1era Ed.
2025

PUERTO MADERO
EDITORIAL

FUNDAMENTOS DE PROGRAMACION CON PYTHON

PATRICIO XAVIER MORENO VALLEJO
GISEL KATERINE BASTIDAS GUACHO
MARIA ELENA VALLEJO SANAGUANO

puertomaderoeditorial.com.ar
La Plata - Argentina

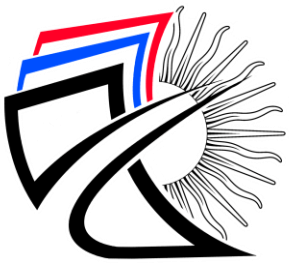
FUNDAMENTOS DE PROGRAMACION CON PYTHON

Patricio Xavier Moreno Vallejo, Gisel Katerine Bastidas Guacho,
Maria Elena Vallejo Sanaguano

ISBN: 978-631-6557-57-5

ISBN: 978-631-6557-57-5

ISBN: 978-631-6557-57-5



PUERTO MADERO
EDITORIAL

FUNDAMENTOS DE PROGRAMACION CON PYTHON

AUTORES:
PATRICIO XAVIER MORENO VALLEJO
GISEL KATERINE BASTIDAS GUACHO
MARIA ELENA VALLEJO SANAGUANO

ISBN: 978-631-6557-57-5

Moreno Vallejo, Patricio Xavier
Fundamentos de programación con Python / Patricio Xavier Moreno Vallejo ; Gisel Katerine Bastidas Guacho ; María Elena Vallejo Sanaguano. - 1a ed. - La Plata : Puerto Madero Editorial Académica, 2025.
Libro digital, PDF/A

Archivo Digital: descarga y online
ISBN 978-631-6557-57-5

1. Acceso a la Información. 2. Computación. 3. Análisis de Datos. I. Bastidas Guacho, Gisel Katerine II. Vallejo Sanaguano, María Elena III. Título
CDD 004.071

Licencia Creative Commons:
Atribución-NoComercial-SinDerivar 4.0 Internacional (CC BY-NC-SA 4.0)

ISBN: 978-631-6557-57-5

ISBN: 978-631-6557-57-5

DEDICATORIA

A la Escuela Superior Politécnica de Chimborazo, alma mater donde cursamos nuestros estudios académicos.

A los niños Leonel Robayo Moreno, Benjamín Moreno Bastidas y a la niña Victoria Robayo Moreno, quienes fueron la inspiración para el desarrollo de esta obra.

ISBN: 978-631-6557-57-5

PUERTO MADERO EDITORIAL

Primera Edición, Julio 2025

FUNDAMENTOS DE PROGRAMACION CON PYTHON
ISBN: 978-631-6557-57-5

Editado por:
Sello editorial: ©Puerto Madero Editorial Académica
Nº de Alta: 933832

Editorial: © Puerto Madero Editorial Académica
CUIL: 20630333971
Calle 45 N491 entre 4 y 5
Dirección de Publicaciones Científicas Puerto Madero Editorial Académica
La Plata, Buenos Aires, Argentina
Teléfono: +54 9 221 314 5902
+54 9 221 531 5142
Código Postal: AR1900

Este libro se sometió a arbitraje bajo el sistema de doble ciego (peer review)

Corrección y diseño:
Puerto Madero Editorial Académica
Diseñador Gráfico: José Luis Santillán Lima

Diseño, Montaje y Producción Editorial:
Puerto Madero Editorial Académica
Diseñador Gráfico: Santillán Lima, José Luis

Director del equipo editorial: Santillán Lima, Juan Carlos

Editor: Santillán Lima, Juan Carlos
Molina Granja, Fernando Tiverio

Hecho en Argentina
Made in Argentina

ISBN: 978-631-6557-57-5

ISBN: 978-631-6557-57-5

AUTORES:

Patricio Xavier Moreno Vallejo

Escuela Superior Politécnica de Chimborazo, Facultad de Administración de Empresas, Carrera de Gestión del Transporte, Panamericana Sur Km 1 1/2, EC060155, Riobamba, Chimborazo, Ecuador

pxmoreno@espoch.edu.ec

https://orcid.org/0000-0002-9317-9884

Gisel Katerine Bastidas Guacho

Escuela Superior Politécnica de Chimborazo, Facultad de Informática y Electrónica, Carrera de Software, Panamericana Sur Km 1 1/2, EC060155, Riobamba, Chimborazo, Ecuador

gis.bastidas@espoch.edu.ec

https://orcid.org/0000-0002-6070-7193

María Elena Vallejo Sanaguano

Escuela Superior Politécnica de Chimborazo, Facultad de Recursos Naturales, Panamericana Sur Km 1 1/2, EC060155, Riobamba, Chimborazo, Ecuador

pmoreno@espoch.edu.ec

https://orcid.org/0000-0001-8255-8953

ISBN: 978-631-6557-57-5

ISBN: 978-631-6557-57-5

CONTENIDO

CONTENIDO XIII

RESUMEN XVII

INTRODUCCIÓN XIX

CAPÍTULO 1 1

1 CONCEPTOS BASICOS 1

1.1 GENERALIDADES DE LA INFORMÁTICA 1

1.1.1 Qué es la informática 1

1.1.2 Qué es una computadora 2

1.1.3 Sistemas de archivos 7

1.1.4 Interpretación de los datos por las computadoras 9

1.1.5 Conversión de un numero decimal a binario 10

1.1.6 Codificación ASCII 11

1.1.7 Codificación Unicode 13

1.1.8 Codificación de colores 14

1.2 APLICACIONES INFORMÁTICAS 16

1.3 CONTEXTO SOCIAL DE LA INFORMÁTICA 19

1.4 DATOS, EXPRESIONES Y FUNCIONES 22

1.4.1 Representación externa e interna de los datos 22

1.4.2 Tipos de datos 22

1.4.3 Tipos de datos numéricos 23

1.4.4 Tipo de dato carácter 24

1.4.5 Constantes y variables 24

1.4.6 Operación de asignación 25

1.4.7 Operación de entrada salida 26

1.4.8 Lenguajes de Programación 27

1.4.9 Comparación de los lenguajes 38

1.4.10 Expresiones 39

1.5 DISEÑO DE ALGORITMOS, PSEUDOCÓDIGO Y DIAGRAMAS DE FLUJO 50

1.5.1 Diseño de algoritmos 50

1.5.2 Pseudocódigo 54

ISBN: 978-631-6557-57-5

1.5.3 Diagramas de flujo *57*

1.6 EJERCICIOS PROPUESTOS 72

CAPÍTULO 2 ..... **78**

2 ESTRUCTURAS DE PROGRAMACION **78**

2.1 ESTRUCTURAS SECUENCIALES 78

2.2 ESTRUCTURAS SELECTIVAS 85

2.3 ESTRUCTURAS DE REPETICIÓN 91

2.4 EJERCICIOS DE APLICACIÓN 105

2.4.1 Ejercicios resueltos *105*

2.4.2 Ejercicios propuestos *163*

CAPÍTULO 3 ..... **165**

3 VECTORES Y MATRICES **165**

3.1 CONCEPTOS GENERALES 165

3.2 ARREGLOS UNIDIMENSIONALES (VECTORES) 165

3.2.1 Declaración de Vectores *166*

3.2.2 Acceso a los elementos *167*

3.2.3 Recorrido de vectores *168*

3.2.4 Operaciones básicas *170*

3.2.5 Funciones y métodos para vectores *172*

3.2.6 Aplicaciones de vectores *174*

3.2.7 Vectores y memoria *176*

***3.3* ARREGLOS BIDIMENSIONALES (MATRICES)** 178

***3.4* EJERCICIOS DE APLICACIÓN** 181

3.4.1 Ejercicios resueltos *181*

3.4.2 Ejercicios propuestos *272*

CAPÍTULO 4 ..... **276**

4 PROGRAMACION MODULAR **276**

4.1 LA PROGRAMACIÓN MODULAR 276

4.1.1 Módulo *276*

4.1.2 Cohesión y acoplamiento *277*

4.1.3 Diseño de módulos *280*

4.1.4 Implementación de módulos *281*

4.1.5 Beneficios de la programación modular 287
4.1.6 Prácticas de programación modular 289
4.1.7 Funciones y subprogramas 293
4.2 EJERCICIOS DE APLICACIÓN 306
4.2.1 Ejercicios resueltos 306
4.2.2 Ejercicios propuestos 345
BIBLIOGRAFÍA 349
DE LOS AUTORES 353
PATRICIO XAVIER MORENO VALLEJO 353
GISEL KATERINE BASTIDAS GUACHO 354
MARIA ELENA VALLEJO SANAGUANO 355

ISBN: 978-631-6557-57-5

ISBN: 978-631-6557-57-5

RESUMEN

El libro Fundamentos de Programación con Python, está diseñado para proporcionar una base sólida en la programación. Está dirigido a estudiantes que buscan adquirir competencias en la creación de algoritmos, la solución de problemas y el desarrollo de software mediante el uso de código en diversos entornos. A lo largo del texto, se cubren los conceptos fundamentales de la informática, incluyendo el análisis de algoritmos, estructuras de datos, diseño de software, y tecnologías de la información.

El libro combina un enfoque teórico y práctico, destacando el uso del lenguaje Python sin descuidar otros lenguajes fundamentales en la enseñanza de programación, como JavaScript, C++ y Java. Abarca desde la sintaxis básica de Python, hasta la implementación de funciones, manejo de listas, diccionarios y estructuras de control como bucles y condicionales. Los capítulos están estructurados para introducir progresivamente temas como el diseño de algoritmos, la modularidad en la programación, el uso de funciones y variables en distintos ámbitos, y la reutilización de código. Esto permite a los lectores consolidar los conocimientos teóricos con ejemplos prácticos.

Una característica clave del libro es el uso de pseudocódigo y diagramas de flujo, que facilitan la comprensión de los conceptos antes de implementarlos en los diferentes lenguajes de programación. El lector encontrará comparaciones entre los cuatro lenguajes, lo que fomenta una comprensión integral y adaptable a diversos entornos tecnológicos.

Al finalizar el libro, los lectores estarán capacitados para escribir código eficiente, claro y comprensible, aplicando buenas prácticas de programación que aseguren el mantenimiento y la escalabilidad de los proyectos. Se enfatiza la importancia de la reutilización del código, así como la claridad en su escritura, con el fin de optimizar el trabajo en equipos de desarrollo y enfrentar proyectos de software con éxito en distintos lenguajes de programación.

ISBN: 978-631-6557-57-5

ISBN: 978-631-6557-57-5

INTRODUCCIÓN

El libro Fundamentos de Programación con Python, está diseñado para ofrecer a los estudiantes una introducción sólida a la programación, utilizando cuatro de los lenguajes más populares y versátiles. A través de cuatro capítulos, este texto cubre desde los conceptos básicos hasta las técnicas de programación modular, con el objetivo de proporcionar una formación completa para aquellos que buscan adquirir habilidades prácticas en la creación de algoritmos y el desarrollo de soluciones de software.

El Capítulo 1, titulado Conceptos Básicos, introduce al lector a la informática y su papel en la sociedad moderna. Aquí, se abordan temas como la estructura y funcionamiento de las computadoras, los sistemas de archivos y los datos. Además, se explica el concepto de algoritmo y su importancia para la resolución de problemas mediante la programación. Este capítulo establece una base fundamental para entender el proceso de desarrollo de software, presentando también el pseudocódigo y los diagramas de flujo como herramientas útiles para visualizar algoritmos antes de implementarlos en un lenguaje de programación.

En el Capítulo 2, Estructuras de Programación, se profundiza en las estructuras de control que permiten manejar el flujo de un programa de manera eficiente. El capítulo cubre las estructuras secuenciales, selectivas y repetitivas, fundamentales para tomar decisiones y realizar operaciones repetitivas dentro de un código. Se presentan ejemplos en Python, JavaScript, C++ y Java, lo que permite al lector comparar y entender cómo estas estructuras se implementan en diferentes lenguajes. Esta sección también incluye ejercicios prácticos que ayudan a reforzar el aprendizaje de estas estructuras esenciales para cualquier desarrollador.

El Capítulo 3, Vectores y Matrices, introduce a los lectores en el manejo de estas importantes estructuras de datos. Los vectores, también conocidos como arreglos unidimensionales, y las matrices, que son arreglos multidimensionales, son herramientas clave para organizar y procesar grandes volúmenes de datos. Este capítulo explica cómo se crean, manipulan y utilizan los vectores y matrices en los cuatro lenguajes abordados en el libro. Además, se presentan ejemplos prácticos para ilustrar cómo realizar

ISBN: 978-631-6557-57-5

operaciones matemáticas y lógicas con estos elementos, permitiendo al lector desarrollar programas más eficientes y organizados.

Finalmente, el Capítulo 4, Programación Modular, termina el libro con un enfoque en la modularidad del código, un concepto clave para el desarrollo de software a gran escala. La programación modular permite dividir un programa en partes más pequeñas y manejables, lo que mejora su legibilidad, facilita su mantenimiento y fomenta la reutilización de código. Este capítulo explica cómo organizar y estructurar el código en módulos y funciones, utilizando ejemplos en Python, JavaScript, C++ y Java para mostrar cómo se puede aplicar este enfoque en diferentes lenguajes. Además, se destacan las buenas prácticas para el desarrollo de software, subrayando la importancia de la encapsulación y la separación de responsabilidades dentro de un programa.

Cada capítulo del libro combina teoría y práctica, con ejercicios al final de cada sección que permiten a los estudiantes aplicar los conocimientos adquiridos. Los ejemplos y ejercicios están diseñados para ser implementados en los cuatro lenguajes de programación cubiertos, lo que facilita la comparación entre ellos y ayuda al lector a desarrollar una comprensión más amplia y profunda de la programación. Algo a destacar es el pseudocódigo y diagramas de flujo presentes en los ejercicios que ayudan a visualizar el desarrollo de programas antes de implementar el código, fomentando una comprensión más profunda de los procesos lógicos detrás de la programación.

A lo largo del libro, el lector no solo aprenderá a escribir código en Python, JavaScript, C++ y Java, sino que también adquirirá habilidades esenciales para resolver problemas complejos, optimizar el rendimiento de los programas y trabajar en proyectos de software más grandes y modulares. El objetivo final es que, al finalizar el libro, los estudiantes estén equipados con una base sólida para enfrentar los desafíos del desarrollo de software en el mundo real, con un enfoque en la eficiencia, la escalabilidad y la reutilización de código.

Este libro es ideal tanto para aquellos que están comenzando en el mundo de la programación como para quienes ya tienen experiencia y buscan mejorar sus habilidades en múltiples lenguajes de programación. La estructura clara y los ejemplos prácticos permiten que el lector avance a su propio ritmo, construyendo una comprensión sólida de

ISBN: 978-631-6557-57-5

los fundamentos de la programación mientras aprende a aplicar estos conocimientos en una variedad de contextos y plataformas

ISBN: 978-631-6557-57-5

.

ISBN: 978-631-6557-57-5

CAPÍTULO 1

1 CONCEPTOS BASICOS

1.1 Generalidades de la Informática

La informática mediante el uso de tecnologías computacionales tiene una gran influencia en el avance y desarrollo de nuestra sociedad actual. A través de la convergencia entre la teoría y la práctica, esta disciplina trata desde el análisis de algoritmos y estructuras de datos, hasta el diseño de complejas arquitecturas de computadoras y el desarrollo de software. Se extienden hacia diversas áreas del conocimiento, como la inteligencia artificial, la ciberseguridad y el procesamiento de grandes volúmenes de datos, la informática se presenta como una ciencia aplicada esencial para la resolución de problemas prácticos, potenciando así la innovación y la eficiencia en múltiples sectores industriales, científicos y educativos.

En el núcleo de la informática se encuentra la computadora, dispositivo capaz de procesar datos para transformarlos en información útil, siguiendo instrucciones específicas dadas por programas de software. Esta máquina electrónica avanzada, compuesta por hardware y software, realiza operaciones fundamentales de entrada, procesamiento y salida de datos, representando el principio del procesamiento de información digital. Su capacidad para ejecutar una amplia gama de tareas complejas, desde el control de procesos industriales hasta la facilitación de la comunicación global, la convierte en una herramienta indispensable en la vida cotidiana. A medida que evoluciona la tecnología, la informática sigue expandiendo sus horizontes, abriendo nuevas vías para el análisis de datos, la seguridad informática y el desarrollo de soluciones tecnológicas innovadoras, lo que refleja su rol vital en el progreso y bienestar de la sociedad moderna.

1.1.1 Qué es la informática

La palabra "informática" proviene de la contracción de dos términos en francés: "information" y "automatique". En español, estos términos se traducen como "información" y "automática". *La informática es la disciplina científica que se encarga del estudio, desarrollo, implementación, soporte y gestión de la información y los datos*

ISBN: 978-631-6557-57-5

mediante el uso de tecnologías de computación, por lo tanto, la informática se refiere al procesamiento automático de la información, y es la ciencia que estudia el tratamiento racional y automático de la información mediante computadoras.

La informática no solo implica la comprensión de los sistemas de computación existentes, sino también la investigación y creación de nuevas tecnologías, métodos y técnicas para el procesamiento y manejo de la información.

Engloba la puesta en práctica de las tecnologías computacionales para resolver problemas específicos y el mantenimiento de estos sistemas para asegurar su correcto funcionamiento y su adaptabilidad a las necesidades cambiantes.

Tiene la capacidad de organizar, almacenar, recuperar, proteger y analizar datos de manera eficaz. Esto es fundamental en la era actual, donde el volumen de datos generados y su importancia para la toma de decisiones son mayores que nunca.

La informática se basa en el uso de ordenadores y software para llevar a cabo sus objetivos. Esto incluye desde hardware básico, como microprocesadores y dispositivos de almacenamiento, hasta software avanzado, como sistemas operativos, aplicaciones de base de datos, y plataformas de desarrollo.

La informática, constituye una ciencia aplicada que abarca teoría y práctica. Se relaciona con el análisis de algoritmos (procedimientos o fórmulas para resolver problemas), estructuras de datos, programación, arquitectura de computadoras, y más recientemente, con áreas como la inteligencia artificial, la ciberseguridad, el procesamiento de datos a gran escala (big data), y el desarrollo de aplicaciones web y móviles.

1.1.2 Qué es una computadora

Una computadora es una máquina electrónica avanzada que procesa datos para convertirlos en información útil. Funciona mediante la interpretación y ejecución de instrucciones específicas contenidas en programas de software. En su forma más elemental, una computadora realiza operaciones fundamentales como entrada de datos (input), su procesamiento y generación de datos de salida (output), siguiendo el principio de "procesamiento de información digital".

Desde un punto de vista técnico, una computadora (figura 1.1) se compone de hardware y software. El hardware incluye el sistema central de procesamiento o CPU

ISBN: 978-631-6557-57-5

(Central Processing Unit), memoria principal para el almacenamiento temporal de información y dispositivos periféricos que facilitan la entrada y salida de datos, como teclados, ratones, monitores, impresoras y dispositivos de almacenamiento secundario. El software, por otro lado, comprende el sistema operativo que coordina el hardware y el software, y las aplicaciones que realizan tareas específicas para el usuario.

Figura 1.1 Computadora y periféricos

Fuente: Héctor Gómez

Las computadoras pueden realizar una amplia gama de tareas complejas más allá de la simple aritmética, desde controlar procesos industriales hasta permitir la comunicación global a través de Internet. Además, son capaces de tomar decisiones lógicas basadas en algoritmos, realizar tareas de aprendizaje automático y procesar grandes cantidades de datos para descubrir patrones y tendencias. Esto las convierte en herramientas indispensables en la vida cotidiana, la industria, la ciencia y la educación.

Figura 1.2 Componentes de hardware

Fuente: Gustavo Castellanos

ISBN: 978-631-6557-57-5

A continuación, se especifican los componentes de hardware (Figura 1.2), que constituyen los elementos físicos de una computadora son:

1. **Unidad Central de Procesamiento (CPU)**

Es el cerebro de la computadora, incluye la *Unidad Aritmética Lógica* (ALU) donde se realizan los cálculos matemáticos y las decisiones lógicas necesarias durante la ejecución de los programas, así como una *Unidad de Control* que dirige y coordina todas las operaciones de la computadora, especifica cuándo y cómo deben realizarse las operaciones de lectura y escritura en la memoria, la ejecución de operaciones en la ALU, y la interpretación de las instrucciones del programa.

2. **Memoria de Acceso Aleatorio (RAM)**

La RAM (Random Access Memory) es la memoria principal de la computadora, utilizada para almacenar temporalmente los datos y programas que la CPU está utilizando activamente. Esto incluye el sistema operativo, las aplicaciones en uso, y los datos en proceso. La característica más importante de la memoria principal es su capacidad para permitir el acceso rápido a los datos por parte de la CPU, lo que es esencial para el rendimiento general del sistema.

Esta memoria es volátil, lo que significa que los datos almacenados en ella se pierden cuando el dispositivo se apaga o se reinicia. Su volatilidad contrasta con la naturaleza permanente del almacenamiento secundario, como los discos duros y las unidades de estado sólido, donde los datos se conservan incluso cuando el dispositivo está apagado.

Existen varios tipos de memoria principal, que incluyen:

a) **DRAM (Dynamic Random-Access Memory):** Es el tipo más común de memoria RAM utilizada en las computadoras personales y servidores. La DRAM necesita ser refrescada constantemente con cargas eléctricas para mantener la información almacenada, lo que la hace relativamente lenta y energéticamente más demandante que otros tipos de RAM.

b) **SRAM (Static Random-Access Memory):** A diferencia de la DRAM, la SRAM no necesita ser refrescada constantemente, lo que la hace más rápida y menos consumidora de energía. Sin embargo, es más costosa de producir, lo que limita

ISBN: 978-631-6557-57-5

su uso a cachés de CPU y como memoria de trabajo en dispositivos pequeños, donde la cantidad de memoria requerida es relativamente baja.

c) **SDRAM (Synchronous Dynamic Random-Access Memory):** Este tipo de memoria es una evolución de la DRAM, diseñada para sincronizarse con el ciclo de reloj de la CPU, lo que mejora la eficiencia en la transferencia de datos. DDR (Double Data Rate) SDRAM es una versión más avanzada que puede transferir datos dos veces por ciclo de reloj, aumentando significativamente el rendimiento. Las generaciones sucesivas de DDR (DDR2, DDR3, DDR4, y más recientemente DDR5) han ofrecido mejoras en velocidad, eficiencia energética y capacidad.

d) **RAM no volátil (NVRAM):** Combina las características de la RAM tradicional con la capacidad de retener datos incluso cuando se apaga la energía. Esto se logra mediante diferentes tecnologías, incluida la memoria flash y tecnologías más especializadas como FeRAM, MRAM o RRAM. Estas memorias son útiles en aplicaciones donde la preservación de datos en caso de interrupción de energía es crítica.

Cada tipo de memoria principal tiene sus propias aplicaciones específicas, ventajas y limitaciones, y se eligen en función de las necesidades de rendimiento, costo y aplicación específica de cada sistema computacional.

3. **Almacenamiento**

Incluye dispositivos como discos duros (*HDD Hard Disk Drive*) que ofrecen una gran capacidad de almacenamiento a un costo relativamente bajo. Es ideal para almacenar grandes cantidades de datos que no requieren acceso rápido, como archivos multimedia. Tiene partes móviles, lo que lo hace más susceptible al daño por golpes o caídas.

Las unidades de estado sólido (*SSD Solid-State Drive*) utilizan memoria flash para almacenar datos, esto elimina las partes móviles presentes en los HDD. Esto las hace más rápidas y resistentes a golpes. Con estas unidades se tiene acceso rápido a los datos, menor tiempo de arranque del sistema, y operación más silenciosa. Son ideales para sistemas operativos, aplicaciones que requieren alto rendimiento, y como unidades de arranque. Aunque los precios han disminuido, las SSD todavía son más costosas por gigabyte que los HDD. Además, tienen un número limitado de ciclos de escritura, aunque esto rara vez es un problema para la mayoría de los usuarios.

ISBN: 978-631-6557-57-5

Una variante más rápida de las SSD son las *NVMe* (Non-Volatile Memory Express) utilizan el bus PCIe de la computadora para ofrecer velocidades de transferencia significativamente más altas que las SSD tradicionales. Ideal para videojuegos de alta gama, edición de video y tareas que requieren grandes transferencias de datos.

Las tarjetas de memoria como *SD* o *microSD* y las unidades *flash USB* ofrecen almacenamiento portátil y fácilmente removible, ideal para transferir archivos entre dispositivos, siendo compatibles con una amplia gama de dispositivos. Tienen capacidades más limitadas en comparación con los HDD y SSD, y pueden ser más fáciles de perder debido a su tamaño compacto.

4. **Placa Base (Motherboard)**

Es la tarjeta principal que conecta todos los componentes de la computadora, incluyendo la CPU, la memoria RAM, las tarjetas de expansión, y los dispositivos de almacenamiento. Distribuye la energía a estos componentes y permite su comunicación.

5. **Tarjeta Gráfica (GPU)**

Es responsable de procesar y generar las imágenes, video y gráficos que se muestran en el monitor. Algunas CPUs tienen capacidades gráficas integradas, pero las tarjetas gráficas dedicadas ofrecen un rendimiento superior para juegos, diseño gráfico y tareas intensivas en video.

6. **Fuente de Alimentación**

Convierte la corriente alterna (AC) del enchufe de la pared en corriente continua (DC) de bajo voltaje, que es la que utilizan los componentes internos de la computadora.

7. **Dispositivos de Entrada**

Incluyen teclado, ratón, cámara web, micrófono y otros dispositivos que permiten al usuario interactuar con la computadora.

8. **Dispositivos de Salida**

Incluyen el monitor, altavoces, impresora y otros dispositivos que permiten a la computadora comunicarse con el usuario, mostrando resultados o produciendo copias físicas.

9. **Unidades Ópticas**

Como las unidades de DVD o Blu-ray, son menos comunes hoy en día, pero pueden usarse para leer y escribir discos ópticos.

ISBN: 978-631-6557-57-5

10. **Tarjetas de Expansión**

Permiten añadir funcionalidades adicionales a la computadora, como tarjetas de sonido, tarjetas de red adicionales, tarjetas de captura de video, etc.

11. **Sistema de Enfriamiento**

Incluye ventiladores, disipadores de calor y, en algunos casos, sistemas de enfriamiento líquido para mantener los componentes a una temperatura operativa segura.

Estos componentes trabajan conjuntamente para permitir que una computadora funcione correctamente, permitiendo la ejecución de software, el procesamiento de datos, y la interacción con el usuario.

1.1.3 Sistemas de archivos

Los sistemas de archivos son componentes esenciales en los sistemas operativos que gestionan cómo se almacenan, organizan y acceden los datos en dispositivos de almacenamiento como discos duros, SSDs, memorias USB, entre otros. Cumplen varias funciones importantes:

Organización de Datos: Permiten estructurar los datos almacenados en directorios (también conocidos como carpetas) y archivos, facilitando su organización, búsqueda y acceso. Esto ayuda a los usuarios y programas a encontrar fácilmente los datos necesarios para su operación.

Gestión del Espacio de Almacenamiento: Controlan cómo se asigna el espacio en el dispositivo de almacenamiento a los archivos y directorios, optimizando el uso del espacio disponible y evitando su desperdicio. Esto incluye el seguimiento de qué sectores del disco están libres y cuáles están ocupados.

Metadatos: Los sistemas de archivos almacenan metadatos junto con los datos reales. Estos metadatos pueden incluir información sobre los propios archivos, como su tamaño, fecha de creación y modificación, permisos de acceso, y el propietario del archivo. Esta información es importante para la gestión de seguridad y para realizar operaciones como búsquedas y organización de archivos.

Seguridad y Acceso: Proporcionan mecanismos para controlar quién puede acceder a los datos y cómo pueden hacerlo. Esto incluye la implementación de permisos de archivo y directorio, que determinan qué usuarios o grupos de usuarios pueden leer, escribir o ejecutar un archivo.

ISBN: 978-631-6557-57-5

Integridad de Datos: Implementan características para asegurar la integridad de los datos, como la detección de errores y, en algunos casos, su corrección. Esto es vital para prevenir la pérdida o corrupción de datos debido a fallos del software, hardware o cortes de energía.

Compatibilidad: Los diferentes sistemas operativos suelen utilizar diferentes sistemas de archivos, lo que afecta a la compatibilidad entre dispositivos y plataformas. Algunos sistemas de archivos están diseñados para ser utilizados en entornos específicos, mientras que otros pueden ser más versátiles y compatibles con múltiples sistemas operativos.

Existen varios sistemas de archivos, cada uno con sus propias características, ventajas y limitaciones, diseñados para diferentes tipos de dispositivos de almacenamiento y usos. Elegir el sistema de archivos adecuado puede tener un impacto significativo en el rendimiento, la eficiencia y la seguridad de los datos almacenados. A continuación, se presenta algunos de los sistemas de archivos más comunes y sus particularidades:

NTFS (New Technology File System): Se utiliza predominantemente en el sistema operativo Windows. Tiene alta eficiencia en el manejo de archivos grandes y pequeños, es bueno en seguridad y soporte para características avanzadas. Su principal limitación es la compatibilidad con otros sistemas operativos sin software adicional.

FAT32 (File Allocation Table 32): Ampliamente usado en dispositivos extraíbles como memorias USB y tarjetas SD, y sistemas operativos incluyendo Windows, Mac OS y algunos sistemas en dispositivos embebidos.

ext4 (Fourth Extended Filesystem): Es el estándar para muchas distribuciones de Linux. Tiene soporte para archivos muy grandes y volúmenes de almacenamiento, asignación de bloques eficiente (lo que mejora el rendimiento y reduce la fragmentación), y registro de transacciones para mayor fiabilidad. La compatibilidad fuera del ecosistema Linux es su limitante puede requerir software especializado.

btrfs (B-Tree Filesystem): Se utiliza en sistemas Linux para servidores y estaciones de trabajo que necesitan capacidades avanzadas de almacenamiento. Incluye características como la deduplicación, compresión de datos en tiempo real, soporte para snapshots y clones, y una gran capacidad para corregir errores y recuperar datos.

ISBN: 978-631-6557-57-5

APFS (Apple File System): Introducido en macOS Sierra y utilizado en macOS, iOS, tvOS y watchOS. Optimizado para SSD, con soporte para cifrado nativo, snapshots, clonación de archivos y directorios, y mejor gestión del espacio. Tiene mejoras significativas en velocidad, eficiencia y fiabilidad en dispositivos de Apple.

HFS+ (Hierarchical File System Plus): Antes de la introducción de APFS, HFS+ era el sistema de archivos estándar en Mac OS. Soporte para archivos grandes y varios atributos de archivo, incluyendo tipos de archivo y creadores, además de compresión de archivos. Es menos eficiente que APFS, especialmente en SSDs, y sin algunas de las características avanzadas de APFS como la optimización para almacenamiento flash.

1.1.4 Interpretación de los datos por las computadoras

Las computadoras interpretan los datos a través de una serie de procesos digitales que transforman la información de entrada en salidas útiles, siguiendo instrucciones precisas proporcionadas por los programas de software. Este proceso se basa en el sistema binario y en la lógica computacional. Aquí hay una explicación paso a paso de cómo las computadoras interpretan los datos:

Sistema Binario

Las computadoras operan utilizando el sistema binario, que es un sistema numérico que utiliza dos dígitos, 0 y 1, conocidos como bits. Cada bit representa un estado de encendido (1) o apagado (0). Los datos de cualquier tipo (texto, imágenes, sonidos, etc.) se convierten en una secuencia binaria que la computadora puede procesar.

Unidades de Datos

Los bits se agrupan en unidades más grandes para su manejo y procesamiento:
Byte: Un grupo de 8 bits, que puede representar un carácter en un texto, por ejemplo.
Palabra: Una cantidad de bits procesados como una unidad por el procesador, que puede ser de 16, 32, 64 bits o más, dependiendo de la arquitectura de la computadora.

Codificación de Datos

Para interpretar los datos, las computadoras utilizan distintos esquemas de codificación:
ASCII: Para textos en inglés, usa 7 o 8 bits para representar cada carácter.
Unicode: Para una amplia variedad de caracteres de distintos idiomas, utilizando diferentes cantidades de bits.

ISBN: 978-631-6557-57-5

JPEG, MPEG: Para imágenes y videos, que codifican visual y auditivamente la información en formatos binarios.

MP3, WAV: Para audio, que codifican las ondas sonoras.

Procesamiento de Datos

Una vez que los datos están en forma binaria, la CPU realiza las siguientes operaciones:

Fetch (Búsqueda): Recupera las instrucciones y los datos de la memoria.

Decode (Decodificación): Decodifica las instrucciones para determinar qué operaciones se deben realizar.

Execute (Ejecución): Realiza las operaciones lógicas o aritméticas en la ALU, o maneja los datos según lo que se requiera.

Store (Almacenamiento): Guarda el resultado de nuevo en la memoria o lo envía a un dispositivo de salida.

Interpretación Lógica

La CPU interpreta las secuencias de instrucciones y datos utilizando lógica computacional. Los circuitos lógicos dentro de la CPU pueden realizar operaciones fundamentales como AND, OR, NOT, y XOR con los bits, lo que permite realizar cálculos y tomar decisiones basadas en condiciones lógicas (Ormaza, 2020).

Resultados

Después de procesar los datos, la computadora puede presentar los resultados al usuario a través de dispositivos de salida (pantallas, impresoras, altavoces) o transmitir la información procesada a otras computadoras o dispositivos.

1.1.5 Conversión de un numero decimal a binario

La conversión de un número decimal a binario se puede realizar utilizando el método de divisiones sucesivas. Este método implica dividir el número decimal entre 2 (la base del sistema binario) y anotar el residuo de cada división. El número binario se forma con estos residuos leídos en orden inverso, desde la última división hasta la primera. Vamos a convertir, por ejemplo, el número decimal 11 al sistema binario:

1. Dividir 11 para 2. El cociente producto de la división es 5 y el residuo es 1.

ISBN: 978-631-6557-57-5

2. Tomar el cociente de la división anterior (5) y se divide para 2. El cociente es 2 y el residuo es 1.
3. Tomas el cociente de la división anterior (2) y se divide para 2. El cociente es 1 y el residuo es 0.
4. Por último, divides 1 para 2. El cociente es 0 y el residuo es 1.

Ahora, se toma los residuos de cada división y se organiza en orden inverso, empezando por el último residuo obtenido hasta el primero: 1011.
Por lo tanto, el número 11 en decimal se convierte en 1011 en binario. Aquí está la representación detallada del proceso:

División	Cociente	Residuo
11/2	5	1
5/2	2	1
2/2	1	0
1/2	0	1

Leyendo los residuos de abajo hacia arriba (1011), se obtiene la representación binaria del número decimal 11.

1.1.6 Codificación ASCII

La codificación ASCII (American Standard Code for Information Interchange) (figura 1.3) es un estándar de codificación de caracteres mediante el cual se representa texto en las computadoras y otros dispositivos de comunicación. Utiliza números para representar letras, dígitos, caracteres de control y símbolos. En el estándar ASCII original, cada carácter se representa con un número de 7 bits, permitiendo 128 combinaciones posibles, que van desde 0 hasta 127.

Veamos un ejemplo práctico para ilustrar cómo funciona la codificación ASCII:

Imagina que quieres codificar la palabra "Hola" en ASCII. Cada letra de la palabra se representa con un número específico según la tabla ASCII:

H: 72

o: 111

ISBN: 978-631-6557-57-5

l: 108

a: 97

Por lo tanto, la palabra "Hola" se codificaría como una secuencia de números basada en la representación ASCII de cada carácter: 72 111 108 97.

El código ASCII

sigla en inglés de American Standard Code for Information Interchange
(Código Estadounidense Estándar para el Intercambio de Información)

Caracteres de control ASCII

DEC	HEX	Símbolo ASCII	
00	00h	NULL	(carácter nulo)
01	01h	SOH	(inicio encabezado)
02	02h	STX	(inicio texto)
03	03h	ETX	(fin de texto)
04	04h	EOT	(fin transmisión)
05	05h	ENQ	(enquiry)
06	06h	ACK	(acknowledgement)
07	07h	BEL	(timbre)
08	08h	BS	(retroceso)
09	09h	HT	(tab horizontal)
10	0Ah	LF	(salto de línea)
11	0Bh	VT	(tab vertical)
12	0Ch	FF	(form feed)
13	0Dh	CR	(retorno de carro)
14	0Eh	SO	(shift Out)
15	0Fh	SI	(shift In)
16	10h	DLE	(data link escape)
17	11h	DC1	(device control 1)
18	12h	DC2	(device control 2)
19	13h	DC3	(device control 3)
20	14h	DC4	(device control 4)
21	15h	NAK	(negative acknowle.)
22	16h	SYN	(synchronous idle)
23	17h	ETB	(end of trans. block)
24	18h	CAN	(cancel)
25	19h	EM	(end of medium)
26	1Ah	SUB	(substitute)
27	1Bh	ESC	(escape)
28	1Ch	FS	(file separator)
29	1Dh	GS	(group separator)
30	1Eh	RS	(record separator)
31	1Fh	US	(unit separator)
127	20h	DEL	(delete)

Caracteres ASCII imprimibles

DEC	HEX	Símbolo	DEC	HEX	Símbolo	DEC	HEX	Símbolo
32	20h	espacio	64	40h	@	96	60h	`
33	21h	!	65	41h	A	97	61h	a
34	22h	"	66	42h	B	98	62h	b
35	23h	#	67	43h	C	99	63h	c
36	24h	$	68	44h	D	100	64h	d
37	25h	%	69	45h	E	101	65h	e
38	26h	&	70	46h	F	102	66h	f
39	27h	'	71	47h	G	103	67h	g
40	28h	(	72	48h	H	104	68h	h
41	29h	)	73	49h	I	105	69h	i
42	2Ah	*	74	4Ah	J	106	6Ah	j
43	2Bh	+	75	4Bh	K	107	6Bh	k
44	2Ch	,	76	4Ch	L	108	6Ch	l
45	2Dh	-	77	4Dh	M	109	6Dh	m
46	2Eh	.	78	4Eh	N	110	6Eh	n
47	2Fh	/	79	4Fh	O	111	6Fh	o
48	30h	0	80	50h	P	112	70h	p
49	31h	1	81	51h	Q	113	71h	q
50	32h	2	82	52h	R	114	72h	r
51	33h	3	83	53h	S	115	73h	s
52	34h	4	84	54h	T	116	74h	t
53	35h	5	85	55h	U	117	75h	u
54	36h	6	86	56h	V	118	76h	v
55	37h	7	87	57h	W	119	77h	w
56	38h	8	88	58h	X	120	78h	x
57	39h	9	89	59h	Y	121	79h	y
58	3Ah	:	90	5Ah	Z	122	7Ah	z
59	3Bh	;	91	5Bh	[	123	7Bh	{
60	3Ch	<	92	5Ch	\	124	7Ch	|
61	3Dh	=	93	5Dh	]	125	7Dh	}
62	3Eh	>	94	5Eh	^	126	7Eh	~
63	3Fh	?	95	5Fh	_			

ASCII extendido

DEC	HEX	Símbolo	DEC	HEX	Símbolo	DEC	HEX	Símbolo	DEC	HEX	Símbolo
128	80h	Ç	160	A0h	á	192	C0h	└	224	E0h	Ó
129	81h	ü	161	A1h	í	193	C1h	┴	225	E1h	ß
130	82h	é	162	A2h	ó	194	C2h	┬	226	E2h	Ô
131	83h	â	163	A3h	ú	195	C3h	├	227	E3h	Ò
132	84h	ä	164	A4h	ñ	196	C4h	─	228	E4h	õ
133	85h	à	165	A5h	Ñ	197	C5h	┼	229	E5h	Õ
134	86h	å	166	A6h	ª	198	C6h	ã	230	E6h	µ
135	87h	ç	167	A7h	º	199	C7h	Ã	231	E7h	þ
136	88h	ê	168	A8h	¿	200	C8h	╚	232	E8h	Þ
137	89h	ë	169	A9h	®	201	C9h	╔	233	E9h	Ú
138	8Ah	è	170	AAh	¬	202	CAh	╩	234	EAh	Û
139	8Bh	ï	171	ABh	½	203	CBh	╦	235	EBh	Ù
140	8Ch	î	172	ACh	¼	204	CCh	╠	236	ECh	ý
141	8Dh	ì	173	ADh	¡	205	CDh	═	237	EDh	Ý
142	8Eh	Ä	174	AEh	«	206	CEh	╬	238	EEh	¯
143	8Fh	Å	175	AFh	»	207	CFh	¤	239	EFh	´
144	90h	É	176	B0h	░	208	D0h	ð	240	F0h	≡
145	91h	æ	177	B1h	▒	209	D1h	Ð	241	F1h	±
146	92h	Æ	178	B2h	▓	210	D2h	Ê	242	F2h	‗
147	93h	ô	179	B3h	│	211	D3h	Ë	243	F3h	¾
148	94h	ö	180	B4h	┤	212	D4h	È	244	F4h	¶
149	95h	ò	181	B5h	Á	213	D5h	ı	245	F5h	§
150	96h	û	182	B6h	Â	214	D6h	Í	246	F6h	÷
151	97h	ù	183	B7h	À	215	D7h	Î	247	F7h	¸
152	98h	ÿ	184	B8h	©	216	D8h	Ï	248	F8h	°
153	99h	Ö	185	B9h	╣	217	D9h	┘	249	F9h	¨
154	9Ah	Ü	186	BAh	║	218	DAh	┌	250	FAh	·
155	9Bh	ø	187	BBh	╗	219	DBh	▀	251	FBh	¹
156	9Ch	£	188	BCh	╝	220	DCh	▄	252	FCh	³
157	9Dh	Ø	189	BDh	¢	221	DDh	¦	253	FDh	²
158	9Eh	×	190	BEh	¥	222	DEh	Ì	254	FEh	■
159	9Fh	ƒ	191	BFh	┐	223	DFh	▀	255	FFh	

Figura 1.3 Código ASCII

Fuente: Enciclopedia concepto

Si quisieras convertir esta secuencia de números de nuevo a texto, simplemente buscarías cada número en la tabla ASCII para encontrar el carácter correspondiente y reconstruir la palabra original.

Es importante notar que ASCII incluye tanto caracteres imprimibles como caracteres de control (que no representan símbolos escritos sino acciones, como el salto de línea). Por ejemplo, el carácter ASCII de valor 10 representa un salto de línea (LF, Line Feed).

La codificación ASCII facilitó la estandarización de la representación de texto en los primeros años de la informática, aunque ha sido extendida y complementada por otros estándares, como Unicode, para abarcar una gama más amplia de caracteres de distintos idiomas y símbolos.

ISBN: 978-631-6557-57-5

1.1.7 Codificación Unicode

Unicode tiene capacidad para codificar más de 1,1 millones de puntos de código, abarcando caracteres de casi todos los sistemas de escritura existentes, además de una amplia gama de símbolos, emojis y otros caracteres especiales. La versión actual de Unicode puede asignar más de 140,000 caracteres.

Sin embargo, a continuación, se ofrece una visión general sobre cómo están organizados algunos rangos importantes dentro de la tabla de Unicode y dónde se puede encontrar recursos para explorarla en detalle.

Rangos Básicos de Unicode:

U+0000 a U+007F: El Plano Básico Multilingüe (BMP), que incluye los caracteres del estándar ASCII, caracteres latinos adicionales, y caracteres de muchos otros alfabetos como el griego, cirílico, hebreo, árabe, entre otros.

U+0080 a U+07FF: Incluye caracteres adicionales para alfabetos de otros idiomas, incluyendo los necesarios para representar la mayoría de las lenguas vivas.

U+0800 a U+FFFF: Este rango incluye símbolos matemáticos, diacríticos, y caracteres de escritura no latina como el tailandés y el lao.

U+10000 a U+1FFFF: Aquí se encuentran los caracteres suplementarios, incluidos los símbolos históricos, musicales, matemáticos, y los emojis.

Ejemplos de Caracteres Unicode

Letras latinas: U+0041 (A), U+0062 (b)

Cifras arábigas: U+0030 (0), U+0039 (9)

Símbolos matemáticos: U+2200 (∀, "para todo")

Emojis: U+1F600 (😀, cara sonriente)

Para explorar la tabla de Unicode y buscar caracteres específicos, puedes utilizar varios recursos en línea. Algunas opciones recomendadas incluyen:

Unicode Consortium: El sitio web oficial de Unicode (unicode.org) proporciona la especificación completa, tablas de caracteres, y herramientas de búsqueda.

FileFormat.info: Ofrece una herramienta de búsqueda de caracteres Unicode donde puedes encontrar información detallada sobre cada punto de código (www.fileformat.info).

ISBN: 978-631-6557-57-5

Dado que Unicode se actualiza regularmente para incluir nuevos caracteres y símbolos, estos recursos en línea son la mejor manera de obtener información actualizada y detallada sobre la codificación de caracteres específicos.

Supongamos que quieres representar el texto "Hola 👋" (que incluye un emoji) utilizando Unicode (Figura 1.4) en el esquema de codificación UTF-8:

H: U+0048

o: U+006F

l: U+006C

a: U+0061

Espacio: U+0020

Emoji 👋 (Mano saludando): U+1F44B

Figura 1.4 Código Unicode Letras Latinas

Fuente: Kreative Entertainment

1.1.8 Codificación de colores

Los colores en el ámbito digital se codifican utilizando diferentes modelos de color, entre los que los más comunes son el RGB (Red, Green, Blue - Rojo, Verde, Azul) y el HEX (Hexadecimal). Estos modelos permiten representar una amplia gama de colores mediante la combinación de valores para cada uno de los colores primarios en el caso del RGB, o mediante una cadena de caracteres en base hexadecimal en el caso del HEX. Aquí te explico cada uno:

ISBN: 978-631-6557-57-5

Modelo RGB

En el modelo RGB (Figura 1.4), cada color se representa mediante una combinación de los tres colores primarios de luz (rojo, verde y azul). Cada uno de estos colores puede tener un valor en el rango de 0 a 255, donde 0 significa ninguna intensidad del color y 255 representa la máxima intensidad. Por ejemplo:

-Blanco: RGB(255, 255, 255) - La combinación de la máxima intensidad de rojo, verde y azul resulta en blanco.

- Negro: RGB(0, 0, 0) - La ausencia de los tres colores resulta en negro.

- Rojo: RGB(255, 0, 0) - Máxima intensidad de rojo con ausencia de verde y azul.

Modelo HEX

El modelo HEX (Figura 1.5) utiliza una representación hexadecimal para indicar los valores de los colores. Consiste en un símbolo `#` seguido de seis dígitos, donde los dos primeros dígitos representan el rojo, los dos siguientes el verde y los dos últimos el azul. Cada par de dígitos está en el rango de `00` a `FF`, que es equivalente al rango de 0 a 255 en decimal. Ejemplos:

- Blanco: #FFFFFF - Máxima intensidad de rojo, verde y azul en hexadecimal.

- Negro: #000000 - Ausencia de rojo, verde y azul.

- Rojo: #FF0000 - Máxima intensidad de rojo con ausencia de verde y azul.

Color 0:	[Color 0]	#FFFFFF	255	255	255
Color 1:	[Color 1]	#000000	0	0	0
Color 2:		#FFFFFF	255	255	255
Color 3:	[Color 3]	#FF0000	255	0	0
Color 4:	[Color 4]	#00FF00	0	255	0
Color 5:	[Color 5]	#0000FF	0	0	255
Color 6:	[Color 6]	#FFFF00	255	255	0
Color 7:	[Color 7]	#FF00FF	255	0	255
Color 8:	[Color 8]	#00FFFF	0	255	255
Color 9:	[Color 9]	#800000	128	0	0
Color 10:	[Color 10]	#008000	0	128	0
Color 11:	[Color 11]	#000080	0	0	128
Color 12:	[Color 12]	#808000	128	128	0
Color 13:	[Color 13]	#800080	128	0	128
Color 14:	[Color 14]	#008080	0	128	128
Color 15:	[Color 15]	#C0C0C0	192	192	192

Figura 1.5 Modelo HEX y Modelo RGB

Fuente: Blog de Javier

Otros Modelos

ISBN: 978-631-6557-57-5

Además de RGB y HEX, existen otros modelos de color como CMYK (Cyan, Magenta, Yellow, Key/Black - usado en la impresión), HSL (Hue, Saturation, Lightness - Tono, Saturación, Luminosidad) y otros, diseñados para propósitos específicos. Cada uno de estos modelos tiene sus propias aplicaciones y ventajas. RGB y HEX son ampliamente utilizados en pantallas digitales y diseño web debido a su simplicidad y compatibilidad universal con dispositivos electrónicos y navegadores web.

1.2 Aplicaciones informáticas

Las aplicaciones informáticas son esenciales en el mundo moderno, debido a que permiten a usuarios y computadoras interactuar para realizar una amplia gama de tareas. Estos programas informáticos se diseñan para cumplir propósitos específicos y pueden variar desde aplicaciones simples hasta suites complejas de software que abarcan múltiples funciones.

