Concepts, techniques, and models of computer programming*. MIT press.

Van Snyder, W. (2007). Scientific programming in fortran. *Scientific Programming*, 15(1), 930816. https://doi.org/10.1155/2007/930816

Villao González, J. I. & López Zambrano, E. M. (2024). Revisión de literatura sobre el uso de machine learning enfocados a la seguridad de la información para minimizar vulnerabilidades en el sector educativo. B.S. thesis, Universidad Politécnica Salesiana. https://dspace.ups.edu.ec/handle/123456789/29275

Vinayakumar, R., Soman, K. & Menon, P. (2018). Alg-design: Facilitates to learn algorithmic thinking for beginners. In *2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, (pp. 1–6). https://doi.org/10.1109/ICCCNT.2018.8493952

Vincent, J. A. & Schwager, M. (2025). Reachable polyhedral marching (rpm): An exact analysis tool for deep-learned control systems. *IEEE Transactions on Neural Networks and Learning Systems*, 36(10), 19225–19239. https://doi.org/10.1109/TNNLS.2025.3571720

Viswanadham, N. & Narahari, Y. (2015). *Performance modeling of automated systems*. PHI Learning Pvt. Ltd.

Voelter, M., Deursen, A. v., Kolb, B. & Eberle, S. (2015). Using c language extensions for developing embedded software: a case study. *SIGPLAN Not.*, 50(10), 655–674. https://doi.org/10.1145/2858965.2814276

Von Neumann, J. & Goldstine, H. H. (1947). Numerical inverting of matrices of high order. *American Mathematical Society.*

Von Neumann, J. & Kurzweil, R. (2012). *The computer and the brain*. Yale university press.

Wampler, D. & Payne, A. (2014). *Programming Scala: Scalability= Functional Programming+ Objects*. .º'Reilly Media, Inc."

Wang, H., You, Q. & Duan, S. (2025a). Synchronous byzantine agreement with o(n) messages and o(1) expected time. *IEEE Transactions on Information Forensics and Security*, 20, 338–349. https://doi.org/10.1109/TIFS.2024.3515854

Wang, L., Feng, Q. & Wang, J. (2025b). Approximation algorithms for facility location and k-median with differential privacy. *Theoretical Computer Science*, 1052, 115417. https://doi.org/10.1016/j.tcs.2025.115417

Wang, S., Mondal, A. & Chattopadhyay, A. (2025c). Optimal toffoli-depth quantum adder. *ACM Transactions on Quantum Computing*, 6(3). https://doi.org/10.1145/3743691

Waszek, D. (2025). Boole's Late Manuscript 'On the Nature of Thought': A Rewriting of the Laws of Thought Without Uninterpretables. *History and Philosophy of Logic*, (pp. 1–50). https://doi.org/10.1080/01445340.2025.2550132

Watson, C. & Li, F. W. (2014). Failure rates in introductory programming revisited. In *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education*, ITiCSE '14, (pp. 39–44), New York, NY, USA. Association for Computing Machinery. https://doi.org/10.1145/2591708.2591749

Wirth, N. (1973). The programming language pascal (revised report). *Berichte der Fachgruppe Computerwissenschaften*, 5. https://doi.org/10.3929/ethz-a-000814158

Wu, B., Bao, W. & Zhou, B. B. (2024). Competitive analysis of online elastic caching of transient data in multi-tiered content delivery network. *IEEE Transactions on Parallel and Distributed Systems*, 35(12), 2449–2462. https://doi.org/10.1109/TPDS.2024.3475412

Xiao, X., Chen, Z., Quan, L., Stojanovic, A. & Wu, L. (2025). A knapsack modelling approach to financial resource allocation problem using a dual search pattern firefly algorithm. *International Journal of Bio-Inspired Computation*, 26(5), 1–13. https://doi.org/10.1504/IJBIC.2025.149184

Xu, R. & Li, Y. (2024). Interpretable Spatial–Temporal Graph Convolutional Network for System Log Anomaly Detection. *Advanced Engineering Informatics*, 62, 102803. https://doi.org/10.1016/j.aei.2024.102803