Las aplicaciones informáticas han evolucionado significativamente desde las herramientas básicas de procesamiento de texto y hojas de cálculo hasta sistemas avanzados que incorporan inteligencia artificial, aprendizaje automático y análisis de grandes volúmenes de datos. Esta evolución ha permitido el desarrollo de aplicaciones especializadas en distintos campos, desde la gestión empresarial hasta la investigación científica, pasando por el entretenimiento y la educación.

a) En la Vida Cotidiana

Las aplicaciones móviles han transformado las rutinas diarias, facilitando desde la gestión de finanzas personales hasta la comunicación instantánea y el acceso a entretenimiento bajo demanda. La integración de aplicaciones con dispositivos portátiles y el Internet de las Cosas (IoT) ha extendido aún más su impacto, permitiendo monitorizar la salud, controlar dispositivos domésticos inteligentes y mejorar la seguridad personal y del hogar.

b) En el Trabajo Remoto y la Colaboración

Aplicaciones de software como plataformas de colaboración en línea, herramientas de gestión de proyectos y sistemas de videoconferencia han sido importantes para el auge del trabajo remoto. Facilitan la comunicación entre equipos distribuidos geográficamente, permiten compartir y editar documentos en tiempo real y ayudan a mantener la productividad sin importar la ubicación de los empleados.

ISBN: 978-631-6557-57-5

c) Aplicaciones Verticales y Horizontales

La distinción entre aplicaciones verticales, diseñadas para necesidades específicas de sectores o profesiones, y aplicaciones horizontales, útiles para un amplio espectro de usuarios, resalta la capacidad de la tecnología informática para adaptarse a distintas necesidades y objetivos. Esto demuestra la flexibilidad y capacidad de personalización del software, permitiendo que empresas y profesionales elijan herramientas que se ajusten precisamente a sus requisitos.

d) Impacto en la Educación

Las aplicaciones educativas han revolucionado el aprendizaje, ofreciendo recursos interactivos, acceso a bibliotecas digitales, plataformas de cursos en línea y herramientas de simulación. Estas aplicaciones no solo complementan la enseñanza tradicional, sino que también abren nuevas vías para el autoaprendizaje y la educación a distancia, democratizando el acceso al conocimiento.

e) Avances en el Comercio Electrónico

El comercio electrónico se ha beneficiado enormemente de las aplicaciones informáticas, desde sistemas de gestión de inventarios hasta plataformas de venta en línea y aplicaciones móviles para compras. La integración de análisis de datos y personalización mediante aplicaciones ha permitido a las empresas entender mejor a sus clientes y ofrecer experiencias de compra más satisfactorias.

f) Contribuciones a la Ciencia y la Medicina

En el ámbito científico y médico, las aplicaciones informáticas han permitido avances significativos, desde el procesamiento de grandes conjuntos de datos en investigaciones hasta la gestión de historiales clínicos electrónicos y el desarrollo de tecnologías de diagnóstico y tratamiento. La telemedicina y las publicaciones electrónicas biomédicas son ejemplos de cómo las aplicaciones informáticas están facilitando el acceso a servicios de salud y a la información científica.

g) Aplicaciones Informáticas más utilizadas

Las aplicaciones informáticas más utilizadas en el mundo varían ampliamente según el contexto, incluyendo ámbitos como la productividad personal y profesional, comunicación, entretenimiento, y educación. A continuación, se detallan algunas de las categorías y aplicaciones más destacadas globalmente:

ISBN: 978-631-6557-57-5

1) **Productividad y Ofimática**

Microsoft Office: Incluye Word, Excel, PowerPoint, y Outlook, entre otros. Es ampliamente utilizado en entornos profesionales y educativos para la creación de documentos, hojas de cálculo, presentaciones y gestión de correo electrónico.

Google Workspace: Ofrece Google Docs, Sheets, Slides, y Gmail, proporcionando herramientas colaborativas basadas en la nube.

LibreOffice: Una suite de ofimática libre y gratuita que incluye procesador de textos, hojas de cálculo, y herramientas de presentación.

2) **Navegadores Web**

Google Chrome: Es el navegador web más utilizado, conocido por su velocidad y extensiones.

Mozilla Firefox: Destacado por su enfoque en la privacidad y personalización.

Safari: Predominantemente usado en dispositivos Apple, valorado por su integración con el ecosistema de Apple.

Microsoft Edge: El navegador sucesor de Internet Explorer, integrado en Windows y enfocado en la seguridad.

3) **Comunicación**

WhatsApp y Telegram: Aplicaciones de mensajería instantánea populares para la comunicación personal y grupal.

Zoom y Microsoft Teams: Plataformas de videoconferencia que se han vuelto esenciales para el trabajo remoto, la educación a distancia, y reuniones virtuales.

Slack: Utilizado en entornos profesionales para la comunicación y colaboración en equipo.

4) **Redes Sociales**

Facebook: Una de las redes sociales más grandes, utilizada para conectar con amigos, familia y comunidades.

Instagram: Popular por compartir fotos y videos, con un fuerte enfoque en la estética visual.

X: Plataforma de microblogging utilizada para compartir actualizaciones breves y seguir las noticias en tiempo real.

LinkedIn: Red profesional para compartir el currículum, buscar empleo y establecer

ISBN: 978-631-6557-57-5

conexiones profesionales.

5) **Entretenimiento y Multimedia**

Spotify y Apple Music: Servicios de streaming de música con vastos catálogos.

Netflix y Disney+: Plataformas de streaming de video que ofrecen películas, series, y documentales.

YouTube: El sitio web de compartición de videos más grande, utilizado para entretenimiento, educación y marketing.

6) **Herramientas de Desarrollo**

Visual Studio Code y Sublime Text: Editores de código populares entre desarrolladores.

Git: Sistema de control de versiones utilizado para el desarrollo colaborativo de software.

Docker: Plataforma para desarrollar, entregar y ejecutar aplicaciones en contenedores.

7) **Seguridad y Mantenimiento**

Antivirus como Avast, Norton, y Kaspersky: Ofrecen protección contra malware y amenazas a la seguridad.

CCleaner: Herramienta para optimizar el sistema y limpiar archivos innecesarios.

Estas aplicaciones demuestran la diversidad y amplitud de las herramientas informáticas disponibles para satisfacer las necesidades diarias de comunicación, trabajo, entretenimiento, y gestión de la información de personas alrededor del mundo. La popularidad de estas aplicaciones varía según las tendencias, necesidades específicas de los usuarios, y la aparición de nuevas tecnologías.

1.3 Contexto social de la Informática

El contexto social de la informática se relaciona a cómo esta disciplina interactúa con la sociedad en general, incluyendo su impacto en la vida cotidiana, la economía, la cultura, y la organización social. La informática, siendo la ciencia que estudia el tratamiento automático de la información por medio de computadoras, tiene un papel fundamental en el desarrollo y transformación de las sociedades modernas. A continuación, detallaremos algunos aspectos clave del contexto social de la informática:

1) **Acceso y Brecha Digital**

La informática ha cambiado la manera en que accedemos a la información y nos

ISBN: 978-631-6557-57-5

comunicamos. Sin embargo, esto también ha dado lugar a la brecha digital, una desigualdad significativa en el acceso y uso de tecnologías de información y comunicación entre diferentes grupos sociales, económicos, y geográficos. Mientras algunas personas y comunidades se benefician enormemente de la tecnología, otras quedan rezagadas.

En varios países, se han lanzado programas de inclusión tecnológica para distribuir dispositivos electrónicos, como tabletas y computadoras, en escuelas de zonas rurales o desfavorecidas para mejorar el acceso a la tecnología y reducir la brecha digital.

En muchas áreas rurales y comunidades de bajos ingresos, el acceso a internet de alta velocidad es limitado o inexistente existiendo desigualdad con las ciudades, lo que afecta la educación, las oportunidades de empleo y el acceso a servicios.

2) Transformación Laboral y Económica

La informática ha sido un motor de transformación en el ámbito laboral y económico, automatizando procesos que antes requerían intervención humana y creando nuevos tipos de empleo, especialmente en el sector de tecnología de información. Sin embargo, también ha llevado a la obsolescencia de ciertos trabajos y a la necesidad de una constante actualización de habilidades por parte de los trabajadores.

La automatización de la manufactura con la introducción de robots y sistemas informáticos en las líneas de producción ha aumentado la eficiencia, pero también ha reducido la necesidad de mano de obra en ciertos sectores.

Plataformas como Uber o Airbnb, basadas en tecnología informática, han creado nuevas formas de trabajo autónomo, aunque también han generado debates sobre seguridad laboral y derechos de los trabajadores.

3) Cambios en la Comunicación y las Relaciones Sociales

Las tecnologías informáticas, especialmente internet y las redes sociales, han revolucionado la manera en que las personas se comunican, interactúan, y forman comunidades. Esto ha tenido efectos tanto positivos, como el fácil acceso a información y la capacidad de conectar con personas alrededor del mundo, como negativos, incluyendo la desinformación y el aislamiento social.

Plataformas como Facebook, Twitter e Instagram han cambiado cómo las

ISBN: 978-631-6557-57-5

personas se comunican, permitiendo la rápida difusión de información, pero también presentando desafíos como la propagación de noticias falsas.

La informática ha posibilitado el teletrabajo, lo que permite a las personas trabajar desde cualquier lugar, afectando el equilibrio entre la vida laboral y personal.

4) Privacidad y Seguridad

La informática también ha planteado desafíos significativos en términos de privacidad y seguridad de la información. La recolección y análisis de grandes volúmenes de datos personales por empresas y gobiernos plantean preocupaciones sobre la privacidad y el uso indebido de esta información.

Casos como el escándalo de Cambridge Analytica, donde se recolectaron sin autorización datos de millones de usuarios de Facebook, resaltan las preocupaciones sobre la privacidad en la era digital.

Ataques de ransomware (tipo de malware que bloquea el acceso a tu información personal o dispositivo y exige el pago de un rescate para recuperarlo) a hospitales y ciudades muestran la vulnerabilidad de las instituciones públicas y privadas ante los ciberdelincuentes, resaltando la importancia de la seguridad informática.

5) Impacto en la Educación

La educación se ha transformado considerablemente con la integración de la informática en el proceso de enseñanza y aprendizaje. Esto incluye desde el uso de computadoras y software educativo en las aulas hasta la educación en línea, que permite el acceso a recursos y conocimientos sin las limitaciones geográficas tradicionales.

Plataformas como Coursera o edX ofrecen acceso a cursos de universidades de prestigio a nivel mundial, democratizando el acceso a la educación de alta calidad.

Software que se ajusta al nivel de conocimiento y aprendizaje del estudiante, personalizando la experiencia educativa y potenciando el aprendizaje individual.

6) Ética y Responsabilidad Social

Finalmente, el contexto social de la informática también engloba cuestiones éticas y de responsabilidad social, como el desarrollo de tecnología de manera sostenible, el respeto por la privacidad, y la inclusión de consideraciones éticas en el diseño de sistemas informáticos.

Iniciativas como el proyecto AI4People buscan establecer marcos éticos para el

ISBN: 978-631-6557-57-5

desarrollo y uso de la inteligencia artificial, promoviendo tecnologías que respeten los derechos humanos y la dignidad.

La inclusión de características de accesibilidad en el diseño de sitios web y aplicaciones, asegurando que personas con discapacidad puedan utilizarlos, refleja una preocupación por la inclusión y la equidad.

1.4 Datos, Expresiones y funciones

1.4.1 Representación externa e interna de los datos

En el contexto de la informática, la representación de los datos puede verse desde dos perspectivas: la externa y la interna. *Externamente, los datos se manifiestan a través de caracteres alfanuméricos, signos, imágenes, sonidos* y más, lo cual facilita la interacción humana con la información.

Por otro lado, cuando estos datos se procesan dentro de un ordenador, adoptan una forma distinta. *Internamente*, un computador opera con una representación binaria, es decir, *todo dato e información se codifica como una secuencia de bits*, *que son dígitos binarios compuestos exclusivamente por ceros (0) y unos (1)*. Esta representación binaria es fundamental, debido a que se ajusta a la arquitectura eléctrica subyacente de las computadoras, donde los dos estados pueden representar encendido (1) o apagado (0).

1.4.2 Tipos de datos

Los tipos de datos son fundamentales en la programación debido a que definen la naturaleza de la información que puede ser procesada por un ordenador (Cerrada & Collado, 2010). Esta clasificación no solo dicta cómo se almacenará y representará la información internamente en la máquina, sino también qué operaciones son posibles realizar con ella.

Hay dos categorías generales de tipos de datos: simples y compuestos. Los tipos *simples*, también conocidos como primitivos, son aquellos que no *están compuestos por otros datos y representan una única información*. Por ejemplo, un número sin decimales se almacena como un dato de *tipo entero*, mientras que un número con decimales se maneja como un dato de *tipo real*. Los tipos de datos *lógicos*, conocidos como *booleanos*, tienen solo dos valores posibles: verdadero o falso, que son especialmente útiles en estructuras de control y decisiones lógicas. El tipo *caracter*, por otro lado, puede referirse a un solo símbolo o letra, mientras que una *cadena* (string) es una secuencia de caracteres

ISBN: 978-631-6557-57-5

que comúnmente se utiliza para representar texto.

En contraposición, los tipos de datos *compuestos* o estructurados *permiten agrupar varios valores, que pueden ser de tipos simples o incluso otros compuestos, para formar una estructura más compleja*. Un ejemplo común de un tipo compuesto es el vector, que puede almacenar una serie de elementos del mismo tipo, como un arreglo de números enteros.

Además de estos, existen otros tipos de datos más complejos que son construidos por los programadores de acuerdo con las necesidades específicas de su software, como son las *estructuras* (structs), que son agrupaciones de tipos simples y compuestos bajo un mismo nombre, y las *clases*, que son la base de la programación orientada a objetos y permiten encapsular datos y comportamientos asociados a estos datos (Shehzad, 2024).

Estas herramientas ofrecen a los programadores la flexibilidad para crear programas eficientes y manejar la información de manera efectiva, permitiendo así que las computadoras realicen tareas complejas y sean una herramienta poderosa en una multitud de aplicaciones.

1.4.3 Tipos de datos numéricos

Los tipos de datos numéricos permiten realizar cálculos y representar cantidades en una amplia gama de aplicaciones. Dentro de los tipos numéricos, encontramos principalmente dos categorías: los enteros y los reales.

El *tipo de dato entero*, comúnmente referido en la programación como `integer` o `int`, abarca números sin fracciones ni decimales y puede incluir tanto valores positivos como negativos. Son la representación de los números enteros en la matemática y su rango está definido por la arquitectura del sistema computacional (por ejemplo, 32-bit o 64-bit) (Cerrada & Collado, 2010). Ejemplos comunes de enteros serían -100, 0, 25 o 2048. Los enteros se usan típicamente cuando se necesitan contar elementos discretos, como usuarios, productos en un inventario o pasos en un algoritmo.

Por otro lado, el *tipo de dato real*, conocido en muchos lenguajes de programación como `real`, `float` o `double`, comprende los números que poseen una parte fraccional, representada por un punto decimal. Esto les permite expresar cantidades más precisas y modelar situaciones que requieren granularidad, como mediciones científicas, transacciones financieras, donde la precisión en las medidas es determinante como la

ISBN: 978-631-6557-57-5

ingeniería, física y economía,. Algunos ejemplos de números reales son 3.1416, -0.001 o 100.25.

1.4.4 Tipo de dato carácter

El tipo de dato *carácter* permite la manipulación de los elementos más básicos de la escritura: los caracteres individuales. Estos pueden ser letras del alfabeto, en mayúsculas o minúsculas, dígitos numéricos del 0 al 9 y una variedad de caracteres especiales que incluyen signos de puntuación y símbolos matemáticos, entre otros.

Un dato de tipo carácter usualmente es representado en un lenguaje de programación como un valor individual encerrado entre comillas simples o dobles, dependiendo del lenguaje (por ejemplo, 'A', 'b', “3”, “$”) (Easttom, 2006). En el contexto de la codificación de caracteres, la representación más común es el estándar ASCII, aunque en ambientes modernos y más internacionales se tiende a usar UTF-8, que incluye un conjunto más amplio de caracteres de múltiples idiomas y símbolos.

Por otro lado, una *cadena* o string es una secuencia de caracteres que trabajan como un colectivo. Es decir, mientras un carácter es un elemento singular ('A'), una cadena es un conjunto de estos elementos formando un texto ('ABCD'). En programación, las cadenas se utilizan para almacenar y manipular texto, como nombres, oraciones y párrafos. La longitud de una cadena se determina por la cantidad de caracteres que contiene, y al igual que los caracteres individuales, se encuentra delimitada por comillas en la mayoría de los lenguajes de programación (Nair, 2009).

La habilidad de trabajar con cadenas es fundamental para el procesamiento de texto, como en la generación de informes, la interacción con usuarios a través de interfaces, o la manipulación de datos en formatos de texto como XML o JSON. Por ejemplo, la cadena "Fundamentos de Programación" tiene una longitud de 27 caracteres, incluyendo los espacios.

1.4.5 Constantes y variables

Una *constante* es un valor que no cambia a lo largo del tiempo durante la ejecución de un programa. Una vez que se le asigna un valor a una constante al inicio, este valor permanecerá inalterable, lo cual es útil para representar datos que no deben modificarse, como el valor de pi (π), la tasa de conversión de monedas fijas, o el número de días en una semana.

ISBN: 978-631-6557-57-5

En contraste, una *variable* es un espacio en la memoria del ordenador donde se puede almacenar un dato que puede cambiar durante la ejecución del programa. Las variables son identificadas por un nombre único y se les asigna un tipo de dato específico que define qué tipo de valores pueden almacenar y qué operaciones se pueden realizar con ellas.

Las variables se utilizan de diversas maneras en un programa:

1. Guardar datos y estados: Las variables permiten mantener y actualizar información a medida que el programa opera, como el puntaje en un juego o la entrada del usuario.
2. Asignar valores de una variable a otra: Esta característica se usa para transferir valores entre variables, realizar cálculos o modificar el estado del programa.
3. Representar valores dentro de una expresión matemática: Las variables se utilizan en operaciones matemáticas para calcular nuevos valores y realizar lógica de decisión.
4. Mostrar valores por pantalla: Se emplean para presentar resultados al usuario, como mensajes de bienvenida, resultados de cálculos, o datos recuperados de una base de datos.

El uso adecuado de constantes y variables es importante para escribir programas claros y mantenibles. Las constantes proporcionan puntos de referencia fijos que facilitan la comprensión del código, mientras que las variables proporcionan la flexibilidad necesaria para manejar datos dinámicos y adaptarse a la entrada del usuario o a cambios en el entorno del programa.

1.4.6 Operación de asignación

La operación de asignación es fundamental en cualquier lenguaje de programación porque permite guardar o actualizar valores en las variables. Esta acción se denota comúnmente con el signo igual (`=`), aunque conceptualmente se representa como una flecha (←) en pseudocódigo o algoritmos para distinguir claramente la asignación de la comparación de igualdad, que también utiliza el símbolo `=` en la mayoría de los lenguajes (Mejia et al., 2024).

Cuando asignamos un valor a una variable, cualquier valor que tenía previamente se sobrescribe, lo que significa que se pierde y es reemplazado por el nuevo. Por eso se dice que la asignación es "destructiva" en el sentido de que el estado anterior de la variable se elimina.

Por ejemplo, si tenemos la variable `edad`, y se quiere asignarle un valor inicial

ISBN: 978-631-6557-57-5

de 30, se escribe en la mayoría de los lenguajes de programación de la siguiente manera:

edad = 30

En pseudocódigo sería:

edad ← 30

Si después, en otra parte del programa, se decide actualizar esa edad a 31, realizaríamos otra asignación:

edad = 31

En pseudocódigo sería:

edad ← 31

Después de esta segunda asignación, el valor de `edad` que era 30 se pierde y se actualiza a 31. La operación de asignación se considera una instrucción o sentencia dentro de un programa, y es una de las acciones más básicas y utilizadas en la programación.

1.4.7 Operación de entrada salida

Las operaciones de entrada y salida permiten la interacción entre el usuario y el programa. La operación de entrada (leer) se utiliza para recibir datos del exterior, como puede ser a través de un teclado o un archivo. Por otro lado, la operación de salida (escribir) se emplea para presentar resultados al usuario, ya sea en la pantalla, mediante una impresión en papel o en un archivo.

Por ejemplo, si se desea pedir al usuario su edad y luego mostrarla en pantalla, el pseudocódigo sería algo como esto:

```
INICIO
   // Pedir al usuario que ingrese su edad
   ESCRIBIR "Ingresa tu edad:"
   LEER edad

   // Mostrar la edad ingresada
   ESCRIBIR "Tu edad es: ", edad
FIN
```

En este ejemplo, `LEER` es la operación de entrada que permite al usuario ingresar un valor que será almacenado en la variable `edad`. La instrucción `ESCRIBIR` es la operación de salida que muestra en pantalla la frase "Tu edad es: " seguida del valor

ISBN: 978-631-6557-57-5

de la variable “edad”.

1.4.8 Lenguajes de Programación

En el transcurso de este texto, nos centraremos en el uso y análisis de varios lenguajes de programación, específicamente Python, Java, C++ y JavaScript. Además, se realiza una comparativa detallada basada en el código del ejemplo en Python con los otros lenguajes de programación.

a) Instalación y utilización de Python

Python es un lenguaje de programación de alto nivel, interpretado y de propósito general, conocido por su legibilidad y sintaxis clara, que facilita tanto el aprendizaje para los programadores principiantes como el desarrollo rápido para los experimentados. Fue creado por Guido van Rossum y su primera publicación fue en 1991 (Meyers, 2023). Python soporta varios paradigmas de programación, incluidos la programación orientada a objetos, imperativa y, en menor medida, funcional.

Las variables en Python no necesitan ser declaradas explícitamente, lo que significa que el tipo de una variable se determina en tiempo de ejecución. Esto hace que el lenguaje sea flexible y fácil de usar, aunque requiere precaución para evitar errores de tipo. Python maneja automáticamente la memoria, lo que significa que los objetos son eliminados o "recolectados" cuando ya no son necesarios, liberando así los recursos sin intervención del programador (Bhimavarapu & Hemanth, 2023).

Python es utilizado ampliamente en diferentes campos y aplicaciones, tales como: desarrollo web, ciencia de datos, análisis de datos, inteligencia artificial, machine learning, escribir scripts para automatización, desarrollo de aplicaciones de escritorio con interfaces gráficas de usuario (Gold, 2024).

Vamos a tener nuestro primer contacto con Python, para lo cual en el navegador búsque **Google Colab** y asomará la siguiente pantalla y dar click en “Te damos la bienvenida a Colaboratory”.

ISBN: 978-631-6557-57-5

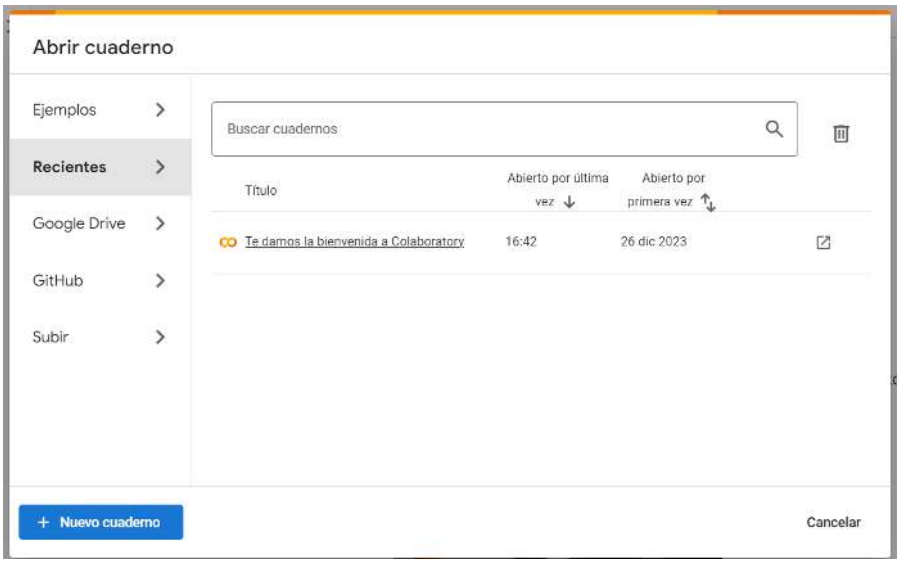


Figura 1.5 Cuaderno de Google Colab

Fuente: Google Colab

Aparecerá la siguiente pantalla donde se debe ubicar el código.

Figura 1.6 Lugar donde se escribe los programas

Fuente: Google Colab

El ejemplo que se mencionó en pseudocódigo se expresa de la siguiente manera en Python:

```
#Pedir al usuario que ingrese su edad
edad = input("Ingresa tu edad:")
#Mostrar la edad ingresada
print("Tu edad es: ", edad)
```

ISBN: 978-631-6557-57-5

En la flecha de la izquiera que esta en el circulo negro dar click para que se ejecute el programa.

```
#Pedir al usuario que ingrese su edad
edad = input("Ingresa tu edad:")
#Mostrar la edad ingresada
print("Tu edad es: ", edad)

Ingresa tu edad:
```

Introduce tu edad y luego pulsar la flecha enter.

```
#Pedir al usuario que ingrese su edad
edad = input("Ingresa tu edad:")
#Mostrar la edad ingresada
print("Tu edad es: ", edad)

Ingresa tu edad:30
Tu edad es:  30
```

Aquí **input** es la función de entrada que captura lo que el usuario digita y lo guarda en la variable **edad**. La función **print** es la operación de salida que imprime el texto "Tu edad es:" seguido del valor que el usuario introdujo, que ahora está almacenado en la variable edad.

a) Instalación de BlueJ y uso de Java

BlueJ es un entorno de desarrollo integrado (IDE, por sus siglas en inglés) diseñado específicamente para la enseñanza y el aprendizaje del lenguaje de programación Java. Es desarrollado por la Universidad de Kent en conjunto con la Universidad Deakin. Lo que hace único a BlueJ es su enfoque educativo, ofreciendo una interfaz gráfica simple y fácil de usar que facilita a los principiantes el aprendizaje de conceptos fundamentales de la programación orientada a objetos.

Una de las características destacadas de BlueJ es su representación visual de las clases y objetos, permitiendo a los usuarios interactuar directamente con la estructura de sus programas a través de diagramas. Esto ayuda a los estudiantes a comprender mejor la relación entre las clases y los objetos, así como los principios de la herencia y el polimorfismo.

ISBN: 978-631-6557-57-5

BlueJ ofrece funcionalidades como la creación, edición, compilación y ejecución de programas Java dentro de un entorno controlado y simplificado. Además, integra herramientas para el testing unitario y la depuración, lo que facilita la identificación y corrección de errores en el código.

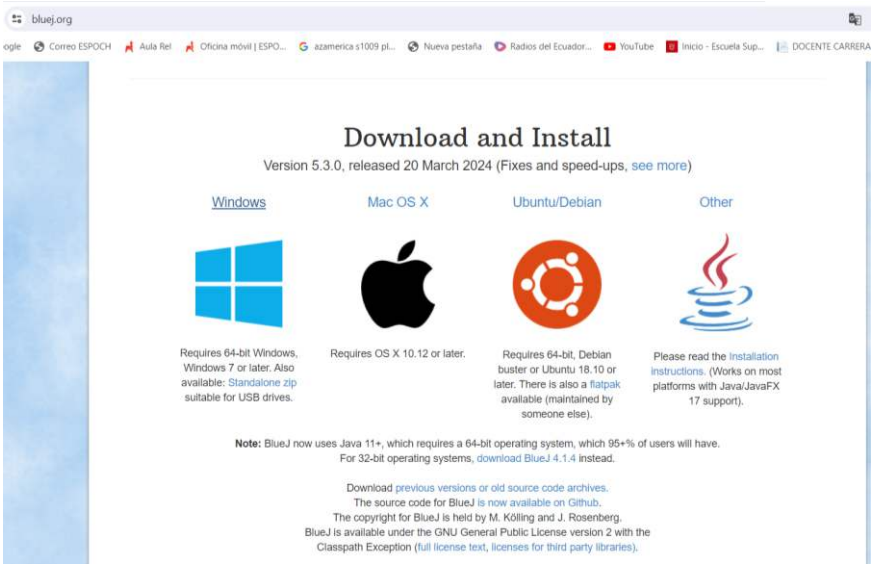


Figura 1.7 Obtener BlueJ

Fuente: BlueJ.org

Es ampliamente utilizado en entornos educativos, es gratuito y está disponible para varias plataformas (Figura 1.7), incluidas Windows, macOS y Linux.

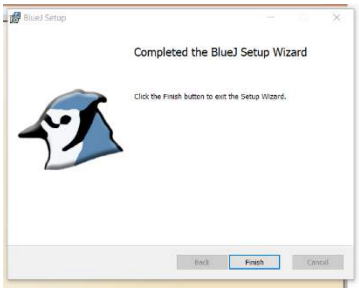


Figura 1.8 Instalación de BlueJ

Fuente: BlueJ

ISBN: 978-631-6557-57-5

Después de obtener de la página oficial www.bluej.org, proceda con la instalación (Figura 1.8)

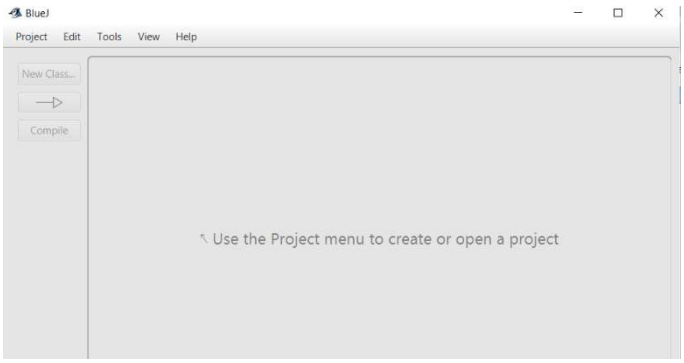


Figura 1.9 Abrir de BlueJ

Fuente: BlueJ

Proceda a abrir BlueJ desde la ventana inferior izquierda si esta en Windows, escribiendo BlueJ y le aparecerá la ventana que muestra la figura 1.9.

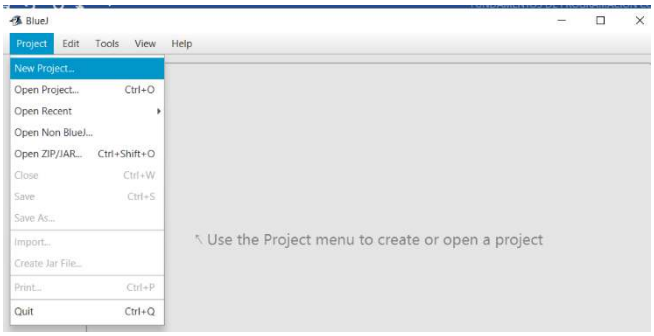


Figura 1.10 Crear un nuevo proyecto

Fuente: BlueJ

Proceda a crear un nuevo proyecto (Figura 1.10) ingresando a Proyect/NewProyect.

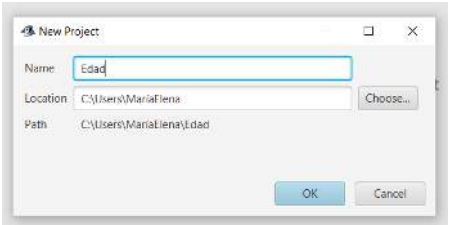


Figura 1.11 Nombre del proyecto

Fuente: BlueJ

ISBN: 978-631-6557-57-5

Se procede a poner un nombre al proyecto y ha escoger la ubicación donde se guardará (Figura 1.11).

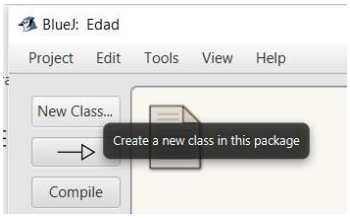


Figura 1.12 Nueva Clase

Fuente: BlueJ

Dar click en New Class para que se cree la clase (Figura 1.12)

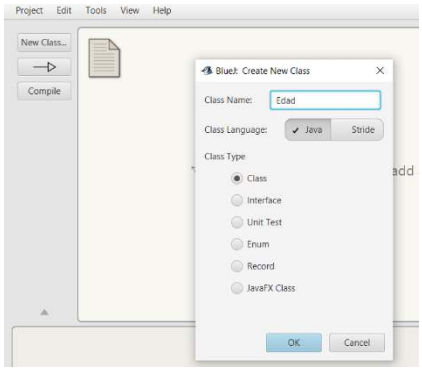


Figura 1.13 Nombre de la Clase

Fuente: BlueJ

Dar click en la forma de hoja que asoma en la ventana y ubicar el nombre de la clase.

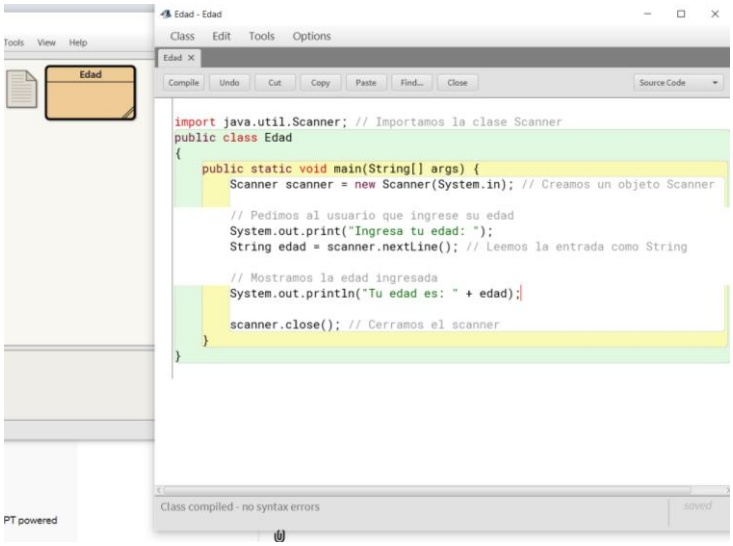


Figura 1.14 Código

Elaborado por los Autores

ISBN: 978-631-6557-57-5

Proceda a ubicar el código y luego escoja el ícono compile (Figura 1.14) para proceder a compilar la clase y ver que se ejecute correctamente el programa.

A continuación se explica línea por línea el código en Java:

```
import java.util.Scanner; // Importamos la clase Scanner
```

Esta línea importa la clase Scanner del paquete java.util, que es necesaria para leer la entrada del usuario desde la consola (Friesen, 2024).

```
public class Edad {
```

Define una clase pública llamada Edad. En Java, cada programa debe tener al menos una clase, y en este caso, Edad es la clase principal donde se ejecuta el programa.

```
  public static void main(String[] args) {
```

Esta línea define el método main, que es el punto de entrada de cualquier programa en Java. Los argumentos del método (String[] args) pueden usarse para recibir argumentos de línea de comandos (Learning, 2000).

```
    Scanner scanner = new Scanner(System.in); // Creamos un objeto Scanner
```

Aquí se crea una instancia de la clase `Scanner`, llamada `scanner`, que se utiliza para leer la entrada del usuario. `System.in` es un `InputStream` que se refiere a la entrada estándar del programa (por lo general, esto es lo que el usuario escribe en la consola).

```
    // Pedimos al usuario que ingrese su edad
    System.out.print("Ingresa tu edad: ");
```

Esta línea imprime un mensaje en la consola sin un salto de línea al final, pidiendo al usuario que ingrese su edad.

```
    String edad = scanner.nextLine(); // Leemos la entrada como String
```

Llama al método nextLine() del objeto scanner para leer una línea completa de entrada (hasta que el usuario presiona Enter) y la almacena en una variable de tipo String llamada edad.

ISBN: 978-631-6557-57-5

```
// Mostramos la edad ingresada
System.out.println("Tu edad es: " + edad);
```

Imprime un mensaje en la consola, concatenando el texto "Tu edad es: " con el valor de la variable edad, mostrando así la edad que el usuario ingresó.

```
scanner.close(); // Cerramos el scanner
```

Finalmente, esta línea cierra el objeto scanner para liberar los recursos que estaba utilizando. Es una buena práctica cerrar el Scanner cuando ya no se necesita, especialmente si está leyendo de un archivo u otra fuente que ocupa recursos del sistema.

```
    }
}
```

Estas llaves cierran el método main y la clase Edad, respectivamente, marcando el fin del bloque de código para cada uno.

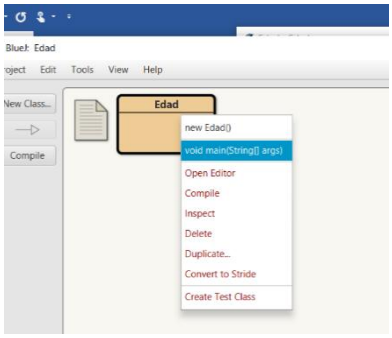
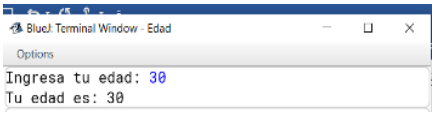


Figura 1.15 Método main

Elaborado por los Autores

Ejecutar el método main (Figura 1.15) de la clase para ver el programa en acción. BlueJ te proporcionará una interfaz para introducir argumentos, pero en este caso, no necesita proporcionar ninguno. Simplemente dar clic en "OK".



Le aparecerá una ventana donde le solicitará que ingrese la edad.

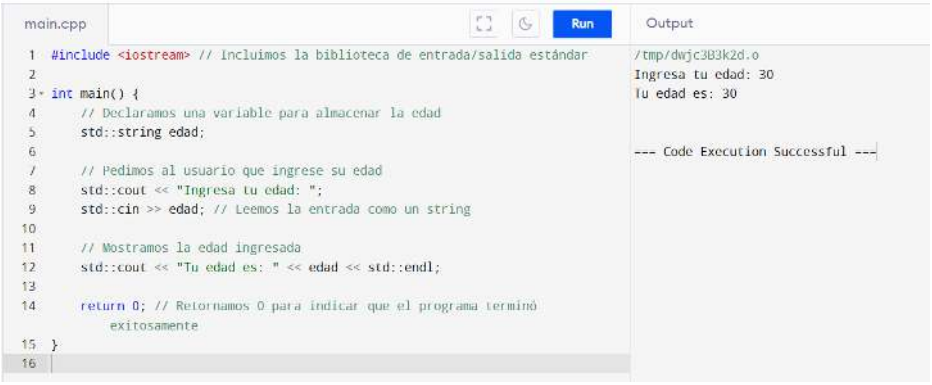
ISBN: 978-631-6557-57-5

b) Instalación y utilización de C++

C++ es un lenguaje de programación de propósito general que fue desarrollado por Bjarne Stroustrup como una extensión del lenguaje C. Se introdujo oficialmente en 1985, y desde entonces ha evolucionado a través de varias versiones estándar, es ampliamente reconocido por su capacidad para ofrecer un equilibrio entre el alto rendimiento y la expresividad en la escritura de código (Malhotra & Malhotra, 2023).

C++ se utiliza en una amplia gama de aplicaciones desde el desarrollo de sistemas operativos, navegadores web, juegos, aplicaciones de escritorio, aplicaciones para servidores hasta sistemas embebidos y aplicaciones de alto rendimiento como clientes de bases de datos y sistemas de comercio electrónico (Benjumea & Roldan, 2011).

Para usar C++ vamos ha ingresar en el navegador a C++ Online Compiler (programiz.com) , que permite compilar las líneas de código escritas en C++. Procedemos a realizar el mismo ejemplo que tenemos en Python.



En la interfaz que nos presenta ubicamos el código y pulsar en Run para hacer que corra el programa. Luego nos va ha mostrar la pantalla de la derecha con la salida (Output).

Explicación del código C++:

#include <iostream>: Esta línea incluye la biblioteca estándar de entrada/salida en C++, que nos permite utilizar std::cin para leer de la entrada estándar (como el teclado) y std::cout para escribir en la salida estándar (como la consola).

int **main**() { ... }: Define la función principal main, que es el punto de entrada de cualquier programa en C++. La ejecución del programa comienza y termina con esta función (Malhotra & Malhotra, 2023).

ISBN: 978-631-6557-57-5

std::string edad;: Declara una variable llamada edad de tipo std::string, que se usará para almacenar la edad ingresada por el usuario. C++ requiere que declaremos el tipo de cada variable explícitamente.

std::cout << "Ingresa tu edad: ";: Imprime el mensaje "Ingresa tu edad: " en la consola. std::cout se utiliza para la salida, y el operador << se utiliza para enviar datos a dicha salida.

std::cin >> edad;: Lee la entrada del usuario desde la consola y la almacena en la variable edad. std::cin se utiliza para la entrada, y el operador >> se utiliza para recibir datos de dicha entrada.

std::cout << "Tu edad es: " **<< edad << std::endl;**: Imprime el mensaje "Tu edad es: " seguido de la edad ingresada por el usuario y un salto de línea (std::endl).

return 0;: Finaliza la función main retornando 0, lo cual indica que el programa terminó exitosamente.
Este programa pedirá al usuario que ingrese su edad y luego imprimirá la edad que fue ingresada, similar a como lo hace el programa original en Python.

c) Instalación y utilización de JavaScript

JavaScript es un lenguaje de programación de alto nivel, interpretado y basado en el concepto de prototipos. Fue creado por Brendan Eich en 1995 y originalmente se llamaba LiveScript antes de ser renombrado a JavaScript. Aunque comparte parte de su nombre y ciertas convenciones de sintaxis con Java, los dos lenguajes son independientes y tienen diferentes propósitos y diseños. JavaScript se ha convertido en uno de los pilares de la World Wide Web, junto con HTML y CSS, proporcionando la estructura, el estilo y la interactividad de las páginas web, respectivamente (Learning, 2000).

JavaScript se ejecuta directamente en el navegador web (lado cliente) sin necesidad de compilación previa, permitiendo scripts dinámicos que pueden modificar el contenido

ISBN: 978-631-6557-57-5

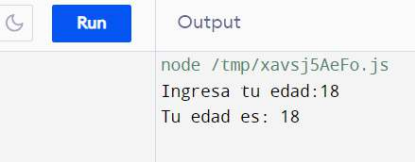
de una página web en tiempo real. Soporta estilos de programación orientada a objetos, imperativa y funcional. Las variables en JavaScript pueden contener datos de cualquier tipo y cambiar su tipo en tiempo de ejecución.

JavaScript permite el acceso y modificación del Document Object Model (DOM), lo que significa que puede cambiar tanto la estructura como el contenido de las páginas web (Barzee, 2017).

Su capacidad para ejecutarse tanto en el cliente como en el servidor permite a los desarrolladores utilizar un único lenguaje de programación a lo largo de todo el proyecto, simplificando el proceso de desarrollo de software.

Para usar JavaScript vamos ha ingresar en el navegador a Online JavaScript Compiler (programiz.com) , que permite compilar las líneas de código escritas en JavaScript. Se procede a realizar el mismo ejemplo que se tiene en Python para observar las diferencias.

Luego de haber ubicado las líneas de código se procede a ejecutar dando click en Run



Para convertir el código de Python a JavaScript, se necesita adaptarlo para ejecutarse en un entorno como un navegador web, ya que JavaScript no tiene una función de entrada directa como input en Python para consolas. La forma más común de recoger una entrada en JavaScript en un entorno de navegador es usando prompt, y para mostrar la salida se puede usar alert o console.log.

prompt("Ingresa tu edad:"): Muestra un cuadro de diálogo con un campo de texto para que el usuario ingrese algo. En este caso, pedimos la edad del usuario. La función prompt devuelve el texto que el usuario ha introducido.

var edad = ...: Guarda el valor ingresado por el usuario en la variable edad.

alert("Tu edad es: " + edad): Muestra un cuadro de diálogo con el mensaje compuesto por "Tu edad es: " seguido del valor de la variable edad.

ISBN: 978-631-6557-57-5

console.log("Tu edad es: " + edad): Esta es otra forma de mostrar la salida, pero en lugar de usar un cuadro de diálogo, imprime el mensaje en la consola del navegador. Esta línea está comentada porque normalmente usarías alert o console.log, no ambos. console.log es útil para la depuración o cuando quieres ver la salida sin interrumpir la interacción del usuario con la página web.

1.4.9 Comparación de los lenguajes

Para comparar la implementación de este sencillo programa en Python, Java, C++, y JavaScript, consideraremos varios aspectos como la cantidad de líneas de código necesarias, la dificultad percibida para un principiante y la recomendación de cada lenguaje para enseñar fundamentos de programación a estudiantes principiantes. La siguiente tabla muestra la comparación:

Lenguaje	Líneas de código	Dificultad para Principiantes	Recomendado para principiantes
Python	2	Baja	Si
Java	8	Media	No
C++	7	Media-Alta	No
JavaScript	2	Baja	Si

Tabla 1.1 Comparación de lenguajes de programación

Elaborado por los Autores

Explicación:

Python: Este lenguaje es conocido por su sintaxis clara y concisa, lo que lo hace muy legible y fácil de aprender para los principiantes. El programa de ejemplo se escribe en muy pocas líneas, y no hay necesidad de manejar tipos de datos complicados para una tarea tan simple.

La facilidad para manejar variables sin la necesidad de declarar su tipo explícitamente y la gestión automática de memoria hacen que Python sea una elección favorable para enseñar los fundamentos de la programación, permitiendo que los estudiantes se enfoquen en la lógica y estructura del código más que en la sintaxis y la gestión de recursos (Budd, 2009).

Java: Requiere más líneas de código porque se necesita definir una clase y un método

ISBN: 978-631-6557-57-5

"main" para ejecutar el programa. Además, la entrada y salida estándar en Java es más verbosa comparada con Python. Java introduce conceptos como clases y objetos desde el principio, lo cual puede ser más difícil para los principiantes (Friesen, 2024).

El nivel de rigor que tiene Java en la definición y manipulación de variables prepara a los estudiantes para una comprensión más fuerte de conceptos de la programación orientada a objetos y puede contribuir a una mejor detección de errores en etapas tempranas del proceso de compilación.

C++: Bastante similar a Java en cuanto a la cantidad de líneas de código y complejidad, pero añade la gestión manual de la memoria y otras complejidades del lenguaje que pueden ser desafiantes para un principiante. Aunque el programa de ejemplo no muestra la complejidad de la gestión de memoria, es un aspecto que los estudiantes tendrán que enfrentar al aprender C++.

JavaScript: Ofrece una simplicidad similar a Python en términos de líneas de código. Al ser el lenguaje de la web, puede ser muy atractivo para los principiantes que deseen ver resultados inmediatos en el navegador. Además, la sintaxis es amigable y permite a los estudiantes experimentar con la programación interactiva de manera sencilla (Barzee, 2017).

Para enseñar fundamentos de programación a estudiantes principiantes, Python y JavaScript son altamente recomendados debido a su sintaxis simple, menor cantidad de líneas de código para realizar tareas básicas y la facilidad con la que los estudiantes pueden empezar a ver resultados tangibles. Python es ampliamente reconocido por su uso en educación inicial en programación, mientras que JavaScript ofrece una excelente entrada al desarrollo web, lo cual es muy relevante hoy en día. Java y C++, aunque fundamentales en muchos aspectos de la programación y con grandes aplicaciones en la industria, pueden presentar una curva de aprendizaje más pronunciada para los principiantes, haciéndolos más adecuados para cursos avanzados.

El desarrollo del libro se basará en Python, pero eso no quiere decir que en momentos oportunos se presenten ejemplos en los otros tres lenguajes.

1.4.10 Expresiones

En programación, las expresiones son fragmentos de código que el compilador o intérprete evalúa para producir un valor (Mejia et al., 2024). Estas se clasifican en

ISBN: 978-631-6557-57-5

distintos tipos según el resultado que generan y los operadores que utilizan:

1) **Expresiones Aritméticas**: Estas incluyen operadores como suma (+), resta (-), multiplicación (*), división (/), y módulo (%) utilizado para obtener el resto de la división, y son usadas para calcular valores numéricos. Por ejemplo, 6 + 3 * 2 es una expresión aritmética que daría como resultado 12, siguiendo la precedencia de operadores.

En las expresiones aritméticas se incluyen otros operadores además de los mencionados. Algunos de estos son:

Exponenciación (**): Usado para elevar un número a la potencia de otro. Por ejemplo, 5 ** 3 resultaría 125.

División entera (//): Similar a la división, pero devuelve el resultado sin decimales, es decir, el resultado entero de la división. Por ejemplo, 5 // 2 resultaría en 2.

Operador unario de positivo (+): No cambia el signo del operando. Por ejemplo, +3 simplemente será 3.

Operador unario de negativo (-): Cambia el signo del operando. Por ejemplo, -3 cambia el signo de 3 a negativo.

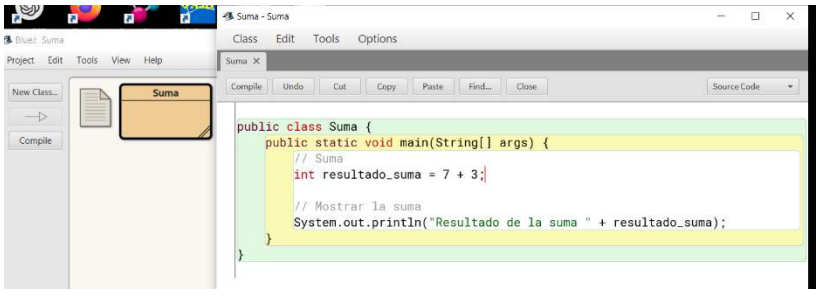
Estos operadores se pueden combinar en expresiones más complejas para realizar cálculos matemáticos de diversa índole dentro de un programa.

A continuación ejemplo en **Python** de la suma de dos números:

```
# Suma
resultado_suma = 7 + 3
print("Resultado de la suma", resultado_suma)
```

```
Resultado de la suma 10
```

Código en **Java** para la suma de los 2 números:



ISBN: 978-631-6557-57-5

Para escribir el siguiente código en **Python** hacerquese a la parte inferior del programa que escribió anteriormente y le asomará + Código + Texto escoger código, para continuar haciendo otros programas, como los que se muestran a continuación sobre multiplicación, exponenciación, determinación del valor del módulo .

```
# Multiplicación
resultado_multiplicacion = 4 * 5
print("Resultado de la multiplicación", resultado_multiplicacion)
```

```
Resultado de la multiplicación 20
```

```
# Exponenciación
resultado_exponenciacion = 5 ** 3
print("Resultado del exponente", resultado_exponenciacion)
```

```
Resultado del exponente 125
```

```
# Módulo
resultado_modulo = 18 % 5
print("Resultado del modulo", resultado_modulo)
```

```
Resultado del modulo 3
```

En **C++**

main.cpp — Run — Output

```
#include <iostream>
#include <cmath> // Necesario para la función pow

int main() {
    // Exponenciación
    double resultado_exponenciacion = pow(5, 3);
    std::cout << "Resultado del exponente: " << resultado_exponenciacion << std
        ::endl;

    return 0;
}
```

```
/tmp/TWDnYmd1Bj.o
Resultado del exponente: 125


=== Code Execution Successful ===
```

Este código incluye la directiva de preprocesador #include <cmath>, que es necesaria para usar la función pow que realiza la exponenciación. Luego, se imprime el resultado utilizando **std::cout**. En C++ se utiliza **std::endl** para un salto de línea al final de la impresión en la consola.

En **JavaScript**

main.js — Run — Output

```
// Módulo
let resultado_modulo = 18 % 5;
console.log("Resultado del módulo", resultado_modulo);
```

```
node /tmp/yDqQnxam1F.js
Resultado del módulo 3
```

El operador % es el operador de módulo, y 18 % 5 calcula el resto de dividir 18 entre 5. Como 18 dividido entre 5 es 3 con un resto de 3, el resultado de 18 % 5 es 3.

Por lo tanto, la salida que se muestra en la consola Resultado del módulo 3, es el

ISBN: 978-631-6557-57-5

resultado de esta operación de módulo.

2) **Expresiones Relacionales:** Utilizan operadores de comparación como igualdad (==), desigualdad (!=), menor que (<), mayor que (>), menor o igual que (<=), y mayor o igual que (>=). Estas expresiones evalúan la relación entre dos operandos y devuelven un valor booleano (True o False). Un ejemplo podría ser edad >= 18, que verifica si la variable edad es mayor o igual a 18.

Ejemplos en Python:

1) Si dos números que se comparan son iguales la respuesta es verdadero, caso contario es falso.

```
# Igualdad
es_igual = (10 == 10)
print( es_igual)

True
```

La línea es_igual = (10 == 10) usa el operador de igualdad == para comparar si 10 es igual a 10, lo cual es verdadero (True). Por lo tanto, la variable es_igual se establece en el valor booleano True.

Luego, la función print(es_igual) imprime el valor de la variable es_igual, que es True.

En la salida se muestra True, confirmando que la expresión 10 == 10 es verdadera.

2) Ver si dos numeros son diferentes

```
# Desigualdad
es_diferente = (8 != 5)
print( es_diferente)

True
```

El código es un ejemplo de una operación de comparación en Python. El comentario en español dice "Desigualdad", que se refiere a verificar si dos valores son diferentes. En el código, es_diferente se establece a la evaluación de (8 != 5), que es una expresión que verifica si 8 es distinto de 5. Dado que 8 y 5 son valores diferentes, la expresión se evaluará como True. Luego, este valor booleano se imprime, lo que resulta en la salida que se ve en la imagen: True.

ISBN: 978-631-6557-57-5

3) Establecer si un número es menor comparado con otro

```
# Menor que
es_menor = (12 < 10)
print( es_menor)
```

```
False
```

```
# Mayor o igual que
es_mayor_o_igual = (5 >= 5)
print(es_mayor_o_igual )
```

```
True
```

El código está evaluando si 12 es menor que 10. La expresión (12 < 10) se evalúa como **False** porque 12 no es menor que 10, es mayor. Por lo tanto, la variable es_menor se establece en False y este valor se imprime en la consola.

Se muestra un código en Python que incluye una operación de comparación con un comentario que dice "Mayor o igual que". El código está comprobando si 5 es mayor o igual a 5. La expresión (5 >= 5) es verdadera porque 5 es igual a 5, y en términos de la comparación "mayor o igual", cuando los números son iguales, la condición también se considera verdadera. Por eso, la variable es_mayor_o_igual se evalúa como True, y el resultado que se imprime en la consola es True.

En **C++**

Igualdad o desigualdad de 2 números

main.cpp

Run

Output

```
#include <iostream>

int main() {
    // Igualdad
    bool es_igual = (10 == 10);

    // Imprimir el resultado
    std::cout << std::boolalpha;  // Esto hará que se imprima "true" o "false" en
        lugar de 1 o 0
    std::cout << "El resultado es: " << es_igual << std::endl;

    return 0;
}
```

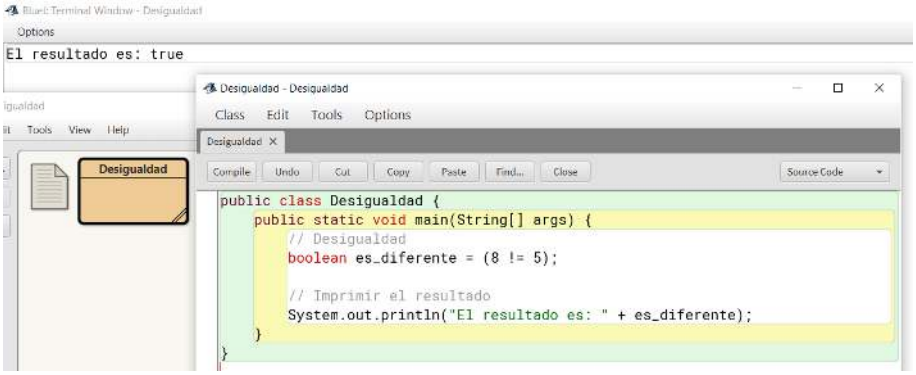
```
/tmp/mKvegH71S7.o
El resultado es: true


=== Code Execution Successful ===
```

En este código de C++, se declara una variable booleana (bool) es_igual que almacena el resultado de la comparación (10 == 10). Luego, se imprime el resultado utilizando **std::cout**, y se utiliza **std::boolalpha** para que el booleano se imprima como true o false en lugar de como un entero. Finalmente, std::endl se usa para añadir un salto de línea al final de la impresión.

ISBN: 978-631-6557-57-5

En **Java**

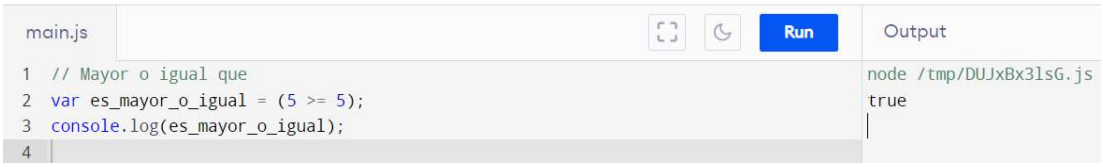
Establecer si dos números son diferentes



En este código Java, se declara una variable booleana (boolean) es_diferente que almacena el resultado de la comparación (8 != 5). Luego, se imprime el resultado en la consola usando **System.out.println**. Java utiliza != para probar la desigualdad, al igual que Python (Sharan & Davis, 2021).

En **JavaScript**

Código para establecer si al comparar dos números son iguales o uno es mayor que el otro.



En este código de JavaScript, se utiliza var para declarar la variable es_mayor_o_igual. Luego asignamos a esa variable el resultado de la comparación (5 >= 5), que evalúa si 5 es mayor o igual a 5, lo cual es true. Finalmente, imprimimos el resultado en la consola con console.log() (Easttom, 2001).

3. **Expresiones Lógicas:** Combinan valores booleanos o expresiones que resultan en valores booleanos utilizando operadores lógicos como AND (&& o and), OR (|| o or), y NOT (! o not). Por ejemplo, aprobado && asistenciaCompleta puede representar una expresión lógica que sólo devuelve True si ambas “aprobado” y “asistenciaCompleta” son verdaderas.

ISBN: 978-631-6557-57-5

Ejemplos:

En Python

```
# AND lógico
condicion_and = (True and False)
print(condicion_and)
```

```
False
```

El código en Python realiza una operación lógica **AND**. La variable condicion_and se establece con el resultado de la operación (True and False). La operación AND lógica solo devuelve True si ambos operandos son verdaderos. Dado que uno de los operandos es False, el resultado de la operación es False. Luego, el valor de condicion_and se imprime, lo que nos da como salida False (Annable, 2024).

```
#AND lógico
condicion_and = (True and True)
print(condicion_and)
```

```
True
```

En el caso de que ambos sean True (True and True), la respuesta es True.

```
# OR lógico
condicion_or = (True or False)
print(condicion_or)
```

```
True
```

El código representa una operación lógica **OR** en Python. La variable condicion_or se establece con el resultado de (True or False). La operación OR lógica devuelve True si al menos uno de los operandos es verdadero. En este caso, como el primer operando es True, el resultado de la operación es True, independientemente del valor del segundo operando. Por lo tanto, el valor de condicion_or se imprime como True.

```
# NOT lógico
condicion_not = not(True)
print(condicion_not)
```

```
False
```

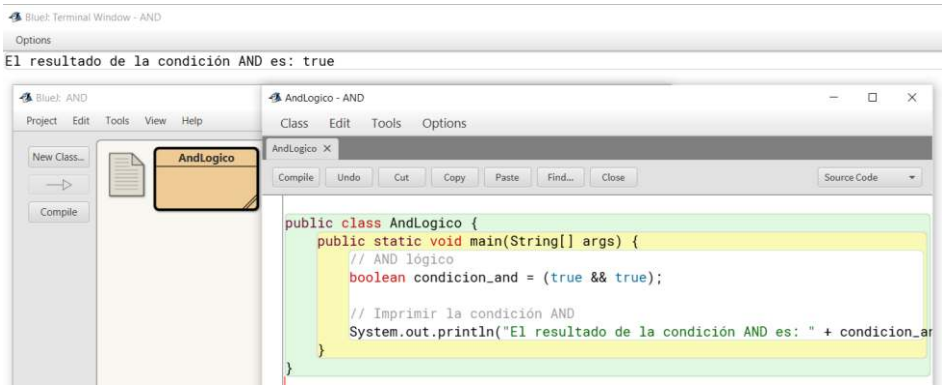
El código muestra una operación lógica **NOT** en Python. La variable

ISBN: 978-631-6557-57-5

condicion_not se establece como el resultado de negar True mediante la palabra clave **not**. La operación NOT lógica invierte el valor de verdad del operando, por lo tanto, not(True) devuelve False. Después, se imprime el valor de condicion_not, resultando en la salida False.

Cada uno de estos tipos de expresiones juega un papel fundamental en la construcción de algoritmos y en la lógica de decisión dentro de los programas (Joyanes, 2020).

En **Java**



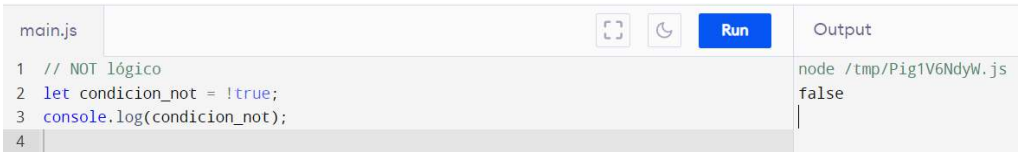
En este código Java, se declara una variable booleana (boolean) condicion_and y se asigna el resultado de la operación lógica (true && true). Luego se imprime el resultado en la consola con System.out.println(). En Java, el operador lógico AND se representa con &&.

En **C++**

Este código en C++ a igual que en Java, JavaScript se utiliza || que es el operador lógico **OR**. Declara una variable booleana (bool) condicion_or que almacena el resultado de la operación (true || false). Luego imprime este resultado a la consola.

ISBN: 978-631-6557-57-5

En **JavaScript**



El código de JavaScript que realiza una operación lógica **NOT**. Se declara una variable condicion_not con la palabra clave let, que se inicializa con el resultado de aplicar not a True (representado en JavaScript como true) (Easttom, 2001). En JavaScript, el operador lógico NOT se representa con el signo de exclamación ! que es similar en Java y C++.

1.1.6 Funciones

Las funciones en programación son similares a las funciones en matemáticas: toman entradas, realizan operaciones con ellas y devuelven salidas. En el contexto de la programación, una función es una sección de código a la que se le da un nombre y que puede ser invocada desde otras partes del programa tantas veces como sea necesario, lo que ayuda a evitar la repetición de código.

Características clave de las funciones:

Nombre: Cada función tiene un identificador único, que es el nombre que se utiliza para invocarla.

Parámetros: Son variables listadas entre los paréntesis en la definición de la función. Estos actúan como variables de entrada para la función.

Cuerpo: El conjunto de instrucciones que se ejecutan cuando se llama a la función. Estas instrucciones definen la tarea que realiza la función.

Valor de retorno: Después de ejecutar el cuerpo, la función puede devolver un valor al punto donde fue llamada utilizando la instrucción return.

Las funciones pueden ser de dos tipos:

Procedimientos: Son funciones que llevan a cabo una tarea, pero no devuelven un valor.

Funciones propiamente dichas: Son las que realizan una operación y devuelven un resultado.

Ejemplo: función para sumar dos números.

En **Python**

ISBN: 978-631-6557-57-5

```
[1] def sumar(a, b):
        resultado = a + b
        return resultado
```

```
# Llamar a la función y pasarle dos números como argumentos
suma = sumar(10, 12)
print(suma)  # Esto imprimirá 22 en la consola

22
```

En el ejemplo, **sumar** es el nombre de la función, **a** y **b** son los parámetros, y **resultado** es el valor que se devuelve con **return**. Cuando se llama a **sumar(10, 12)**, la función recibe **10** y **12** como argumentos, los suma y devuelve el resultado, que luego se imprime en la consola.

En **C++**

main.cpp — Run — Output

```
#include <iostream>

int sumar(int a, int b) {
    int resultado = a + b;
    return resultado;
}

int main() {
    int suma = sumar(10, 12);
    std::cout << suma; // Esto imprimirá 22 en la consola
    return 0;
}
```

```
/tmp/szMdM9s7uP.o
22

=== Code Execution Successful
```

Este programa en C++ se compone de dos partes principales: la definición de una función llamada **sumar** y la función **main**, que es el punto de entrada del programa.

#include <iostream>: Esta directiva del preprocesador le indica al compilador que incluya la biblioteca estándar de entrada y salida. Esto es necesario para poder usar **std::cout** para imprimir en la consola.

int sumar(int a, int b): Esta es la definición de la función **sumar**, que toma dos argumentos de tipo entero, a y b. La función calcula la suma de a y b, y guarda el resultado en la variable **resultado**, también de tipo entero.

return resultado;: La función sumar termina devolviendo el valor de resultado al código que la llamó.

int main(): main es la función principal que se ejecuta al iniciar el programa. Es donde inicia y termina la ejecución de un programa en C++.

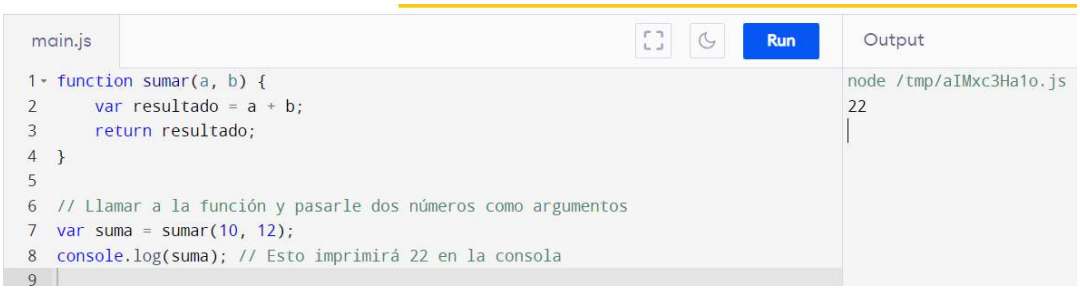
ISBN: 978-631-6557-57-5

int suma = sumar(10, 12);: Dentro de main, se declara una variable entera llamada suma y se le asigna el valor devuelto por la función sumar cuando se llama con los argumentos 10 y 12.

std::cout << suma;: Este código imprime el valor de la variable suma en la consola. Como suma tiene el valor de la suma de 10 y 12, imprimirá 22.

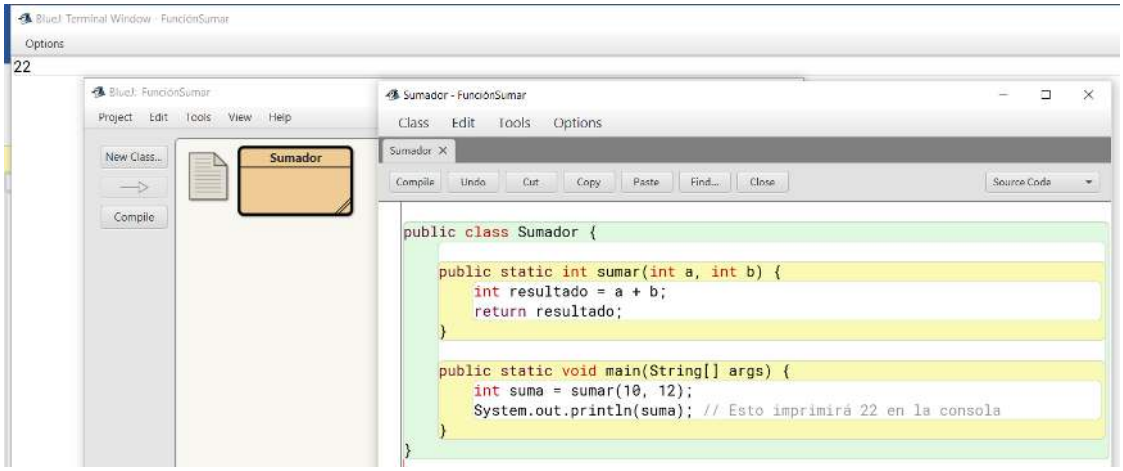
return 0;: Finalmente, main devuelve 0, lo cual es una convención que indica que el programa ha terminado correctamente.

En **JavaScript**



En JavaScript, la función sumar realiza la misma operación que la función en Python: toma dos argumentos y devuelve su suma. Se utiliza var para declarar variables y console.log para imprimir el resultado en la consola.

En **Java**



En Java, se crea una clase, en este caso se le ha llamado **Sumador**. Dentro de esta clase, hay un método **sumar** que toma dos enteros como parámetros y devuelve su suma. El método **main** es el punto de entrada del programa, donde se llama al método sumar y se muestra el resultado utilizando System.out.println.

ISBN: 978-631-6557-57-5

1.5 Diseño de algoritmos, Pseudocódigo y Diagramas de flujo

1.5.1 Diseño de algoritmos

Un algoritmo es, en esencia, un conjunto ordenado y finito de operaciones que permite solucionar un problema específico. La construcción de un algoritmo eficiente y efectivo requiere un proceso cuidadoso que va desde la comprensión profunda del problema a resolver hasta la implementación del algoritmo en un lenguaje de programación específico (Joyanes, 2020). Este proceso se puede desglosar en tres niveles principales de descripción: de alto nivel, formal e implementación.

1. **Descripción de Alto Nivel**

En el nivel más alto, el primer paso es definir claramente el problema a solucionar. Esto implica entender las necesidades y restricciones, así como los objetivos que se buscan alcanzar. Posteriormente, se selecciona un modelo matemático o se describe el algoritmo de manera verbal, utilizando ilustraciones si es necesario, pero omitiendo detalles técnicos o específicos de implementación.

Ejemplo 1

Definición del Problema: Un cliente realiza un pedido a una fábrica, y dependiendo de su solvencia, la fábrica aceptará o rechazará el pedido.

Datos de Entrada: Pedido del cliente, ficha del cliente.

Proceso: Examinar la ficha del cliente para verificar solvencia.

Datos de Salida: Aceptación o rechazo del pedido.

Ejemplo 2

Definición del Problema: Preparar y poner la mesa para la comida.

Datos de Entrada: Vajilla, vasos, cubiertos, servilletas, número de comensales.

Proceso: Colocar sobre la mesa los elementos necesarios según el número de comensales.

Datos de Salida: Mesa correctamente puesta.

2. **Descripción Formal**

A continuación, se procede a describir el algoritmo en un nivel más formal, utilizando pseudocódigo o diagramas de flujo. Esta descripción detalla la secuencia de pasos o instrucciones que se deben seguir para alcanzar la solución del problema, estructurando las operaciones de manera lógica y secuencial.

ISBN: 978-631-6557-57-5

Ejemplo 1

Inicio

Leer el pedido del cliente

Examinar la ficha del cliente

Si el cliente es solvente Entonces

  Aceptar el pedido

Sino

  Rechazar el pedido

Fin

Ejemplo 2

Inicio

Poner el mantel

Para cada tipo de elemento (servilletas, vasos, platos, cubiertos) hacer

  Repetir

    Tomar un elemento del tipo correspondiente

  Hasta que el número de elementos sea igual al de comensales

Fin

3. Implementación

Finalmente, el algoritmo se traduce a un lenguaje de programación específico, transformando las descripciones previas en código ejecutable por una computadora. Esta etapa requiere conocimientos técnicos sobre el lenguaje elegido y sobre principios de programación.

Ejemplo 1:

```
pedido = leer_pedido()
cliente = obtener_ficha_cliente(pedido.cliente_id)
if cliente.es_solvente:
  aceptar_pedido(pedido)
else:
  rechazar_pedido(pedido)
```

El código es un fragmento de Python diseñado para gestionar pedidos basándose en la solvencia de un cliente. Aquí te explico paso a paso qué hace cada línea:

ISBN: 978-631-6557-57-5

1. **pedido = leer_pedido()**

- Esta línea llama a la función leer_pedido(), la cual se supone que devuelve algún tipo de objeto o diccionario que representa un pedido realizado por un cliente. La información exacta que retorna dependería de cómo esté implementada dicha función. El resultado de esta función se almacena en la variable pedido.

Los diccionarios se utilizan para almacenar datos de manera que se puedan encontrar de modo eficiente utilizando la clave.

El objeto o diccionario contendrá datos que representan el pedido, como pueden ser el ID del pedido, los artículos pedidos, la cantidad de cada artículo, el precio, la información del cliente, etc.

2. **cliente = obtener_ficha_cliente(pedido.cliente_id)**

- Aquí se llama a la función obtener_ficha_cliente(), pasando como argumento el atributo cliente_id del objeto pedido obtenido en el paso anterior. Esta función está diseñada para buscar y retornar información sobre un cliente específico, probablemente desde una base de datos o un sistema de almacenamiento de datos. La información del cliente se almacena en la variable cliente.

3. **if cliente.es_solvente:**

- Esta línea inicia una estructura condicional (if) que verifica si el cliente asociado con el pedido es solvente. Se asume que cliente es un objeto que tiene un atributo es_solvente, el cual es un valor booleano (True si el cliente es solvente, False en caso contrario).

4. **aceptar_pedido(pedido)**

- Dentro del bloque if, si se cumple que el cliente es solvente (cliente.es_solvente evalúa a True), se ejecuta la función aceptar_pedido(), pasando el objeto pedido como argumento. Esta función realizaría las operaciones necesarias para procesar y aceptar el pedido, como actualizar una base de datos, enviar notificaciones, etc.

5. **else:**

- Esta parte del condicional se ejecuta si el cliente no es solvente (cliente.es_solvente evalúa a False).

6. **rechazar_pedido(pedido)**

- Si el cliente no es solvente, se llama a la función rechazar_pedido(), pasando el mismo objeto pedido como argumento. Esta función se encargaría de las acciones necesarias para

ISBN: 978-631-6557-57-5

rechazar el pedido, que podrían incluir notificar al cliente, registrar el rechazo en una base de datos, entre otros.

Ejemplo 2:

```
poner_mantel()
elementos = ['servilletas', 'vasos', 'platos', 'cubiertos']
for elemento in elementos:
    cantidad = 0
    while cantidad < num_comensales:
        colocar_elemento(elemento)
        cantidad += 1
```

Este fragmento de código en Python describe un proceso para preparar una mesa para cierto número de comensales, siguiendo una serie de pasos específicos para colocar varios elementos esenciales como servilletas, vasos, platos y cubiertos. Aquí se explica cada parte del código:

1. **poner_mantel()**

- Esta función, que se supone definida en otra parte del código, se encarga de poner un mantel en la mesa. No se especifican detalles sobre cómo lo hace, debido a que depende de la implementación de la función. Podría ser tan simple como imprimir un mensaje que diga que el mantel ha sido colocado, o podría involucrar una serie de pasos más complejos en un contexto más amplio (como en una simulación o en una interfaz de usuario gráfica).

2. **elementos = ['servilletas', 'vasos', 'platos', 'cubiertos']**

- Aquí se crea una lista llamada elementos que contiene strings representando los distintos tipos de elementos que se van a colocar en la mesa: servilletas, vasos, platos y cubiertos.

3. **for elemento in elementos:**

- Este es un bucle for que itera sobre cada elemento en la lista elementos. En cada iteración, la variable elemento tomará el valor de uno de los elementos de la lista, permitiendo realizar acciones específicas para cada tipo de elemento (servilletas, vasos, platos, cubiertos) en las iteraciones sucesivas.

4. **cantidad = 0**

ISBN: 978-631-6557-57-5

- Al inicio de cada iteración del bucle for, se inicializa una variable cantidad a 0. Esta variable llevará la cuenta de cuántos elementos de un tipo específico han sido colocados en la mesa.

5. **while cantidad < num_comensales:**

- Se introduce un bucle while que se ejecutará mientras que el valor de cantidad sea menor que num_comensales. La variable num_comensales se asume que ha sido definida en otra parte del código y representa el número total de personas para las que se está preparando la mesa.

6. **colocar_elemento(elemento)**

- Dentro del bucle while, se llama a la función colocar_elemento(), pasando como argumento la variable elemento, que representa el tipo de elemento que se está colocando en la mesa en esa iteración. La función colocar_elemento() se encargaría de realizar las acciones necesarias para colocar efectivamente el elemento en la mesa. La naturaleza exacta de estas acciones dependerá de la implementación de la función.

7. **cantidad += 1**

- Después de colocar cada elemento, se incrementa el valor de cantidad en 1. Esto actualiza el conteo de cuántos elementos de ese tipo se han colocado hasta el momento.

El código, por tanto, describe un procedimiento para colocar igual número de servilletas, vasos, platos y cubiertos en una mesa, de acuerdo con el número de comensales. Se asegura de que para cada comensal haya un conjunto completo de estos elementos, utilizando para ello una combinación de bucles for y while, que serán explicados en detalle más adelante, al igual que el if – else utilizado en el ejemplo 1.

El *diseño de algoritmos* es un proceso iterativo y refinado que va *de lo general a lo específico*, permitiendo abordar problemas complejos de manera estructurada. Comenzar por una *descripción de alto nivel* facilita la comprensión del problema y su solución, la *descripción formal* especifica el cómo se soluciona, y la i*mplementación* pone en práctica la solución de manera concreta y ejecutable (Chinnathambi, 2024)**.**

1.5.2 Pseudocódigo

El pseudocódigo es una forma de expresar algoritmos que es más fácil de entender que el código de programación real para personas que están diseñando y desarrollando algoritmos. No está destinado a ser ejecutado por una computadora, sino a ser leído por

ISBN: 978-631-6557-57-5

humanos y por eso utiliza un lenguaje cercano al natural con algunas estructuras de control tomadas de la programación (Easttom, 2006).

El pseudocódigo al no estar atado a las sintaxis específicas de los lenguajes de programación, permite a los desarrolladores concentrarse en la lógica subyacente del problema. Es una forma de representar los pasos que el algoritmo debe seguir para llegar a una solución de manera que sean fáciles de leer y entender.

Características del Pseudocódigo:

- **Legibilidad:** Su estructura y sintaxis intuitivas permiten una comprensión clara del flujo del algoritmo.

-**Independencia:** Se escribe sin adherirse a las reglas de ningún lenguaje de programación específico.

- **Flexibilidad:** Aunque no es un código ejecutable, sus expresiones y estructuras se asemejan a las usadas en programación real.

-**Precisión:** Aunque utiliza un lenguaje cercano al natural, el pseudocódigo es preciso en su descripción de la lógica del algoritmo.

-**Estándar Informal:** No existe un estándar único para el pseudocódigo; su estilo puede variar entre diferentes programadores y organizaciones, pero suele seguir convenciones comunes que facilitan su entendimiento.

Componentes del Pseudocódigo:

-**Variables:** Se declaran al principio, especificando su nombre y tipo.

-**Instrucciones:** Se siguen secuencialmente y pueden incluir estructuras de control como bucles y condicionales.

-**Comentarios:** Se utilizan para explicar partes del algoritmo y mejorar su comprensión.

Reglas para la Escritura del Pseudocódigo:

1. **Identificadores:** Los nombres de variables y funciones deben ser descriptivos y no deben coincidir con palabras reservadas. Se recomienda un límite de 50 caracteres y pueden contener letras, dígitos y guiones bajos.

2. **Operadores y Signos de Puntuación:** Se utilizan para realizar operaciones matemáticas y para estructurar el código, respectivamente.

3. **Comentarios:** Se indican con // para comentarios de una sola línea y con {} para comentarios de múltiples líneas.

ISBN: 978-631-6557-57-5

Ejemplo 1: Cálculo del Perímetro de un Rectángulo

```
Algoritmo perímetro_rectángulo
  definir base, altura, perímetro Como Entero
  Escribir "Ingrese el valor de la base"
  Leer base
  Escribir "Ingrese el valor de la altura"
  Leer altura
  perímetro <- 2 * base + 2 * altura
  Escribir "El perímetro del rectángulo es: ", perímetro
FinAlgoritmo
```

La interpretación del pseudocódigo que describe un algoritmo sencillo para calcular el perímetro de un rectángulo es el siguiente:

1. **definir base, altura, perímetro Como Entero:** Aquí se están declarando tres variables (base, altura y perímetro) como enteros. Esto significa que estas variables almacenarán valores numéricos enteros.
2. **Escribir "Ingrese el valor de la base":** Este comando muestra un mensaje al usuario pidiéndole que ingrese el valor de la base del rectángulo.
3. **Leer base:** Este comando permite al usuario ingresar un valor que se almacenará en la variable base.
4. **Escribir "Ingrese el valor de la altura":** Similar al paso 2, esto muestra un mensaje al usuario para que ingrese el valor de la altura del rectángulo.
5. **Leer altura:** Aquí el usuario ingresa un valor que se almacenará en la variable altura.
6. **perímetro <- 2 * base + 2 * altura:** Esta línea calcula el perímetro del rectángulo utilizando la fórmula matemática del perímetro P = 2*(base + altura). El resultado se almacena en la variable perímetro.
7. **Escribir "El perímetro del rectángulo es: ", perímetro:** Finalmente, se muestra al usuario el perímetro calculado, utilizando el valor almacenado en la variable perímetro.
8. **FinAlgoritmo:** Esta línea indica el final del algoritmo.

ISBN: 978-631-6557-57-5

Ejemplo 2: Suma de Dos Números Enteros

```
Algoritmo suma_dos_numeros
  definir numero1, numero2, suma Como Entero
  Escribir "Ingrese el valor del primer número a sumar"
  Leer numero1
  Escribir "Ingrese el valor del segundo número a sumar"
  Leer numero2
  suma <- numero1 + numero2
  Escribir "La suma de los dos números ingresados es: ", suma
FinAlgoritmo
```

A continuación, se explica el algoritmo:

1. **definir numero1, numero2, suma Como Entero:** En esta línea, se declaran tres variables llamadas numero1, numero2 y suma. Todas estas variables son de tipo entero, lo que significa que están diseñadas para almacenar números enteros.

2. **Escribir "Ingrese el valor del primer número a sumar":** Esta línea muestra un mensaje en pantalla solicitando al usuario que introduzca el primer número que desea sumar.
3. **Leer numero1:** Aquí el algoritmo espera que el usuario ingrese un valor que se almacenará en la variable numero1.

4. **Escribir "Ingrese el valor del segundo número a sumar":** Similar al paso anterior, este comando muestra un mensaje solicitando al usuario que ingrese el segundo número.
5. **Leer numero2:** El usuario introduce un segundo valor que se almacenará en la variable numero2.
6. **suma <- numero1 + numero2:** Esta línea realiza la operación matemática de suma entre numero1 y numero2, y el resultado se almacena en la variable suma.

7. **Escribir "La suma de los dos números ingresados es: ", suma:** Finalmente, el algoritmo muestra el resultado de la suma, informando al usuario del valor calculado que está almacenado en la variable **suma**.

1.5.3 Diagramas de flujo

Los diagramas de flujo son herramientas visuales utilizadas para representar algoritmos y procesos, facilitando la comprensión del flujo de operaciones y decisiones. Estos diagramas siguen una serie de convenciones estándar que permiten a cualquier

ISBN: 978-631-6557-57-5

persona entender la secuencia lógica de un procedimiento, independientemente de su conocimiento en programación (Ormaza, 2020).

Reglas para la Construcción de Diagramas de Flujo

1. **Inicio y Fin:** Todo diagrama de flujo debe tener un único punto de inicio y un único punto final.

2. **Flujo:** El flujo del proceso se indica con flechas que conectan los distintos elementos. Estas líneas deben ser rectas y solo en dirección vertical u horizontal.

3. **Conectividad:** Las líneas de flujo siempre deben estar conectadas a los símbolos que representan pasos o decisiones.

4. **Orientación:** Un diagrama de flujo se lee de arriba hacia abajo y de izquierda a derecha.

5. **Notación Universal:** Los símbolos utilizados son estándar y no dependen de ningún lenguaje de programación específico. Deben ser claros y concisos para que la lógica pueda ser codificada posteriormente en cualquier lenguaje de programación.

6. **Comentarios:** Deben usarse para explicar operaciones complejas dentro del diagrama de flujo, aportando claridad al propósito y la función de ciertos pasos.

Símbolos Comunes en Diagramas de Flujo

- **Óvalo:** Indica el inicio y el fin del diagrama de flujo.
- **Rectángulo:** Representa una operación o acción a realizar.
- **Rombo:** Denota una decisión, donde el flujo se divide en dos o más caminos en función de una condición.
- **Paralelogramo:** Se usa para entrada y salida de datos.
- **Flechas:** Dirigen el flujo del proceso de un paso a otro.

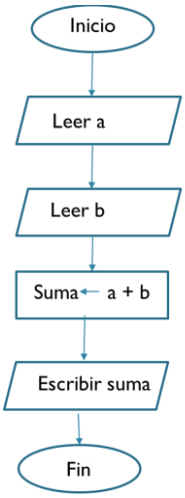
Ejemplo 1: Diagrama de Flujo para Sumar Dos Números Enteros

El objetivo es crear un diagrama de flujo para un algoritmo simple que lea dos números enteros y luego escriba la suma de estos números. El diagrama de flujo para este algoritmo incluiría:

1. Un símbolo de inicio.
2. Dos paralelogramos para las operaciones de entrada de los números.
3. Un rectángulo para la operación de suma.
4. Un paralelogramo para mostrar el resultado de la suma.

ISBN: 978-631-6557-57-5

5. Un símbolo de fin.



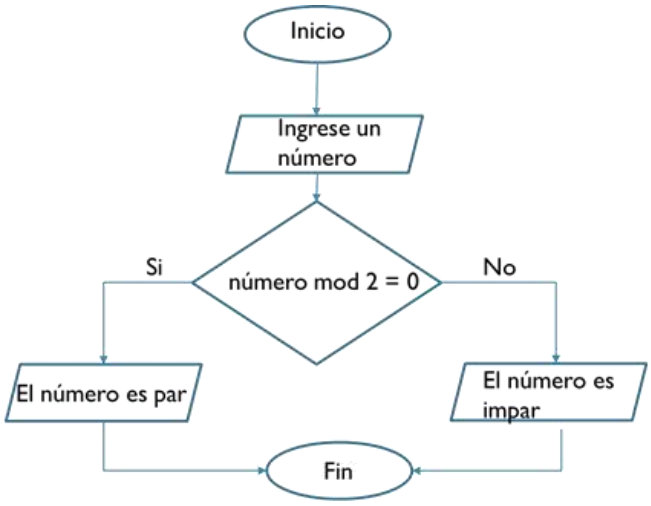
Interpretación del diagrama de flujo:

- Comienza con un óvalo etiquetado **Inicio**.
- A continuación, dos paralelogramos con las etiquetas **Leer a** que es el primer número y **Leer b** que es el segundo número.
- Luego, un rectángulo con la operación **suma**.
- Después, un paralelogramo con la instrucción **Escribir suma**".
- Finaliza con un óvalo etiquetado **Fin**.

Este diagrama es lineal y no requiere decisiones; por tanto, no incluye rombos. Las flechas indicarían el flujo de arriba hacia abajo, conectando cada uno de los símbolos en el orden en que se ejecutan las operaciones.

Ejemplo 2: Ingrese un número y comprobar si ese número es par o impar.

ISBN: 978-631-6557-57-5



Para determinar si un número es par o impar mediante un diagrama de flujo, el proceso seguiría los siguientes pasos.

1. **Inicio:** Este es el punto de partida del diagrama de flujo. Indica el comienzo del proceso.
2. **Entrada de datos:** Aquí se solicita ingresar el número que se quiere evaluar. Este paso se representa con un paralelogramo, que indica una operación de entrada/salida. Se podría etiquetar como "Ingrese un número".
3. **Operación de evaluación:** En este paso se realiza la operación matemática para determinar si el número es par o impar. Esto se representa con un rombo, que indica una decisión. La pregunta sería "¿El número es par?", lo cual se puede determinar con la operación número mod 2 = 0. Si la respuesta es verdadera, el número es par. Si la respuesta es falsa, el número es impar.
4. **Salida si es par:** Si el resultado de la evaluación es verdadero (el número es par), el flujo se dirige hacia una operación de salida, donde se indica que el número es par. Este paso también se representa con un paralelogramo y podría etiquetarse como "El número es par".
5. **Salida si es impar:** Si el resultado de la evaluación es falso (el número es impar), el flujo se dirige hacia otra operación de salida, donde se indica que el número es impar. Al igual que el paso anterior, se utiliza un paralelogramo para este propósito, etiquetándose como "El número es impar".

ISBN: 978-631-6557-57-5

6. **Fin:** Tanto el flujo de "El número es par" como el de "El número es impar" convergen en un punto final que indica el término del proceso. Este se representa con un óvalo y se etiqueta como "Fin".

Este proceso en un diagrama de flujo ayudaría a visualizar de manera clara y ordenada los pasos necesarios para determinar si un número es par o impar.

Los diagramas de flujo son muy útiles en la fase de diseño y análisis de sistemas, debido a que proporcionan una vista rápida y clara de un algoritmo o proceso, ayudando a identificar y corregir errores lógicos antes de la implementación en código. Además, facilitan la comunicación entre los miembros del equipo de desarrollo y pueden ser usados como guía durante la fase de codificación y documentación.

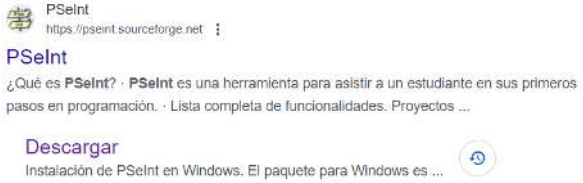
1.5.2 Herramienta PSeInt

PSeInt (Pseudo Intérprete) es una herramienta diseñada para ayudar a los estudiantes que están comenzando a aprender los conceptos de la programación, este software se utiliza para interpretar y ejecutar pseudocódigo.

PSeInt está diseñado específicamente para hablar en términos de estructuras de programación básicas como if-then-else, while, repeat-until, for, entre otros. La idea es que los estudiantes puedan enfocarse en aprender la lógica de la programación sin tener que preocuparse por la sintaxis específica de un lenguaje de programación.

La herramienta ofrece una interfaz donde los usuarios pueden escribir su pseudocódigo, y luego el programa puede ayudar a detectar errores lógicos, ejecutar el pseudocódigo paso a paso, y visualizar lo que está sucediendo con la ejecución en una manera que es fácil de entender para alguien que está aprendiendo. Además, PSeInt suele proporcionar sugerencias y ayuda en la construcción de algoritmos y, en algunas versiones, permite convertir pseudocódigo en código de algunos lenguajes de programación reales.

El programa se obtiene del Internet, buscando PSeInt.



Cuando ejecute el programa observara una ventana como la que se muestra a

ISBN: 978-631-6557-57-5

continuación:

En esta ventana debe ubicar el código y para ejecutar debe pulsar en el ícono (Ejecutar) de color verde.

Y para mostrar el diagrama de flujo debe pulsar en el ícono (Dibujar Diagrama de Flujo) antepenúltimo.

1.6 Ejercicios de aplicación

1.6.1 Ejercicios resueltos

1. ¿Cómo las innovaciones en la tecnología de almacenamiento, específicamente las transiciones de HDD a SSD y luego a NVMe, han impactado en el diseño y rendimiento de los sistemas operativos modernos?

Para resolver esta pregunta, primero es importante entender la evolución de las tecnologías de almacenamiento y cómo cada una ha influido en el rendimiento y diseño de los sistemas operativos.

1) **Paso del HDD a SSD:**

- Los Discos Duros (HDD) han sido la base del almacenamiento en computadoras durante décadas, utilizando platos magnéticos giratorios para leer y escribir datos. Sin embargo, su naturaleza mecánica limita la velocidad de acceso y la durabilidad.

- Las Unidades de Estado Sólido (SSD) marcaron una gran mejora en velocidad gracias a la ausencia de partes móviles, reduciendo significativamente los tiempos de arranque del sistema y la carga de aplicaciones. Los SSD usan memoria flash para almacenar

ISBN: 978-631-6557-57-5

datos, lo que permite un acceso más rápido a los archivos.

Impacto en los Sistemas Operativos:

Los sistemas operativos se han adaptado para aprovechar la velocidad de los SSD, mejorando no solo el tiempo de arranque sino también la rapidez en la ejecución de programas y en la respuesta del sistema. Además, se han optimizado las operaciones de lectura y escritura para minimizar el desgaste de las celdas de memoria, un aspecto crítico en los SSD.

2) **La Transición a NVMe:**

- NVMe (Non-Volatile Memory Express) es un protocolo diseñado para maximizar las ventajas de las unidades SSD basadas en memoria flash a través de interfaces como PCIe, ofreciendo velocidades aún mayores que las SSD tradicionales. Esto es posible gracias a una ruta de acceso más directa y eficiente a la CPU y a la capacidad de manejar una mayor cantidad de colas de comandos simultáneamente.

Impacto en los Sistemas Operativos:

- Para sacar partido de las capacidades de NVMe, los sistemas operativos han incorporado soporte nativo para este protocolo, lo que permite una gestión más eficiente del almacenamiento y un mejor rendimiento general. Esto ha tenido implicaciones significativas en el diseño de sistemas operativos, especialmente en la gestión de archivos y en la eficiencia del sistema de entrada/salida (I/O).

- Además, la adopción de NVMe ha estimulado el desarrollo de nuevas tecnologías de virtualización y ha permitido avances significativos en bases de datos y aplicaciones que requieren alta velocidad de acceso a datos, como el procesamiento de grandes volúmenes de datos (big data) y aplicaciones de aprendizaje automático.

2. ¿Cómo afecta el tamaño de la palabra (16, 32, 64 bits) al tipo de operaciones que puede realizar una computadora y su rendimiento general?

El tamaño de la palabra, que se refiere a la cantidad de bits que una CPU puede procesar de una vez, afecta significativamente al tipo de operaciones que puede realizar una computadora y a su rendimiento general de varias maneras:

d) **Capacidad de Procesamiento y Velocidad:** Cuanto mayor es el tamaño de la palabra (16, 32, 64 bits o más), mayor es la cantidad de información que la CPU puede manejar simultáneamente. Esto significa que las operaciones, especialmente las que involucran

ISBN: 978-631-6557-57-5

grandes cantidades de datos, pueden completarse más rápidamente. Por ejemplo, en una arquitectura de 64 bits, la CPU puede procesar datos de 64 bits en un solo ciclo de reloj, a diferencia de una CPU de 32 bits, que requeriría dos ciclos para procesar la misma cantidad de datos.

e) **Manejo de Memoria:** El tamaño de la palabra también influye en la cantidad máxima de memoria que la computadora puede utilizar eficazmente. Una arquitectura de 32 bits tiene un espacio de direccionamiento de memoria teóricamente limitado a 4 GB (2^32 posiciones de memoria), mientras que una arquitectura de 64 bits expande enormemente este límite a 16 exabytes (2^64 posiciones de memoria). Esto permite que sistemas con arquitecturas de 64 bits manejen aplicaciones más grandes y más complejas, así como grandes bases de datos y archivos.

f) **Compatibilidad con Software:** Algunas aplicaciones están diseñadas específicamente para aprovechar las capacidades de las arquitecturas de 32 o 64 bits. Las aplicaciones optimizadas para 64 bits pueden realizar cálculos más complejos y manejar más datos a la vez que sus contrapartes de 32 bits. Sin embargo, las aplicaciones de 64 bits no pueden ejecutarse en un sistema operativo de 32 bits debido a la incompatibilidad en el manejo y procesamiento de datos.

g) **Rendimiento en Aplicaciones Específicas:** El rendimiento en aplicaciones específicas, como el procesamiento de gráficos, simulaciones y aplicaciones científicas, puede mejorar significativamente con tamaños de palabra más grandes. Esto se debe a que pueden procesar cantidades mayores de datos por ciclo de instrucción, lo que es esencial para operaciones que requieren un alto grado de precisión numérica o manejan grandes volúmenes de datos.

h) **Eficiencia Energética:** Aunque un tamaño de palabra más grande puede implicar un mayor consumo de energía por ciclo de reloj debido al aumento en el procesamiento de datos, las mejoras en la tecnología de fabricación de chips y la optimización de software pueden mitigar este efecto. Además, al completar las tareas más rápidamente, puede ser posible que la CPU regrese antes a un estado de baja energía, mejorando la eficiencia energética general del sistema.

3. Explique cómo las operaciones lógicas (AND, OR, NOT, XOR) permiten a las

ISBN: 978-631-6557-57-5

computadoras realizar cálculos y tomar decisiones. Proporcionen ejemplos de cómo se podrían usar estas operaciones en programas informáticos.

Las operaciones lógicas (AND, OR, NOT, XOR) son fundamentales para el procesamiento de datos en las computadoras, permitiéndoles realizar cálculos y tomar decisiones basadas en condiciones lógicas. Estas operaciones trabajan a nivel de bits, el nivel más básico de información en la computación, donde el valor es 0 o 1 (Ormaza, 2020). Veamos cómo cada una de estas operaciones contribuye al funcionamiento de las computadoras y algunos ejemplos de su uso:

1) **AND (Y):** Esta operación devuelve 1 solo si ambos bits de entrada son 1. Si cualquiera de los bits es 0, el resultado es 0. Es útil para verificar si varias condiciones son verdaderas al mismo tiempo.

- Ejemplo en programación: Determinar si un número es par y mayor que 100. Podría usarse la operación AND para comprobar si el número cumple ambas condiciones simultáneamente.

2) **OR (O):** Devuelve 1 si al menos uno de los bits de entrada es 1. Es útil para verificar si alguna de varias condiciones es verdadera.

- Ejemplo en programación: Determinar si un usuario tiene permiso para acceder a un recurso. Se podría usar la operación OR para comprobar si el usuario tiene alguno de varios roles permitidos.

3) **NOT (NO):** Esta operación invierte el bit de entrada; si el bit es 1, devuelve 0, y viceversa. Es útil para cambiar el estado de una condición o para crear condiciones negativas.

- Ejemplo en programación: Cambiar el estado de una bandera de control. Si tienes una bandera que indica si un mensaje debe mostrarse (1) o no (0), usar NOT invertiría este estado.

4. **XOR (O exclusivo):** Devuelve 1 solo si los bits de entrada son diferentes. Si ambos bits son iguales (ambos 0 o ambos 1), el resultado es 0. Es útil para verificar que dos condiciones no sean iguales entre sí.

- Ejemplo en programación: Detectar cambios en el estado. Si se compara el estado anterior y el estado actual de un bit con XOR, el resultado será 1 solo si el estado ha cambiado, lo cual es útil para detectar flancos de subida o bajada en señales o para

ISBN: 978-631-6557-57-5

implementar algoritmos de cifrado.

Aplicaciones Prácticas:

-Control de Flujo: Las operaciones lógicas se utilizan en estructuras de control de flujo, como las instrucciones if y while, para combinar múltiples condiciones y decidir el camino que debe tomar el programa.

-Manipulación de Bits: En tareas de bajo nivel, como la programación de controladores de dispositivos, la manipulación directa de bits mediante operaciones lógicas permite configurar y leer estados de hardware de manera eficiente.

-Seguridad y Cifrado: Las operaciones lógicas, especialmente XOR, juegan un papel crucial en algoritmos de cifrado, donde se utilizan para alterar los datos de manera predecible y reversible solo si se conoce la clave correcta.