Yevick, D. (2005). *A First Course in Computational Physics and Object-Oriented Programming with C++ Hardback with CD-ROM*. Cambridge University Press

Yoder, C. M. & Schrag, M. L. (1978). Nassi-shneiderman charts an alternative to flowcharts for design. In *Proceedings of the Software Quality Assurance Workshop on Functional and Performance Issues*, (pp. 79–86), New York, NY, USA. Association for Computing Machinery. https://dl.acm.org/doi/pdf/10.1145/800283.811104

Yolanda Burgos-Lopez, M., Ibarra-Herrera, C. C. & Chuck-Hernandez, C. (2024). Fighting the gender drop off from STEM careers throughout the University-Industry Collaboration: Effects of Corporate Mentorship. In *2024 World Engineering Education Forum-Global Engineering Deans Council, WEEF-GEDC*, World Engineering Education Forum - Global Engineering Deans Council, 345 E 47TH ST, New York, NY 10017 USA. IEEE. https://doi.org/10.1109/WEEF-GEDC63419.2024.10854963

Yue, J., Bi, X. & Chen, Z. (2024). Heterogeneous-branch integration framework: Introducing first-order predicate logic in Logical Reasoning Question Answering. *Neurocomputing*, 609. https://doi.org/10.1016/j.neucom.2024.128504

Yutong, C. & Jin, H. (2024). A bibliometric and visualization analysis of artificial intelligence and machine learning. In *2024 IEEE 4th International Conference on Electronic Communications, Internet of Things and Big Data (ICEIB)*, (pp. 219–222). https://doi.org/10.1109/ICEIB61477.2024.10602604

Yıldız, Ö. F. (2025). The Medieval Islamicate Terms in Science. *Indonesian Journal of Islamization Studies*, 3(1 SE - Articles), 53–77. https://doi.org/10.21111/injas.v3i1.14353

Zabala-Vargas, S. A., Garcia-Mora, L. H., Ardila-Segovia, D. A. & de Benito-Crosetti, B. L. (2019). Motivation increase of mathematics students in Engineering - A proposal from Game Based Learning. In *2019 International Symposium on Engineering Accreditation and Education, ICACIT*, 345 E 47TH ST, NEW YORK, NY 10017 USA. IEEE. https://doi.org/10.1109/ICACIT46824.2019.9130297

Zambrano Peñafiel, G. M. (2024). Revisión sistemática las tecnologías aplicadas para mitigar el covid-19 con inteligencia artificial desde un enfoque al aprendizaje profundo. B.S. thesis, Universidad Politécnica Salesiana. https://dspace.ups.edu.ec/handle/123456789/28113

Zapata Martínez, J. & Llerena-Izquierdo, J. (2023). Las TIC después del COVID-19: la perspectiva de los profesores universitarios. *CODES*, 5(SE - Artículos). https://doi.org/10.15443/codes1988

Zerega-Prado, J. & Llerena-Izquierdo, J. (2022). Arquitectura de consolidación de la información para seguros de la salud mediante Big Data. *Memoria Investigaciones en Ingeniería*, 0(23), 18–31. https://doi.org/10.36561/ING.23.3

Zhang, S. C., Ding, Y. B., Huo, T. X., Duan, S. K. & Wang, L. D. (2025). PointAttention: Rethinking Feature Representation and Propagation in Point Cloud. *IEEE TRANSACTIONS ON MULTIMEDIA*, 27, 327–339. https://doi.org/10.1109/TMM.2024.3521745

Zhang, Y. & Vatedka, S. (2024). Multiple packing: Lower bounds via error exponents. *IEEE Transactions on Information Theory*, 70(2), 1008–1039. https://doi.org/10.1109/TIT.2023.3334032

Zhu, N., Zhao, F., Yu, Y. & Wang, L. (2024). A hierarchical reinforcement learning-aware hyper-heuristic algorithm with fitness landscape analysis. *Swarm and Evolutionary Computation*, 90, 101669. https://doi.org/10.1016/j.swevo.2024.101669