4. Convertir el número decimal 23 a binario.

División	Cociente	Residuo
23/2	11	1
11/2	5	1
5/2	2	1
2/2	1	0
½	0	1

Se procede a leer los residuos de abajo hacia arriba y se tiene 10111, que son 5 bits, por lo tanto, se deduce que el número 23 ocupa 2 bytes y cada byte está formado por cuatro bits, como existen 5 bits se completa con ceros el byte y se obtiene la representación binaria del número decimal 23 como 0001 0111.

5. Codifica la palabra "Universidad" en ASCII y presenta los valores numéricos correspondientes.

La palabra "Universidad" codificada en ASCII corresponde a los siguientes valores numéricos: 85, 110, 105, 118, 101, 114, 115, 105, 100, 97, 100. Cada número representa el valor ASCII de cada letra en la palabra.

6. Representa la frase "Buen día " utilizando los códigos Unicode correspondientes.

La frase representada utilizando los códigos Unicode correspondientes es:

B: U+0042

ISBN: 978-631-6557-57-5

u: U+0075

e: U+0065

n: U+006E

(espacio): U+0020

d: U+0064

í: U+00ED

a: U+0061

(espacio): U+0020

(sol con cara): U+1F31E

Cada "U+XXXX" representa el valor Unicode del carácter en la frase

7. Convierte los siguientes colores de RGB a HEX: RGB(34,139,34), RGB(70,130,180).

Para RGB(34,139,34): #228B22

Para RGB(70,130,180): #4682B4

Cada código HEX representa la misma combinación de colores que la correspondiente notación RGB

8. Crear un programa en Python que pida al usuario ingresar un número entero y luego lo convierta a un tipo de dato flotante. Mostrar ambos valores.

1) Pseudocódigo

```
Proceso Convertir_Entero_A_Flotante
    Definir numero_entero, numero_flotante Como Real
    Escribir "Por favor, ingresa un número entero: "
    Leer numero_entero
    numero_flotante <- numero_entero
    Escribir "Número entero: ", numero_entero
    Escribir "Número flotante: ", numero_flotante
FinProceso
```

Interpretación del pseudocódigo:

Definir dos variables de tipo Real, numero_entero y numero_flotante.

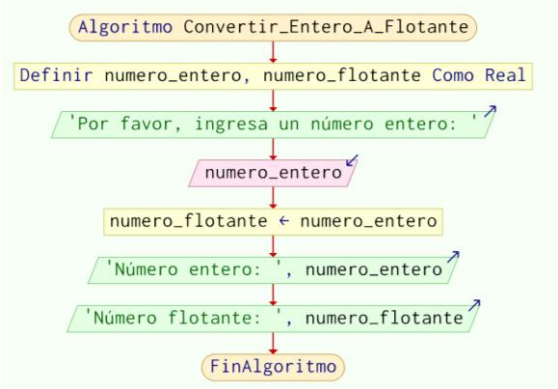
Solicitar al usuario que ingrese un número entero y lo guarda en numero_entero.

Asignar el valor de numero_entero a numero_flotante.

Mostrar ambos valores en pantalla.

ISBN: 978-631-6557-57-5

2) **Diagrama de Flujo**



Interpretación

Inicio del Algoritmo: Se define el comienzo del algoritmo con el nombre "Convertir_Entero_A_Flotante".

Definición de Variables: Se definen dos variables, numero_entero y numero_flotante, ambas de tipo Real. Aunque se va a ingresar un número entero, se define como Real para facilitar la conversión posterior.

Entrada de Datos: Se solicita al usuario que ingrese un número entero a través de un mensaje "Por favor, ingresa un número entero: ". Este valor se guarda en la variable numero_entero.

Asignación de Valor: Se realiza la asignación del valor de numero_entero a numero_flotante. Dado que ambas variables son de tipo Real, esta asignación es directa y convierte el valor a flotante si es necesario.

Salida de Datos: Se muestran dos mensajes:

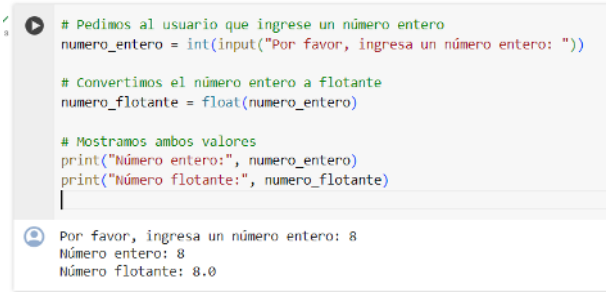
- El primero muestra el texto "Número entero: " seguido del valor de la variable numero_entero.
- El segundo muestra el texto "Número flotante: " seguido del valor de la variable numero_flotante.

Fin del Algoritmo: Se marca el fin del algoritmo con "FinAlgoritmo".

Este flujo permite al usuario ver el número que ingresó y su correspondiente representación en formato flotante.

3) **Implementación en Python**

ISBN: 978-631-6557-57-5



```
# Pedimos al usuario que ingrese un número entero
numero_entero = int(input("Por favor, ingresa un número entero: "))

# Convertimos el número entero a flotante
numero_flotante = float(numero_entero)

# Mostramos ambos valores
print("Número entero:", numero_entero)
print("Número flotante:", numero_flotante)
```

```
Por favor, ingresa un número entero: 8
Número entero: 8
Número flotante: 8.0
```

El código que se muestra en Python realiza lo siguiente:

a) Pedir al usuario que ingrese un número entero:

- Se muestra el mensaje "Por favor, ingresa un número entero:".
- Se espera que el usuario ingrese un valor, y dicho valor se convierte a un entero utilizando la función int.
- El número entero se almacena en la variable numero_entero.

b) Convertir el número entero a flotante:

- Se convierte el valor almacenado en numero_entero a un número flotante utilizando la función float.
- El número flotante resultante se almacena en la variable numero_flotante.

c) Mostrar ambos valores, el entero y el flotante:

- Se imprime el valor de numero_entero con el mensaje "Número entero:".
- Se imprime el valor de numero_flotante con el mensaje "Número flotante:".

Esto muestra que el número entero se ha convertido correctamente a su equivalente flotante, que es 8.0.

9. Hacer un programa en JavaScript en el cual dado un número flotante presente el número flotante y la parte entera del número.

1) Pseudocódigo

```
Proceso Convertir_Flotante_A_Entero
    Definir numero_flotante Como Real
    Definir numero_entero Como Entero
    Escribir "Por favor, ingresa un número flotante: "
    Leer numero_flotante
    numero_entero <- trunc(numero_flotante)
    Escribir "Número flotante: ", numero_flotante
    Escribir "Parte entera del número: ", numero_entero
```

ISBN: 978-631-6557-57-5

FinProceso

Interpretación del Pseudocódigo

Proceso Convertir_Flotante_A_Entero: Esta es la declaración del inicio del proceso, que hemos llamado "Convertir_Flotante_A_Entero".

Definir numero_flotante Como Real: Aquí se declara una variable llamada numero_flotante, la cual será de tipo Real. Esto significa que puede almacenar números con decimales.

Definir numero_entero Como Entero: Se declara otra variable llamada numero_entero de tipo Entero. Esta variable solo podrá almacenar números sin parte decimal.

Escribir "Por favor, ingresa un número flotante: ": Con esta línea se muestra un mensaje en pantalla para que el usuario sepa que debe ingresar un número flotante.

Leer numero_flotante: Esta línea sirve para leer un número del usuario y almacenarlo en la variable numero_flotante.

numero_entero <- trunc(numero_flotante): La función trunc toma la parte entera del número flotante y la almacena en la variable numero_entero. Por ejemplo, si numero_flotante es 3.1416, entonces numero_entero será 3.

Escribir "Número flotante: ", numero_flotante: Esta línea imprime en pantalla el mensaje "Número flotante: " seguido del valor de la variable numero_flotante que el usuario ingresó originalmente.

Escribir "Parte entera del número: ", numero_entero: Finalmente, se muestra en pantalla el mensaje "Parte entera del número: " junto con el valor de numero_entero, que es la parte entera del número que se ingresó.

FinProceso: Esta línea indica el fin del proceso que hemos definido.

El propósito general del pseudocódigo es recibir un número con decimales del usuario, separar y mostrar tanto el número flotante completo como solo la parte entera de este número.

2) **Diagrama de flujo**

El diagrama de flujo representa el siguiente proceso:

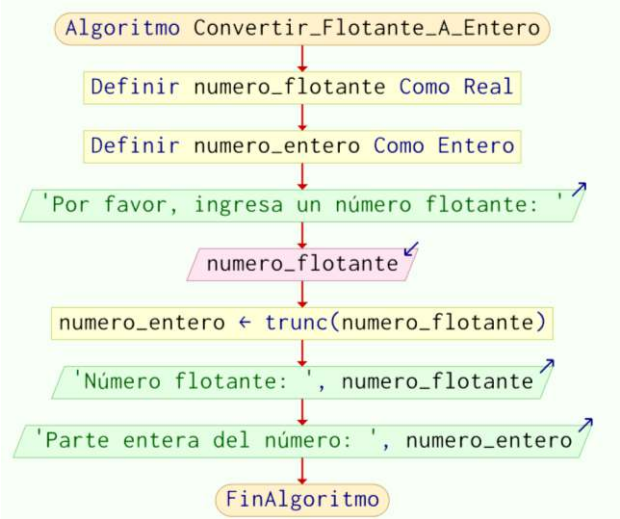
Inicio del Algoritmo: El proceso se llama "Convertir_Flotante_A_Entero".

Definición de Variables:

ISBN: 978-631-6557-57-5

- numero_flotante como Real: Se declara una variable para almacenar el número con decimales que introducirá el usuario.

- numero_entero como Entero: Se declara una variable para guardar la parte entera del número flotante.



Solicitud de Entrada al Usuario:

- Se le pide al usuario que ingrese un número flotante mostrando el mensaje "Por favor, ingresa un número flotante: ".

Entrada del Usuario:

- El número flotante ingresado por el usuario se asigna a la variable numero_flotante.

Conversión a Entero:

- Se usa la función trunc para convertir el numero_flotante en un entero, descartando la parte decimal, y el resultado se asigna a numero_entero.

Mostrar Resultados:

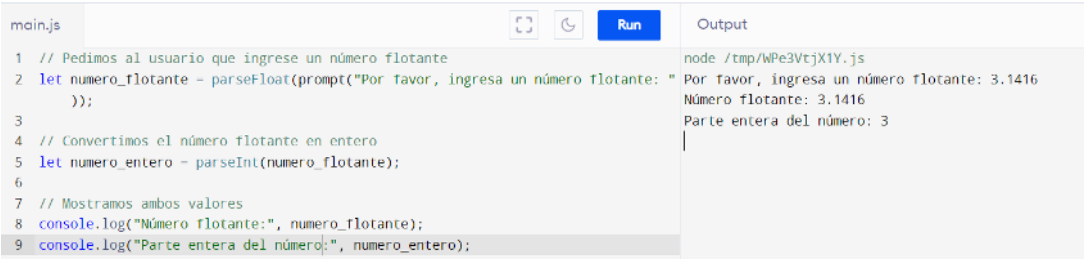
- Se muestra en pantalla el numero_flotante con el mensaje "Número flotante: ".

- Luego se muestra el numero_entero con el mensaje "Parte entera del número: ".

Fin del Algoritmo: Se indica que el proceso ha terminado con "FinAlgoritmo".

3) Implementación

ISBN: 978-631-6557-57-5



En JavaScript, let es una palabra clave que se utiliza para declarar variables con un alcance limitado a un bloque de código, una expresión o una instrucción. Esto significa que la variable solo puede ser accedida y utilizada dentro del bloque en el que se ha definido.

-Utiliza prompt() para pedir al usuario que ingrese un número entero.

-Utiliza parseFloat() para asegurarse de que la entrada sea tratada como un número flotante.

Convierte ese número flotante a un número entero con parseInt().

Finalmente, imprimirá ambos valores en la consola del navegador con console.log().

1.6 Ejercicios propuestos

1 ¿Cómo describiría la diferencia entre hardware y software a alguien que no está familiarizado con las computadoras? Incluya ejemplos de cada uno.

2 ¿Cuáles son las funciones principales de la Unidad Central de Procesamiento (CPU) en una computadora?

3 ¿Cómo impacta la velocidad de la CPU en el rendimiento general de una computadora?

4 Explicar la diferencia entre memoria RAM volátil y almacenamiento secundario no volátil. ¿Por qué es importante esta distinción?

5 Describa las diferencias entre DRAM, SRAM, y SDRAM. ¿En qué situaciones es preferible usar cada tipo?

6 ¿Cuáles son las ventajas y desventajas de un disco duro (HDD) comparado con una unidad de estado sólido (SSD)?

7 ¿Cuál es el papel de la placa base en una computadora y cómo afecta a la compatibilidad entre diferentes componentes de hardware?

8 Delibera sobre la importancia de las tarjetas gráficas dedicadas en comparación

ISBN: 978-631-6557-57-5

con las capacidades gráficas integradas en algunas CPUs.

9 Enumera algunos dispositivos de entrada y salida y explica cómo facilitan la interacción entre el usuario y la computadora.

10 ¿Cómo ha evolucionado la tecnología de los dispositivos de entrada y salida en la última década?

11 ¿Por qué es importante el sistema de archivos en un sistema operativo? Compara dos sistemas de archivos y discute sus diferencias y aplicaciones.

12 Investiga cómo han evolucionado las computadoras desde sus inicios hasta la actualidad, enfocándote en los cambios tecnológicos clave y cómo estos han impactado en la sociedad.

13 Explora cómo las unidades de estado sólido (SSD) han transformado la industria de las computadoras, desde mejoras en el rendimiento hasta cambios en el diseño de sistemas operativos y aplicaciones.

14 Investiga cómo el hardware moderno, especialmente las GPUs, ha permitido avances en inteligencia artificial y aprendizaje automático. Examina casos de uso específicos.

15 Analiza cómo los sistemas operativos interactúan con el hardware de la computadora y gestionan los recursos del sistema. Puedes enfocarte en un sistema operativo específico o comparar varios.

16 Explora los principios básicos de la computación cuántica y su potencial para transformar campos como la criptografía, la simulación de materiales y la optimización de problemas.

17 Examina el impacto ambiental de la producción y desecho de hardware informático, así como las estrategias para mitigarlo, incluyendo reciclaje, diseño sostenible y legislación relevante.

18 Investiga sobre las últimas tecnologías en interfaces hombre-máquina, como realidad virtual, realidad aumentada, interfaces cerebro-computadora, y cómo estas podrían cambiar nuestra interacción con las computadoras.

19 Convertir el número decimal 111 a binario.

20 ¿Por qué las computadoras utilizan el sistema binario para procesar datos? Expliquen la importancia de los bits en este contexto.

ISBN: 978-631-6557-57-5

21 Compare y contraste los esquemas de codificación ASCII y Unicode. ¿Por qué es importante Unicode en el contexto global actual?

22 Explique cómo los formatos de codificación como JPEG o MP3 permiten que las computadoras manejen eficientemente datos complejos como imágenes y sonidos.

23 ¿Cuál es el valor RGB para el color blanco?

a) RGB(0, 0, 0)

b) RGB(255, 255, 255)

c) RGB(255, 0, 0)

d) RGB(0, 255, 0)

24 ¿Qué representa el código HEX #FF0000?

a) Azul

b) Verde

c) Rojo

d) Negro

25 35. Decodifica la siguiente secuencia de números ASCII en texto: 84 101 99 110 111 108 111 103 105 97.

26 Investiga y describe dos ejemplos de cómo se utilizaba la codificación ASCII en las primeras computadoras o dispositivos de comunicación.

27 ¿Cuántos caracteres puede codificar Unicode?

a) Más de 50,000

b) Más de 140,000

c) Más de 1,1 millones

d) Más de 2 millones

28 ¿Qué rango de Unicode incluye los emojis?

a) U+0000 a U+007F

b) U+0080 a U+07FF

c) U+0800 a U+FFFF

d) U+10000 a U+1FFFF

29 Explora y describe la evolución de Unicode y cómo ha permitido una mayor inclusividad de caracteres de distintas culturas y lenguajes en la informática.

30 ¿Cuál es el valor RGB para el color blanco?

ISBN: 978-631-6557-57-5

a) RGB(0, 0, 0)

b) RGB(255, 255, 255)

c) RGB(255, 0, 0)

d) RGB(0, 255, 0)

31 ¿Qué representa el código HEX #FF0000?

a) Azul

b) Verde

c) Rojo

d) Negro

32 Compara los modelos de color RGB y HEX con el modelo CMYK, explicando las diferencias principales y en qué contextos se utiliza cada uno.

33 Describe cómo las aplicaciones móviles han transformado las rutinas diarias de las personas. Proporciona al menos tres ejemplos.

34 ¿Cuál es la diferencia entre aplicaciones verticales y aplicaciones horizontales? Da un ejemplo de cada una.

35 Explica cómo las aplicaciones informáticas han contribuido al auge del trabajo remoto y la colaboración.

36 ¿De qué manera las aplicaciones informáticas han impactado en la educación?

37 Realiza una investigación sobre cómo las aplicaciones informáticas han transformado el comercio electrónico. Puedes enfocarte en aspectos como la personalización, la logística y la experiencia del usuario.

38 Investigar las diferencias en la representación interna de los tipos de datos numéricos en Python, Java, C++ y JavaScript y presenta tus hallazgos.

39 Investigar cómo se implementan los conceptos de POO (Programación Orientada a Objetos) en Java y C++ y realiza un comparativo de su sintaxis y funcionalidades.

40 Realice el código para la suma de 2 números en C++ y JavaScript.

41 Realizar el código en C++ para el operador lógico AND.

42 Realizar el código en JavScript para el operador lógico OR.

43 Realizar el código en Java para el operador lógico NOT.

44 Explica la diferencia entre una variable y una constante en programación.

ISBN: 978-631-6557-57-5

45 Escribe un programa en Python que pida al usuario su nombre y edad, luego imprime un mensaje que diga: "Hola [nombre], tienes [edad] años".

46 Crea un diagrama de flujo que represente el proceso de decisión para saber si un número ingresado por el usuario es par o impar.

47 Diseña un algoritmo en pseudocódigo que calcule el área de un círculo recibiendo el radio como dato de entrada y que devuelva el área como salida.

48 Analiza y compara los tipos de operaciones que se pueden realizar con los diferentes tipos de datos numéricos en Java y Python.

49 Escribir el pseudocódigo y realizar el diagrama de flujo para el cálculo del volumen de una pirámide.

50 Escribir el pseudocódigo y realizar el diagrama de flujo para la multiplicación de dos números enteros.

51 Escribir el pseudocódigo y realizar el diagrama de flujo para pedir al usuario una temperatura en grados Celsius y convertirla a grados Fahrenheit. La fórmula de conversión es Fahrenheit=Celsius × 5/9 + 32.

52 Escribir el pseudocódigo y realizar el diagrama de flujo para solicitar al usuario ingresar varios números (la cantidad debe ser definida por el usuario) y calcular la media aritmética de esos números.

53 Realizar el pseudocódigo y el diagrama de flujo para solicitar al usuario un número entero y calcular la factorial de ese número.

54 Realizar el pseudocódigo y el diagrama de flujo para pedir al usuario que ingrese una lista de números y luego encontrar el valor máximo y mínimo de esa lista.

ISBN: 978-631-6557-57-5

ISBN: 978-631-6557-57-5

CAPÍTULO 2

2 ESTRUCTURAS DE PROGRAMACION

La programación en Python, al igual que en otros lenguajes, se apoya en estructuras de control para el flujo lógico de un programa, haciéndolo más legible y fácil de modificar. Estas estructuras son importantes para desarrollar programas estructurados y se clasifican en:

Secuenciales: Donde las acciones se ejecutan una tras otra, en un flujo lineal.

Selectivas: Permiten tomar decisiones dentro del programa, ejecutando bloques de código específicos basados en condiciones.

Repetitivas: Facilitan la ejecución de un conjunto de instrucciones múltiples veces, bajo ciertas condiciones.

Las estructuras selectivas utilizan instrucciones como if y sus variantes (if-then, if-then-else), aunque Python no utiliza la instrucción switch tradicional, se pueden lograr funcionalidades similares con diccionarios o estructuras de control múltiple. Estas estructuras son fundamentales para abordar problemas de decisión y cálculo en la programación.

Para comprender y aplicar estas estructuras de control, se recurre frecuentemente a herramientas como los diagramas de flujo y el pseudocódigo, facilitando la visualización del flujo lógico de los programas.

El término "flujo de control" se refiere a la secuencia en la que se ejecutan las instrucciones de un programa. Por defecto, el flujo es secuencial, es decir, las instrucciones se ejecutan una después de la otra en el orden en que aparecen en el código. Sin embargo, las estructuras de selección y repetición alteran este flujo de manera controlada y predefinida, permitiendo a los programas reaccionar a condiciones variables y ejecutar instrucciones de forma no secuencial.

2.1 Estructuras secuenciales

Una estructura secuencial representa el flujo básico de ejecución donde cada acción o instrucción se lleva a cabo una tras otra en un orden lineal y predefinido. En este tipo de estructura, las tareas se organizan de manera que el resultado de una acción sirve como punto de partida o entrada para la siguiente, creando una cadena de procesos

ISBN: 978-631-6557-57-5

interconectados que avanzan paso a paso hacia la terminación del programa. Este enfoque secuencial asegura que haya una clara línea de progreso, donde cada etapa del proceso depende de la correcta ejecución de la anterior, garantizando que el programa fluya de manera ordenada desde su inicio hasta su finalización.

Dentro de una estructura secuencial la "entrada" se refiere a los datos o información que se introducen en el programa, ya sea por el usuario, a través de archivos, o cualquier otro medio externo. Estos datos son procesados secuencialmente por el programa, pasando a través de diversas operaciones o cálculos definidos por el desarrollador. Finalmente, el proceso concluye con una "salida", que es el resultado de todas las operaciones llevadas a cabo por el programa. Esta salida puede tomar múltiples formas, como la visualización de resultados en pantalla, la generación de un archivo de datos, o incluso el envío de información a otros sistemas o módulos para su posterior procesamiento.

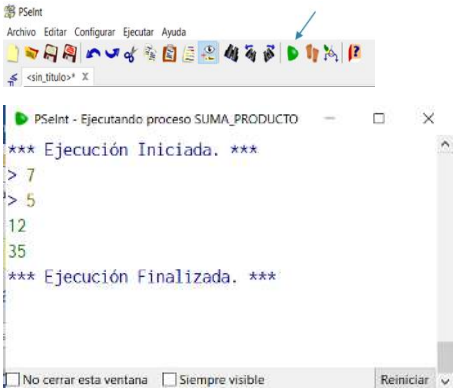
La simplicidad de la estructura secuencial la hace ideal para programas y algoritmos que siguen un flujo lineal de tareas, donde la complejidad y la necesidad de tomar decisiones basadas en condiciones no son predominantes. Sin embargo, incluso en programas más complejos que requieren de estructuras de control adicionales como las selectivas o repetitivas, la base de su ejecución sigue siendo, en esencia, una serie de pasos secuenciales.

Ejemplo 1: Calcular la suma y producto de dos números

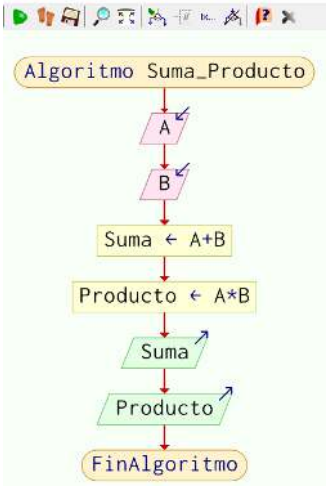
Pseudocódigo

Para verificar que el pseudocódigo está bien escrito pulsar el ícono Ejecutar. Le va a solicitar que introduzca los valores de A y B.

ISBN: 978-631-6557-57-5



Luego se procede a generar el diagrama de flujo



La implementación en Python

```
# Solicitar al usuario que ingrese el primer número
A = float(input("Introduce el primer número: "))

# Solicitar al usuario que ingrese el segundo número
B = float(input("Introduce el segundo número: "))

# Calcular la suma de los dos números
suma = A + B

# Calcular el producto de los dos números
producto = A * B

# Imprimir los resultados
print("La suma de los números es:", suma)
print("El producto de los números es:", producto)
```

```
Introduce el primer número: 7
Introduce el segundo número: 5
La suma de los números es: 12.0
El producto de los números es: 35.0
```

ISBN: 978-631-6557-57-5

Este script es un ejemplo claro de una estructura secuencial, donde cada tarea se realiza una tras otra: primero se solicitan los datos, luego se realizan los cálculos, y finalmente se muestran los resultados.

Un script es un archivo que contiene código Python destinado a ser ejecutado directamente. Estos archivos suelen tener la extensión .py (para trabajar con Python fuera de línea en la PC puede descargar la última versión de Python e instalar, una breve explicación al respecto se indica más adelante) y pueden contener secuencias de comandos (instrucciones), definiciones de funciones, y también pueden incluir clases, que son ejecutadas por el intérprete de Python para realizar una tarea específica.

Números de punto flotante (float): Representa un número flotante, es decir, un número que puede tener una parte fraccional (decimal) además de su parte entera, ejemplo: 3.1416, -3.0. Los números flotantes se utilizan en Python para realizar cálculos que requieren precisión más allá de los números enteros, como operaciones matemáticas que involucran fracciones o cuando se trabaja con medidas que no son enteras (Matuszek, 2022).

Enteros (int): Representan números enteros positivos o negativos sin parte decimal. Por ejemplo, -3, 0, 1024 son todos enteros. Python puede manejar enteros de tamaño arbitrario, es decir, no hay un límite fijo en su valor, más allá de las restricciones de memoria de tu sistema.

Números complejos (complex): Son aquellos que tienen una parte real y una parte imaginaria, y se utilizan en campos que requieren cálculos complejos como la ingeniería eléctrica, la física, y otros dominios científicos. Se representan como real + imagj, donde real es la parte real e imag es la parte imaginaria. Por ejemplo, 3+4j es un número complejo.

Decimal (decimal.Decimal): Proporcionado por el módulo decimal, este tipo es útil para hacer cálculos financieros y monetarios precisos, donde se requiere que las operaciones sean exactas y no sufran los problemas de precisión de los flotantes.

Fracciones (fractions.Fraction): Proporcionado por el módulo fractions, este tipo representa números racionales (fracciones) utilizando dos enteros, el numerador y el denominador.

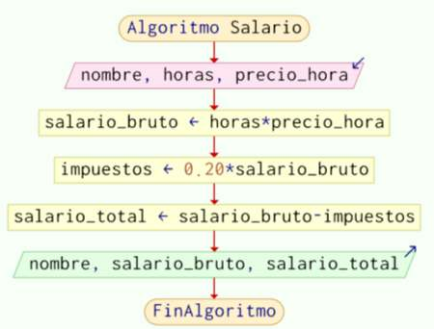
ISBN: 978-631-6557-57-5

Ejemplo 2: Calcular el salario neto de un trabajador en función del número de horas trabajadas, precio de la hora de trabajo y, considerando unos descuentos fijos, el sueldo bruto en concepto de impuestos (20%).

Pseudocódigo

```
Algoritmo Salario
    leer nombre,horas,precio_hora
    salario_bruto ← horas * precio_hora
    impuestos ← 0.20 * salario_bruto
    salario_total ← salario_bruto - impuestos
    escribir nombre,salario_bruto,salario_total
FinAlgoritmo
```

Diagrama de flujo

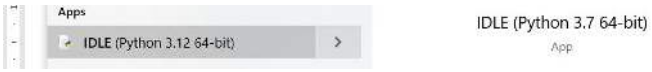


La implementación se realiza utilizando Python. En este caso se va emplear el Python instalado en la PC, para lo cual se debe buscar la versión de Python en el Internet.

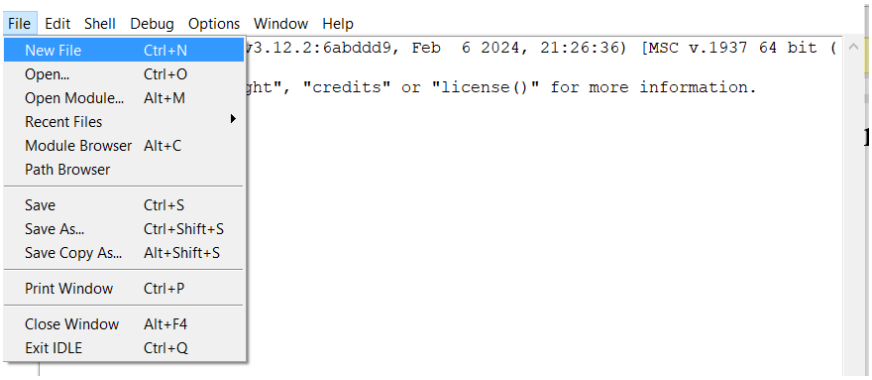
Proceder a descargar la versión de Python con la que desee trabajar

Una vez descargado Python, en la ventana de Windows de su PC de la barra de tareas buscar el idle y dar click en el idle de la versión que se vaya a utilizar.

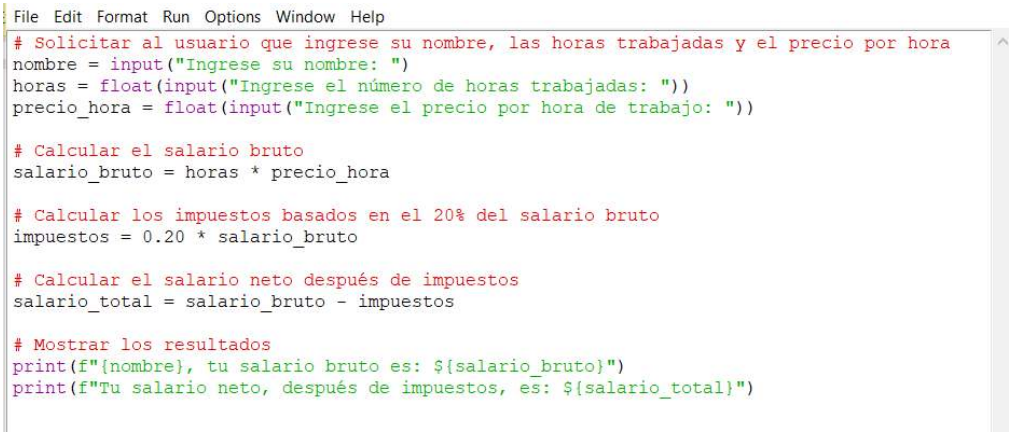
ISBN: 978-631-6557-57-5



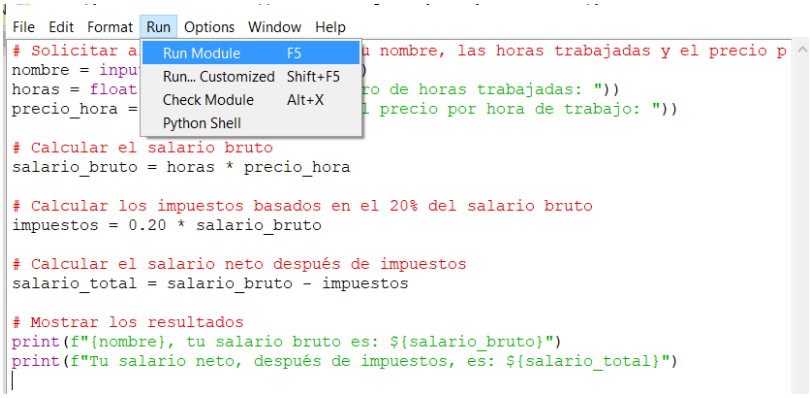
A continuación, aparece la ventana de Python, ir a File/New File para crear una nueva ventana para escribir el código y cuando requiera grabar el archivo ir a File/Save y ubicar el nombre del archivo con extensión .py (ejemplo: salario.py).



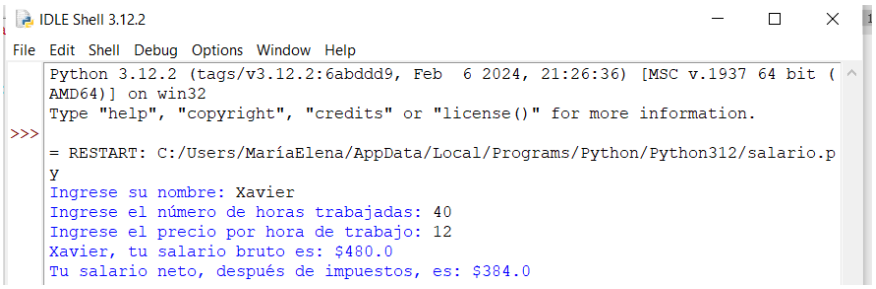
Escribir el código



Para ejecutar el programa ir a Run/Run Module



Aquí se observa el resultado de correr el programa.



Interpretación del código:

El código en Python solicita al usuario que proporcione tres partes de información: su nombre, el número de horas que ha trabajado y su tarifa de pago por hora. Luego, realiza los siguientes pasos:

1. Entrada de Datos:

- Solicita al usuario que escriba su nombre y lo almacena en la variable nombre.

- Solicita al usuario que ingrese la cantidad de horas trabajadas y convierte esa entrada a un número de punto flotante (decimal) y lo almacena en la variable horas.

- Solicita al usuario el precio que se paga por hora de trabajo, convierte esta entrada a un número de punto flotante y lo almacena en la variable precio_hora.

2. Cálculo del Salario Bruto:

- Calcula el salario bruto multiplicando el número de horas trabajadas (horas) por el precio por hora (precio_hora) y almacena el resultado en salario_bruto.

3. Cálculo de Impuestos:

ISBN: 978-631-6557-57-5

- Calcula los impuestos como el 20% del salario bruto. Esto se hace multiplicando salario_bruto por 0.20 y el resultado se almacena en la variable impuestos.

4. Cálculo del Salario Neto:

- Determina el salario neto, que es el salario después de descontar los impuestos, restando impuestos de salario_bruto y almacena el resultado en salario_total.

5. Presentación de Resultados:

- Imprime el salario bruto, mostrando el nombre del usuario y el monto del salario bruto calculado.
- Imprime el salario neto, es decir, el salario después de impuestos.

2.2 Estructuras selectivas

Las estructuras selectivas permiten incorporar la toma de decisiones lógicas dentro de los programas. Estas estructuras evalúan condiciones para decidir qué secuencia de instrucciones ejecutar. La esencia de una estructura selectiva es la capacidad de dirigir el flujo de ejecución del programa en diferentes direcciones, basándose en el resultado de la evaluación de una o más condiciones.

La utilización de estructuras selectivas es indispensable cuando se enfrenta a situaciones en las que el comportamiento del algoritmo debe variar dependiendo de los datos de entrada o del estado actual del sistema. En esencia, permiten que un algoritmo actúe de manera diferente ante diferentes situaciones, aumentando significativamente su flexibilidad y aplicabilidad.

Tipos de Estructuras Selectivas

1. **Simples:** Son la forma más básica de estructura selectiva, donde se evalúa una condición y, si esta se cumple (es decir, es verdadera), se ejecuta un conjunto específico de instrucciones. Si la condición es falsa, el algoritmo continúa su ejecución sin entrar en el bloque de código asociado a la condición verdadera. En pseudocódigo se representa típicamente con las palabras clave si (if) y entonces (then).

Pseudocódigo

```
si (condición) entonces
  // Instrucciones a ejecutar si la condición es verdadera
```

ISBN: 978-631-6557-57-5

fin_si

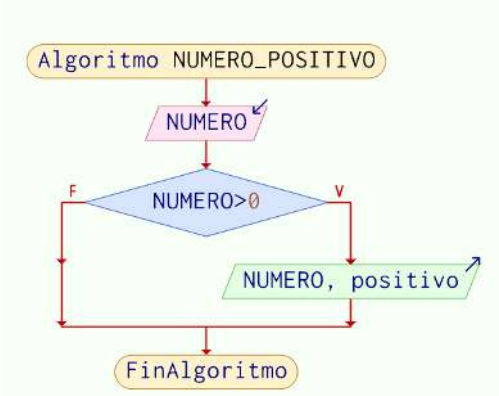
En Python, el equivalente de **IF THEN** es simplemente **IF**

Ejemplo: Establecer si un número ingresado por teclado es positivo

Pseudocódigo

```
Algoritmo Número_Positivo
    Leer número
    Si número > 0
    Escribir número positivo
    FinSi
    FinAlgoritmo
```

Diagrama de flujo



Implementación

```
# Solicitar al usuario que ingrese un número
numero = int (input("Por favor, ingresa un número: "))
# Evaluar si el número es positivo
if numero > 0:
    print("El número es positivo.")
```

```
Por favor, ingresa un número: 4
El número es positivo.
```

2. **Dobles:** Amplían las estructuras simples al introducir una acción alternativa que se ejecuta cuando la condición evaluada resulta ser falsa. Esto se logra mediante el uso de una cláusula adicional, usualmente denominada **si_no** (else) en pseudocódigo. De esta forma, se pueden manejar dos flujos de ejecución diferentes: uno para cuando la condición es verdadera y otro para cuando es falsa.

ISBN: 978-631-6557-57-5

Pseudocódigo

```
si (condición) entonces
   // Instrucciones si la condición es verdadera
si_no
   // Instrucciones si la condición es falsa
fin_si
```

Ejemplo: Ingresar la edad de una persona y determinar si es mayor o menor de edad

Pseudocódigo

```
Algoritmo Mayor_o_Menor_Edad
Leer edad
Si edad ≥ 18 Entonces
    Escribir Mayor de edad
SiNo
    Escribir Menor de edad
FinSi
FinAlgoritmo
```

Diagrama de flujo

Implementación

```
# Solicitar al usuario que ingrese su edad
edad = int(input("Por favor, ingresa tu edad: "))

# Evaluar si la persona es mayor o menor de edad
if edad >= 18:
    print("Mayor de edad")
else:
    print("Menor de edad")
```

```
Por favor, ingresa tu edad: 9
Menor de edad
```

Este código comienza pidiendo al usuario que ingrese su edad a través de la consola y utiliza la función **input** para leer esta entrada. Como **input** devuelve una cadena de texto (string), utilizamos **int()** para convertir esa cadena en un número entero, lo que nos permite hacer la comparación con el número 18. Después, se usa una estructura condicional **if-else** para evaluar si la **edad** ingresada cumple la condición de ser igual o superior a 18, imprimiendo "Mayor de edad" si se cumple la condición, o "Menor de edad" si no se cumple.

Una cadena de texto, conocida como **string** en inglés, es una secuencia de caracteres utilizada para representar texto en la programación de computadoras. Los caracteres pueden ser letras, números, símbolos o espacios. Las cadenas se utilizan en casi todos los lenguajes de

ISBN: 978-631-6557-57-5

programación modernos para almacenar y manipular texto, como nombres, mensajes y contenido de archivos de texto.

3. **Múltiples:** Son necesarias cuando hay más de dos posibles flujos de ejecución basados en múltiples condiciones. En estos casos, se pueden utilizar estructuras como "según sea" o "switch-case" en algunos lenguajes de programación, que permiten evaluar una expresión y ejecutar diferentes bloques de instrucciones dependiendo del resultado de esa evaluación. Esta estructura es particularmente útil para simplificar el código cuando se deben considerar múltiples alternativas.

Pseudocódigo

```
según sea (expresión)
   caso valor1:
      // Instrucciones para el caso 1
   caso valor2:
      // Instrucciones para el caso 2
   ...
   caso valorN:
      // Instrucciones para el caso N
   caso contrario:
      // Instrucciones si ningún caso anterior coincide
fin_según
```

ISBN: 978-631-6557-57-5

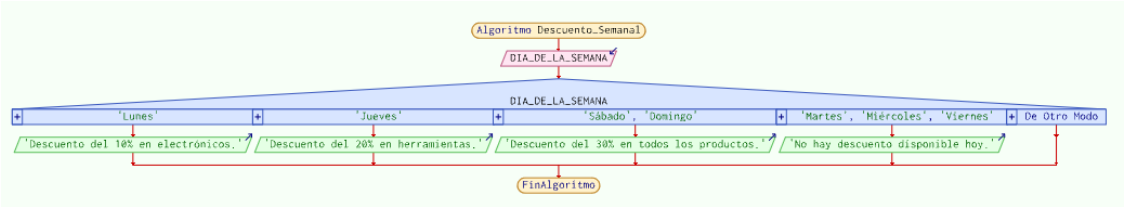
Ejemplo: descuentos según el día de la semana.

Pseudocódigo

```
Algoritmo Descuento_Semanal
    Leer día_de_la_semana
Según día_de_la_semana
Caso "Lunes":
    Escribir "Descuento del 10% en electrónicos."
Caso "Jueves":
    Escribir "Descuento del 20% en herramientas."
Caso "Sábado", "Domingo":
    Escribir "Descuento del 30% en todos los productos."
Caso "Martes", "Miércoles", "Viernes":
    Escribir "No hay descuento disponible hoy."
Fin Según
FinAlgoritmo
```

En este ejemplo, el algoritmo comienza leyendo el día de la semana. Luego, utiliza una estructura de control **Según** para evaluar el valor de **día_de_la_semana** y seleccionar el bloque de código correspondiente a ese día. Cada **Caso** representa un día de la semana, y para cada uno, se especifica un descuento particular aplicable a ciertas categorías de productos.

Diagrama de flujo



ISBN: 978-631-6557-57-5

Implementación

```
# Solicitar al usuario que ingrese el día de la semana
dia_de_la_semana = input("Ingrese el día de la semana: ")

# Convertir a minúsculas para estandarizar la entrada del usuario
dia_de_la_semana = dia_de_la_semana.lower()

# Aplicar los descuentos según el día de la semana
if dia_de_la_semana == "lunes":
    print("Descuento del 10% en electrónicos.")
elif dia_de_la_semana == "jueves":
    print("Descuento del 20% en herramientas.")
elif dia_de_la_semana == "sábado" or dia_de_la_semana == "domingo":
    print("Descuento del 30% en todos los productos.")
elif dia_de_la_semana in ["martes", "miércoles", "viernes"]:
    print("No hay descuento disponible hoy.")
else:
    print("El día ingresado no es válido.")
```

```
Ingrese el día de la semana: Martes
No hay descuento disponible hoy.
```

En este script se solicita al usuario que ingrese el día de la semana a través de la consola, luego se convierte la entrada del usuario a minúsculas para facilitar la comparación, independientemente de cómo el usuario haya ingresado el texto (mayúsculas, minúsculas, o una mezcla de ambas).Se usa una serie de sentencias **if-elif-else** para comparar el día ingresado con los diferentes casos y determinar el descuento aplicable. En caso de que el día de la semana sea sábado o domingo, aplicamos un descuento del 30% en todos los productos. Para los días martes, miércoles y viernes, indicamos que no hay descuentos disponibles. Si el día ingresado no coincide con ningún caso esperado, el programa informa que el día no es válido.

El **elif** es la abreviatura de **else if** , se traduce como "en caso contrario, si". Se utiliza para evaluar otra condición en caso de que la condición anterior en la cadena **if-elif-else** sea falsa. Se puede tener tantas sentencias **elif** como se necesite para evaluar condiciones adicionales.

La correcta utilización de las estructuras selectivas es importante para el desarrollo de algoritmos eficientes y efectivos, permitiendo que los programas se adapten y respondan adecuadamente a diferentes situaciones y datos de entrada. Su flexibilidad y potencial

ISBN: 978-631-6557-57-5

para gestionar complejidades hacen de las estructuras selectivas una herramienta indispensable de cualquier programador.

2.3 Estructuras de repetición

Las estructuras repetitivas, permiten ejecutar un conjunto de instrucciones de manera iterativa, es decir, repetidas veces, bajo ciertas condiciones establecidas. Esta capacidad de repetición se aprovecha ampliamente en la computación para realizar tareas de manera eficiente, como procesar colecciones de datos, realizar cálculos iterativos, o manejar eventos dinámicos en aplicaciones de software.

Conceptos Clave de las Estructuras Repetitivas

Iteración: Es la repetición de un bloque de instrucciones en un programa.

Bucle o Lazo: Es la estructura que permite la iteración.

Condición de Continuación: Es la condición que se evalúa para determinar si el bucle debe continuar ejecutándose o detenerse.

Tipos de Estructuras Repetitivas

Existen varias estructuras repetitivas, cada una con características que las hacen adecuadas para diferentes situaciones:

1. Mientras (while): Repite un bloque de instrucciones mientras la condición especificada sea verdadera. La evaluación de la condición ocurre al principio del bucle, lo que significa que es posible que el cuerpo del bucle no se ejecute si la condición inicial es falsa.

ISBN: 978-631-6557-57-5

Ejemplo: Realizar la suma de los números que se indica. Terminar la operación cuando se pide sumar 0.

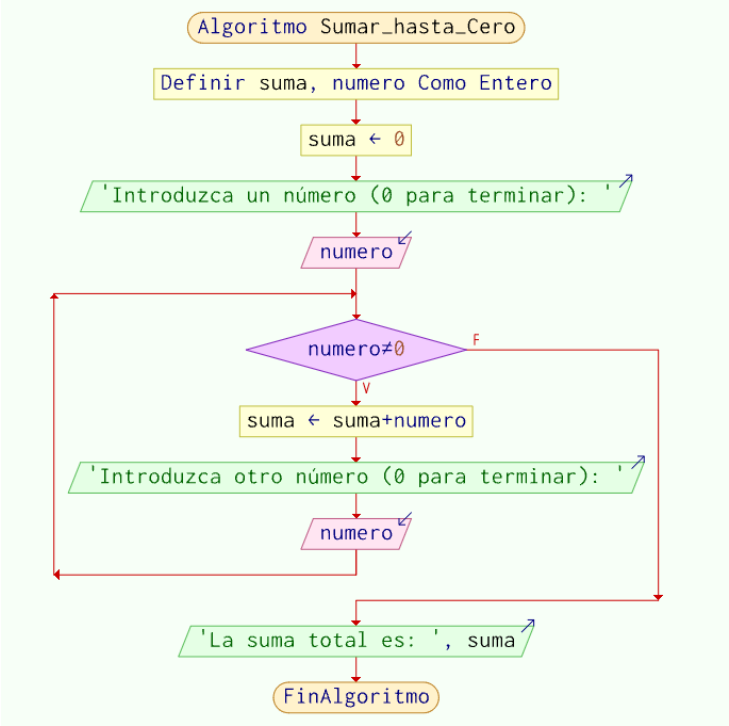
Pseudocódigo

```
Algoritmo Sumar_hasta_Cero
    Definir suma, numero Como Entero;
    suma ← 0;
    Escribir "Introduzca un número (0 para terminar): ";
    Leer numero;

    Mientras numero ≠ 0 Hacer
        suma ← suma + numero;
        Escribir "Introduzca otro número (0 para terminar): ";
        Leer numero;
    Fin Mientras

    Escribir "La suma total es: ", suma;
Fin Algoritmo
```

Este pseudocódigo se inicia definiendo las variables **suma** y **numero** como enteros, y se establece **suma** en 0. Luego, se usa un bucle **Mientras** para pedir al usuario que introduzca números. El bucle se repite mientras el número introducido sea diferente de 0, sumando cada número introducido a **suma**. Cuando el usuario introduce 0, el bucle termina y se muestra la suma total de los números introducidos.

ISBN: 978-631-6557-57-5

Diagrama de flujo



El diagrama de flujo representa visualmente el pseudocódigo para un algoritmo llamado "Sumar_hasta_Cero". La lógica que se sigue es la siguiente:

1) Se inicializan dos variables como enteros: “suma” y “numero”.

2) Se asigna a la variable “suma” el valor de 0 para comenzar la acumulación de la suma.

3) Se pide al usuario que introduzca un número con la instrucción "Introduzca un número (0 para terminar):". Este valor se almacena en la variable “numero”.

4) Hay un punto de decisión: si el valor de “numero” es diferente de cero (representado por la condición numero <> 0), se entra en un bucle.

5) Dentro del bucle:

- Se suma el valor de la variable “numero” a la variable “suma” (indicado por suma <- suma + numero).
- Se solicita al usuario que introduzca otro número con la misma instrucción anterior. Este nuevo valor reemplaza al anterior en la variable “numero”.

6) El bucle se repite hasta que el usuario introduce un 0, lo que hace que la condición “numero <> 0” se evalúe como falsa (F en el diagrama) y se salga del bucle.

7) Al salir del bucle, se muestra al usuario el resultado total de la suma con la instrucción "La suma total es:", seguido del valor de la variable “suma”.

ISBN: 978-631-6557-57-5

8) Finalmente, el algoritmo llega a su fin, como indica el bloque "FinAlgoritmo".

Implementación

```
suma = 0

# Solicita al usuario el primer número
numero = int(input("Introduzca un número (0 para terminar): "))

# Inicia el bucle que sigue solicitando números y sumándolos a 'suma'
# hasta que el usuario introduzca 0
while numero != 0:
    suma += numero
    numero = int(input("Introduzca otro número (0 para terminar): "))

# Imprime la suma total de los números introducidos
print("La suma total es:", suma)
```

```
Introduzca un número (0 para terminar): 2
Introduzca otro número (0 para terminar): 6
Introduzca otro número (0 para terminar): 5
Introduzca otro número (0 para terminar): 0
La suma total es: 13
```

Este script comienza inicializando una variable **suma** a 0, que almacenará la suma total de los números introducidos. Luego, entra en un bucle **while** que solicita al usuario que introduzca números de forma repetida. Cada número introducido se suma a **suma**. El bucle continúa hasta que el usuario introduce el número 0, momento en el cual el programa termina el bucle y muestra la suma total de los números introducidos.