Zhu, Y. J., Sui, S. S., Du, W. S., Li, X. & Liu, P. (2025). Picking point localization method of table grape picking robot based on you only look once version 8 nano. *Engineering Applications of Artificial Intelligence*, 146. https://doi.org/10.1016/j.engappai.2025.110266

Autor

Joe Llerena Izquierdo, MSc.
Universidad Politécnica Salesiana
email: jllerena@ups.edu.ec, ORCID: 0000-0001-9907-7048



De una familia guayaquileña con carisma salesiano, su padre, hermanas y hermano de colegios de salesianas y salesianos, siendo su tía una Hija de María Auxiliadora, Sor Martha Llerena Romo, FMA. Exalumno del Colegio Cristóbal Colón, 13 años de fidelidad. Especialidad Físico Matemático, promoción 49 (1993). Realiza su voluntariado en Esmeraldas, parroquia La Tolita, San Juan Bosco, vía a Atacames en 1996 junto al padre Jaime Calero, sdb (†). Fue coordinador de los Oratorios: Don Bosco (Cristóbal Colón), siendo su director el P. Marco Paredes, sdb., del Oratorio Don Bosco (La Delicia), con su director el coadjutor Sr. Ángel Robusti, sdb (†), y del Oratorio Santiago Apóstol, con su director el P. Rafael Guevara, sdb (†), entre 1997 y 2004. En 2006 realizó el curso Docentes de Salesianidad nivel 1, en el Centro Regional de Formación Permanente en Quito, con el P. Fernando Peraza Leal, sdb (†). Docente en el Cristóbal Colón (1994-2010) junto a los salesianos de Don Bosco (SDB) y docente en el colegio Fiscomisional Santa María Mazzarello (2003-2015) junto a las Hijas de María Auxiliadora (FMA).

Ingeniero en Computación por la Escuela Superior Politécnica del Litoral (ESPOL, 2001) y Magíster en Sistemas de Información Gerencial (ESPOL, 2007); además, es Máster en Administración de Empresas (UPS, 2012) y Máster en Ciencias de la Educación (UPS, 2012) por la Universidad Politécnica Salesiana, Ph.D.(c) del Programa de Doctorado de Educación y TIC (e-Learning) de la Universitat Oberta de Catalunya (UOC), Barcelona, España (2026). Desde 1995 se ha dedicado a la docencia en instituciones de educación media y superior. Miembro IEEE de Sección Ecuador R9 desde el 2008. Crea la Rama Estudiantil IEEE en la UPS sede Guayaquil en el 2009. Por su eficaz voluntariado y ser motivador de jóvenes universitarios, fue nombrado Coordinador de Actividades Estudiantiles (SAC a nivel nacional) de la Sección Ecuador (2010-2012). Miembro de la Sociedad IEEE en Educación. En el 2023 tiene el privilegio de ser Miembro Senior IEEE a nivel mundial.

Labora en la Universidad Politécnica Salesiana, sede Guayaquil (2000). Profesor en la Carrera de Computación. Coordinador del claustro de Computación Aplicada y Metodologías de la Computación (2019). Director Técnico de la Unadedvi (2010-2017). En 2011 le fue otorgado el reconocimiento Valeriano Gavinelli Bovio por su compromiso y disponibilidad para encontrar soluciones creativas mediante la práctica de valores salesianos.

Coordinador del Grupo de Investigación en Enseñanza y Aprendizaje de las Ciencias para la Ingeniería (GIEACI, 2014) y miembro del Grupo de Innovación Educativa en Tecnologías de la Información y Comunicaciones (Gie-Ticea, 2019). En 2025, es parte del equipo de posgrado, coordina las maestrías en Software y, en Tecnologías de la Información y Comunicación para la Educación. Miembro del equipo de Mentores en Investigación con mayor índice, i1. Galardonado en la categoría de mayor producción de alto impacto (2022 al 2025). Autor de libros académicos y, artículos de alto impacto como primer autor, coautor, último autor y autor único, según las métricas en Web of Science y Scopus. Revisor de artículos en revistas relevantes de cuartil Q1. Sus intereses en investigación se relacionan con la educación en la ingeniería.