2. Hacer-Mientras (do-while): Similar al bucle "mientras", pero la condición se evalúa al final del bucle. Esto asegura que el cuerpo del bucle se ejecute al menos una vez, independientemente de la condición.

3. Repetir-Hasta (repeat-until): Continúa la ejecución del bloque de instrucciones hasta que se cumpla una condición específica. Esta estructura garantiza que el bloque de instrucciones se ejecute al menos una vez, similar a "hacer-mientras".

Pseudocódigo

```
Algoritmo RepetirMensaje
    Repetir

        Escribir "Nuestro mundo es maravilloso. ¿Desea ver el mensaje otra vez? (s/n): "
        Leer respuesta
    hasta que respuesta ≠ "s"
FinAlgoritmo
```

ISBN: 978-631-6557-57-5

El pseudocódigo es un ejemplo de un bucle `repetir-hasta` que se encuentra en lenguajes de programación o herramientas didácticas como PSeInt, el cual es similar a un bucle do-while que se encuentra en otros lenguajes de programación. Este es el flujo que describe el pseudocódigo:

1) Se inicia el algoritmo llamado "RepetirMensaje".

2) Comienza el bucle “repetir”.

3) Se muestra un mensaje al usuario: "Nuestro mundo es maravilloso. ¿Desea ver el mensaje otra vez? (s/n): ".

4) Se espera a que el usuario introduzca una respuesta, que se almacena en la variable “respuesta”.

5) El bucle continuará “repetir” mientras que la condición respuesta <> "s" (respuesta diferente a "s") no se cumpla.

6) Si el usuario escribe "s", se repite el bucle, mostrando nuevamente el mensaje y pidiendo otra respuesta.

7) Si el usuario escribe algo diferente a "s", se sale del bucle y termina el algoritmo con "FinAlgoritmo".

Este bucle garantiza que el mensaje se muestre al menos una vez y seguirá mostrándose mientras el usuario responda con "s". Cuando el usuario responda con cualquier otra cosa, el bucle terminará.

Diagrama de flujo

El diagrama de flujo ilustra el algoritmo llamado "RepetirMensaje" que implementa la lógica de bucle **repeat-until** que se encuentra en algunos lenguajes de programación. Aquí está la interpretación de ese diagrama de flujo:

1) El algoritmo inicia y de inmediato muestra el mensaje "Nuestro mundo es maravilloso. ¿Desea ver el mensaje otra vez? (s/n):".

ISBN: 978-631-6557-57-5

2) Después de mostrar el mensaje, se espera la entrada del usuario, que se guarda en la variable **respuesta**.
3) A continuación, el flujo de control pasa a una evaluación condicional donde se verifica si **respuesta** es igual a "s".
4) Si **respuesta** es igual a "s" (lo cual se marcaría con una "v" verdadero en el diagrama), se repite el bucle: se vuelve a mostrar el mensaje y se solicita de nuevo una respuesta al usuario.
5) Si **respuesta** no es igual a "s" (marcado con una "f" de falso), el algoritmo termina, como indica el bloque "FinalAgoritmo".

Implementación

```
respuesta = 's'
while respuesta == 's':
    print("Nuestro mundo es maravilloso. ¿Desea ver el mensaje otra vez? (s/n): ")
    respuesta = input()
```

```
Nuestro mundo es maravilloso. ¿Desea ver el saludo otra vez? (s/n):
s
Nuestro mundo es maravilloso. ¿Desea ver el saludo otra vez? (s/n):
s
Nuestro mundo es maravilloso. ¿Desea ver el saludo otra vez? (s/n):
n
```

En este código escrito en python, el bucle **while** se ejecuta inicialmente porque la condición se evalúa después del primer mensaje, imitando el comportamiento de un bucle **do-while** o **repeat-until**.

4. Desde/Para (for): Ideal para situaciones donde se conoce el número exacto de iteraciones de antemano. Esta estructura permite inicializar un contador, definir la condición de continuación y especificar un paso de incremento o decremento en una sola línea.

Ejemplo 1: Imprimir los números del 1 al 5

Pseudocódigo

```
1  Algoritmo imprimir_numeros
2      Para numero ← 1 Hasta 5 Con Paso 1 Hacer
3          Escribir numero
4      Fin Para
5  FinAlgoritmo
```

1) Inicia el algoritmo llamado imprimir_numeros
2) Se inicia un bucle **Para** que asigna a la variable **numero** los valores desde 1 hasta

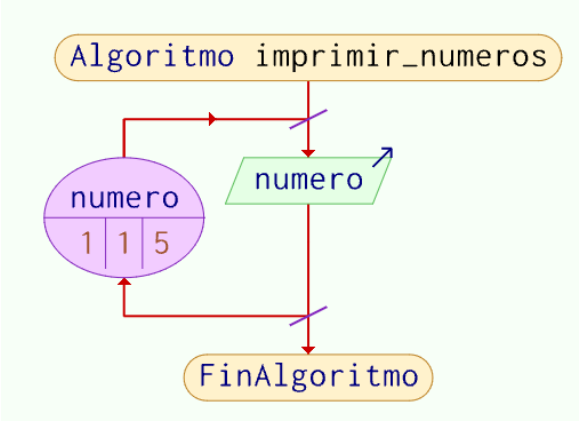
ISBN: 978-631-6557-57-5

5, incrementando de uno en uno en cada iteración (esto es lo que significa **Con Paso 1**).

3) Dentro del bucle, se ejecuta la instrucción **Escribir numero**, que imprime el valor actual de **numero**.
4) Después de imprimir cada número, el bucle continua con el siguiente valor hasta que se han impreso todos los números del 1 al 5.
5) Una vez que el bucle ha terminado de ejecutarse, también termina el algoritmo con **FinAlgoritmo**.

El efecto de este pseudocódigo sería mostrar en pantalla los números del 1 al 5, cada uno en una línea nueva, que es exactamente lo que haría el código Python correspondiente que has mencionado previamente.

Diagrama de flujo



1) El algoritmo inicia y entra en un bucle, el cual está representado por la estructura ovalada con la etiqueta **numero** que contiene los valores 1, 1, 5. Esto sugiere que la variable **numero** se inicializa en 1, tiene un valor final de 5, y se incrementa de uno en uno en cada iteración del bucle. Esta es una representación común de un bucle **for** en diagramas de flujo.
2) Dentro del bucle, el número actual representado por la variable **numero** se imprime o escribe.
3) El flujo se repite, incrementando el valor de **numero** por uno cada vez, hasta que el valor de **numero** excede 5.
4) Cuando **numero** es mayor que 5, el bucle termina (esto se muestra con la línea

ISBN: 978-631-6557-57-5

que sale del proceso con la etiqueta **numero** que no vuelve a la ovalada, sino que procede al final del algoritmo).

5) El algoritmo concluye, como se indica con el rectángulo de esquinas redondeadas con la etiqueta **FinAlgoritmo**.

Implementación

```
for i in range(1, 6):
    print(i)
```

```
1
2
3
4
5
```

Este código utiliza un bucle **for** para iterar sobre una secuencia de números generada por la función **range**. La función **range(1, 6)** genera una secuencia de números que comienza en 1 y termina en 5 (el 6 es exclusivo y no se incluye en la secuencia).

El bucle **for** asigna cada número de la secuencia a la variable **i** y luego ejecuta el cuerpo del bucle para cada valor de **i**. Dentro del cuerpo del bucle, se tiene una instrucción **print(i)** que imprime el valor actual de **i**.

Como resultado, la salida del código será la impresión de los números del 1 al 5, cada uno en una nueva línea, tal como se ve en la salida mostrada debajo del código. Este bucle se ejecuta exactamente 5 veces, una vez para cada número en la secuencia generada por **range**.

ISBN: 978-631-6557-57-5

Ejemplo 2: Dada la lista [2,4,5,6] hacer un programa que sume los elementos de la lista.

Pseudocódigo

```
Algoritmo SumarElementos
    Dimension numeros[4];
    Definir suma Como Entero;
    Definir i Como Entero;

    numeros[1] ← 2;
    numeros[2] ← 4;
    numeros[3] ← 5;
    numeros[4] ← 6;
    suma ← 0;

    Para i ← 1 Hasta 4 Con Paso 1 Hacer
        suma ← suma + numeros[i];
    Fin Para

    Escribir "La suma de los elementos es: ", suma;
FinAlgoritmo
```

Este pseudocódigo describe un algoritmo simple para sumar los elementos de un arreglo de números enteros. A continuación se explica cada parte del pseudocódigo:

1) Declaración de Variables:

- Dimension numeros[4]; establece que “numeros“ es un arreglo con capacidad para cuatro elementos enteros.
- Definir suma Como Entero; declara una variable “suma” que se utilizará para acumular la suma de los elementos del arreglo.
- Definir i Como Entero; declara una variable “i” que se utilizará como contador en el bucle “Para”.

2) Inicialización de Arreglo:

- numeros[1] <- 2; asigna el valor 2 al primer elemento del arreglo “números”.
- numeros[2] <- 4; asigna el valor 4 al segundo elemento del arreglo “números”.
- numeros[3] <- 5;` asigna el valor 5 al tercer elemento del arreglo “números”.
- numeros[4] <- 6; asigna el valor 6 al cuarto y último elemento del arreglo “números”.

3) Inicialización de la Suma:

- suma <- 0; inicializa la variable “suma” a 0.

4) Bucle Para Sumar Elementos:

- La estructura de control “Para i <- 1 Hasta 4 Con Paso 1 Hacer” comienza un bucle que asigna valores de 1 a 4 a la variable `i`. En cada iteración del bucle, se incrementa “i”

ISBN: 978-631-6557-57-5

en 1 (“Con Paso 1”).

- Dentro del bucle, “suma <- suma + numeros[i];” actualiza la variable “suma” con la suma del valor actual de “suma” y el elemento del arreglo “números” en la posición “i”.

5) Mostrar Resultado:

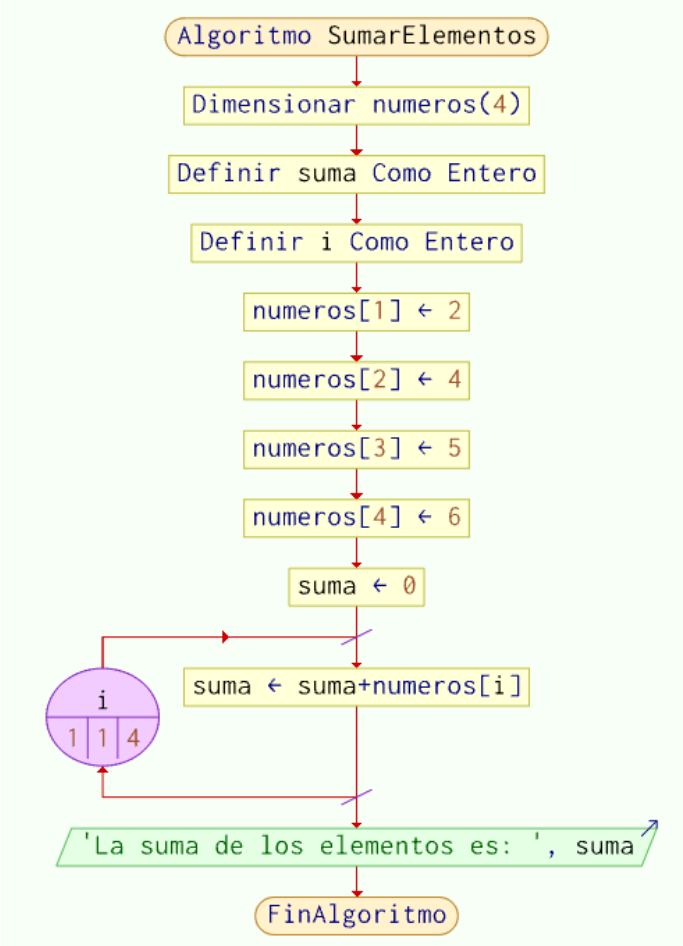
- Después de salir del bucle, Escribir "La suma de los elementos es: ", suma; muestra el resultado acumulado en la variable “suma”.

6) Fin del Algoritmo:

- “FinAlgoritmo” indica que el algoritmo ha terminado.

Es importante notar que este pseudocódigo está escrito para PSeInt, un programa educativo que facilita el aprendizaje de la lógica de programación, y por tanto utiliza una sintaxis que puede variar ligeramente en comparación con lenguajes de programación reales. En lenguajes como Python, la indexación de los arreglos comienza en 0 en lugar de 1, y no es necesario declarar el tamaño del arreglo antes de asignar valores.

Diagrama de flujo



ISBN: 978-631-6557-57-5

Este diagrama de flujo representa el proceso de sumar los elementos de un arreglo numérico. Aquí está la interpretación paso a paso:

1) Dimensionar numeros(4): Se declara un arreglo llamado “numeros” con 4 posiciones.

2) Definir suma Como Entero: Se declara una variable “suma” de tipo entero.

3) Definir i Como Entero: Se declara una variable “i” de tipo entero, la cual se utilizará como contador en el bucle.

4) numeros[1] <- 2: Se asigna el valor 2 a la primera posición del arreglo “numeros”.

5) numeros[2] <- 4: Se asigna el valor 4 a la segunda posición del arreglo “numeros”.

6) numeros[3] <- 5: Se asigna el valor 5 a la tercera posición del arreglo “numeros”.

7) numeros[4] <- 6: Se asigna el valor 6 a la cuarta y última posición del arreglo “numeros“.

8) suma <- 0: Se inicializa la variable `suma` con el valor 0.

9) El bucle **Para** comienza con i igual a 1 y termina con i igual a 4, incrementándose i en 1 con cada iteración (esto se muestra en el diagrama con el rango de i de 1 a 4 y un bucle interno).

- Dentro del bucle, suma <- suma + numeros[i]: Se suma el valor de la posición actual del arreglo `numeros` indexada por `i` a la variable `suma`.

10) Después de salir del bucle, Escribir "La suma de los elementos es: ", suma: Se muestra el resultado acumulado de la suma.

11) FinAlgoritmo: Indica el fin del algoritmo.

La ejecución de este diagrama de flujo resultará en sumar los números 2, 4, 5 y 6 para obtener un total que será mostrado al usuario. La suma total sería 17.

ISBN: 978-631-6557-57-5

Implementación

```
# Definición del arreglo y la variable de suma
numeros = [2, 4, 5, 6]
suma = 0

# Bucle para sumar los elementos del arreglo
for i in range(len(numeros)):
    suma += numeros[i]

# Impresión del resultado
print("La suma de los elementos es:", suma)
```

```
La suma de los elementos es: 17
```

Este código en Python realiza la suma de los elementos de una lista. Aquí está la explicación:

1) **Definición del arreglo y la variable de suma:** Se crea una lista llamada **numeros** que contiene cuatro elementos: 2, 4, 5 y 6. También se inicializa una variable **suma** en 0. Esta variable almacenará la suma total de los elementos de la lista.

```
numeros = [2, 4, 5, 6]
suma = 0
```

2) **Bucle para sumar los elementos del arreglo:** Se usa un bucle **for** para iterar a través de la lista. El bucle utiliza la función range y la función len. len(**numeros**) devuelve la longitud de la lista **numeros**, que es 4 en este caso. range(len(numeros)) genera una secuencia de números desde 0 hasta 3 (la longitud de la lista menos uno). El bucle for repite su bloque de código interno una vez para cada número en esta secuencia.

```
for i in range(len(numeros)):
    suma += numeros[i]
```

Dentro del bucle, se accede a cada elemento de la lista **numeros** usando numeros[i], donde 'i' es la variable del bucle que va de 0 a 3, y se suma este valor a la variable **suma**.

3) **Impresión del resultado:** Finalmente, después de que el bucle ha terminado, se imprime el valor de la variable **suma**, que ahora contiene la suma total de todos los elementos de la lista **numeros**.

```
print("La suma de los elementos es:", suma)
```

ISBN: 978-631-6557-57-5

El resultado que muestra este código es "La suma de los elementos es: 17", que es la suma de los números 2, 4, 5 y 6.

Consideraciones en estructuras de repetición

En las estructuras de repetición se debe tener en cuenta las siguientes consideraciones:

Bucles Infinitos: Un bucle se convierte en infinito si la condición de terminación nunca se cumple. Estos bucles son errores comunes en la programación y deben evitarse, a menos que se desee una ejecución continua (por ejemplo, en un servidor).

Uso de Centinelas: Un valor centinela se utiliza para indicar el fin de un conjunto de datos en bucles que procesan secuencias de entrada. Es un método eficaz para controlar la ejecución del bucle cuando el número de iteraciones no se conoce de antemano.

Optimización del Rendimiento: En algunos casos, un bucle puede hacer que un programa sea ineficiente, especialmente si contiene operaciones costosas dentro de su cuerpo. Evalúa la necesidad de cada operación dentro del bucle y busca maneras de reducir la carga computacional.

Claridad y Mantenibilidad: Aunque las estructuras repetitivas son herramientas poderosas, su uso indebido puede llevar a código difícil de entender y mantener. Utiliza comentarios y divide el código en funciones o métodos cuando sea apropiado para mantener la claridad.

La correcta utilización de las estructuras repetitivas es esencial para el desarrollo de software eficiente. Permiten a los desarrolladores escribir código más limpio, legible, más comprensible y más mantenible, facilitando la implementación de algoritmos complejos y el procesamiento de grandes volúmenes de datos.

Aplicaciones de Estructuras Repetitivas

Las estructuras repetitivas tienen una amplia gama de aplicaciones en el ámbito de la programación. Algunas de sus aplicaciones más comunes incluyen:

a) **Procesamiento de Datos:** En el procesamiento de datos, las estructuras repetitivas se utilizan para manejar grandes volúmenes de información. Por ejemplo, se podría tener un archivo con miles de números y necesitar calcular estadísticas, como la suma, el promedio o incluso distribuciones más complejas.

ISBN: 978-631-6557-57-5

Un bucle puede leer cada número y realizar cálculos acumulativos o aplicar algoritmos para clasificar o filtrar datos, todo esto mientras avanza a través de la lista de números.

b) **Control de Interfaces de Usuario:** Las aplicaciones con interfaz gráfica (GUI) son interactivas y esperan la interacción del usuario para responder. Una estructura repetitiva puede estar en constante ejecución, revisando si el usuario ha hecho clic en un botón, ha introducido texto o ha realizado alguna otra acción. Dependiendo de las interacciones, el programa puede actualizar la GUI, responder a comandos o procesar entradas. Este tipo de estructura repetitiva a menudo se encuentra en un bucle de eventos o bucle de mensajes dentro del programa.

c) **Juegos:** En el desarrollo de juegos, las estructuras repetitivas forman parte del "game loop" o bucle de juego, el corazón de cualquier juego interactivo. En cada iteración del bucle, el juego procesa las entradas del usuario (como movimientos del ratón, teclas presionadas, toques en la pantalla), actualiza el estado del juego (como la posición de los personajes, puntuaciones, o colisiones) y luego produce una nueva imagen para mostrar en pantalla. Este ciclo se repite varias veces por segundo, creando la ilusión de movimiento y respuesta inmediata a las acciones del usuario.

d) **Simulaciones:** Las estructuras repetitivas son importantes en la simulación de procesos que varían con el tiempo o están basados en iteraciones. Se puede simular el crecimiento poblacional, el cambio climático o sistemas económicos, repitiendo un conjunto de operaciones muchas veces, donde cada iteración representa un paso en el tiempo o una prueba dentro de un escenario simulado. En cada paso, se pueden actualizar las variables de acuerdo con modelos matemáticos para ver cómo evolucionan a lo largo del tiempo o bajo diferentes condiciones.

ISBN: 978-631-6557-57-5

2.4 Ejercicios de aplicación

2.4.1 Ejercicios resueltos

1) Calcular el perímetro de un círculo.

Pseudocódigo

```
Algoritmo CalcularPerimetroCirculo
    // Definir variables
    Definir radio, perimetro Como Real

    // Solicitar el radio del círculo al usuario
    Escribir "Indique el radio del círculo: "
    Leer radio

    // Calcular el perímetro del círculo
    perimetro ← 2 * PI * radio

    // Mostrar el resultado
    Escribir "El perímetro del círculo es ", perimetro
FinAlgoritmo
```

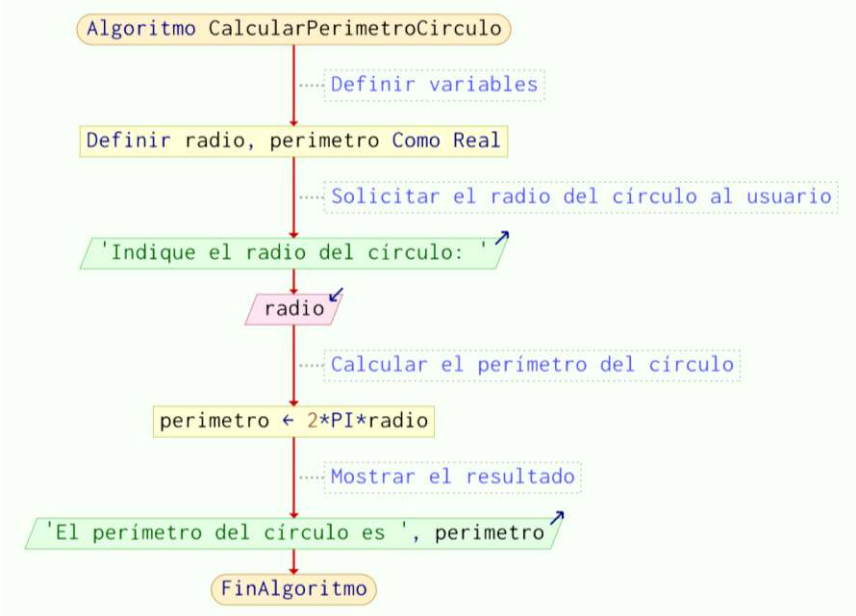
Las variables radio y perimetro se declaran con la palabra clave Real, indicando que son números decimales.

Para solicitar datos del usuario se utilizan las instrucciones Escribir para mostrar un mensaje y Leer para recibir la entrada.

El cálculo del perímetro utiliza PI como constante predefinida en PSeInt para representar el valor de π.

Finalmente, se utiliza Escribir para mostrar el resultado del cálculo.

Diagrama de flujo



ISBN: 978-631-6557-57-5

Implementación

En Python

```python
import math

# Se pide el radio del círculo al usuario
radio = float(input("Indique el radio del círculo: "))

# Se calcula el perímetro
perimetro = 2 * math.pi * radio

# Se muestra el resultado
print("El perímetro del círculo es", perimetro)
```

```
Indique el radio del círculo: 3
El perímetro del círculo es 18.84955592153876
```

Se importa el módulo math, que contiene funciones matemáticas útiles, incluida la constante pi, math.pi proporciona el valor de π.

En C++

main.cpp Run

```cpp
#include <iostream>
#include <cmath> // Para usar M_PI, que es la representación de pi

int main() {
    float radio, perimetro;

    // Pedir el radio al usuario
    std::cout << "Indique el radio del círculo: ";
    std::cin >> radio;

    // Calcular el perímetro
    perimetro = 2 * M_PI * radio;

    // Mostrar el resultado
    std::cout << "El perímetro del círculo es " << perimetro << std::endl;

    return 0;
}
```

Se incluyen las cabeceras <iostream> para operaciones de entrada y salida y <cmath> para constantes matemáticas y funciones.

ISBN: 978-631-6557-57-5

Se utiliza std::cout para imprimir mensajes en la consola y std::cin para leer la entrada del usuario.

M_PI es una definición de pi proporcionada por la biblioteca <cmath>.

No es necesario importar toda una biblioteca para el valor de pi, ya que M_PI está definido en <cmath>.

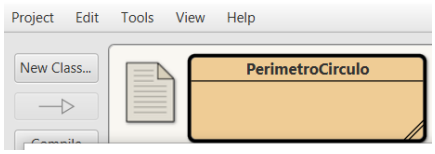
Se utiliza std::endl para agregar un salto de línea al final de la salida en la consola.

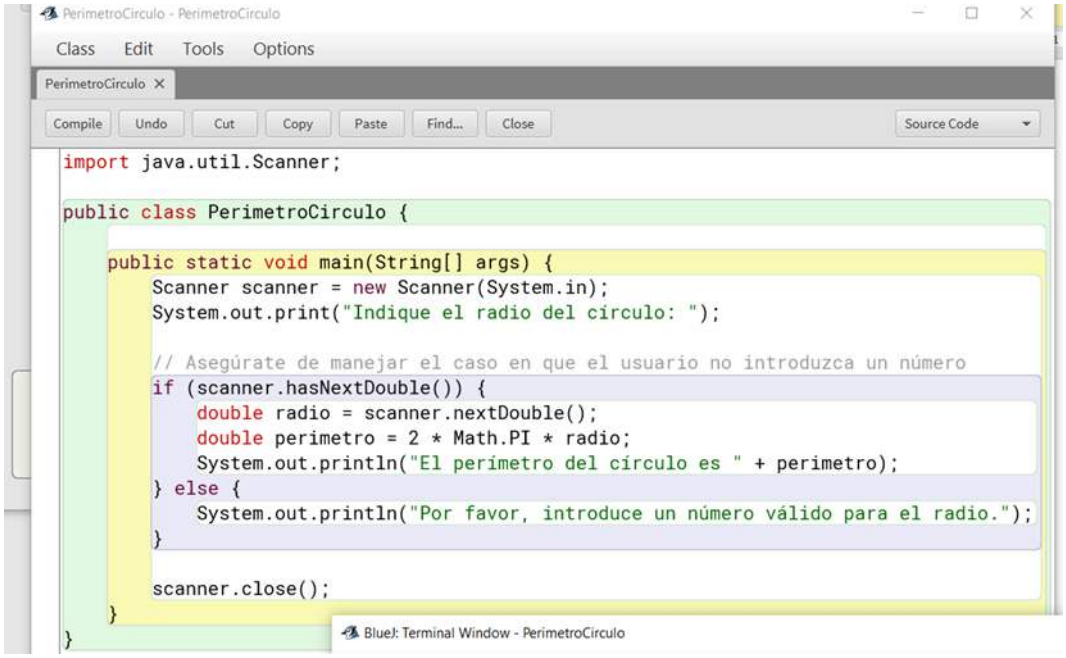
En C++, todas las funciones deben estar dentro de una función principal, que es int main() en este caso.

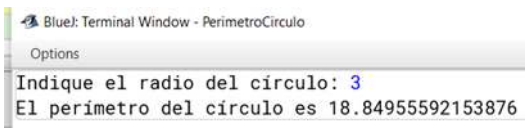
Se declara el tipo de las variables (float) antes de usarlas.

En C++, es común terminar la función main con un return 0; para indicar que el programa ha terminado correctamente.

En Java







ISBN: 978-631-6557-57-5

2) Hacer un programa para calcular el índice de masa corporal (ICM).

Pseudocódigo

```
Algoritmo CalcularIMC
    // Definir variables
    Definir peso, altura, imc Como Real

    // Solicitar el peso y la altura al usuario
    Escribir "Introduce tu peso en kilogramos: "
    Leer peso
    Escribir "Introduce tu altura en metros: "
    Leer altura

    // Calcular el IMC
    imc ← peso / (altura ↑ 2)

    // Mostrar el resultado
    Escribir "Tu índice de masa corporal es ", imc
FinAlgoritmo
```

Definir variables: Se usa Definir seguido del nombre de la variable y Como Real para aclarar que son números reales. PSeInt asume el tipo de la variable basándose en los datos que se asignan.

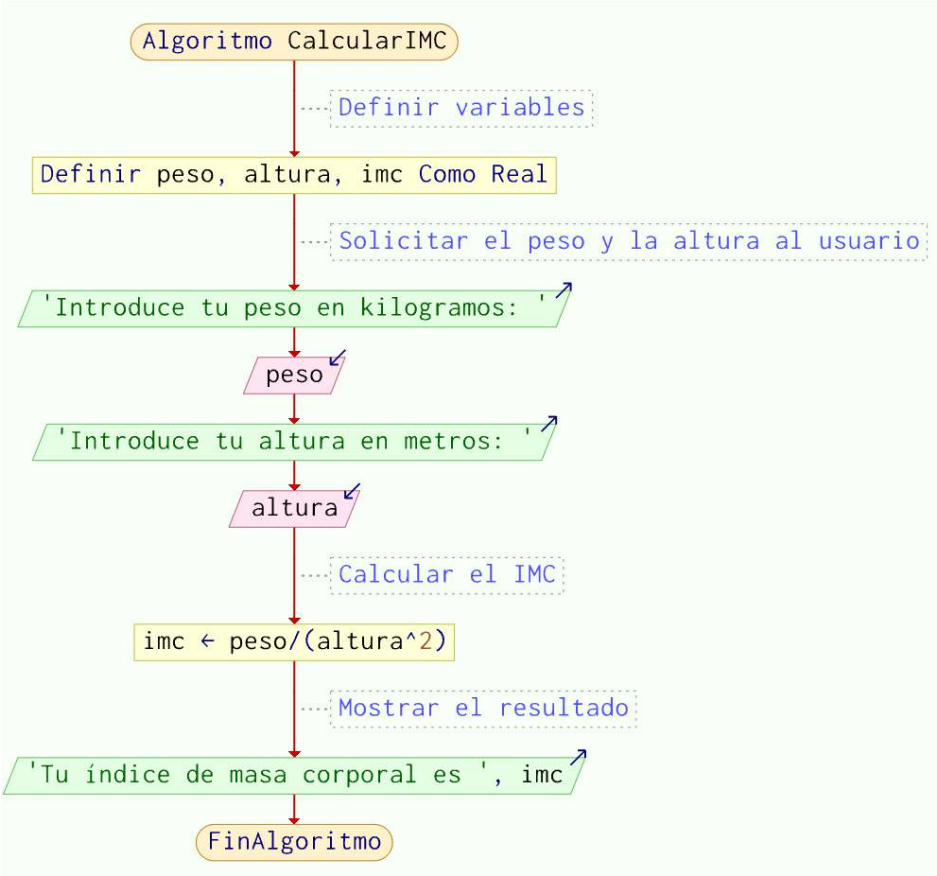
Solicitar datos: Escribir se utiliza para mostrar un mensaje en pantalla y Leer para capturar la entrada del usuario.

Calcular el IMC: Se usa el operador de potencia ^ que es el equivalente en PSeInt del operador ** en Python para elevar un número a una potencia.

Mostrar el resultado: Se utiliza Escribir para mostrar el resultado del cálculo del IMC.

ISBN: 978-631-6557-57-5

Diagrama de flujo



Implementación

```
# Se pide el peso en kilogramos y la altura en metros
peso = float(input("Introduce tu peso en kilogramos: "))
altura = float(input("Introduce tu altura en metros: "))

# Se calcula el IMC
imc = peso / (altura**2)

# Se muestra el resultado
print("Tu índice de masa corporal es", imc)
```

```
Introduce tu peso en kilogramos: 79.38
Introduce tu altura en metros: 1.72
Tu índice de masa corporal es 26.83207138994051
```

3) Escribir un programa que calcule el promedio de la nota obtenida con los siguientes indicadores: examen 1, examen 2, proyecto final.

ISBN: 978-631-6557-57-5

Pseudocódigo

```
Algoritmo CalcularNotaFinal
    // Definir variables
    Definir examen1, peso_examen1, examen2, peso_examen2, proyecto_final, peso_proyecto, nota_final Como Real

    // Solicitar las notas y sus pesos
    Escribir "Nota del examen 1: "
    Leer examen1
    Escribir "Peso del examen 1 (en porcentaje, ej. 9 para 35%=0.35): "
    Leer peso_examen1

    Escribir "Nota del examen 2: "
    Leer examen2
    Escribir "Peso del examen 2 (en porcentaje, ej. 8 para 35%=0.35): "
    Leer peso_examen2

    Escribir "Nota del proyecto final: "
    Leer proyecto_final
    Escribir "Peso del proyecto final (en porcentaje, ej. 9 para 30%=0.30): "
    Leer peso_proyecto

    // Calcular la nota final
    nota_final ← examen1 * peso_examen1 + examen2 * peso_examen2 + proyecto_final * peso_proyecto  / (peso_examen1 + peso_examen2 + peso_proyecto)

    // Mostrar la nota final
    Escribir "La nota final del estudiante es ", nota_final
FinAlgoritmo
```

1) El algoritmo comienza definiendo variables para las notas de los exámenes, los pesos de cada evaluación y la nota final como números reales.

2) El programa solicita al usuario que introduzca la nota del primer examen y su peso, que debe ser introducido en formato decimal (es decir, 35% como 0.35).

3) Repite el proceso para el segundo examen.

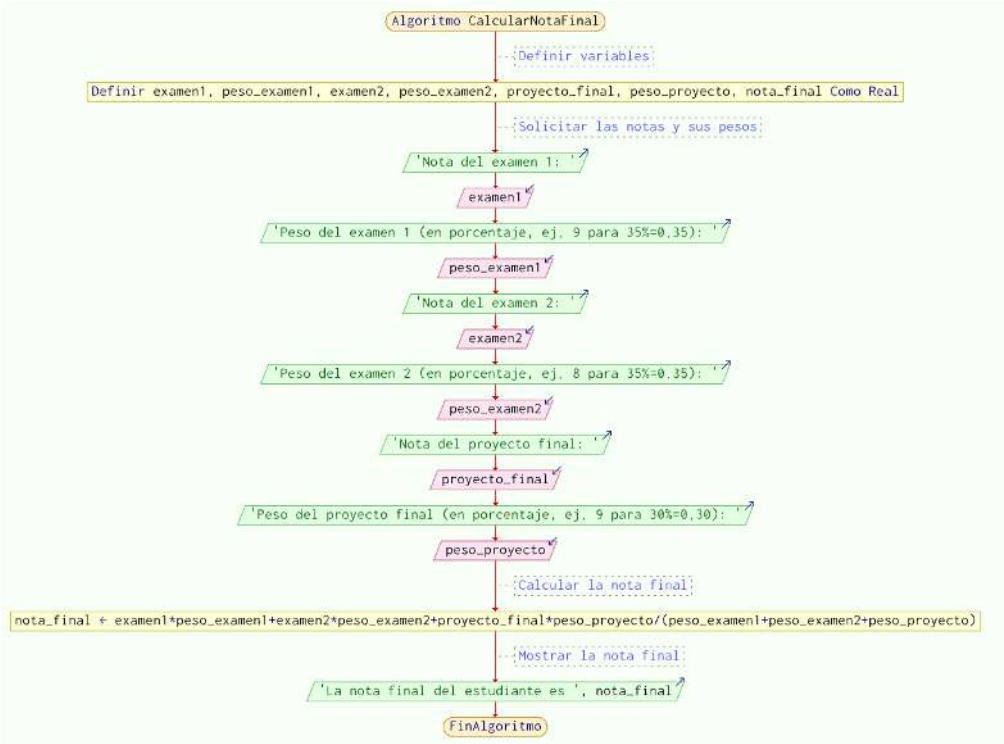
4) Continúa solicitando la nota del proyecto final y su peso, también en formato decimal.

5) Calcula la nota final utilizando una fórmula que multiplica cada nota por su peso y divide el resultado por la suma de los pesos para obtener un promedio ponderado.

6. Finalmente, muestra la nota final del estudiante.

Diagrama de flujo

ISBN: 978-631-6557-57-5



Implementación

```
# Solicitar las notas y sus pesos
examen1 = float(input("Nota del examen 1: "))
peso_examen1 = float(input("Peso del examen 1: "))
examen2 = float(input("Nota del examen 2: "))
peso_examen2 = float(input("Peso del examen 2: "))
proyecto_final = float(input("Nota del proyecto final: "))
peso_proyecto = float(input("Peso del proyecto final: "))

# Calcular la nota final
nota_final = (examen1 * peso_examen1 + examen2 * peso_examen2 + proyecto_final * peso_proyecto) / (peso_examen1 + peso_examen2 + peso_proyecto)

# Mostrar la nota final
print("La nota final del estudiante es", round(nota_final, 2))
```

```
Nota del examen 1: 9
Peso del examen 1: 0.35
Nota del examen 2: 8
Peso del examen 2: 0.35
Nota del proyecto final: 9
Peso del proyecto final: 0.30
La nota final del estudiante es 8.65
```

4) Hacer un programa que calcule el interés compuesto para una inversión inicial, tasa de interés anual, número de veces que se capitaliza por año y el número total de años.

ISBN: 978-631-6557-57-5

Pseudocódigo

```
Algoritmo CalcularCapitalFinal
    // Definir variables
    Definir capital_inicial Como Real
    Definir tasa_interes Como Real
    Definir capital_final Como Real
    Definir veces_año Como Entero
    Definir años Como Entero

    // Solicitar datos al usuario
    Escribir "Introduce el capital inicial: "
    Leer capital_inicial
    Escribir "Introduce la tasa de interés anual (como decimal): "
    Leer tasa_interes
    Escribir "Introduce el número de veces que el interés se capitaliza por año: "
    Leer veces_año
    Escribir "Introduce el número de años: "
    Leer años

    // Calcular el capital final
    capital_final ← capital_inicial * (1 + tasa_interes / veces_año) ↑ (veces_año * años)

    // Mostrar el resultado
    Escribir "El capital acumulado después de ", años, " años es ", capital_final
FinAlgoritmo
```

Se definen cinco variables. capital_inicial, tasa_interes y capital_final como números reales (Real), y veces_año y años como números enteros (Entero).

El algoritmo solicita al usuario que ingrese el capital inicial de su inversión.

A continuación, solicita la tasa de interés anual, la cual debe ser proporcionada en forma decimal (por ejemplo, 10% como 0.10).

Luego, pide el número de veces que el interés se capitaliza por año. Este sería un número entero como 1 para anual, 6 para semestral, 12 para mensual, etc.

Después, solicita el número de años durante los cuales se realizará la inversión.

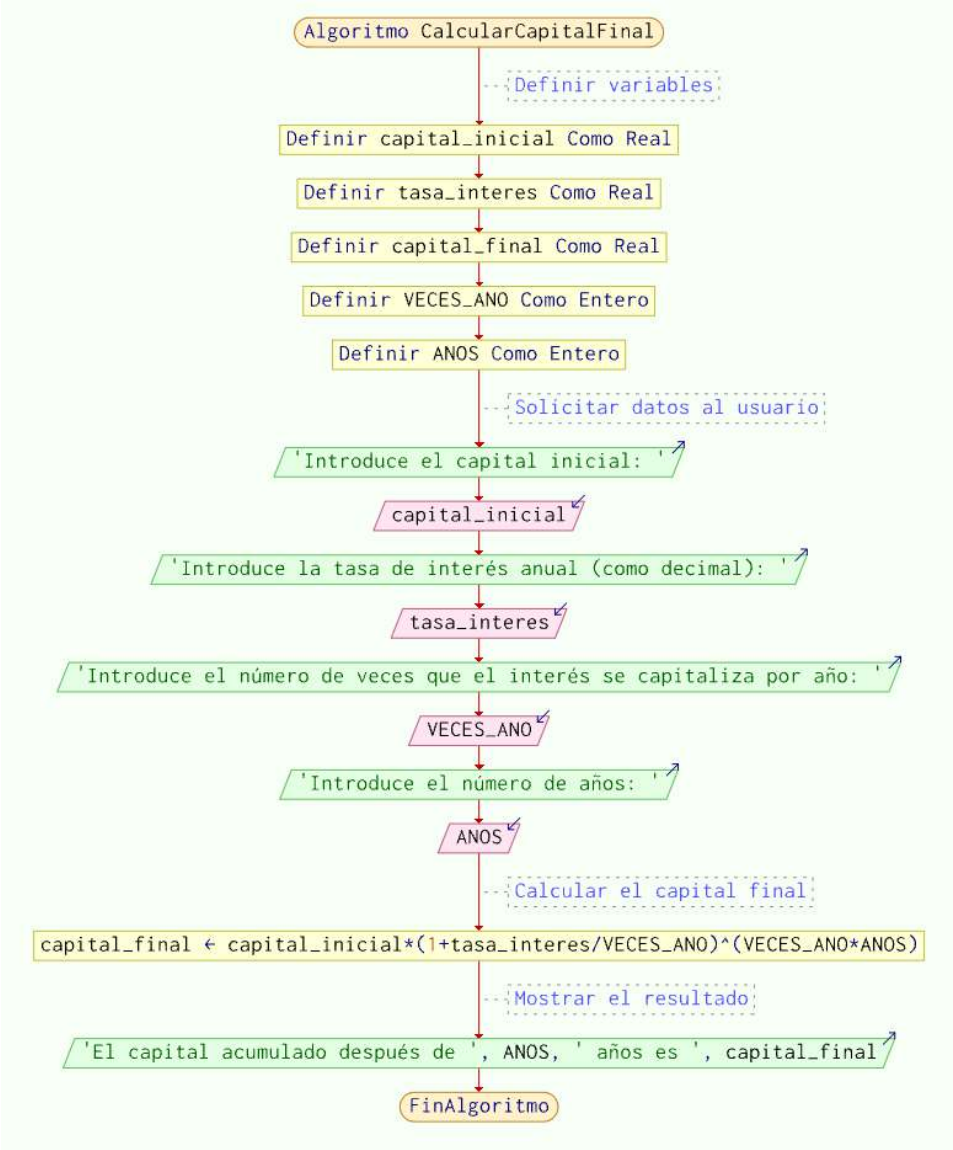
Con todos los datos ingresados, el algoritmo calcula el capital final aplicando la fórmula de interés compuesto.

Finalmente, el algoritmo imprime el resultado del capital acumulado después del número de años especificado.

El operador ^ se utiliza para representar la operación de potencia en PSeInt, correspondiente al operador ** en Python. El resultado se muestra sin redondear; si se quisiera redondear a dos decimales, se debería modificar el algoritmo para utilizar una función de redondeo adecuada para PSeInt, como Truncar.

ISBN: 978-631-6557-57-5

Diagrama de flujo



Implementación

```
# Solicitar datos al usuario
capital_inicial = float(input("Introduce el capital inicial: "))
tasa_interes = float(input("Introduce la tasa de interés anual (como decimal): "))
veces_año = int(input("Introduce el número de veces que el interés se capitaliza por año: "))
años = int(input("Introduce el número de años: "))

# Calcular el capital final
capital_final = capital_inicial * ((1 + tasa_interes / veces_año) ** (veces_año * años))

# Mostrar el resultado
print("El capital acumulado después de", años, "años es", round(capital_final, 2))
```

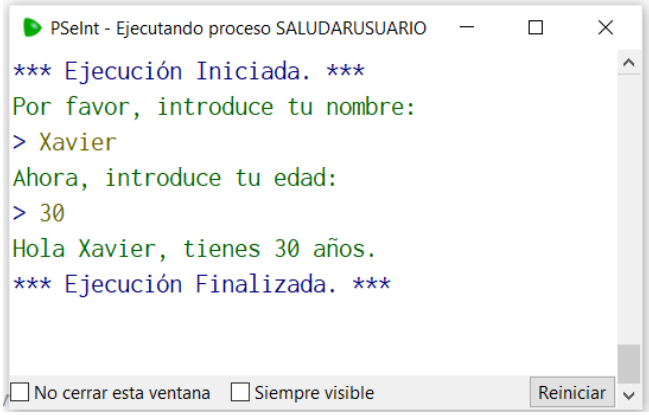
ISBN: 978-631-6557-57-5

```
Introduce el capital inicial: 15000
Introduce la tasa de interés anual (como decimal): 0.10
Introduce el número de veces que el interés se capitaliza por año: 1
Introduce el número de años: 2
El capital acumulado después de 2 años es 18150.0
```

5) Escriba un programa en JavaScript que pida al usuario su nombre y edad, luego imprime un mensaje que diga: "Hola [nombre], tienes [edad] años".

Pseudocódigo

```
// Programa que pide al usuario su nombre y edad y luego imprime un mensaje de saludo
Proceso SaludarUsuario
    Definir nombre Como Cadena;
    Definir edad Como Entero;
    // Pedir al usuario que ingrese su nombre
    Escribir "Por favor, introduce tu nombre:";
    Leer nombre;
    // Pedir al usuario que ingrese su edad
    Escribir "Ahora, introduce tu edad:";
    Leer edad;
    // Imprimir el mensaje de saludo con el nombre y la edad proporcionados
    Escribir "Hola ", nombre, ", tienes ", edad, " años.";
FinProceso
```



ISBN: 978-631-6557-57-5

Diagrama de flujo

Implementación

main.js

Run

```
// Programa en JavaScript que pide al usuario su nombre y edad y luego imprime un
    mensaje de saludo

// Pedir al usuario que ingrese su nombre
var nombre = prompt("Por favor, introduce tu nombre:");

// Pedir al usuario que ingrese su edad
var edad = prompt("Ahora, introduce tu edad:");

// Imprimir el mensaje de saludo con el nombre y la edad proporcionados
alert("Hola " + nombre + ", tienes " + edad + " años.");

```

```
Por favor, introduce tu nombre:Patricio
Ahora, introduce tu edad:31
Hola Patricio, tienes 31 años.
```

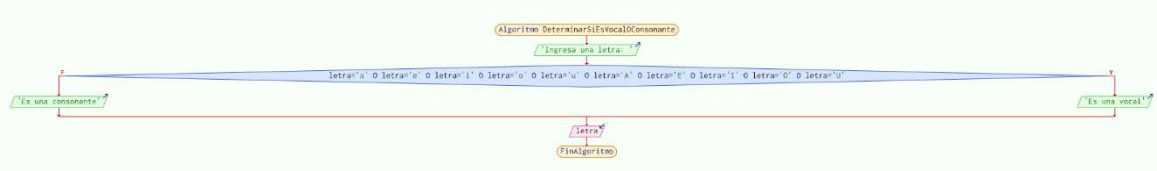
6) Comprobar si una letra es vocal o consonante.

Pseudocódigo

```
Algoritmo DeterminarSiEsVocalOConsonante
    Escribir "Ingresa una letra: "
    Leer letra

    Si letra = "a" O letra = "e" O letra = "i" O letra = "o" O letra = "u" O letra = "A" O letra = "E" O letra = "I" O letra = "O" O letra = "U"  Entonces
        Escribir "Es una vocal"

    Sino
        Escribir "Es una consonante"
    FinSi
FinAlgoritmo
```

ISBN: 978-631-6557-57-5

Diagrama de flujo



Implementación

En Python

```
letra = input("Ingresa una letra: ").lower()
if letra in "aeiou":
    print("Es una vocal")
else:
    print("Es una consonante")
```

```
Ingresa una letra: u
Es una vocal
```

El código en Python realiza las siguientes acciones:

1) Pide al usuario que ingrese una letra y almacena esta entrada en la variable letra.

2) Convierte la letra ingresada a minúscula utilizando el método .lower() para facilitar la comparación.

3) Evalúa si la letra convertida a minúscula está entre las vocales "aeiou" utilizando la palabra clave in.

4) Si la letra está en la cadena de vocales, imprime el mensaje "Es una vocal".

5) Si la letra no está en la cadena de vocales, por descarte, imprime el mensaje "Es una consonante".

En JavaScript

```
let letra = prompt("Ingresa una letra: ").toLowerCase();

if ("aeiou".includes(letra)) {
    alert("Es una vocal");
} else {
    alert("Es una consonante");
}
```

ISBN: 978-631-6557-57-5

```
Ingresa una letra: I
Es una vocal
```

Pide al usuario que ingrese una letra con un prompt.

Convierte la entrada a minúsculas con .toLowerCase() para que la comparación sea insensible a mayúsculas.

Usa .includes() para comprobar si la letra ingresada está dentro de las vocales "aeiou".

Si es así, muestra una alerta que dice "Es una vocal".

Si no, muestra una alerta que dice "Es una consonante".

En C++

main.cpp

```cpp
#include <iostream>
#include <string>
#include <cctype>

int main() {
    char letra;
    std::string vocales = "aeiou";

    std::cout << "Ingresa una letra: ";
    std::cin >> letra;
    letra = tolower(letra); // Convierte la letra a minúscula

    if (vocales.find(letra) != std::string::npos) {
        std::cout << "Es una vocal" << std::endl;
    } else {
        std::cout << "Es una consonante" << std::endl;
    }

    return 0;
}
```

```
Ingresa una letra: E
Es una vocal
```

1) Se incluyen tres bibliotecas:

- <iostream> para la entrada y salida estándar.
- <string> para utilizar objetos de tipo `std::string`.
- <cctype> para funciones de caracterización de caracteres, como tolower que convierte letras a minúsculas.

2) Se declara la función principal int main(), que es el punto de inicio de ejecución del programa.

ISBN: 978-631-6557-57-5

3) Dentro de main, se declara una variable letra de tipo char para almacenar un único carácter.

4) Se crea una cadena de texto std::string vocales que contiene todas las vocales en minúsculas.

5) Se pide al usuario que ingrese una letra con el mensaje "Ingresa una letra: " usando std::cout.

6) Se lee la letra ingresada por el usuario y se almacena en la variable letra usando std::cin.

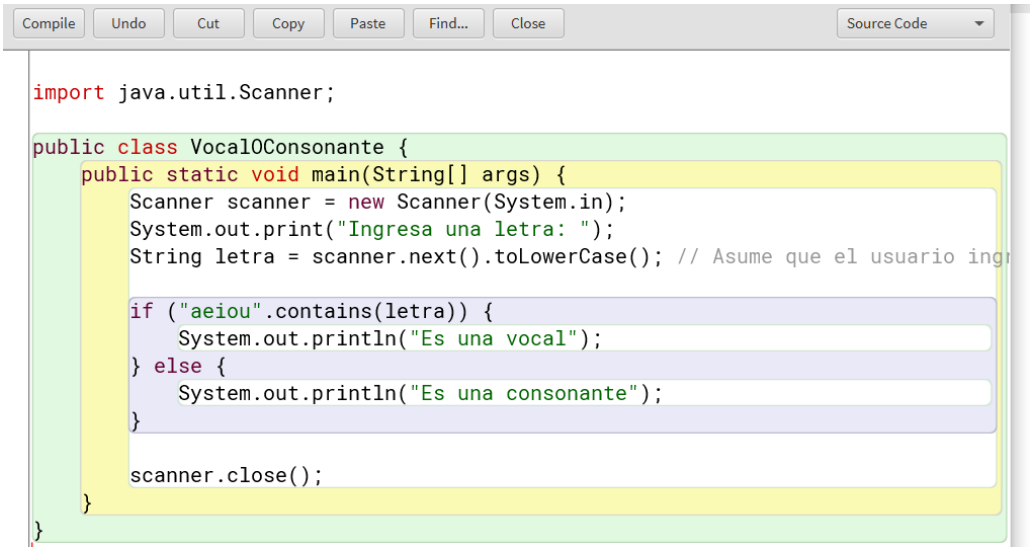
7) La letra ingresada se convierte a minúscula usando la función tolower para manejar de forma uniforme la comparación con las vocales.

8) Se utiliza vocales.find(letra) para buscar la letra en la cadena de vocales. Si find no devuelve std::string::npos, significa que la letra fue encontrada dentro de la cadena vocales.

9) Si la letra está en la cadena vocales, se imprime "Es una vocal". Si no, se imprime "Es una consonante".

10) Finalmente, el programa retorna 0, lo que indica que ha terminado exitosamente.

En Java




```java
import java.util.Scanner;

public class VocalOConsonante {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Ingresa una letra: ");
        String letra = scanner.next().toLowerCase(); // Asume que el usuario ingr

        if ("aeiou".contains(letra)) {
            System.out.println("Es una vocal");
        } else {
            System.out.println("Es una consonante");
        }

        scanner.close();
    }
}
```

```
Ingresa una letra: Z
Es una consonante
```

ISBN: 978-631-6557-57-5

1) Se importa la clase Scanner del paquete java.util, que se utiliza para leer la entrada del usuario desde la consola.
2) Se declara una clase pública llamada VocalOConsonante.
3) Dentro de la clase, se define el método main, que es el punto de entrada de cualquier aplicación en Java.
4) Se crea un objeto scanner de la clase Scanner para leer la entrada de texto proporcionada por el usuario a través de la consola.
5) Se solicita al usuario que ingrese una letra mostrando el mensaje "Ingresa una letra: " en la consola.
6) Se lee la entrada del usuario con scanner.next(), que toma el siguiente token de entrada como una cadena String. Se asume que el usuario ingresará solo un carácter. Esta entrada se convierte a minúsculas utilizando el método toLowerCase() para uniformizar la comparación con las vocales.
7) Se verifica si la cadena de vocales "aeiou" contiene la letra ingresada con el método contains. Si la letra está contenida en la cadena, se imprime en consola "Es una vocal".
8) Si la letra no está contenida en la cadena de vocales, se ejecuta la cláusula else, y se imprime en consola "Es una consonante".
9) Finalmente, se cierra el objeto scanner con el método close() para liberar los recursos asociados a este objeto.

7) Realizar un programa en Java para que reciba dos números como parámetros y retorne la suma de ambos.

Pseudocódigo

```
Algoritmo SumarNumeros
  // Definir variables
  Definir numero1, numero2, suma Como Entero
  // Solicitar al usuario que ingrese los números
  Escribir "Ingrese el primer número:"
  Leer numero1
  Escribir "Ingrese el segundo número:"
  Leer numero2
```

ISBN: 978-631-6557-57-5

```
// Realizar la suma de los números ingresados
suma <- numero1 + numero2
// Mostrar el resultado de la suma
Escribir "La suma de ", numero1, " y ", numero2, " es: ", suma
FinAlgoritmo
```

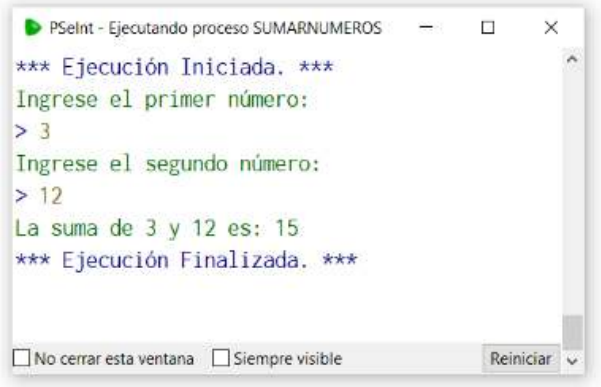
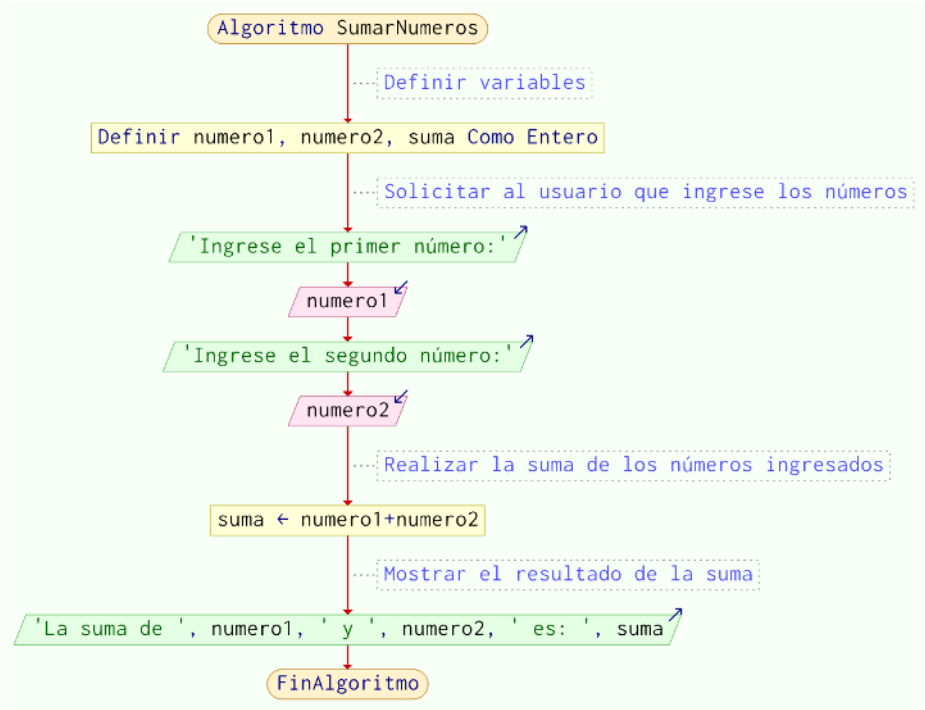
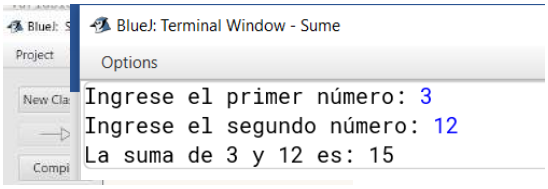


Diagrama de flujo



ISBN: 978-631-6557-57-5

Implementación



Compile | Undo | Cut | Copy | Paste | Find... | Close

```java
import java.util.Scanner;

public class Sume {
    public static void main(String[] args) {
        // Crear una instancia de Scanner para leer desde el teclado
        Scanner scanner = new Scanner(System.in);

        // Definir variables
        int numero1, numero2, suma;

        // Solicitar al usuario que ingrese el primer número
        System.out.print("Ingrese el primer número: ");
        numero1 = scanner.nextInt();

        // Solicitar al usuario que ingrese el segundo número
        System.out.print("Ingrese el segundo número: ");
        numero2 = scanner.nextInt();

        // Realizar la suma de los números ingresados
        suma = numero1 + numero2;

        // Mostrar el resultado de la suma
        System.out.println("La suma de " + numero1 + " y " + numero2 + " es: " + suma);

        // Cerrar el scanner
        scanner.close();
    }
}
```

8) Realizar un programa en Python en el cual una cadena de texto se muestre de forma inversa.

Pseudocódigo

Algoritmo Invertir_Cadena

Definir texto, texto_invertido Como Cadena

Definir TAMAÑO, i Como Entero

Escribir "Ingrese una cadena de texto:"

Leer texto

TAMAÑO <- Longitud(texto)

texto_invertido <- ""

Para i <- TAMAÑO Hasta 1 Con Paso -1 Hacer

ISBN: 978-631-6557-57-5

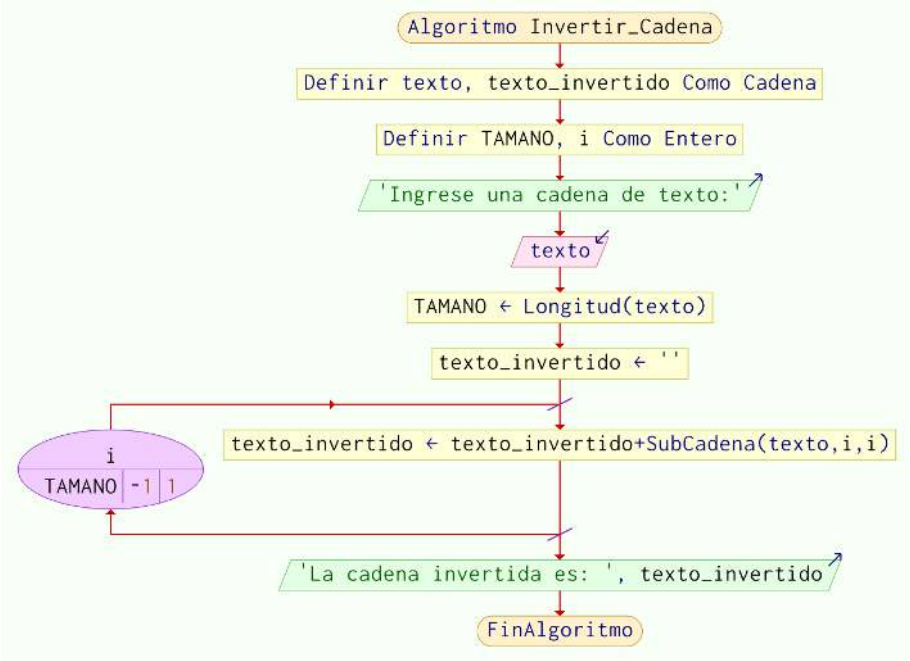
texto_invertido <- texto_invertido + SubCadena(texto, i, i)

Fin Para

Escribir "La cadena invertida es: ", texto_invertido

FinAlgoritmo

Diagrama de flujo



Implementación en Python

```
# Pedir al usuario que ingrese una cadena de texto
texto = input("Ingrese una cadena de texto: ")

# Inicializar la cadena invertida vacía
texto_invertido = ""

# Obtener el tamaño de la cadena de texto
TAMANO = len(texto)

# Invertir la cadena utilizando un bucle for que recorre el texto de atrás hacia adelante
for i in range(TAMANO - 1, -1, -1):
    texto_invertido += texto[i]

# Mostrar la cadena invertida
print("La cadena invertida es: ", texto_invertido)
```

```
Ingrese una cadena de texto: HOLA
La cadena invertida es:  ALOH
```

Se utiliza un bucle for que empieza en el último índice de la cadena (longitud - 1) y termina en el índice 0, decreciendo en 1 en cada iteración. Cada carácter se concatena a

ISBN: 978-631-6557-57-5

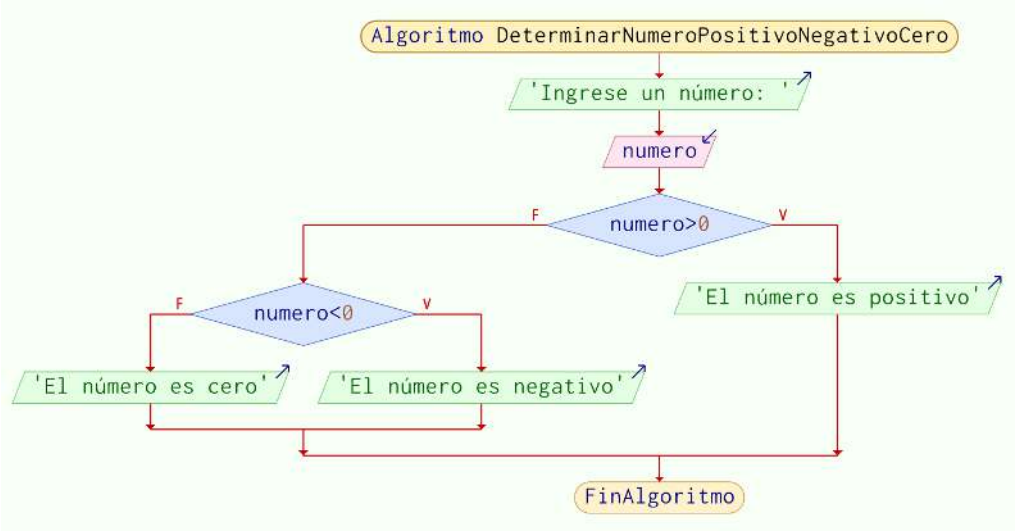
la variable texto_invertido. Finalmente, se imprime el resultado.

9) Determinar si un número dado es positivo, negativo o cero.

Pseudocódigo

```
Algoritmo DeterminarNumeroPositivoNegativoCero
    Escribir "Ingrese un número: "
    Leer numero

    Si numero > 0 Entonces
        Escribir "El número es positivo"
    Sino
        Si numero < 0 Entonces
            Escribir "El número es negativo"
        Sino
            Escribir "El número es cero"
        FinSi
    FinSi
FinAlgoritmo
```

Diagrama de flujo



El algoritmo comienza con el título "Algoritmo DeterminarNumeroPositivoNegativoCero".

Se solicita al usuario que ingrese un número con la instrucción "Ingrese un número:".

3) Luego, se toma el valor ingresado por el usuario y se almacena en la variable numero.

4) Se verifica si numero > 0 (número es mayor que cero). Si la condición es verdadera (v), se sigue la línea que lleva a la instrucción "El número es positivo" y luego se dirige hacia el fin del algoritmo. Si la condición es falsa (f), se procede al siguiente paso.

ISBN: 978-631-6557-57-5

5) Se verifica si numero < 0 (número es menor que cero). Si esta condición es verdadera (v), se sigue la línea que lleva a la instrucción "El número es negativo" y luego se dirige hacia el fin del algoritmo. Si la condición es falsa (f), se procede al siguiente paso.

6) Si ninguna de las condiciones anteriores es verdadera, significa que el número es cero. Se muestra el mensaje "El número es cero".

7) Finalmente, independientemente del camino seguido, todos llevan a "FinAlgoritmo", que indica el fin del proceso.

Implementación

En Python

```
numero = int(input("Ingresa un número: "))
if numero > 0:
    print("El número es positivo")
elif numero < 0:
    print("El número es negativo")
else:
    print("El número es cero")
```

```
Ingresa un número: 3
El número es positivo
```

En JavaScript

main.js

```
var numero = prompt("Ingrese un número: ");
numero = parseInt(numero);

if (numero > 0) {
   alert("El número es positivo");
} else if (numero < 0) {
   alert("El número es negativo");
} else {
   alert("El número es cero");
}
```

```
Ingrese un número: -2
El número es negativo
```

Solicita al usuario que ingrese un número utilizando prompt.

Convierte la entrada del usuario a un número entero con parseInt.

Comprueba si el número es mayor que cero, menor que cero o igual a cero.

ISBN: 978-631-6557-57-5

En C++

main.cpp

```cpp
#include <iostream>

int main() {
    int numero;

    std::cout << "Ingrese un número: ";
    std::cin >> numero;

    if (numero > 0) {
        std::cout << "El número es positivo" << std::endl;
    } else if (numero < 0) {
        std::cout << "El número es negativo" << std::endl;
    } else {
        std::cout << "El número es cero" << std::endl;
    }

    return 0;
}
```

```
Ingrese un número: 0
El número es cero
```

Se incluye la biblioteca iostream para permitir la entrada y salida de datos.

Se inicia la función principal main, que es el punto de entrada de un programa en C++.

Se declara la variable numero para almacenar la entrada del usuario.

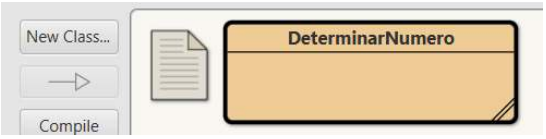
Se solicita al usuario que ingrese un número con std::cout y se captura con std::cin.

Se utiliza una estructura if-else para determinar si el número es positivo, negativo o cero.

Se muestra el resultado con std::cout.

El programa termina con un return 0, indicando que el programa ha terminado correctamente.

En Java



ISBN: 978-631-6557-57-5

```java
import java.util.Scanner;

public class DeterminarNumero {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int numero;

        System.out.print("Ingrese un número: ");
        numero = scanner.nextInt();

        if (numero > 0) {
            System.out.println("El número es positivo");
        } else if (numero < 0) {
            System.out.println("El número es negativo");
        } else {
            System.out.println("El número es cero");
        }

        scanner.close();
    }
}
```

```
Ingrese un número: -8
El número es negativo
```

Importa la clase Scanner para leer la entrada del usuario.

Declara la clase DeterminarNumero.

Dentro de la clase, el método main es el punto de entrada del programa.

Crea una instancia de Scanner para leer desde la entrada estándar (la consola).

Pide al usuario que ingrese un número y lo almacena en la variable numero.

Usa una estructura if-else para comprobar si el número es mayor, menor o igual a cero.

Imprime el resultado correspondiente en la consola.

Finalmente, cierra el objeto scanner para liberar recursos.

10) Hacer un programa que verifique si un año es bisiesto

Pseudocódigo

1) El algoritmo comienza con el título DeterminarSiUnAñoEsBisiesto.

2) Luego, el algoritmo pide al usuario que ingrese un año mostrando el mensaje: "Ingresa un año: ".

3) La respuesta del usuario se lee y se almacena en la variable año.

ISBN: 978-631-6557-57-5

4) A continuación, el algoritmo evalúa la condición compuesta por dos partes unidas por un O lógico:

- La primera parte verifica si el año es divisible por 4 (año % 4 = 0) y no es divisible por 100 (año % 100 <> 0). El símbolo % representa la operación de módulo (resto de la división), = se utiliza para la comparación de igualdad y <> se usa para la comparación de no igualdad.

- La segunda parte verifica si el año es divisible por 400 (año % 400 = 0).

```
Algoritmo DeterminarSiUnAñoEsBisiesto
    Escribir "Ingresa un año: "
    Leer año

    Si (año % 4 = 0 Y año % 100 ≠ 0) O (año % 400 = 0) Entonces
        Escribir "El año es bisiesto"
    Sino
        Escribir "El año no es bisiesto"
    FinSi
FinAlgoritmo

```

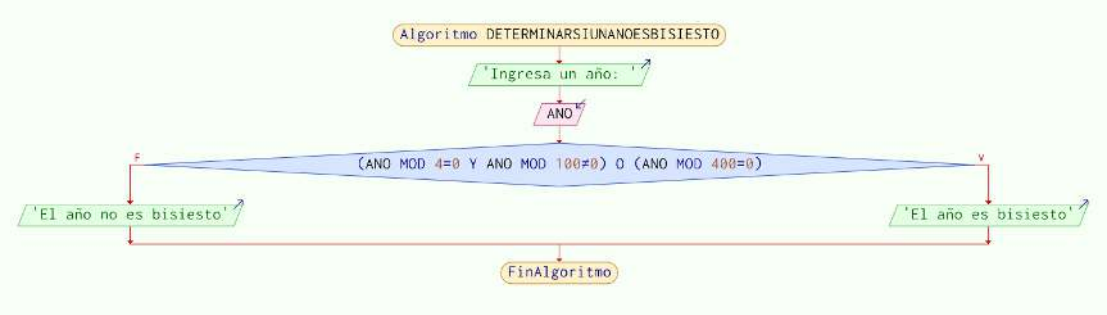
5) Si alguna de estas dos partes de la condición es verdadera (si el año es divisible por 4 y no por 100, o es divisible por 400), entonces el algoritmo concluye que el año es bisiesto y muestra el mensaje "El año es bisiesto".

6) Si la condición compuesta no se cumple (lo que significa que el año no es divisible por 4, o es divisible por 100 pero no por 400), el algoritmo concluye que el año no es bisiesto y muestra el mensaje "El año no es bisiesto".

7) Finalmente, el algoritmo termina después de mostrar uno de los dos mensajes, indicado por FinAlgoritmo.

ISBN: 978-631-6557-57-5

Diagrama de flujo



Implementación

En Python

```
año = int(input("Ingresa un año: "))
if (año % 4 == 0 and año % 100 != 0) or (año % 400 == 0):
    print("El año es bisiesto")
else:
    print("El año no es bisiesto")
```

```
Ingresa un año: 2024
El año es bisiesto
```

En JavaScript

```
    // Se usa prompt para obtener el año del usuario
let año = prompt("Ingresa un año:");

// Se convierte el valor ingresado a número, debido a que prompt devuelve una
    cadena
año = Number(año);

// Se verifica si el año es bisiesto según las reglas establecidas
if ((año % 4 === 0 && año % 100 !== 0) || año % 400 === 0) {
    alert("El año es bisiesto");
} else {
    alert("El año no es bisiesto");
}
```

```
Ingresa un año:2040
El año es bisiesto
```

11) Hacer un programa que cuente del 1 al 13 de 2 en 2

Pseudocódigo

El pseudocódigo representa un algoritmo que cuenta de 1 a 13 de dos en dos. Aquí está la interpretación del pseudocódigo línea por línea:

ISBN: 978-631-6557-57-5

```
Algoritmo ContarDeDosEnDosHastaTrece
    contador ← 1
    Mientras contador ≤ 13 Hacer
        Escribir contador
        contador ← contador + 2
    Fin Mientras
FinAlgoritmo
```

1) Algoritmo ContarDeDosEnDosHastaTrece: Es el inicio del algoritmo y el título describe su función, que es contar de dos en dos hasta llegar a trece.

2) contador <- 1: Inicializa una variable llamada contador y le asigna el valor de 1.

3) Mientras contador <= 13 Hacer: Inicia un bucle Mientras que se ejecutará mientras el valor de contador sea menor o igual a 13.

4) Escribir contador: Dentro del bucle, el algoritmo escribe el valor actual de contador.

5) contador <- contador + 2: Aumenta el valor de contador en 2.

6) Fin Mientras: Indica el fin del bucle Mientras.

7) FinAlgoritmo: Marca el fin del algoritmo.

Diagrama de flujo

Implementación

En Python

```
contador = 1
while contador <= 13:
    print(contador)
    contador += 2
```

ISBN: 978-631-6557-57-5

```
1
3
5
7
9
11
13
```

1) contador = 1: Se inicializa la variable contador en 1. Esta variable llevará la cuenta de los números en el bucle.

2. while contador <= 13: Comienza un bucle while que se ejecutará siempre que el valor de contador sea menor o igual a 13.

3. print(contador): Dentro del bucle, se imprime el valor actual de contador. Como contador comienza en 1, la primera vez que se ejecuta esta línea se imprimirá 1.

4) contador += 2: Se incrementa el valor de contador en 2. Esto significa que después de imprimir el valor de contador, se sumará 2 antes de la próxima iteración del bucle.

En JavaScript

```javascript
let contador = 1;
while (contador <= 13) {
    console.log(contador);
    contador += 2;
}
```

Este código en JavaScript funcionará de manera similar al código Python original: inicializa una variable contador en 1, entra en un bucle while que se ejecuta mientras contador sea menor o igual a 13, y en cada iteración del bucle, imprime el valor actual de contador en la consola y luego incrementa contador en 2. Los resultados impresos serán los números impares entre 1 y 13.

En C++

```cpp
#include <iostream>

int main() {
    int contador = 1;
    while (contador <= 13) {
        std::cout << contador << std::endl;
        contador += 2;
    }
    return 0;
}
```

int contador = 1;: Se declara una variable entera llamada contador y se le asigna un valor inicial de 1.

ISBN: 978-631-6557-57-5

while (contador <= 13) {: Se inicia un bucle while que se repetirá mientras el valor de contador sea menor o igual a 13.

std::cout << contador << std::endl;: Dentro del bucle, el programa imprime el valor actual de contador en la consola y luego inserta un salto de línea (std::endl).

contador += 2;: Se incrementa el valor de contador en 2.

En Java

```java
public class Contador {
    public static void main(String[] args) {
        int contador = 1;
        while (contador <= 13) {
            System.out.println(contador);
            contador += 2;
        }
    }
}
```

Define una clase pública llamada Contador.

Dentro de la clase Contador, el método main es declarado.

Declara la variable contador y la inicializa con el valor de 1.

Ejecuta un bucle while que se repite mientras el valor de contador sea menor o igual a 13.

Dentro del bucle, imprime el valor actual de contador en la consola y luego incrementa contador en 2.

Una vez que contador es mayor que 13, el bucle termina y con él el programa.

12) Mostrar los elementos de una lista uno por uno

Pseudocódigo

```
Algoritmo ImprimirElementosDeLaLista
    Dimension lista[3]

    lista[1] ← "María"
    lista[2] ← "Gisel"
    lista[3] ← "Xavier"
    indice ← 1

    Mientras indice ≤ 3 Hacer
        Escribir lista[indice]
        indice ← indice + 1
    Fin Mientras
Fin Algoritmo
```

ISBN: 978-631-6557-57-5

1) Se declara una lista de tamaño 3, que es capaz de contener tres elementos.

2) Se asignan los valores "María", "Gisel" y "Xavier" a las tres posiciones de la lista (lista[1], lista[2] y lista[3], respectivamente).

3) Se inicializa la variable indice con el valor 1.

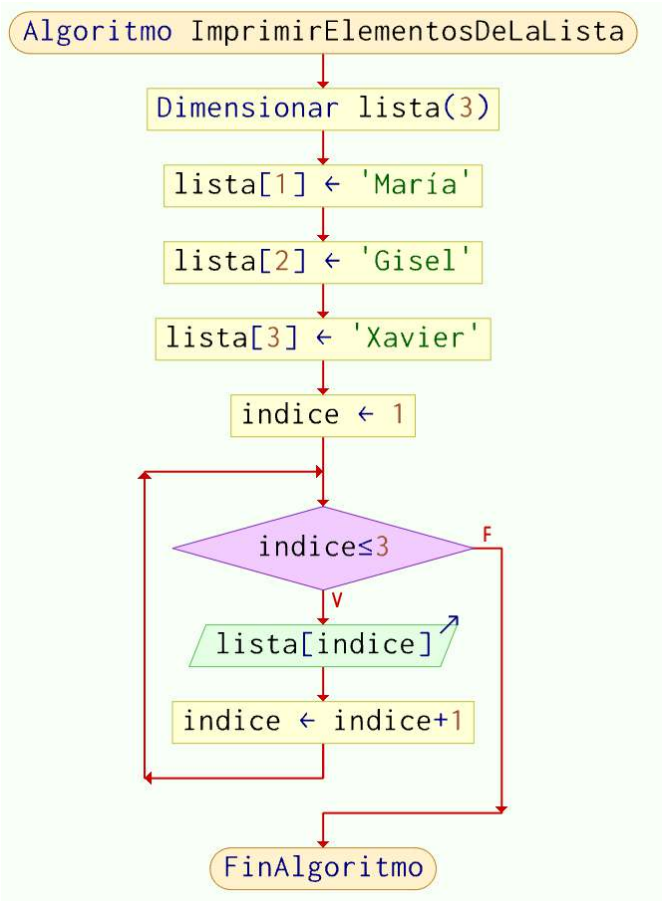
4) Se inicia un bucle Mientras que ejecuta sus instrucciones internas mientras que el valor de indice sea menor o igual a 3.

5) Dentro del bucle, se utiliza la instrucción Escribir para imprimir el valor actual de la lista correspondiente al indice actual.

6) Tras imprimir, se incrementa el valor de índice en 1, para pasar al siguiente elemento de la lista en la próxima iteración del bucle.

7) Una vez que indice supera el valor 3, el bucle Mientras termina.

Diagrama de flujo



ISBN: 978-631-6557-57-5

Implementación

En Python

```
lista = ["María", "Gisel", "Xavier"]
indice = 0
while indice < len(lista):
    print(lista[indice])
    indice += 1
```

```
María
Gisel
Xavier
```

1) Se crea una lista llamada lista que contiene tres elementos: las cadenas "María", "Gisel" y "Xavier".

2) Se inicializa una variable llamada indice con el valor de 0.

3) Se entra en un bucle while que se ejecutará mientras indice sea menor que la longitud de lista (que es len(lista) y corresponde al número de elementos en la lista.

4) Dentro del bucle, se imprime el elemento de lista que está en la posición indicada por indice.

5) Después de imprimir, se incrementa indice en 1.

6. Este proceso se repite hasta que indice es igual a la longitud de la lista, en cuyo punto el bucle while deja de ejecutarse.

En JavaScript

```
let lista = ["María", "Gisel", "Xavier"];
let indice = 0;

while (indice < lista.length) {
    console.log(lista[indice]);
    indice++;
}
```

Declara un array llamado lista que contiene los nombres "María", "Gisel" y "Xavier".

Inicializa una variable llamada indice con el valor 0.

Ejecuta un bucle while que continuará mientras el valor de indice sea menor que la longitud del array lista.

ISBN: 978-631-6557-57-5

Dentro del bucle, se imprime el elemento actual de la lista usando console.log.

Incrementa el valor de indice en 1 después de cada impresión para avanzar al siguiente elemento de la lista.

El bucle se repite hasta que indice sea igual a la longitud de la lista, en cuyo punto el bucle termina y por consiguiente el script también.

En C++

```cpp
#include <iostream>
#include <vector>
#include <string>

int main() {
    std::vector<std::string> lista = {"María", "Gisel", "Xavier"};
    int indice = 0;

    while (indice < lista.size()) {
        std::cout << lista[indice] << std::endl;
        indice += 1;
    }

    return 0;
}
```

Incluye las bibliotecas necesarias para manejar entradas/salidas y la clase de vector, que se utiliza para almacenar la lista de strings.

Declara un vector de strings llamado lista e inicializa con los nombres "María", "Gisel" y "Xavier".

Declara una variable indice y la inicializa con 0.

Ejecuta un bucle while que verifica si el indice es menor que el tamaño del vector lista utilizando lista.size().

Imprime el elemento actual del vector lista correspondiente al indice y luego incrementa indice en 1.

El bucle continúa hasta que indice sea igual al tamaño del vector, en cuyo punto termina el bucle y el programa.

13) Hacer un bucle que genere de forma aleatorio números y que mientras no asome por ejemplo el 4 no se detenga.

ISBN: 978-631-6557-57-5

Pseudocódigo

```
Algoritmo GenerarNumerosAleatoriosHastaCuatro
    Definir numero Como Entero;
    numero ← 0;
    Mientras numero ≠ 4 Hacer
        numero ← Aleatorio(1,10);
        Escribir numero;
    Fin Mientras
FinAlgoritmo
```

Diagrama de flujo

Se define la variable numero como un entero y la inicializamos en 0.

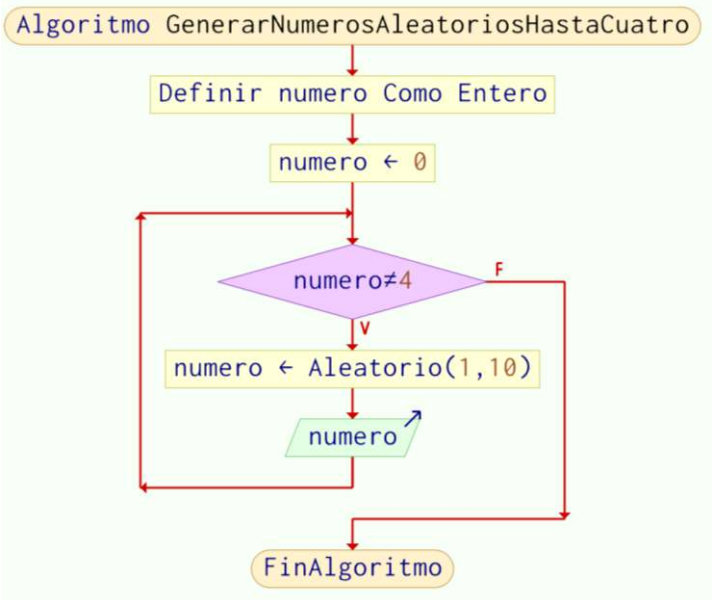
Luego, se entra en un bucle Mientras que se ejecuta mientras el valor de numero sea distinto (<>) de 4.

Dentro del bucle, asignamos a numero un valor aleatorio entre 1 y 10 usando la función Aleatorio(1,10).

A continuación, se escribe el valor de numero.

El bucle continuará hasta que numero sea igual a 4.

Diagrama de flujo



Implementación

1) Importa el módulo random, que proporciona funciones para generar números aleatorios.

2) Inicializa una variable llamada numero con el valor 0.

ISBN: 978-631-6557-57-5

3) Entra en un bucle while que seguirá ejecutándose mientras que numero sea diferente de 4 (`numero != 4`).

```
import random

numero = 0
while numero != 4:
    numero = random.randint(1, 10)
    print(numero)
```

```
9
5
10
10
10
8
8
8
3
9
8
4
```

4) Dentro del bucle, la función random.randint(1, 10) genera un número entero aleatorio entre 1 y 10 (inclusive) y asigna este nuevo valor a la variable numero.

5) Imprime el valor actual de numero.

6) El bucle terminará solo cuando la función randint genere el número 4. Mientras el número generado sea diferente de 4, el bucle seguirá ejecutándose.

En JavaScript

```
let numero = 0;

while (numero !== 4) {
    numero = Math.floor(Math.random() * 10) + 1;
    console.log(numero);
}
```

Inicializa una variable numero con el valor 0.

Comienza un bucle while que se ejecutará mientras numero sea diferente de 4.

Dentro del bucle, Math.random() genera un número aleatorio entre 0 (inclusive) y 1 (exclusivo), luego se multiplica por 10 para cambiar el rango a entre 0 y 10, y Math.floor() redondea hacia abajo al entero más cercano, para obtener un rango de 0 a 9. Al sumar 1,

ISBN: 978-631-6557-57-5

ajustamos el rango a entre 1 y 10.

Se imprime el valor de numero en la consola con console.log.

Si numero es 4, el bucle while terminará. Si no, el bucle continuará generando y mostrando números aleatorios.

En C++

```cpp
#include <iostream>
#include <cstdlib>
#include <ctime>

int main() {
    // Inicializa el generador de números aleatorios.
    std::srand(std::time(0));

    int numero = 0;

    while (numero != 4) {
        // Genera un número aleatorio entre 1 y 10.
        numero = std::rand() % 10 + 1;

        // Imprime el número.
        std::cout << numero << std::endl;
    }

    return 0;
}

```

Incluye las bibliotecas iostream para la entrada y salida de datos, cstdlib para la función rand() que genera números aleatorios, y ctime para la función time() que se utiliza para inicializar la semilla del generador de números aleatorios con el tiempo actual.

Inicializa la semilla del generador de números aleatorios con el valor actual de tiempo para asegurarse de que los números aleatorios sean diferentes en cada ejecución del programa.

Declara una variable entera numero y la inicializa con 0.

Ejecuta un bucle while que sigue iterando mientras numero sea diferente de 4.

Dentro del bucle, genera un número aleatorio entre 1 y 10 usando rand() y lo asigna a numero.

Imprime numero en la consola.

Si numero es igual a 4, el bucle while terminará y con él el programa.

14) Hacer un programa que sume los números ingresados hasta que se ingrese un

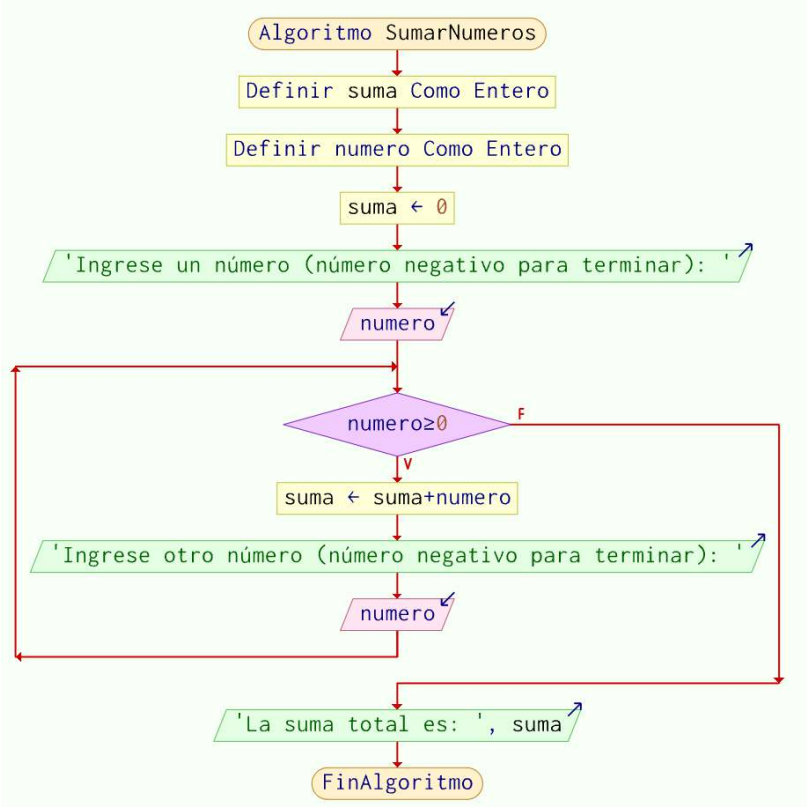
ISBN: 978-631-6557-57-5

número negativo.

Pseudocódigo

```
Algoritmo SumarNumeros
    Definir suma Como Entero;
    Definir numero Como Entero;

    suma ← 0;
    Escribir "Ingrese un número (número negativo para terminar): ";
    Leer numero;

    Mientras numero ≥ 0 Hacer
        suma ← suma + numero;
        Escribir "Ingrese otro número (número negativo para terminar): ";
        Leer numero;
    FinMientras

    Escribir "La suma total es: ", suma;
FinAlgoritmo
```

Diagrama de flujo



ISBN: 978-631-6557-57-5

Implementación

```
suma = 0
numero = int(input("Ingrese un número (número negativo para terminar): "))
while numero >= 0:
    suma += numero
    numero = int(input("Ingrese otro número (número negativo para terminar): "))
print("La suma total es:", suma)
```

```
Ingrese un número (número negativo para terminar): 5
Ingrese otro número (número negativo para terminar): 2
Ingrese otro número (número negativo para terminar): 8
Ingrese otro número (número negativo para terminar): -4
La suma total es: 15
```

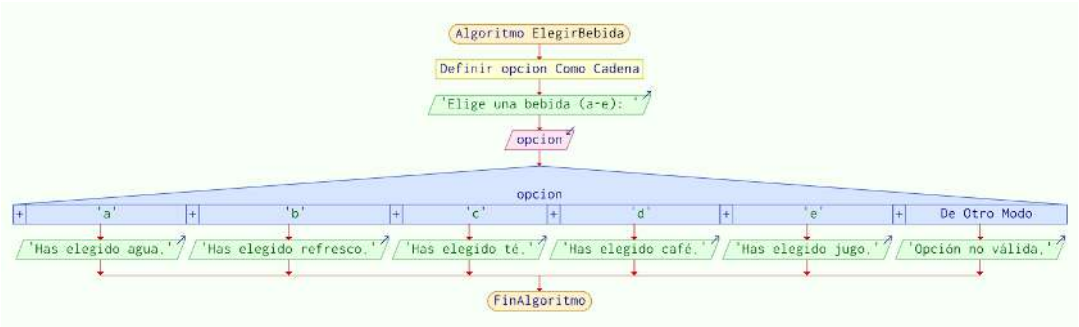
15) Realizar un programa que permita seleccionar una bebida

Pseudocódigo

```
Algoritmo ElegirBebida
    Definir opcion Como Caracter

    Escribir "Elige una bebida (a-e): "
    Leer opcion

    Segun opcion Hacer
        'a':
            Escribir "Has elegido agua."
        'b':
            Escribir "Has elegido refresco."
        'c':
            Escribir "Has elegido té."
        'd':
            Escribir "Has elegido café."
        'e':
            Escribir "Has elegido jugo."
        De Otro Modo:
            Escribir "Opción no válida."
    Fin Segun
FinAlgoritmo
```

ISBN: 978-631-6557-57-5

Diagrama de flujo



Implementación

En C++

main.cpp

```cpp
#include <iostream> // Necesario para std::cout y std::cin

int main() {
    char opcion;
    std::cout << "Elige una bebida (a-e): ";
    std::cin >> opcion;

    switch (opcion) {
        case 'a': std::cout << "Has elegido agua.\n"; break;
        case 'b': std::cout << "Has elegido refresco.\n"; break;
        case 'c': std::cout << "Has elegido té.\n"; break;
        case 'd': std::cout << "Has elegido café.\n"; break;
        case 'e': std::cout << "Has elegido jugo.\n"; break;
        default: std::cout << "Opción no válida.\n";
    }

    return 0;
}
```

```
Elige una bebida (a-e): c
Has elegido té.
```

char opcion;: Se declara una variable de tipo char llamada opcion para almacenar la opción de bebida que el usuario elija.

std::cout << "Elige una bebida (a-e): ";: Se imprime en la consola el mensaje para pedir al usuario que elija una bebida.

ISBN: 978-631-6557-57-5

std::cin >> opcion;: Se lee la entrada del teclado y se guarda en la variable opcion.

switch (opcion) {: Se inicia una declaración switch que permitirá ejecutar diferentes bloques de código en función del valor de opcion.

case 'a': std::cout << "Has elegido agua.\n"; break;: Si opcion es a, se imprime "Has elegido agua." y se termina el switch con break.

Las siguientes líneas case 'b' hasta case 'e': Funcionan de manera similar para las otras opciones de bebidas, imprimiendo el mensaje correspondiente a cada una.

default: std::cout << "Opción no válida.\n";: Si ninguna de las opciones anteriores coincide con el valor de opcion, se ejecuta el caso default, imprimiendo "Opción no válida.".

return 0;: La función main termina y devuelve 0, lo que indica que el programa se ha ejecutado correctamente.

En Java

```java
import java.util.Scanner;

public class SeleccionDeBebida {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Elige una bebida (a-e): ");
        String opcion = scanner.nextLine();

        switch (opcion) {
            case "a":
                System.out.println("Has elegido agua.");
                break;
            case "b":
                System.out.println("Has elegido refresco.");
                break;
            case "c":
                System.out.println("Has elegido té.");
                break;
            case "d":
                System.out.println("Has elegido café.");
                break;
            case "e":
                System.out.println("Has elegido jugo.");
                break;
            default:
                System.out.println("Opción no válida.");
                break;
        }
```

ISBN: 978-631-6557-57-5

```
        scanner.close();
    }
}
```

Este código en Java hace lo siguiente:

Importa el Scanner que se usa para leer la entrada del usuario.

Declara la clase SeleccionDeBebida y el método main, que es el punto de entrada de cualquier aplicación Java.

Crea un objeto scanner para capturar la entrada del usuario.

Pide al usuario que elija una bebida y guarda su elección en la variable opcion.

Usa una declaración switch para comparar la opcion con los casos previstos y ejecuta el código correspondiente a cada caso. Si la opcion no coincide con ninguno de los casos, ejecuta el caso default.

Cierra el scanner para evitar posibles fugas de recursos.

En JavaScript

main.js

```
let opcion = prompt("Elige una bebida (a-e): ");

switch (opcion) {
    case 'a':
        alert("Has elegido agua.");
        break;
    case 'b':
        alert("Has elegido refresco.");
        break;
    case 'c':
        alert("Has elegido té.");
        break;
    case 'd':
        alert("Has elegido café.");
        break;
    case 'e':
        alert("Has elegido jugo.");
        break;
    default:
        alert("Opción no válida.");
        break;
}
```

Solicita al usuario que introduzca una opción mediante un prompt.

ISBN: 978-631-6557-57-5

Evalúa la opción seleccionada con un switch.

Muestra una alerta correspondiente a la elección del usuario.

Si la opción introducida no es válida, muestra una alerta de "Opción no válida".

En Python

```python
opcion = input("Elige una bebida (a-e): ")

if opcion == 'a':
    print("Has elegido agua.")
elif opcion == 'b':
    print("Has elegido refresco.")
elif opcion == 'c':
    print("Has elegido té.")
elif opcion == 'd':
    print("Has elegido café.")
elif opcion == 'e':
    print("Has elegido jugo.")
else:
    print("Opción no válida.")
```

Es lo más parecido al uso del switch.

16) Realizar un programa en el cual se indique la edad y se muestre la etapa de la vida en la que se encuentra.

Pseudocódigo

Se debe adaptar el uso de switch y los rangos, debido a que PSeInt no maneja directamente los rangos en las estructuras Segun. Una forma común de manejar esto en PSeInt es utilizando una serie de Si...SinoSi...Sino.

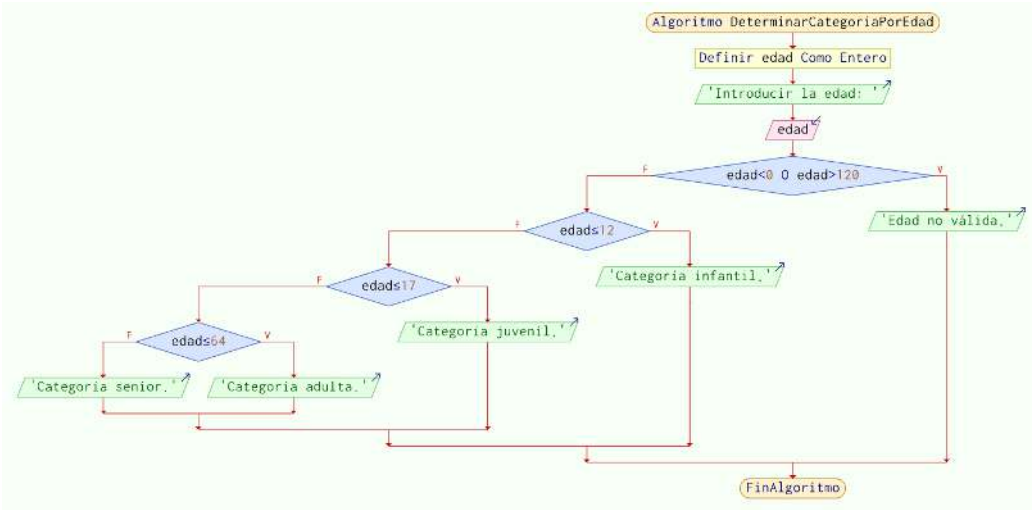
```
Algoritmo DeterminarCategoriaPorEdad
    Definir edad Como Entero;

    Escribir "Introducir la edad: ";
    Leer edad;

    Si edad < 0 O edad > 120 Entonces
        Escribir "Edad no válida.";
    Sino
        Si edad ≤ 12 Entonces
            Escribir "Categoría infantil.";
        Sino
            Si edad ≤ 17 Entonces
                Escribir "Categoría juvenil.";
            Sino
                Si edad ≤ 64 Entonces
                    Escribir "Categoría adulta.";
                Sino
                    Escribir "Categoría senior.";
                FinSi
            FinSi
        FinSi
    FinSi
FinAlgoritmo
```

ISBN: 978-631-6557-57-5

La estructura anidada de Si...Sino valida primero si la edad es menor que 0 o mayor que 120, en cuyo caso imprime que la edad no es válida. Si la edad está dentro del rango válido, entonces procede a evaluar en qué categoría se clasifica según los rangos de edad definidos.

Las estructuras anidadas garantizan que solo se evalúe la siguiente condición si la anterior no se cumplió, asegurando así una evaluación ordenada y excluyente, similar a la lógica de un switch con rangos.

Diagrama de flujo



Implementación en C++

```cpp
#include <iostream>
int main() {
    int edad;
    std::cout << "Introducir la edad: ";
    std::cin >> edad;

    switch (edad) {
        case 0 ... 12:
            std::cout << "Categoria infantil.\n";
            break;
        case 13 ... 17:
            std::cout << "Categoría juvenil.\n";
            break;
        case 18 ... 64:
            std::cout << "Categoría adulta.\n";
            break;
        case 65 ... 120:
            std::cout << "Categoría senior.\n";
            break;
        default:
            std::cout << "Edad no válida.\n";
            break;
    }
```

ISBN: 978-631-6557-57-5

```
24      return 0;
25  }
```

```
Introducir la edad: 19
Categoría adulta.
```

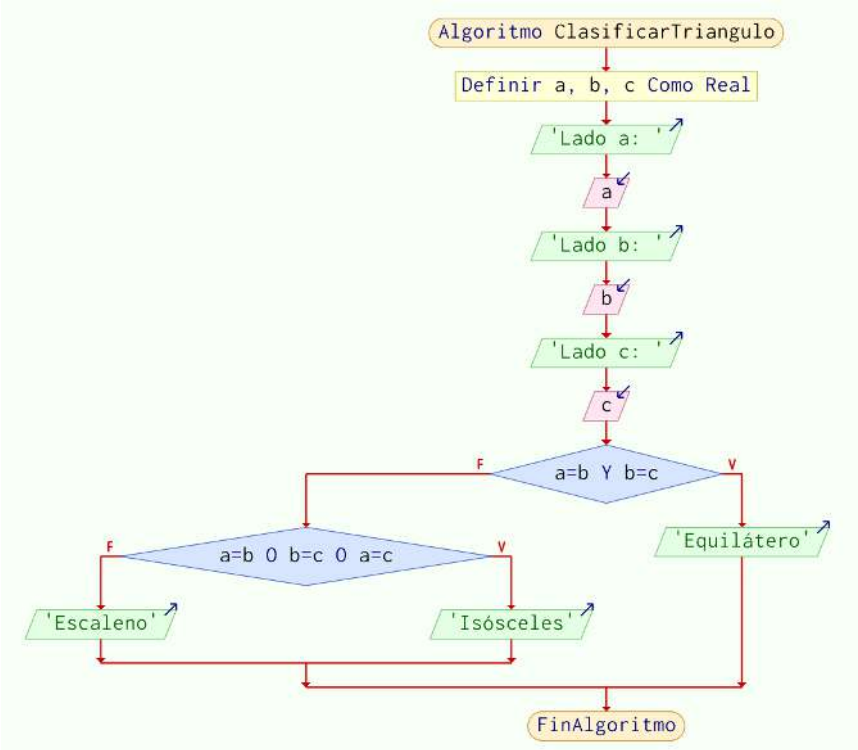
Introducir los lados de un triángulo. El programa debe indicar que tipo de triángulo es.

Implementación

```
1  Algoritmo ClasificarTriangulo
2      Definir a, b, c Como Real;
3
4      Escribir "Lado a: ";
5      Leer a;
6      Escribir "Lado b: ";
7      Leer b;
8      Escribir "Lado c: ";
9      Leer c;
10
11     Si a = b Y b = c Entonces
12         Escribir "Equilátero";
13     Sino
14         Si a = b O b = c O a = c Entonces
15             Escribir "Isósceles";
16         Sino
17             Escribir "Escaleno";
18         FinSi
19     FinSi
20 FinAlgoritmo
```

ISBN: 978-631-6557-57-5

Diagrama de flujo



En Python

```
a, b, c = sorted([float(input("Lado a: ")), float(input("Lado b: ")), float(input("Lado c: "))])

if a == b == c:
    print("Equilátero")
elif a == b or b == c or a == c:
    print("Isósceles")
else:
    print("Escaleno")
```

```
Lado a: 10
Lado b: 10
Lado c: 10
Equilátero
```

1) Entrada de Datos: Pide al usuario que ingrese los valores de los tres lados de un triángulo, denominados a, b, y c. Los valores son tomados como flotantes (números decimales), lo que permite que el usuario introduzca valores como 6.5, 11.8, etc.

2) Ordenamiento: La función sorted() se utiliza para ordenar los lados del triángulo en orden ascendente. Esto es útil, por ejemplo, para simplificar ciertas comprobaciones o cálculos futuros, aunque para este propósito específico de clasificar el tipo de triángulo

ISBN: 978-631-6557-57-5

por sus lados, no es estrictamente necesario ordenarlos.

3) Clasificación del Triángulo:

- Equilátero: La primera condición if a == b == c verifica si todos los lados son iguales. Si es así, imprime "Equilátero", indicando que el triángulo tiene todos sus lados de igual longitud.

- Isósceles: El elif a == b or b == c or a == c verifica si al menos dos lados son iguales. Si esta condición se cumple, imprime "Isósceles", lo que significa que el triángulo tiene dos lados iguales y uno diferente.

- Escaleno: La condición else se ejecuta si las condiciones anteriores no se cumplen, lo que significa que ninguno de los lados es igual a otro. En este caso, imprime "Escaleno", indicando que el triángulo tiene todos los lados de diferentes longitudes.

En JavaScript

main.js

```
let a = parseFloat(prompt("Lado a: "));
let b = parseFloat(prompt("Lado b: "));
let c = parseFloat(prompt("Lado c: "));

// Ordenar los lados para garantizar la comparación adecuada
let lados = [a, b, c].sort((x, y) => x - y);
a = lados[0];
b = lados[1];
c = lados[2];

if (a === b && b === c) {
    console.log("Equilátero");
} else if (a === b || b === c || a === c) {
    console.log("Isósceles");
} else {
    console.log("Escaleno");
}
```

Entrada de Datos: Se usa prompt para solicitar al usuario que ingrese las medidas de los tres lados del triángulo, y se convierten a tipo flotante con parseFloat para manejar adecuadamente los decimales.

Ordenación de Lados: Los lados se almacenan en un array que luego se ordena usando sort para facilitar la comparación. Esto asegura que a <= b <= c, lo cual es útil para la simplificación de las condiciones lógicas, especialmente si decides agregar validaciones de triángulos (como verificar que la suma de los dos lados menores sea mayor que el lado más largo).

ISBN: 978-631-6557-57-5

Evaluación de Condiciones:

Equilátero: Se verifica si todos los lados son iguales.

Isósceles: Se verifica si al menos dos lados son iguales.

Escaleno: Se concluye que todos los lados son diferentes si las condiciones anteriores no se cumplen.

En Java

```java
import java.util.Scanner;
import java.util.Arrays;

public class ClasificacionTriangulo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Lado a: ");
        double a = scanner.nextDouble();
        System.out.print("Lado b: ");
        double b = scanner.nextDouble();
        System.out.print("Lado c: ");
        double c = scanner.nextDouble();

        double[] lados = {a, b, c};
        Arrays.sort(lados); // Ordena los lados para garantizar que a <= b <= c

        a = lados[0];
        b = lados[1];
        c = lados[2];

        if (a == b && b == c) {
            System.out.println("Equilátero");
        } else if (a == b || b == c || a == c) {
            System.out.println("Isósceles");
        } else {
            System.out.println("Escaleno");
        }

        scanner.close();
    }
}
```

Importaciones: Se importan las clases Scanner para la entrada de datos y Arrays para poder ordenar los lados del triángulo y realizar comparaciones.

Entrada de Datos: Se usa Scanner para obtener las longitudes de los tres lados del triángulo del usuario. Se solicitan al usuario los lados a, b, y c del triángulo. Los valores se leen como tipo double, que es adecuado para números que pueden tener decimales.

Ordenación de los Lados: Se almacenan los lados en un array double y se ordena el array usando Arrays.sort() para facilitar la comparación. Esto asegura que a <= b <= c, lo cual simplifica las condiciones siguientes. Lo cual garantiza que el lado menor está en lados[0], el siguiente en lados[1], y el mayor en lados[2].

ISBN: 978-631-6557-57-5

Condiciones:

Equilátero: Se verifica si los tres lados son iguales.

Isósceles: Se verifica si al menos dos de los lados son iguales.

Escaleno: Se determina si no se cumplen las condiciones anteriores, indicando que todos los lados son diferentes.

Cierre del Scanner: Es importante cerrar el Scanner al final del programa para evitar fugas de recursos, mediante scanner.close() (Easttom, 2003).

En C++

```cpp
#include <iostream>
#include <algorithm> // Para std::sort
#include <vector>

int main() {
    double a, b, c;
    std::cout << "Lado a: ";
    std::cin >> a;
    std::cout << "Lado b: ";
    std::cin >> b;
    std::cout << "Lado c: ";
    std::cin >> c;

    // Almacenar los lados en un vector y ordenarlos
    std::vector<double> lados = {a, b, c};
    std::sort(lados.begin(), lados.end());

    // Reasignar a, b y c de forma ordenada
    a = lados[0];
    b = lados[1];
    c = lados[2];
```

```cpp

    if (a == b && b == c) {
        std::cout << "Equilátero\n";
    } else if (a == b || b == c || a == c) {
        std::cout << "Isósceles\n";
    } else {
        std::cout << "Escaleno\n";
    }

    return 0;
}
```

Inclusión de bibliotecas: Se incluye <iostream> para la entrada y salida estándar, y <algorithm> para usar funciones como std::sort. También se incluye <vector> porque se utiliza un vector para almacenar los lados del triángulo.

Entrada de datos: Se solicita al usuario que introduzca las medidas de los lados a, b y c del triángulo.

Ordenación de los lados: Los lados se almacenan en un vector lados, que luego se ordena con std::sort para garantizar que a <= b <= c. Esto facilita las comparaciones

ISBN: 978-631-6557-57-5

subsiguientes.

Reasignación de valores ordenados: Después de ordenar el vector, se reasignan los valores de a, b y c en orden ascendente.

Evaluación de condiciones:

Equilátero: Todos los lados son iguales. Se verifica con la condición a == b && b == c.

Isósceles: Al menos dos lados son iguales. Se verifica con la condición a == b || b == c || a == c.

Escaleno: Ninguno de los lados es igual a otro. Se determina con el else al final, que se ejecuta si las condiciones anteriores no son verdaderas.

Finalización del programa: El programa finaliza con return 0;, indicando que terminó correctamente.

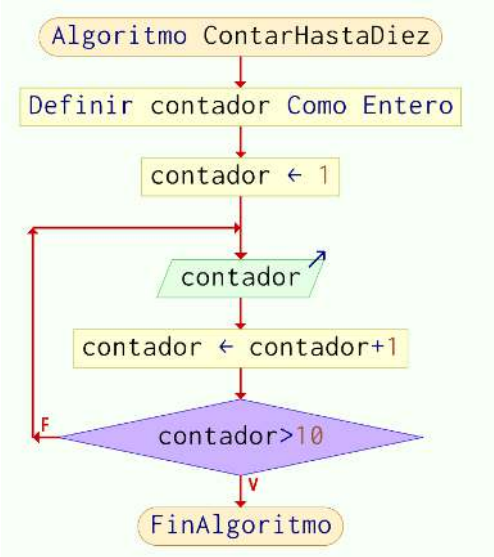
17) Imprimir de forma secuencial del 1 al 10, utilizando do-while.

Pseudocódigo

```
1  Algoritmo ContarHastaDiez
2      Definir contador Como Entero;
3      contador ← 1;
4      Repetir
5          Escribir contador;
6          contador ← contador + 1;
7      Hasta Que contador > 10
8  FinAlgoritmo
```

ISBN: 978-631-6557-57-5

Diagrama de flujo



Implementación

En C++

```cpp
#include <iostream>

int main() {
    int contador = 1;
    do {
        std::cout << contador << std::endl;
        contador++;
    } while (contador <= 10);
    return 0;
}
```

```
1
2
3
4
5
6
7
8
9
10
```

1) Declaración de una variable contador: Se declara e inicializa la variable contador con el valor 1.

2) Ejecución del bloque do-while:

- Bloque do: Este bloque se ejecuta al menos una vez, independientemente de la condición del while. Dentro del bloque, se realiza lo siguiente:

- std::cout << contador << std::endl;: Imprime el valor actual de contador seguido de

ISBN: 978-631-6557-57-5

un salto de línea (std::endl). Al empezar, contador vale 1, por lo que se imprime "1".

- contador++;: Incrementa el valor de contador en 1. Es decir, después de imprimir, contador se incrementa a 2, luego a 3, y así sucesivamente cada vez que se repite el bucle.

- Condición while: Después de cada ejecución del bloque do, se evalúa la condición contador <= 10. Mientras esta condición sea verdadera, el bucle continuará ejecutándose. Esto significa que el bucle se repetirá 10 veces en total, incrementando contador desde 1 hasta 10 y imprimiendo cada valor.

3) Finalización del programa:

El programa termina devolviendo 0, lo cual es un código de salida estándar que indica que el programa ha finalizado correctamente.

En Java

```
public class Contador1al10 {
    public static void main(String[] args) {
        int contador = 1;
        do {
            System.out.println(contador);
            contador++;
        } while (contador <= 10);
    }
}
```

El programa imprime los números del 1 al 10, cada uno en una nueva línea.

Debido a que la condición del while se evalúa después del bloque do, el bloque de código dentro del do se ejecuta al menos una vez, asegurando que incluso si la condición fuera falsa desde el principio (lo cual no ocurre en este caso), el código dentro de do se ejecutaría una vez.

Inicialización de la Variable:

int contador = 1;: Se declara e inicializa una variable contador con el valor 1. Esta variable se utilizará para contar de 1 a 10.

Bucle do-while:

Bloque do:

System.out.println(contador);: Imprime el valor actual de contador en la consola.

contador++;: Incrementa el valor de contador en 1 después de imprimirlo. Esto significa que contador aumentará en cada iteración del bucle hasta que alcance el valor 10.

Condición while:

ISBN: 978-631-6557-57-5

while (contador <= 10);: Después de cada ejecución del bloque do, se verifica esta condición. Si el valor de contador es menor o igual a 10, el bucle se repetirá. El bucle terminará cuando contador sea mayor que 10.

En JavaScript

```
1  let contador = 1;
2  do {
3      console.log(contador);
4      contador++;
5  } while (contador <= 10);
```

El programa inicia con contador establecido en 1.

Entra al bucle do, imprime el valor 1, y luego incrementa contador a 2.

La condición while verifica si contador es menor o igual a 10. Como 2 es menor que 10, el bucle se repite.

Este proceso continúa hasta que contador se incrementa a 11, en ese momento, la condición while se vuelve falsa y el bucle se detiene.

En la consola se verán los números del 1 al 10, cada uno en una nueva línea.

En Python

Python no tiene una estructura do-while nativa, pero se puede emular su comportamiento usando un bucle while con una condición que siempre es verdadera al inicio, y un break si la condición de parada no se cumple (Hazrat, 2023).

```
contador = 1
while True:
    print(contador)
    contador += 1
    if contador > 10:
        break
```

contador = 1: Aquí se declara e inicializa la variable contador con el valor 1. Esta variable se usa para llevar la cuenta de las iteraciones del bucle.

while True: Se establece un bucle while que se ejecutará indefinidamente debido a que la condición siempre es True. Este tipo de bucle se llama "bucle infinito".

ISBN: 978-631-6557-57-5

print(contador): En cada iteración del bucle, el programa imprime el valor actual de contador. Inicialmente, imprime 1.

contador += 1: Después de imprimir el valor, se incrementa contador en 1. Esto actualiza el valor de contador para la próxima iteración del bucle.

if contador > 10: Dentro del bucle, se verifica si contador ha superado el valor de 10.

break: Si la condición es verdadera (es decir, contador es mayor que 10), se ejecuta break, que interrumpe el bucle while y termina su ejecución.

18) Hacer un programa que tenga dos opciones a elegir continuar o salir. Mientras elija continuar debe seguir preguntando y cuando se pulse salir debe terminar.

Pseudocódigo

```
Proceso MenuOpciones
    Definir opcion Como Caracter
    Repetir
        Escribir "Elige una opción: "
        Escribir "1. Continuar"
        Escribir "2. Salir"
        Leer opcion
        Escribir "Opción seleccionada: ", opcion
    Hasta Que opcion = "2"
FinProceso
```

Se define un proceso llamado MenuOpciones

Se declara una variable opcion de tipo Caracter.

Inicia un bucle Repetir que se ejecutará al menos una vez.

Muestra las opciones del menú al usuario.

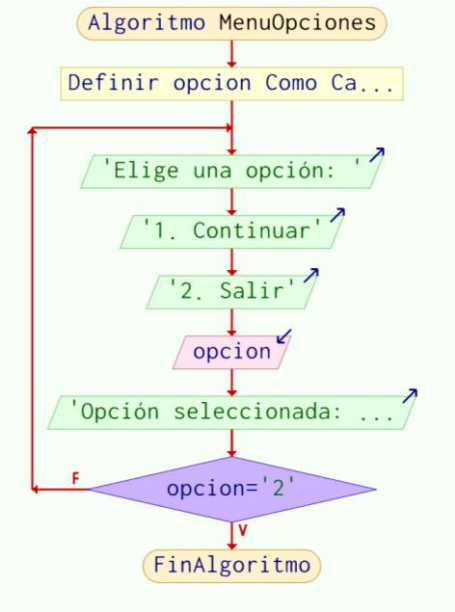
Lee la opción seleccionada por el usuario y la almacena en la variable opcion.

Muestra la opción seleccionada por el usuario.

El bucle se repite hasta que la opción seleccionada sea "2".

Diagrama de flujo

ISBN: 978-631-6557-57-5



Implementación

En JavaScript

```
1  let opcion;
2  do {
3      opcion = prompt("Elige una opción: \n1. Continuar \n2. Salir \n");
4      console.log("Opción seleccionada: " + opcion);
5  } while (opcion != "2");
```

```
Elige una opción:
1. Continuar
2. Salir
1
Opción seleccionada: 1
Elige una opción:
1. Continuar
2. Salir
2
Opción seleccionada: 2
```

Declaración de la Variable:

let opcion; - Se declara una variable opcion que almacenará la elección del usuario.

Bucle do-while:

do { ... } while (opcion != "2"); - Este bucle se ejecutará al menos una vez y continuará ejecutándose mientras la condición especificada en el while sea verdadera. La condición aquí es que la opcion sea diferente de "2".

En C++

ISBN: 978-631-6557-57-5

main.cpp

```cpp
#include <iostream>
#include <string>

int main() {
    std::string opcion;
    do {
        std::cout << "Elige una opción: \n1. Continuar \n2. Salir \n";
        std::getline(std::cin, opcion);
        std::cout << "Opción seleccionada: " << opcion << std::endl;
    } while (opcion != "2");
    return 0;
}
```

C++ utiliza std::getline(std::cin, opcion) para leer la entrada del usuario, incluidas las líneas con espacios (Halterman, 2014).

Se verifica la condición con while (opcion != "2") para determinar si se debe continuar o salir del bucle.

En Java

```java
import java.util.Scanner;

public class MenuInteractivo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String opcion;
        do {
            System.out.println("Elige una opción: \n1. Continuar \n2. Salir");
            opcion = scanner.nextLine();
            System.out.println("Opción seleccionada: " + opcion);
        } while (!opcion.equals("2"));
        scanner.close();
    }
}
```

Se declara String opcion para almacenar la entrada del usuario

Dentro del bucle do, se muestra al usuario un mensaje pidiéndole que elija una opción.

scanner.nextLine() lee la línea completa de la entrada del usuario y la asigna a la variable opcion.

Se imprime la opción seleccionada.

El bucle continúa mientras la opción seleccionada no sea "2". La condición !opcion.equals("2") significa que el bucle seguirá ejecutándose mientras la opción no sea igual a "2".

En Python

ISBN: 978-631-6557-57-5

```
opcion = ''
while True:
    opcion = input("Elige una opción: \n1. Continuar \n2. Salir \n")
    print(f"Opción seleccionada: {opcion}")
    if opcion == '2':
        break
```

Se utiliza input() para capturar la entrada del usuario y print() para mostrar la opción seleccionada.

19) Realice un programa que imprima el rango de números desde el 4 hasta el 8.

Pseudocódigo

```
Proceso MostrarRango
    Para i ← 4 Hasta 8 Con Paso 1 Hacer
        Escribir i
    FinPara
FinProceso
```

Se utiliza un bucle Para para iterar sobre una secuencia de números.

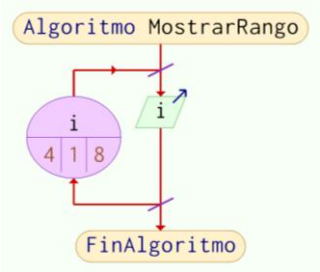
i <- 4: Inicializa la variable i con el valor 4.

Hasta 8: Indica que el bucle continuará hasta que i sea 8.

Con Paso 1: Incrementa i en 1 en cada iteración del bucle.

Dentro del bucle, Escribir i imprime el valor de la variable i, que es el número actual de la secuencia.

Diagrama de flujo



Implementación

En Python

```
for i in range(4, 9):
    print(i)
```

ISBN: 978-631-6557-57-5

```
4
5
6
7
8
```

La función range genera una secuencia de números, en este caso, range(4, 9) genera una secuencia de números comenzando desde 4 hasta 8 (el 9 no se incluye).

Se utiliza un bucle for para iterar sobre cada número en la secuencia generada por range(4, 9).

La variable i toma el valor de cada número en la secuencia en cada iteración del bucle.

Dentro del bucle, print(i) imprime el valor de la variable i, que es el número actual de la secuencia.

En JavaScript

```
for (let i = 4; i < 9; i++) {
    console.log(i);
}
```

Se declara el bucle for

let i = 4: Inicializa la variable i con el valor 4.

i < 9: Condición que mantiene el bucle ejecutándose mientras i sea menor que 9.

i++: Incrementa la variable i en 1 después de cada iteración del bucle.

Dentro del bucle, console.log(i) imprime el valor de la variable i, que es el número actual de la secuencia.

En Java

```
public class MostrarRango {
    public static void main(String[] args) {
        // Bucle for desde 4 hasta 8
        for (int i = 4; i < 9; i++) {
            System.out.println(i);
        }
    }
}
```

El bucle for:

int i = 4: Inicializa la variable i con el valor 4.

i < 9: Condición que mantiene el bucle ejecutándose mientras i sea menor que 9.

i++: Incrementa la variable i en 1 después de cada iteración del bucle.

ISBN: 978-631-6557-57-5

En C++

```cpp
#include <iostream>

int main() {
    // Bucle for desde 4 hasta 8
    for (int i = 4; i < 9; i++) {
        std::cout << i << std::endl;
    }
    return 0;
}
```

Bucle for:

int i = 4: Inicializa la variable i con el valor 4.

i < 9: Condición que mantiene el bucle ejecutándose mientras i sea menor que 9.

i++: Incrementa la variable i en 1 después de cada iteración del bucle.

Dentro del bucle, std::cout << i << std::endl; imprime el valor de la variable i, que es el número actual de la secuencia.

20) Hacer el cuadrado de los números del 1 al 10.

Pseudocódigo

```
Proceso cuadrados
    Definir i Como Entero
    Definir cuadrado Como Entero

    Dimension cuadrado[10]

    Para i ← 1 Hasta 10 Con Paso 1 Hacer
        cuadrado[i] ← i * i
    FinPara

    Para i ← 1 Hasta 10 Con Paso 1 Hacer
        Escribir cuadrado[i]
    FinPara
FinProceso
```

Se definen dos variables enteras: i y cuadrado. La variable i se utilizará como índice en los bucles, y cuadrado se utilizará como un arreglo para almacenar los resultados.

Se define cuadrado como un arreglo de 10 elementos. Esto es suficiente para almacenar los cuadrados de los números del 1 al 10.

Para i <- 1 Hasta 10 Con Paso 1 Hacer: Este bucle se ejecuta desde i = 1 hasta i = 10, incrementando i en 1 en cada iteración.

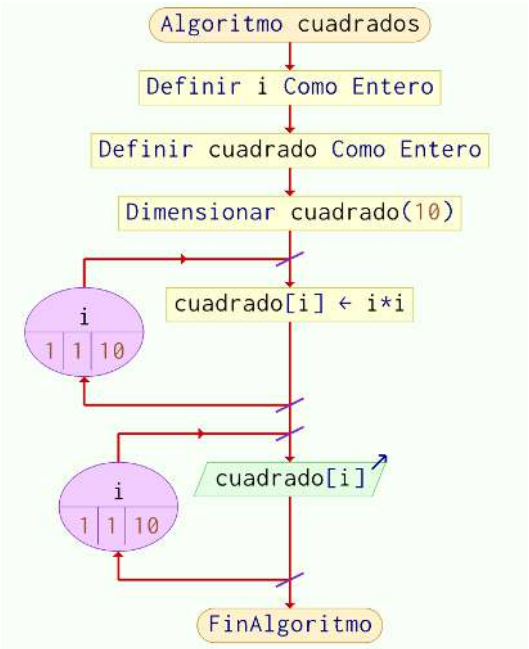
cuadrado[i] <- i * i: En cada iteración, se calcula el cuadrado de i y se almacena en la posición i del arreglo cuadrado.

ISBN: 978-631-6557-57-5

Para imprimir se rerquiere otro bucle Para i <- 1 Hasta 10 Con Paso 1 Hacer que se ejecuta desde i = 1 hasta i = 10.

Escribir cuadrado[i]: En cada iteración, se imprime el valor almacenado en la posición i del arreglo cuadrado.

Diagrama de flujo



En Python

```
cuadrados = []
for i in range(1, 11):
    cuadrados.append(i**2)
print(cuadrados)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Se crea una lista vacía llamada cuadrados que se utilizará para almacenar los resultados. El bucle for itera sobre un rango de números del 1 al 10. La función range(1, 11) genera una secuencia de números comenzando en 1 y terminando en 10 (el 11 no está incluido). Dentro del bucle, para cada valor de i en el rango especificado, se calcula su cuadrado utilizando el operador **, que significa "elevado a la potencia de". El resultado de i**2

ISBN: 978-631-6557-57-5

se agrega a la lista cuadrados usando el método append().

Finalmente, se imprime la lista cuadrados que ahora contiene los cuadrados de los números del 1 al 10.

En JavaScript

```
let cuadrados = [];

for (let i = 1; i <= 10; i++) {
    cuadrados.push(i * i);
}

console.log(cuadrados);
```

Se crea un arreglo vacío llamado cuadrados que se utilizará para almacenar los resultados.

Se establece un bucle for que itera sobre un rango de números del 1 al 10, inclusive.

Dentro del bucle, para cada valor de i en el rango especificado, se calcula su cuadrado utilizando el operador *, y el resultado se agrega al arreglo cuadrados usando el método push().

Finalmente, se imprime el arreglo cuadrados que ahora contiene los cuadrados de los números del 1 al 10.

En Java

```
import java.util.ArrayList;

public class Cuadrados {
    public static void main(String[] args) {
        ArrayList<Integer> cuadrados = new ArrayList<>();

        for (int i = 1; i <= 10; i++) {
            cuadrados.add(i * i);
        }

        System.out.println(cuadrados);
    }
}
```

Se importa la clase ArrayList de la biblioteca estándar de Java para utilizarla en la creación de una lista dinámica.

Se define una clase pública llamada Cuadrados y el método principal main que es el punto de entrada del programa.

Se crea un ArrayList de enteros llamado cuadrados para almacenar los resultados.

ISBN: 978-631-6557-57-5

En esta línea se establece un bucle for que itera desde i = 1 hasta i = 10, inclusive.

Dentro del bucle, para cada valor de i, se calcula su cuadrado utilizando el operador * y se agrega al ArrayList cuadrados usando el método add.

Finalmente, se imprime el ArrayList cuadrados que ahora contiene los cuadrados de los números del 1 al 10.

En C++

```cpp
#include <iostream>
#include <vector>

int main() {
    std::vector<int> cuadrados;

    for (int i = 1; i <= 10; i++) {
        cuadrados.push_back(i * i);
    }

    for (int i = 0; i < cuadrados.size(); i++) {
        std::cout << cuadrados[i] << " ";
    }

    return 0;
}
```

#include <iostream> es necesario para las operaciones de entrada y salida.

#include <vector> es necesario para usar la clase std::vector, que es un contenedor dinámico similar a las listas en Python.

Se crea un std::vector de enteros llamado cuadrados para almacenar los resultados.

El bucle itera desde i = 1 hasta i = 10, inclusive. En cada iteración, calcula el cuadrado de i y lo agrega al vector cuadrados utilizando el método push_back.

El segundo bucle itera sobre los elementos del vector cuadrados e imprime cada uno de ellos, seguido de un espacio.

ISBN: 978-631-6557-57-5

2.4.2 Ejercicios propuestos

1) Escriba un programa para convertir de una longitud de metros a pies.
2) Realice un programa para calcular el volumen de una esfera.
3) Escribir un programa que pida al usuario una cantidad total de segundos y que convierta esta cantidad a días, horas, minutos.
4) Implementar un programa que calcule la depreciación lineal de un activo a lo largo del tiempo.
5) Comprobar si un número ingresado es par o impar
6) Verificar si un alumno aprueba o reprueba una asignatura el valor mínimo debe ser igual a 7 para aprobar.
7) Clasificar películas si es igual o mayor a 18 años puede ver películas para adultos, si la edad es 12 años o mayor puede ver películas de adolescentes, de otras edades pueden ver películas para todo público.
8) Calcule el descuento de una compra a partir de 500 dólares que sea el 15%. Le debe indicar el valor a pagar con el descuento realizado, si no le hace descuento debe haber un mensaje indicando que no hay descuento en la compra.
9) Dado tres números determine cuál es el mayor.
10) Buscar el primer número divisible para 5 en un rango hasta 100.
11) Hacer un programa que sume los números que se van ingresando por teclado hasta que se digite el 0.
12) Hacer un programa que le permita al usuario escribir hasta que escriba la palabra salir.
13) Realizar un programa que decremente en 1 partiendo desde 20.
14) Usando while realizar un programa que calcule el factorial de un número.
15) Iterar hasta encontrar un elemento en la lista, e indicar el índice esto es la posición en la lista que ocupa el elemento, completar el código.

ISBN: 978-631-6557-57-5

```python
nombres = ["Vicky", "Benjamin", "Leonel", "Samanta"]
nombre_buscado = "Benjamin"
indice = 0
while indice < len(nombres) and nombres[indice] != nombre_buscado:
    indice += 1
if indice < len(nombres):
    print(f"{nombre_buscado}                        ")
else:
    print(f"             no encontrado.")
```

```
Benjamin encontrado en el índice 1
```

16) Completar el siguiente programa, el cual al ingresar un número le indica el día de la semana que corresponde.

```cpp
int dia;
std::cout << "Ingresa el número del día (1-7): ";
std::cin >> dia;

switch (dia) {
    case 1: std::cout << "Lunes\n"; break;
    case 2: std::cout << "Martes\n"; break;
    // ... casos para miércoles, jueves y viernes
    case 6: std::cout << "Sábado\n"; break;
    case 7: std::cout << "Domingo\n"; break;
    default: std::cout << "Número no válido.\n";
  }
```

17) Realizar un programa que permite indicar el número del mes y como resultado entregue la estación a la que pertenece.

18) Hacer un programa que le pida que introduzca el nivel de acceso, que puede ser: administrador, usuario, invitado. En el caso de administrador debe devolver acceso total, para usuario debe indicar acceso limitado, y para invitado debe devolver tiene acceso a solo lectura.

19) Realice un programa usando do-while. Que solicite al usuario cuantas veces desea que se repita un mensaje, el mensaje se deberá mostrarse las veces solicitadas por el usuario.

20) Hacer un programa que itere sobre la cadena “Hola”. Utilizar el for.

21) Use range en Python para iterar 4 veces.

ISBN: 978-631-6557-57-5

CAPÍTULO 3

3 VECTORES Y MATRICES

3.1 Conceptos generales

Las matrices y los vectores son conceptos fundamentales en el campo de las matemáticas, especialmente en las áreas de álgebra lineal y cálculos numéricos, teniendo también una importancia en diversas aplicaciones de la física, la ingeniería, la estadística, y la informática, entre otras disciplinas.

Un vector es una entidad matemática que posee magnitud y dirección. En el contexto más simple, especialmente en el plano bidimensional, se puede visualizar como una flecha que parte de un punto a otro. Sin embargo, en matemáticas y ciencias de la computación, un vector se generaliza como un arreglo de números (componentes del vector) que representan coordenadas en un espacio de varias dimensiones. Por ejemplo, un vector en un espacio tridimensional se representa con tres componentes. Los vectores son esenciales en la representación de cantidades físicas que tienen tanto magnitud como dirección, como la velocidad, la fuerza, y el desplazamiento.

Una matriz es una colección rectangular de números, símbolos, o expresiones, dispuestos en filas y columnas. Por ejemplo, una matriz de tamaño 3x2 es una matriz con tres filas y dos columnas. Las matrices son herramientas poderosas en álgebra lineal, utilizadas para resolver sistemas de ecuaciones lineales, realizar transformaciones lineales, entre otras aplicaciones. Además, las matrices juegan un papel decisivo en ciencias de la computación, especialmente en el procesamiento de imágenes, gráficos por computadora, redes neuronales y algoritmos de aprendizaje automático, donde se utilizan para representar y manipular datos de forma eficiente.

3.2 Arreglos unidimensionales (vectores)

En programación, un vector (también conocido como arreglo unidimensional) es una colección de elementos que son del mismo tipo. Se almacena en una secuencia contigua de memoria y cada elemento puede ser accedido directamente mediante un índice, que es un número entero que representa la posición del elemento dentro del vector.

ISBN: 978-631-6557-57-5

3.2.1 Declaración de Vectores

La declaración de un vector varía según el lenguaje de programación, pero generalmente implica especificar el tipo de datos de los elementos que almacenará el vector y su tamaño (es decir, la cantidad de elementos que puede contener). A continuación, se presentan ejemplos en algunos lenguajes de programación populares:

C++

En C y C++, los vectores se declaran especificando el tipo de datos, seguido del nombre del vector y el tamaño entre corchetes (Halterman, 2014). Por ejemplo, para declarar un vector de 10 enteros:

```
int vector[10];
```

Java

En Java, los vectores (arreglos) se declaran de manera similar, pero la especificación del tamaño se realiza al momento de su creación (instanciación) con el operador “new”.

```
int[] vector = new int[10];
```

Python

Python maneja las colecciones de datos con más flexibilidad, y los vectores pueden ser representados como listas. No es necesario declarar el tamaño ni el tipo de datos por adelantado (Hood & Kölling, 2023):

```
vector = [0] * 10  # Crea una lista de 10 elementos, todos inicializados en 0
```

O simplemente:

```
vector = []  # Crea una lista vacía
```

JavaScript

En JavaScript, los vectores se representan mediante arrays y, al igual que en Python, no es necesario especificar el tamaño o tipo de datos de antemano:

```
let vector = new Array(10); // Crea un array de 10 elementos
```

O de forma más simplificada:

ISBN: 978-631-6557-57-5

```
let vector = []; // Crea un array vacío
```

La necesidad de especificar el tipo de datos en la declaración de un vector (como en C++, y Java) asegura el tipado estático, lo que significa que el compilador conoce el tipo de datos de los elementos del vector y puede realizar comprobaciones de tipo en tiempo de compilación. Esto contrasta con lenguajes dinámicos como Python y JavaScript, donde el tipo de los elementos se puede cambiar en tiempo de ejecución y los vectores pueden almacenar elementos de diferentes tipos.

3.2.2 Acceso a los elementos

El acceso a los elementos de un vector es una de las operaciones esenciales en la programación, debido a que permite leer o modificar los valores almacenados en el vector. Esta operación se realiza mediante el uso de índices. Entender cómo funcionan los índices y cómo se utilizan permite trabajar eficazmente con vectores.

Un índice es un entero que especifica la posición de un elemento dentro de un vector. Los índices permiten acceder directamente a cualquier elemento del vector sin necesidad de recorrer los elementos anteriores. La mayoría de los lenguajes de programación modernos utilizan la indexación basada en cero, lo que significa que el primer elemento del vector se encuentra en la posición 0, el segundo en la posición 1, y así sucesivamente (Nair, 2009).

A continuación, se muestra cómo se accede a los elementos de un vector en varios lenguajes de programación:

C++

En C++, el acceso a un elemento se realiza utilizando el nombre del vector seguido por el índice entre corchetes (Easttom, 2003). Por ejemplo, para acceder al tercer elemento de un vector llamado **vector**, se usaría:

```
int valor = vector[2]; // Lee el tercer elemento
vector[2] = 10; // Modifica el tercer elemento
```

Java

El acceso a los elementos en Java es similar al de C++, utilizando también corchetes para especificar el índice:

```
int valor = vector[2]; // Lee el tercer elemento
```

ISBN: 978-631-6557-57-5

```
vector[2] = 10; // Modifica el tercer elemento
```

Python

Python utiliza una sintaxis similar para acceder a los elementos de una lista (que se usa para representar vectores), pero ofrece características adicionales como la indexación negativa, donde **-1** representa el último elemento, **-2** el penúltimo, y así sucesivamente (Barry, 2023):

```
valor = vector[2] # Lee el tercer elemento
vector[2] = 10 # Modifica el tercer elemento
ultimo = vector[-1] # Lee el último elemento
```

JavaScript

En JavaScript, el acceso a los elementos de un array se realiza de manera similar:

```
let valor = vector[2]; // Lee el tercer elemento
vector[2] = 10; // Modifica el tercer elemento
```

Se debe tener en cuenta lo siguiente cuando se trabaja con índices:

Rango de Índices: Debe asegurarse de que el índice utilizado esté dentro del rango válido del vector. Acceder a un vector con un índice fuera de rango puede resultar en un error de tiempo de ejecución, como un "ArrayIndexOutOfBoundsException" en Java o un "IndexError" en Python.

Modificación de Elementos: Los vectores permiten modificar los elementos existentes a través de su índice. Esta característica es utilizada para actualizar valores o realizar operaciones en los datos almacenados.

Lectura de Elementos: La lectura de elementos mediante índices es una operación común para procesar los datos almacenados en un vector, como calcular la suma de sus elementos o buscar un valor específico.

3.2.3 Recorrido de vectores

El recorrido de vectores permite iterar sobre cada elemento de un vector para realizar diversas operaciones, como la búsqueda, modificación o suma de sus elementos. Existen diferentes técnicas para recorrer un vector, dependiendo del lenguaje de programación y del tipo de tarea que se desea realizar. A continuación, se describen las

ISBN: 978-631-6557-57-5

técnicas más comunes utilizando bucles for, while, y foreach.

Bucle For

El bucle for es quizás la forma más común de recorrer un vector. La idea básica es iniciar un contador en 0 (o 1, dependiendo de si el lenguaje indexa los vectores comenzando por 0 o 1), incrementar este contador en cada iteración hasta que alcance el tamaño del vector, accediendo en cada paso al elemento correspondiente del vector.

Ejemplo en C:

```
Int vector[5] = {1, 2, 3, 4, 5};
for (int i = 0; i < 5; i++) {
    printf("%d\n", vector[i]);
}
```

Ejemplo en Python:

```
vector = [1, 2, 3, 4, 5]
for i in range(len(vector)):
print(vector[i])
```

Bucle While

Aunque menos común para recorrer vectores, el bucle while puede utilizarse para iterar sobre un vector, especialmente cuando no se conoce de antemano el número de elementos a procesar o cuando se quiere aplicar alguna condición particular para detener el recorrido.

Ejemplo en Java:

```
int[] vector = {1, 2, 3, 4, 5};
int i = 0;
while (i < vector.length) {
    System.out.println(vector[i]);
    i++;
}
```

Bucle Foreach

El bucle foreach, disponible en varios lenguajes de programación modernos,

ISBN: 978-631-6557-57-5

ofrece una forma más directa y menos propensa a errores de iterar sobre cada elemento de un vector sin necesidad de usar un índice explícito. Este tipo de bucle toma cada elemento del vector, uno por uno, y ejecuta un bloque de código para dicho elemento (Nakov, 2013).

Ejemplo en C#:

```
int[] vector = {1, 2, 3, 4, 5};
foreach (int elemento in vector) {
    Console.WriteLine(elemento);
}
```

Ejemplo en JavaScript:

```
let vector = [1, 2, 3, 4, 5];
vector.forEach(function(elemento) {
   console.log(elemento);
});
```

3.2.4 Operaciones básicas

Las operaciones básicas que se pueden realizar con vectores permiten la manipulación de datos y la implementación de algoritmos en programación (Chinnathambi, 2024). Estas operaciones incluyen la inserción, eliminación, modificación de elementos, así como la búsqueda y ordenación de los mismos. Cada una de estas operaciones tiene sus particularidades y aplicaciones específicas, dependiendo del contexto en el que se utilicen.

1) Inserción de Elementos

La inserción implica agregar un nuevo elemento al vector. Dependiendo del lenguaje de programación y del tipo de vector (estático o dinámico), esto puede realizarse de diferentes maneras. En vectores estáticos, el tamaño del vector está fijado, por lo que directamente no se pueden agregar más elementos una vez que se ha alcanzado el tamaño máximo. En vectores dinámicos o listas, se pueden agregar elementos sin preocuparse por el tamaño actual del vector.

Ejemplo en Python (Lista Dinámica):

```
vector = [1, 2, 3]
```

ISBN: 978-631-6557-57-5

```
vector.append(4)  # Añade el elemento 4 al final del vector
```

2) Eliminación de Elementos

La eliminación consiste en quitar un elemento del vector. Esto puede implicar simplemente eliminar el elemento o, en algunos casos, reorganizar los elementos restantes para llenar el espacio vacío que deja el elemento eliminado.

Ejemplo en Java (Usando ArrayList):

```
ArrayList<Integer> vector = new ArrayList<>(Arrays.asList(1, 2, 3, 4));
vector.remove(Integer.valueOf(3));  // Elimina el número 3 del vector
```

3) Modificación de Elementos

Modificar un elemento significa cambiar el valor de uno de los elementos del vector. Esta operación es directa y solo requiere especificar el índice del elemento a modificar y el nuevo valor.

Ejemplo en C++:

```
int vector[] = {1, 2, 3, 4};
vector[2] = 5;  // Cambia el tercer elemento (índice 2) a 5
```

4) Búsqueda de Elementos

La búsqueda de elementos en un vector puede realizarse mediante búsqueda lineal o binaria. La búsqueda lineal recorre el vector elemento por elemento hasta encontrar el valor buscado, mientras que la búsqueda binaria es más eficiente en vectores ordenados, dividiendo repetidamente a la mitad el rango de búsqueda hasta encontrar el elemento.

Búsqueda Lineal en Python:

```
def busqueda_lineal(vector, elemento):
    for i in range(len(vector)):
        if vector[i] == elemento:
            return i
    return -1
```

Búsqueda Binaria en Python:

```
def busqueda_binaria(vector, elemento):
    izquierda, derecha = 0, len(vector) - 1
    while izquierda <= derecha:
```

ISBN: 978-631-6557-57-5

```
            medio = (izquierda + derecha) // 2
            if vector[medio] == elemento:
                return medio
            elif vector[medio] < elemento:
                izquierda = medio + 1
            else:
                derecha = medio - 1
        return -1
```

5) Ordenación de Vectores

La ordenación es el proceso de reorganizar los elementos de un vector en un orden específico (por lo general, ascendente o descendente). Existen varios algoritmos de ordenación, siendo los más comunes el ordenamiento por burbuja, selección, inserción, y algoritmos más eficientes como quicksort y mergesort (Chinnathambi, 2024).

Ordenamiento por Burbuja en Python:

```
def ordenamiento_burbuja(vector):
    n = len(vector)
    for i in range(n):
        for j in range(0, n-i-1):
                if vector[j] > vector[j+1]:
                    vector[j], vector[j+1] = vector[j+1], vector[j]
```

3.2.5 Funciones y métodos para vectores

Los lenguajes de programación modernos suelen ofrecer una amplia gama de funciones y métodos integrados para trabajar con vectores (o arreglos, listas, según el contexto del lenguaje), lo que facilita enormemente la manipulación de estos. Estas funciones y métodos abarcan operaciones comunes como la ordenación, la inversión de elementos, y la búsqueda dentro de los vectores, entre otras. Vamos a detallar algunas de estas operaciones en distintos lenguajes de programación:

a. Ordenación

Muchos lenguajes de programación ofrecen métodos integrados para ordenar vectores o listas sin necesidad de implementar algoritmos de ordenación manualmente.

ISBN: 978-631-6557-57-5

Python: La lista en Python tiene el método ".sort()" para ordenarla in situ, o se puede usar la función "sorted()" para obtener una nueva lista ordenada.

```
miLista = [3, 1, 4, 1, 5, 9, 2]
miLista.sort()  # Ordena la lista in situ
nuevaLista = sorted(miLista)  # Crea una nueva lista ordenada
```

JavaScript: Los arrays en JavaScript pueden ser ordenados con el método ".sort()". Este método puede también aceptar una función comparadora para criterios de ordenación personalizados. (Easttom, 2001)

```
let miArray = [3, 1, 4, 1, 5, 9, 2];
miArray.sort((a, b) => a - b);  // Ordena el array de forma ascendente
```

b. Inversión

Invertir un vector implica cambiar el orden de sus elementos, de modo que el primer elemento se convierta en el último, el segundo en el penúltimo, y así sucesivamente.

Python:

El método **.reverse()** invierte los elementos de una lista in situ. También se puede utilizar la sintaxis de slicing para crear una copia invertida de la lista.

```
miLista = [1, 2, 3, 4, 5]
miLista.reverse()  # Invierte in situ
listaInvertida = miLista[::-1]  # Crea una copia invertida
```

JavaScript:

El método **.reverse()** invierte los elementos de un array in situ.

```
let miArray = [1, 2, 3, 4, 5];
    miArray.reverse();  // Invierte el array in situ
```

c. Búsqueda

La búsqueda de un elemento específico dentro de un vector es otra operación común. La implementación específica y el rendimiento pueden variar según el método y el lenguaje utilizado.

Python:

Para buscar un elemento y obtener su índice, se puede utilizar el método **.index()**. Si el

ISBN: 978-631-6557-57-5

elemento no existe, se lanzará una excepción.

```
miLista = [1, 2, 3, 4, 5]
        indice = miLista.index(3)  # Devuelve 2, que es el índice de '3' en la lista
```

JavaScript:

Para buscar un elemento, se pueden utilizar métodos como **.indexOf()** o **.findIndex()**. El método **.indexOf()** devuelve el índice del primer elemento que coincida o -1 si no se encuentra (Annable, 2019).

```
let miArray = [1, 2, 3, 4, 5];
let indice = miArray.indexOf(3);  // Devuelve 2
```

3.2.6 Aplicaciones de vectores

Los vectores, con su capacidad para almacenar y gestionar colecciones de datos de manera eficiente, encuentran aplicaciones prácticas en una amplia gama de áreas en la programación. Desde el manejo básico de listas de datos hasta la simulación de estructuras de datos más complejas y el almacenamiento temporal de datos para su procesamiento, los vectores son herramientas versátiles en el desarrollo de software. Veamos algunas de estas aplicaciones en detalle:

1) Manejo de Listas de Datos

En casi cualquier aplicación de software, manejar listas de datos es una necesidad común. Los vectores se utilizan para almacenar listas de usuarios, transacciones, productos, y más. Por ejemplo, en una aplicación de comercio electrónico, un vector puede contener objetos que representan productos en el carrito de compras de un usuario, permitiendo operaciones como agregar, eliminar y modificar productos.

2) Simulación de Estructuras de Datos Complejas

Aunque existen estructuras de datos específicas para representar pilas, colas, árboles, y más, los vectores pueden usarse para simular estas estructuras de una manera más simple y directa, especialmente en contextos educativos o en situaciones donde la implementación completa de la estructura de datos sería una exageración.

Pilas: Se pueden simular usando vectores, aprovechando operaciones de inserción y eliminación al final del vector (push y pop). Esto replica el comportamiento LIFO (Last

ISBN: 978-631-6557-57-5

In, First Out) de las pilas.

En python

```
pila = []
pila.append("A")  # push
pila.pop()        # pop
```

Colas: Similar a las pilas, pero utilizando operaciones al principio y al final del vector para simular el comportamiento FIFO (First In, First Out) de las colas. En algunos lenguajes, esto puede ser menos eficiente sin el uso de estructuras de datos específicas diseñadas para colas.

En python

```
cola = []
cola.append("A")  # enqueue
cola.pop(0)       # dequeue
```

3) Almacenamiento Temporal de Datos

Los vectores son ideales para el almacenamiento temporal de datos durante el procesamiento. Por ejemplo, al realizar análisis de datos o procesamiento de señales, es común leer un conjunto de datos en un vector, procesar estos datos (por ejemplo, filtrar, ordenar, transformar), y luego escribir el resultado o pasar a otro proceso. Esta capacidad para "almacenar" datos temporalmente es importante en la mayoría de los programas que realizan cálculos o manipulación de datos.

4) Gráficos y Simulaciones

En el desarrollo de juegos y gráficos por computadora, los vectores son esenciales para almacenar y manipular coordenadas de vértices, colores, y texturas. Además, en simulaciones físicas, los vectores representan fuerzas, velocidades, y otras propiedades físicas que necesitan ser calculadas y actualizadas constantemente.

5) Bases de Datos en Memoria

Para aplicaciones que requieren acceso rápido a los datos sin la latencia asociada con las bases de datos basadas en disco, los vectores pueden ser utilizados para construir estructuras de datos en memoria que permiten inserciones rápidas, búsquedas, y eliminaciones.

ISBN: 978-631-6557-57-5

3.2.7 Vectores y memoria

La manera en que los vectores ocupan espacio en la memoria y su impacto en el rendimiento de un programa es un tema a considerar en la programación y el diseño de algoritmos. Entender la diferencia entre vectores estáticos y dinámicos, así como sus implicaciones en términos de uso de memoria y eficiencia, es importante para tomar decisiones informadas durante el desarrollo de software.

a) Vectores Estáticos

Los vectores estáticos tienen un tamaño fijo que se define en tiempo de compilación. Esto significa que la cantidad de memoria que ocuparán es conocida y asignada al momento de ejecutar el programa. Dado que su tamaño no cambia durante la ejecución, los vectores estáticos ofrecen ventajas en términos de rendimiento, esto se debe a que el acceso a sus elementos puede ser muy rápido por la predictibilidad de su ubicación en memoria.

Sin embargo, la principal desventaja de los vectores estáticos es su inflexibilidad. Si no se utiliza todo el espacio reservado, se desperdicia memoria; y si se necesita más espacio del asignado inicialmente, no es posible expandir el vector, lo que podría requerir la creación de un nuevo vector más grande y la copia de los datos, una operación costosa en términos de rendimiento.

b) Vectores Dinámicos

A diferencia de los estáticos, los vectores dinámicos pueden cambiar de tamaño durante la ejecución del programa. Esto se logra mediante la asignación de memoria en el heap (montículo), que es una región de memoria utilizada para la asignación dinámica. Los vectores dinámicos inician con un tamaño inicial y, cuando se necesita más espacio, pueden expandirse, típicamente duplicando su capacidad para minimizar el número de reasignaciones.

La flexibilidad de los vectores dinámicos los hace muy útiles para situaciones en las que el número de elementos a almacenar no es conocido de antemano o puede cambiar frecuentemente. Sin embargo, esta flexibilidad viene con un costo: las operaciones de expansión y reasignación de memoria pueden ser costosas,

ISBN: 978-631-6557-57-5

especialmente si ocurren con frecuencia. Además, el acceso a los elementos puede ser ligeramente más lento en comparación con los vectores estáticos, debido a la indirección adicional necesaria para acceder a la memoria dinámica.

c) Impacto en el Rendimiento

El uso de vectores estáticos o dinámicos tiene diferentes impactos en el rendimiento del programa:

Uso de memoria: Los vectores estáticos pueden llevar al desperdicio de memoria si no se utilizan completamente, mientras que los vectores dinámicos pueden ser más eficientes en este aspecto, ya que su tamaño se ajusta según las necesidades. Sin embargo, la sobrecarga de administración de memoria dinámica también puede aumentar el uso de memoria.

Velocidad de acceso: Los vectores estáticos ofrecen un acceso muy rápido a sus elementos, ya que la ubicación de cada elemento en memoria es fija y conocida. En contraste, aunque los vectores dinámicos generalmente también permiten un acceso rápido, puede haber un ligero retraso adicional debido a la indirección necesaria para acceder a elementos en la memoria dinámica.

Tiempo de ejecución: Las operaciones de expansión de vectores dinámicos, que pueden incluir la reubicación de memoria y la copia de elementos existentes a una nueva ubicación de memoria, pueden ser costosas. Estas operaciones afectan el rendimiento, especialmente en aplicaciones donde las inserciones son frecuentes y el tamaño del vector varía significativamente.

d) Consideraciones de Diseño

La elección entre usar vectores estáticos o dinámicos dependerá de varios factores, incluyendo:

Requerimientos de memoria y rendimiento: Si la eficiencia de memoria y el tiempo de acceso son críticos, y el tamaño del conjunto de datos es conocido y constante, los vectores estáticos pueden ser la mejor opción. Por otro lado, si el tamaño del conjunto de datos puede cambiar o no se conoce de antemano, los vectores

ISBN: 978-631-6557-57-5

dinámicos ofrecen la flexibilidad necesaria a costa de un posible impacto en el rendimiento.

Frecuencia de operaciones de modificación: En contextos donde las inserciones y eliminaciones son operaciones frecuentes y el tamaño del conjunto de datos es volátil, los vectores dinámicos son preferibles debido a su capacidad para ajustarse al tamaño de datos actual.

Entorno de ejecución y restricciones del lenguaje de programación: Algunos entornos o lenguajes de programación pueden imponer restricciones sobre el uso de memoria dinámica o estática, o pueden ofrecer optimizaciones específicas que favorecen uno u otro tipo de vector.

3.3 Arreglos bidimensionales (matrices)

Los arreglos bidimensionales, comúnmente conocidos como matrices, son una estructura de datos que permite almacenar y organizar datos en forma de tabla, compuesta por filas y columnas. Esta estructura es una extensión natural de los arreglos unidimensionales (vectores), permitiendo representar datos más complejos y relaciones entre ellos de una manera estructurada y accesible.

Una matriz se puede visualizar como un arreglo de arreglos, donde cada elemento del arreglo principal representa una fila de la matriz y, a su vez, cada uno de estos elementos es un arreglo que representa las columnas de dicha fila. Matemáticamente, una matriz de dimensiones m x n tiene m filas y n columnas, y se puede acceder a cada elemento individual de la matriz especificando dos índices: el primero para la fila y el segundo para la columna.

a) Declaración de Matrices

La declaración de una matriz varía según el lenguaje de programación utilizado. A continuación, se presentan ejemplos:

C++: Se especifica el tipo de datos, seguido del nombre de la matriz y las dimensiones entre corchetes.

```
int miMatriz[3][4];  // Declara una matriz de enteros de 3 filas y 4 columnas
```

ISBN: 978-631-6557-57-5

Java: Similar a C y C++, pero la asignación de espacio puede hacerse en el momento de la declaración o posteriormente con el operador "new".

```
int[][] miMatriz = new int[3][4];  // Declara e inicializa una matriz de 3x4
```

Python: Las matrices se pueden representar usando listas anidadas, sin necesidad de declarar su tamaño de antemano.

```
miMatriz = [[0 for x in range(4)] for y in range(3)]  # Crea una matriz 3x4 con todos los elementos inicializados en 0
```

b) Acceso y Modificación de Elementos

El acceso y la modificación de los elementos de una matriz se realizan especificando los índices de fila y columna. Por ejemplo, para acceder al elemento en la segunda fila y tercera columna de **miMatriz**, se utilizaría miMatriz[1][2] en C, C++, y Java, o de manera similar en otros lenguajes que soporten esta estructura de datos.

c) Recorrido de Matrices

Aprender a recorrer una matriz es fundamental para realizar operaciones con sus elementos, como cálculos sumatorios, búsquedas de valores, y transformaciones de datos. Usualmente, se recorre una matriz mediante bucles anidados, donde un bucle recorre las filas y otro las columnas.

En Python

```
for i in range(num_filas):
for j in range(num_columnas):
    # Acceso al elemento de la matriz en la posición [i][j]
```

d) Operaciones con Matrices

Además de la inserción, eliminación y modificación de elementos individuales, en fundamentos de programación se estudian operaciones más complejas como:

ISBN: 978-631-6557-57-5

Suma y resta de matrices: Solo es posible entre matrices del mismo tamaño, realizando la operación elemento a elemento.

Multiplicación de matrices: Se realiza según las reglas del álgebra lineal, donde el elemento en la posición **i**, **j** de la matriz resultante es el producto punto de la fila **i** de la primera matriz y la columna **j** de la segunda matriz.

Transposición: Consiste en cambiar filas por columnas en una matriz, resultando en una nueva matriz donde el elemento **i**, **j** original pasa a ser el elemento **j**, **i**.

e) Matrices Especiales

Existen varios tipos de matrices con propiedades particulares que se deben conocer, como:

Matrices cuadradas: Son aquellas cuyo número de filas es igual al número de columnas. Este tipo de matrices tiene propiedades y operaciones especiales, como el cálculo del determinante y la matriz inversa.

Matrices diagonales, identidad y cero: Son tipos específicos de matrices cuadradas con características únicas en cuanto a la distribución de sus elementos.

Matrices dispersas: Son matrices donde la mayoría de sus elementos son ceros. Existen técnicas y estructuras de datos especiales para almacenar y manipular eficientemente matrices dispersas.

f) Representación de Matrices en Diferentes Lenguajes de Programación

Cada lenguaje de programación tiene sus propias particularidades para trabajar con matrices. Por ejemplo, en lenguajes de alto nivel como Python, las matrices pueden ser fácilmente representadas y manipuladas usando listas anidadas o bibliotecas especializadas como NumPy, que ofrece una amplia gama de operaciones y funciones matemáticas para trabajar con matrices de manera eficiente.

g) Algoritmos sobre Matrices

En los fundamentos de programación, también es importante aprender algoritmos

básicos que operan sobre matrices, incluyendo:

Algoritmos de búsqueda: Para encontrar un elemento específico o elementos que cumplen cierta condición dentro de una matriz.

Algoritmos de ordenación: Aplicables a matrices o a sus filas/columnas individuales.

Transformaciones de matrices: Como rotaciones, volteos y más, especialmente útiles en gráficos por computadora y procesamiento de imágenes

h) Aplicaciones de Matrices

Las matrices tienen numerosas aplicaciones en computación y matemáticas, incluyendo:

Representación de Datos Complejos: Las matrices se utilizan para almacenar datos que naturalmente se organizan en forma tabular, como hojas de cálculo, tablas de bases de datos, y gráficos de píxeles en imágenes digitales.

Operaciones Matemáticas: En álgebra lineal, las matrices son fundamentales para representar sistemas de ecuaciones lineales, transformaciones lineales, y otras operaciones matemáticas complejas.

Gráficos por Computadora: Se utilizan matrices para realizar operaciones de transformación en gráficos 3D, como rotaciones, escalado, y traslaciones de objetos.

Procesamiento de Señales e Imágenes: Las matrices son clave en algoritmos para el procesamiento de señales e imágenes, permitiendo la implementación de filtros, transformadas, y otras técnicas analíticas.

3.4 Ejercicios de aplicación

3.4.1 Ejercicios resueltos

1) Sumar los elementos de la lista [6, 22, 34, 8, 23]

Pseudocódigo

ISBN: 978-631-6557-57-5

```
Algoritmo SumaLista
    // Declarar el tamaño del arreglo
    Dimension lista[5]

    // Inicializar el arreglo
    lista[1] ← 6
    lista[2] ← 22
    lista[3] ← 34
    lista[4] ← 8
    lista[5] ← 23

    // Inicializar la variable de suma
    suma ← 0

    // Iterar sobre cada elemento del arreglo
    Para i ← 1 Hasta 5 Hacer
        suma ← suma + lista[i]
    Fin Para

    // Imprimir el resultado
    Escribir suma
FinAlgoritmo
```

Se declara un arreglo llamado lista con 5 elementos.

Se asignan valores específicos a cada posición del arreglo lista. Es importante notar que en PSeint los índices de los arreglos comienzan en 1, no en 0 como en muchos otros lenguajes de programación.

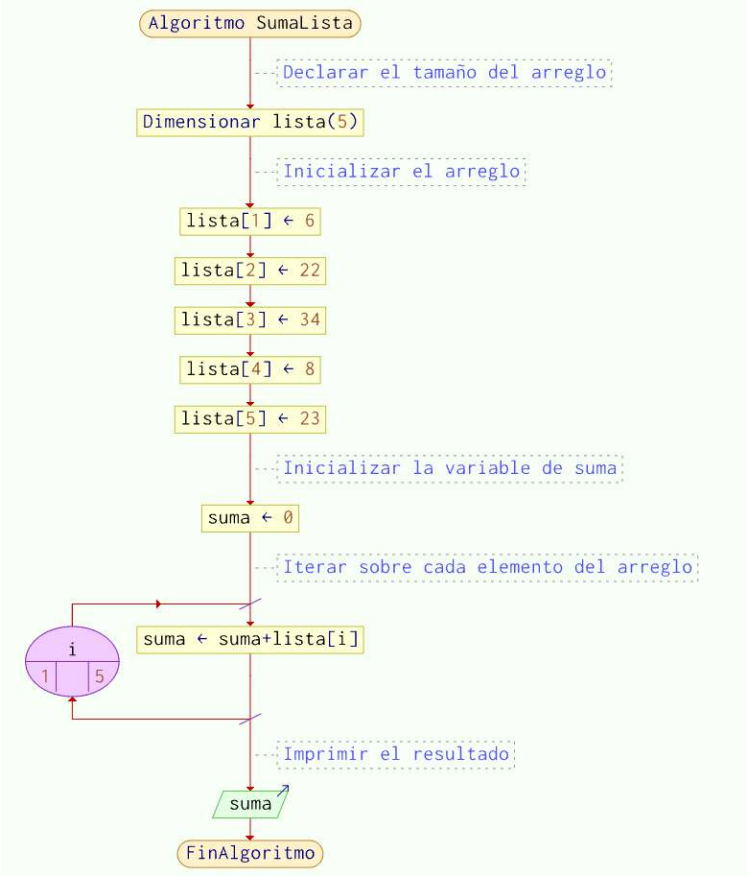
Se inicializa la variable suma en 0. Esta variable se usará para acumular la suma de los elementos del arreglo.

El bloque de código Para define un bucle que iterará desde 1 hasta 5 (inclusive), es decir, sobre todos los índices del arreglo lista. En cada iteración, el valor del elemento lista[i] se suma a la variable **suma**.

Por último se imprime el valor acumulado en la variable **suma**.

ISBN: 978-631-6557-57-5

Diagrama de flujo



Implementación

En Python

```
lista = [6, 22, 34, 8, 23]
suma = 0
for num in lista:
    suma += num
print(suma)
```

```
93
```

Se crea una lista llamada lista que contiene los elementos [6, 22, 34, 8, 23].

Se inicializa una variable llamada suma con el valor 0. Esta variable se utilizará para almacenar la suma de los elementos de la lista.

En JavaScript

for num in lista: Este es el inicio de un bucle for que itera sobre cada elemento de la lista lista. En cada iteración, la variable num toma el valor del siguiente elemento en la lista.

ISBN: 978-631-6557-57-5

suma += num: Dentro del bucle, se actualiza el valor de suma añadiendo el valor actual de num. El operador += es una forma abreviada de escribir suma = suma + num.

Después de que el bucle haya terminado (es decir, después de haber sumado todos los elementos de la lista), se imprime el valor final de suma.

En JavaScript

```
let lista = [6, 22, 34, 8, 23];
let suma = 0;

for (let num of lista) {
    suma += num;
}

console.log(suma);
```

Se utiliza let para declarar la variable lista que contiene una lista de números: [6, 22, 34, 8, 23].

Se declara la variable suma y se inicializa con 0. Esta variable se utilizará para acumular la suma de los elementos de la lista.

for (let num of lista) es una estructura de bucle que itera sobre cada elemento de la lista denominada lista.

En cada iteración, la variable num toma el valor del siguiente elemento de la lista.

En C++

```
#include <iostream>
using namespace std;

int main() {
    // Declarar e inicializar la lista
    int lista[] = {6, 22, 34, 8, 23};
    int suma = 0;

    // Iterar sobre cada elemento de la lista
    for (int num : lista) {
        suma += num;
    }

    // Imprimir el resultado
    cout << suma << endl;

    return 0;
}
```

La biblioteca iostream permite realizar operaciones de entrada y salida, como imprimir en la consola con cout.

using namespace std se utiliza en el espacio de nombres std, lo cual permite usar directamente elementos como cout sin necesidad de escribir std::cout.

La función main es el punto de entrada de un programa en C++. La ejecución del

ISBN: 978-631-6557-57-5

programa comienza aquí.

int lista[] = {6, 22, 34, 8, 23}; se utiliza para declarar un arreglo de enteros llamado lista e inicializa sus elementos con los valores 6, 22, 34, 8, y 23.

int suma = 0; se declara una variable entera llamada suma y la inicializa con 0. Esta variable se usará para almacenar la suma de los elementos de la lista.

for (int num : lista) es un bucle for basado en rango que itera sobre cada elemento del arreglo lista.

En cada iteración, la variable num toma el valor del siguiente elemento en lista.

suma += num; es una forma abreviada de escribir suma = suma + num, y añade el valor actual de num a suma.

cout << suma << endl; imprime el valor de suma en la consola seguido de un salto de línea. cout es el flujo de salida estándar en C++.

return 0; indica que el programa ha terminado correctamente y devuelve 0 al sistema operativo.

En Java

```
1  public class SumaLista {
2      public static void main(String[] args) {
3          // Declarar e inicializar la lista
4          int[] lista = {6, 22, 34, 8, 23};
5          int suma = 0;
6
7          // Iterar sobre cada elemento de la lista
8          for (int num : lista) {
9              suma += num;
10         }
11
12         // Imprimir el resultado
13         System.out.println(suma);
14     }
15 }
16
```

public class SumaLista {: Define una clase pública llamada SumaLista. En Java, todos los programas deben estar contenidos en una clase (Sharan & Davis, 2021).

public static void main(String[] args) {: Define el método main, que es el punto de entrada del programa. La ejecución del programa comienza aquí.

public: Indica que el método es accesible desde cualquier parte.

static: Significa que el método puede ser llamado sin crear una instancia de la clase.

void: Indica que el método no devuelve ningún valor.

String[] args: Permite pasar argumentos al programa desde la línea de comandos.

ISBN: 978-631-6557-57-5

int[] lista = {6, 22, 34, 8, 23};: Declara un arreglo de enteros llamado lista e inicializa sus elementos con los valores 6, 22, 34, 8, y 23.

int suma = 0;: Declara una variable entera llamada suma y la inicializa con 0. Esta variable se utilizará para almacenar la suma de los elementos de la lista.

for (int num : lista) {: Utiliza un bucle for-each para iterar sobre cada elemento del arreglo lista. En cada iteración, la variable num toma el valor del siguiente elemento en la lista.

suma += num;: Dentro del bucle, se actualiza el valor de suma añadiendo el valor actual de num.

System.out.println(suma);: Imprime el valor de suma en la consola seguido de un salto de línea. System.out.println es el método estándar para imprimir en la consola en Java.

2) Del vector [1,4,7,12,15,20] mostrar el elemento que se encuentra en la tercera posición.

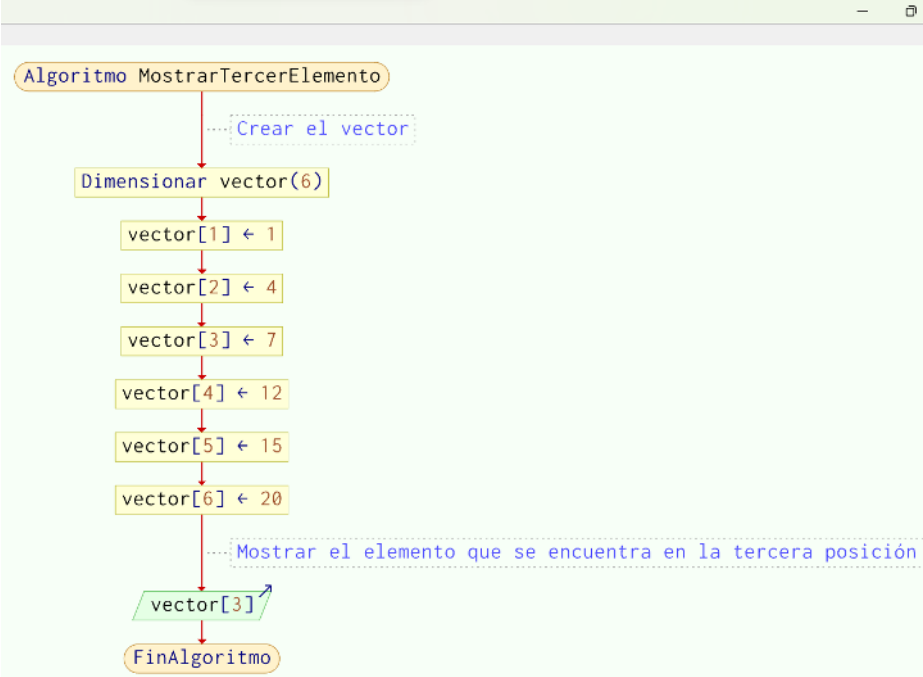
Pseudocódigo

```
Algoritmo MostrarTercerElemento
    //Crear el vector
    Dimension vector[6]
    vector[1] = 1
    vector[2] = 4
    vector[3] = 7
    vector[4] = 12
    vector[5] = 15
    vector[6] = 20

    //Mostrar el elemento que se encuentra en la tercera posición (índice 3 en PSeInt)
    Escribir vector[3]
FinAlgoritmo
```

En PSeInt, los índices de los arreglos comienzan en 1, por lo que el tercer elemento está en el índice 3. Este pseudocódigo define correctamente un arreglo (vector), lo inicializa con los valores proporcionados, y luego imprime el tercer elemento del arreglo, en este caso el 7.

ISBN: 978-631-6557-57-5

Diagrama de flujo



Implementación

En Python

```
# Crear el vector
vector = [1, 4, 7, 12, 15, 20]

# Mostrar el elemento que se encuentra en la tercera posición (índice 2)
print(vector[2])
```

```
7
```

En Python, los índices de las listas comienzan en 0, por lo que el elemento en la tercera posición está en el índice 2. Este código imprimirá el valor 7 en la consola.

En JavaScript

```
1  // Crear el vector
2  let vector = [1, 4, 7, 12, 15, 20];
3
4  // Mostrar el elemento que se encuentra en la tercera posición
       (índice 2)
5  console.log(vector[2]);
```

Este código define un vector con los valores [1, 4, 7, 12, 15, 20] y luego imprime el tercer elemento del vector en la consola, que es 7. En JavaScript, al igual que en Python, los índices de los arrays comienzan en 0, por lo que el tercer elemento se encuentra en el índice 2.

ISBN: 978-631-6557-57-5

En Java

```java
public class Main {
    public static void main(String[] args) {
        // Crear el vector
        int[] vector = {1, 4, 7, 12, 15, 20};

        // Mostrar el elemento que se encuentra en la tercera posición (índice 2)
        System.out.println(vector[2]);
    }
}
```

int[] se utiliza para declarar una variable de tipo arreglo de enteros. El tipo de datos int indica que el arreglo contendrá elementos que son números enteros. En Java, al igual que en Python y JavaScript, los índices de los arrays comienzan en 0, por lo que el tercer elemento se encuentra en el índice 2.

En C++

```cpp
#include <iostream>
using namespace std;

int main() {
    // Crear el vector
    int vector[] = {1, 4, 7, 12, 15, 20};

    // Mostrar el elemento que se encuentra en la tercera
        posición (índice 2)
    cout << vector[2] << endl;

    return 0;
}
```

La línea using namespace std; en C++ se utiliza para evitar la necesidad de escribir std:: antes de cada objeto o función que pertenece al espacio de nombres estándar (standard namespace).

El espacio de nombres estándar (std) incluye muchas funciones, objetos y tipos que se utilizan comúnmente en programas C++. Por ejemplo, std::cout, std::cin, std::string, etc. Si no se usa using namespace std;, tendrías que preceder cada uno de estos con std::.

int: Se declara un arreglo de enteros.

vector[]: El nombre del arreglo es vector y los corchetes indican que es un arreglo. El tamaño se deduce de la lista de inicialización.

{1, 4, 7, 12, 15, 20}: Se inicializa el arreglo con estos seis valores. Como hemos proporcionado seis valores, el tamaño del arreglo se deduce automáticamente a 6.

3) Use en JavaScript la siguiente intrucción let miVector = new Array(10) para llenar un vector del 1 al 10.

Pseudocódigo

ISBN: 978-631-6557-57-5

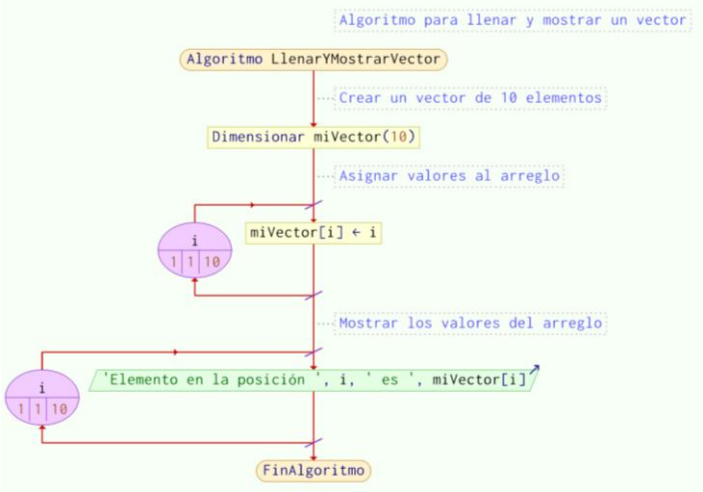
```
// Algoritmo para llenar y mostrar un vector
Algoritmo LlenarYMostrarVector

    // Crear un vector de 10 elementos
    Dimension miVector[10]

    // Asignar valores al arreglo
    Para i = 1 Hasta 10 Con Paso 1 Hacer
        miVector[i] = i
    FinPara

    // Mostrar los valores del arreglo
    Para i = 1 Hasta 10 Con Paso 1 Hacer
        Escribir "Elemento en la posición ", i, " es ", miVector[i]
    FinPara

FinAlgoritmo
```

```
*** Ejecución Iniciada. ***
Elemento en la posición 1 es 1
Elemento en la posición 2 es 2
Elemento en la posición 3 es 3
Elemento en la posición 4 es 4
Elemento en la posición 5 es 5
Elemento en la posición 6 es 6
Elemento en la posición 7 es 7
Elemento en la posición 8 es 8
Elemento en la posición 9 es 9
Elemento en la posición 10 es 10
*** Ejecución Finalizada. ***
```

- Definición del vector: La instrucción Dimension miVector[10] se utiliza para definir un vector de 10 elementos.
- Bucles Para: Los bucles ahora comienzan en 1 y terminan en 10, para ajustarse al rango de índices permitido en PSeInt.
- Asignación de valores: El índice i empieza en 1, por lo que miVector[i] = i llenará el vector con los valores del 1 al 10.
- Mostrar los valores: Se recorre el vector del índice 1 al 10 para mostrar cada elemento.

ISBN: 978-631-6557-57-5

Diagrama de flujo



Implementación

```
1  let miVector = new Array(10);
2
3  // Asignar valores al arreglo
4  for (let i = 0; i < miVector.length; i++) {
5      miVector[i] = i + 1; // Ejemplo: llenando el vector con
           valores del 1 al 10
6  }
7
8  // Mostrar los valores del arreglo
9  for (let i = 0; i < miVector.length; i++) {
10     console.log(`Elemento en la posición ${i} es ${miVector[i]}`
           );
11 }
```

```
Elemento en la posición 0 es 1
Elemento en la posición 1 es 2
Elemento en la posición 2 es 3
Elemento en la posición 3 es 4
Elemento en la posición 4 es 5
Elemento en la posición 5 es 6
Elemento en la posición 6 es 7
Elemento en la posición 7 es 8
Elemento en la posición 8 es 9
Elemento en la posición 9 es 10
```

let miVector = new Array(10);: Crea un nuevo arreglo miVector con 10 elementos, todos inicializados con undefined. undefined en JavaScript es un indicador de que una variable o un elemento de un arreglo ha sido declarado, pero no tiene un valor asignado. Cuando se crea un arreglo con new Array(length), todos sus elementos están en este estado undefined hasta que se les asigne un valor explícito.

for (let i = 0; i < miVector.length; i++) { miVector[i] = i + 1; }: Llena el arreglo miVector con valores del 1 al 10.

ISBN: 978-631-6557-57-5

for (let i = 0; i < miVector.length; i++) { console.log(Elemento en la posición ${i} es ${miVector[i]}); }: Recorre el arreglo y muestra cada elemento en la consola.

4) Hacer un programa que recorra cada elemento de una lista y los vaya imprimiendo.

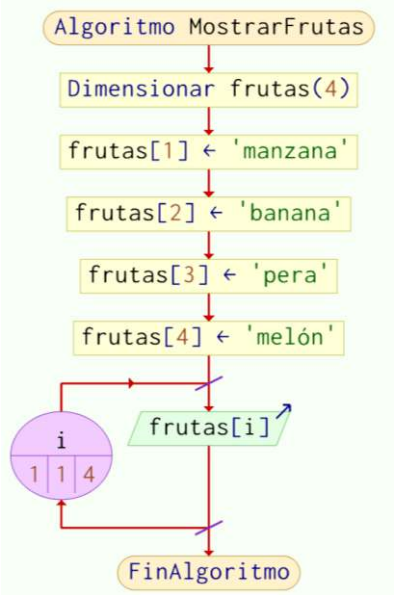
Pseudocódigo

```
Proceso MostrarFrutas
    Dimension frutas[4]

    frutas[1] ← "manzana"
    frutas[2] ← "banana"
    frutas[3] ← "pera"
    frutas[4] ← "melón"

    Para i ← 1 Hasta 4 Con Paso 1 Hacer
        Escribir frutas[i]
    FinPara
FinProceso
```

- Se define un proceso llamado MostrarFrutas.
- Dimension frutas[4] declara un arreglo llamado frutas con 4 elementos. Se asignan valores a cada elemento del arreglo frutas individualmente.
- Se utiliza un bucle Para para iterar sobre cada índice del arreglo frutas. i es la variable de control que toma valores desde 1 hasta 4.
- Dentro del bucle, Escribir frutas[i] imprime el valor del elemento actual del arreglo frutas en la posición i.
- FinPara marca el final del bucle Para.
- FinProceso marca el final del proceso MostrarFrutas.

ISBN: 978-631-6557-57-5

Diagrama de flujo



Implementación

En Python

```
frutas = ["manzana", "banana", "pera", "melón"]
for fruta in frutas:
    print(fruta)
```

```
manzana
banana
pera
melón
```

Se utiliza un bucle for para iterar sobre cada elemento de la lista frutas. En cada iteración del bucle, la variable fruta toma el valor de uno de los elementos de la lista frutas.

En JavaScript

main.js

```
const frutas = ["manzana", "banana", "pera", "melón"];
for (const fruta of frutas) {
    console.log(fruta);
}
```

```
manzana
banana
pera
melón
```

ISBN: 978-631-6557-57-5

La palabra clave const se utiliza para declarar variables que son de solo lectura, lo que significa que una vez que se asigna un valor a una variable declarada con const, ese valor no puede ser reasignado.

Se utiliza un bucle for-of para iterar sobre cada elemento del array frutas. En cada iteración del bucle, la variable fruta toma el valor de uno de los elementos del array frutas.

En Java

```
public class Frutas {
    public static void main(String[] args) {
        String[] frutas = {"manzana", "banana", "pera",
"melón"};

        for (String fruta : frutas) {
            System.out.println(fruta);
        }
    }
}
```

```
manzana
banana
pera
melón
```

- Se puede usar Java en línea como en este ejemplo que se presenta.
- Se define un array String llamado frutas que contiene cuatro elementos: "manzana", "banana", "pera" y "melón".
- El bucle for-each (aunque en Java simplemente se utiliza la sintaxis for con dos puntos :) se introduce para simplificar la iteración sobre arrays y colecciones cuando no se necesita el índice explícito. Es más legible y reduce el riesgo de errores de índice fuera de los límites. En el ejemplo se utiliza un bucle for-each para iterar sobre cada elemento del array frutas. En cada iteración del bucle, la variable fruta toma el valor de uno de los elementos del array frutas.

ISBN: 978-631-6557-57-5

```java
public class Frutas {
    public static void main(String[] args) {
        String[] frutas = {"manzana", "banana", "pera",
"melón"};

        for (int i = 0; i < frutas.length; i++) {
            System.out.println(frutas[i]);
        }
    }
}
```

- Aquí se utiliza un bucle for tradicional para iterar sobre el array frutas.
- int i = 0: Inicializa la variable de control i a 0.
- i < frutas.length: Condición que mantiene el bucle ejecutándose mientras i sea menor que la longitud del array frutas.
- i++: Incrementa la variable i en 1 después de cada iteración.
- Dentro del bucle, System.out.println(frutas[i]) imprime el elemento actual del array frutas en la posición i, frutas[i] accede al elemento del array en la posición i.

En C++

```cpp
#include <iostream>
#include <string>

int main() {
    std::string frutas[] = {"manzana", "banana", "pera", "melón"}
        ;

    for (const std::string& fruta : frutas) {
        std::cout << fruta << std::endl;
    }

    return 0;
}
```

- #include <iostream> es necesario para utilizar std::cout para imprimir en la consola.
- #include <string> es necesario para trabajar con el tipo std::string.
- Se declara un array de cadenas de texto std::string llamado frutas y se inicializa con cuatro elementos.
- Se utiliza un bucle "for-each" para iterar sobre cada elemento del array frutas.
- const std::string& fruta declara una referencia constante a cada elemento del array durante la iteración, lo que evita la copia de cadenas y permite la lectura.
- Dentro del bucle, std::cout << fruta << std::endl; imprime el valor de la variable fruta, que es el elemento actual del array, seguido de un salto de línea.

ISBN: 978-631-6557-57-5

```cpp
#include <iostream>
#include <string>

int main() {
    std::string frutas[] = {"manzana", "banana", "pera", "melón"};
    int numFrutas = sizeof(frutas) / sizeof(frutas[0]);

    for (int i = 0; i < numFrutas; i++) {
        std::cout << frutas[i] << std::endl;
    }

    return 0;
}
```

El código muestra un for tradicional y para poder utilizarlo se debe calcular el número de elementos en el array frutas. sizeof(frutas) da el tamaño total del array en bytes, y sizeof(frutas[0]) da el tamaño del primer elemento. Al dividir estos dos valores, obtenemos el número de elementos en el array.

En el bucle for tradicional para iterar sobre cada elemento del array frutas, se realiza lo siguiente:

int i = 0: Inicializa la variable de control i a 0.

i < numFrutas: Condición que mantiene el bucle ejecutándose mientras i sea menor que el número de elementos en el array frutas.

i++: Incrementa la variable i en 1 después de cada iteración.

Dentro del bucle, std::cout << frutas[i] << std::endl; imprime el valor del elemento actual del array frutas en la posición i.

5) Dado el vector [2, 4, 6, 8, 10, 12, 14, 16] encontrar el elemento que está en la posición 5, modificar el elemento que esta en la posición 3 y ubicar el número 18, determinar cual es el último elemento del vector.

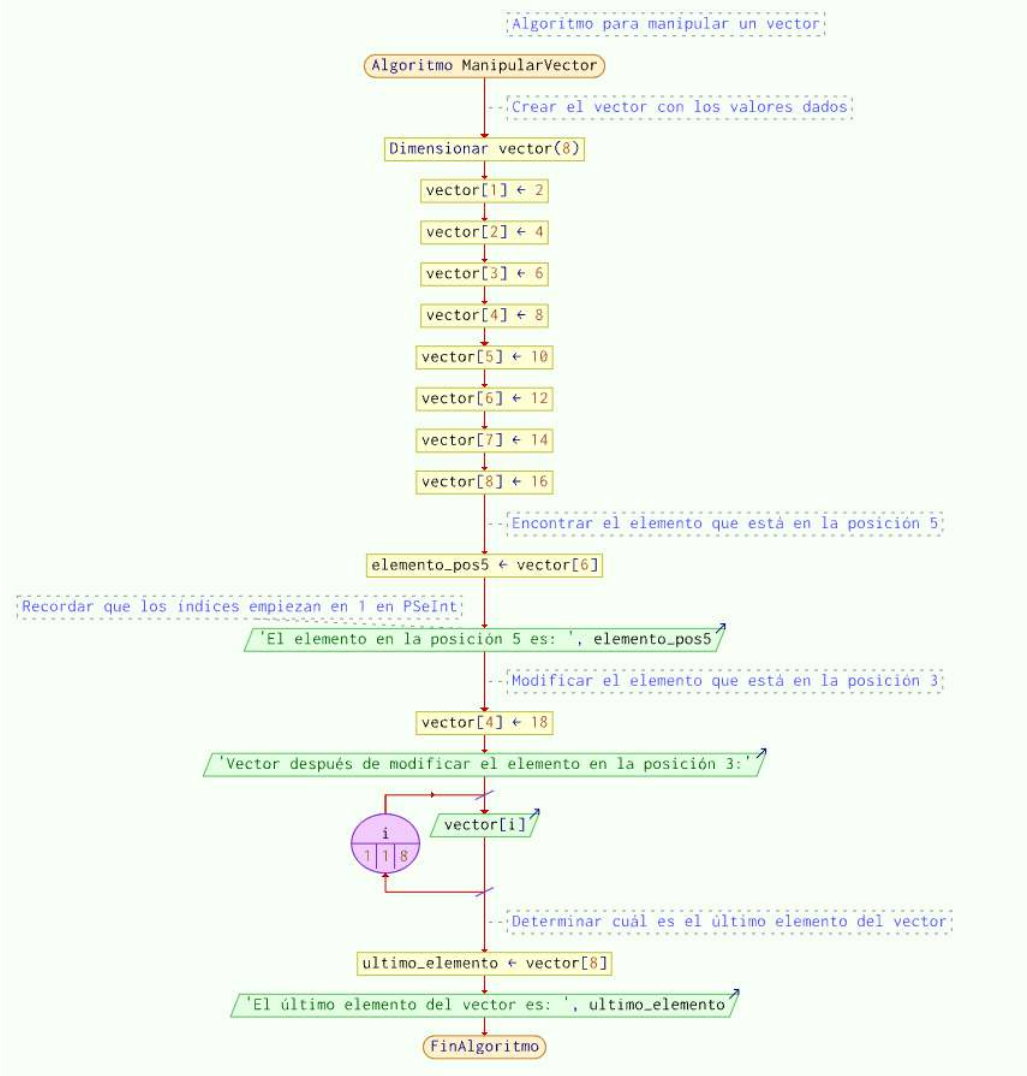
ISBN: 978-631-6557-57-5

Pseudocódigo

```
// Algoritmo para manipular un vector
Algoritmo ManipularVector

    // Crear el vector con los valores dados
    Dimension vector[8]
    vector[1] = 2
    vector[2] = 4
    vector[3] = 6
    vector[4] = 8
    vector[5] = 10
    vector[6] = 12
    vector[7] = 14
    vector[8] = 16

    // Encontrar el elemento que está en la posición 5
    elemento_pos5 = vector[6] // Recordar que los índices empiezan en 1 en PSeInt
    Escribir "El elemento en la posición 5 es: ", elemento_pos5

    // Modificar el elemento que está en la posición 3
    vector[4] = 18
    Escribir "Vector después de modificar el elemento en la posición 3:"
    Para i = 1 Hasta 8 Con Paso 1 Hacer
        Escribir vector[i]
    FinPara

    // Determinar cuál es el último elemento del vector
    ultimo_elemento = vector[8]
    Escribir "El último elemento del vector es: ", ultimo_elemento

FinAlgoritmo
```

ISBN: 978-631-6557-57-5

Diagrama de flujo



Implementación

En Python

```python
# Crear el vector con los valores dados
vector = [2, 4, 6, 8, 10, 12, 14, 16]

# Encontrar el elemento que está en la posición 5
elemento_pos5 = vector[5]  # Recordar que los índices empiezan en 0
print("El elemento en la posición 5 es:", elemento_pos5)

# Modificar el elemento que está en la posición 3
vector[3] = 18
print("Vector después de modificar el elemento en la posición 3:", vector)

# Determinar cuál es el último elemento del vector
ultimo_elemento = vector[-1]
print("El último elemento del vector es:", ultimo_elemento)
```

ISBN: 978-631-6557-57-5

```
El elemento en la posición 5 es: 12
Vector después de modificar el elemento en la posición 3: [2, 4, 6, 18, 10, 12, 14, 16]
El último elemento del vector es: 16
```

- Se crea una lista llamada vector con los valores específicos [2, 4, 6, 8, 10, 12, 14, 16].
- elemento_pos5 = vector[5]: Asigna el valor del sexto elemento del vector (índice 5) a la variable elemento_pos5. En este caso, elemento_pos5 será 12.
- print("El elemento en la posición 5 es:", elemento_pos5): Imprime el valor del sexto elemento en la consola.
- vector[3] = 18: Modifica el cuarto elemento del vector (índice 3) y le asigna el valor 18. Antes, el valor en esa posición era 8.
- print("Vector después de modificar el elemento en la posición 3:", vector): Imprime el vector modificado en la consola. El nuevo vector será [2, 4, 6, 18, 10, 12, 14, 16].
- ultimo_elemento = vector[-1]: Asigna el valor del último elemento del vector a la variable ultimo_elemento. En este caso, ultimo_elemento será 16.
- print("El último elemento del vector es:", ultimo_elemento): Imprime el valor del último elemento en la consola.

En JavaScript

```javascript
// Crear el vector con los valores dados
let vector = [2, 4, 6, 8, 10, 12, 14, 16];
// Encontrar el elemento que está en la posición 5
let elemento_pos5 = vector[5];  // Recordar que los índices empiezan en 0
console.log("El elemento en la posición 5 es:", elemento_pos5);

// Modificar el elemento que está en la posición 3
vector[3] = 18;
console.log("Vector después de modificar el elemento en la posición 3:", vector);

// Determinar cuál es el último elemento del vector
let ultimo_elemento = vector[vector.length - 1];
console.log("El último elemento del vector es:", ultimo_elemento);
```

ISBN: 978-631-6557-57-5

En Java

```java
public class ManipularVector {
    public static void main(String[] args) {
        // Crear el vector con los valores dados
        int[] vector = {2, 4, 6, 8, 10, 12, 14, 16};

        // Encontrar el elemento que está en la posición 5
        int elemento_pos5 = vector[5];  // Recordar que los índices empiezan en 0
        System.out.println("El elemento en la posición 5 es: " + elemento_pos5);

        // Modificar el elemento que está en la posición 3
        vector[3] = 18;
        System.out.println("Vector después de modificar el elemento en la posición 3:"
        for (int i = 0; i < vector.length; i++) {
            System.out.print(vector[i] + " ");
        }
        System.out.println();

        // Determinar cuál es el último elemento del vector
        int ultimo_elemento = vector[vector.length - 1];
        System.out.println("El último elemento del vector es: " + ultimo_elemento);
    }
}
```

6) Encontrar el valor máximo de la lista [34,46,8,79,52]

Pseudocódigo

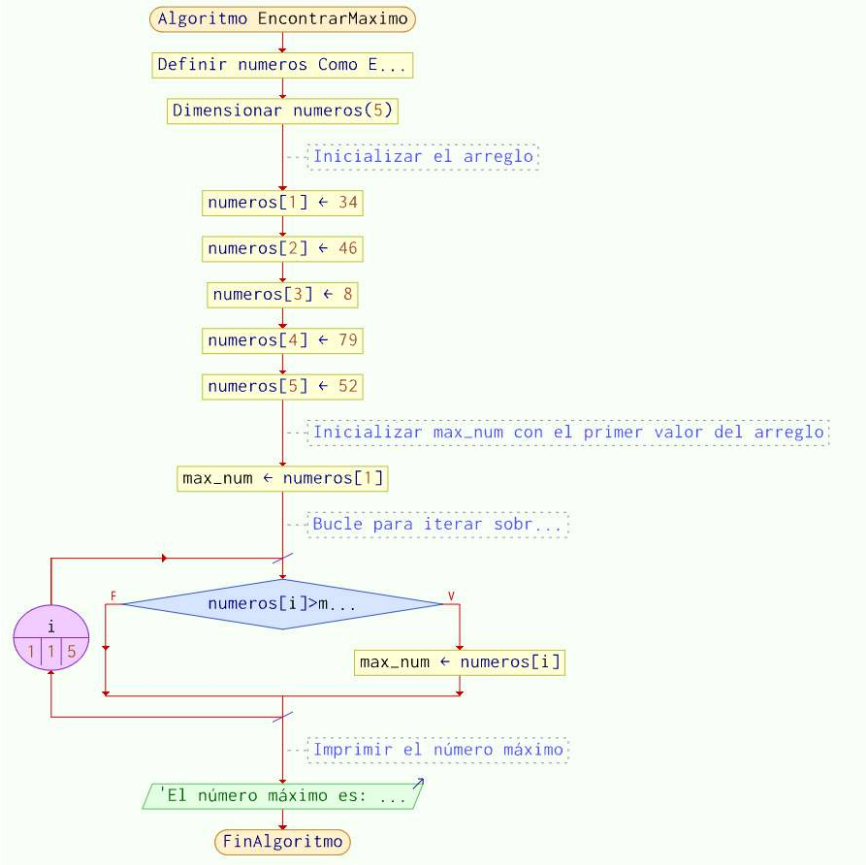
```
Proceso EncontrarMaximo
    Definir numeros Como Entero
    Dimension numeros[5]

    // Inicializar el arreglo
    numeros[1] = 34
    numeros[2] = 46
    numeros[3] = 8
    numeros[4] = 79
    numeros[5] = 52

    // Inicializar max_num con el primer valor del arreglo
    max_num = numeros[1]

    // Bucle para iterar sobre el arreglo
    Para i = 1 Hasta 5 Con Paso 1 Hacer
        Si numeros[i] > max_num Entonces
            max_num = numeros[i]
        FinSi
    FinPara

    // Imprimir el número máximo
    Escribir "El número máximo es: ", max_num
FinProceso
```

- Se declara un arreglo de enteros llamado numeros con 5 elementos. Dimension numeros[5] reserva espacio para 5 elementos en el arreglo.
- Se asignan valores específicos a cada elemento del arreglo numeros. En PSeInt, los índices de los arreglos comienzan en 1.

ISBN: 978-631-6557-57-5

- Se inicializa la variable max_num con el primer valor del arreglo numeros (es decir, 34). Esto establece un valor de referencia inicial para la comparación en el bucle.
- Se utiliza un bucle Para para iterar sobre los índices del arreglo numeros desde 1 hasta 5. Con Paso 1 indica que el índice i se incrementa en 1 en cada iteración.
- Dentro del bucle, se evalúa si el valor actual del arreglo numeros (es decir, numeros[i]) es mayor que el valor actual de max_num. Si es así, max_num se actualiza con el nuevo valor mayor.
- Después de que el bucle ha terminado de iterar sobre todos los elementos del arreglo, se imprime el valor de max_num, que es el número máximo encontrado en el arreglo.

Diagrama de flujo



ISBN: 978-631-6557-57-5

En Python

```
numeros = [34,46,8,79,52]
max_num = numeros[0]
for num in numeros:
    if num > max_num:
        max_num = num
print("El número máximo es:", max_num)
```

- Se crea una lista llamada numeros que contiene los valores [34, 46, 8, 79, 52].
- Se inicializa la variable max_num con el primer valor de la lista numeros, que es 34, para lo cual se utiliza numeros[0] que es un valor inicial para poder comparar con el valor inicial de la lista.
- Se utiliza un bucle for para iterar sobre cada elemento de la lista numeros.
- En cada iteración, la variable num toma el valor del elemento actual de la lista.
- Dentro del bucle, se evalúa si el valor de num es mayor que el valor actual de max_num.
- Si num es mayor que max_num, entonces se actualiza max_num para que tome el valor de num.
- Después de que el bucle ha terminado de iterar sobre todos los elementos de la lista, se imprime el valor de max_num, que es el número máximo encontrado en la lista.

En JavaScript

```
1  let numeros = [34, 46, 8, 79, 52];
2  let max_num = numeros[0];
3
4  for (let num of numeros) {
5      if (num > max_num) {
6          max_num = num;
7      }
8  }
9
10 console.log("El número máximo es:", max_num);
```

```
El número máximo es: 79
```

- Se crea un arreglo llamado numeros que contiene los valores [34, 46, 8, 79, 52].
- Se inicializa la variable max_num con el primer valor del arreglo numeros, que es 34.

ISBN: 978-631-6557-57-5

- Se utiliza un bucle for...of para iterar sobre cada elemento del arreglo numeros.
- En cada iteración, la variable num toma el valor del elemento actual del arreglo.
- Dentro del bucle, se evalúa si el valor de num es mayor que el valor actual de max_num.
- Si num es mayor que max_num, entonces max_num se actualiza para que tome el valor de num.

En Java

```java
public class EncontrarMaximo {
    public static void main(String[] args) {
        // Inicializar el arreglo de números
        int[] numeros = {34, 46, 8, 79, 52};

        // Inicializar max_num con el primer valor del arreglo
        int max_num = numeros[0];

        // Bucle para iterar sobre el arreglo
        for (int num : numeros) {
            if (num > max_num) {
                max_num = num;
            }
        }

        // Imprimir el número máximo
        System.out.println("El número máximo es: " + max_num);
    }
}
```

- Se crea un arreglo de enteros llamado numeros que contiene los valores [34, 46, 8, 79, 52].
- Se inicializa la variable max_num con el primer valor del arreglo numeros, que es 34.
- Se utiliza un bucle for-each para iterar sobre cada elemento del arreglo numeros.
- En cada iteración, la variable num toma el valor del elemento actual del arreglo.
- Dentro del bucle, se evalúa si el valor de num es mayor que el valor actual de max_num. Si num es mayor que max_num, entonces max_num se actualiza para que tome el valor de num.
- Después de que el bucle ha terminado de iterar sobre todos los elementos del arreglo, se imprime el valor de max_num, que es el número máximo encontrado en el arreglo.

ISBN: 978-631-6557-57-5

En C++

```cpp
#include <iostream>
using namespace std;

int main() {
    // Inicializar el arreglo de números
    int numeros[] = {34, 46, 8, 79, 52};
    int size = sizeof(numeros) / sizeof(numeros[0]);

    // Inicializar max_num con el primer valor del arreglo
    int max_num = numeros[0];

    // Bucle para iterar sobre el arreglo
    for (int i = 0; i < size; i++) {
        if (numeros[i] > max_num) {
            max_num = numeros[i];
        }
    }

    // Imprimir el número máximo
    cout << "El número máximo es: " << max_num << endl;

    return 0;
}
```

- #include <iostream> es necesario para utilizar cout para imprimir en la consola.
- using namespace std; permite usar las funciones y objetos del espacio de nombres estándar sin necesidad de prefijarlos con std::.
- Se crea un arreglo de enteros llamado numeros que contiene los valores [34, 46, 8, 79, 52].
- size se calcula para obtener el número de elementos en el arreglo numeros.
- Se inicializa la variable max_num con el primer valor del arreglo numeros, que es 34.
- Se utiliza un bucle for para iterar sobre cada elemento del arreglo numeros.
- En cada iteración, la variable i toma el valor del índice actual del arreglo.
- Dentro del bucle, se evalúa si el valor de numeros[i] es mayor que el valor actual de max_num. Si numeros[i] es mayor que max_num, entonces max_num se actualiza para que tome el valor de numeros[i].
- Después de que el bucle ha terminado de iterar sobre todos los elementos del arreglo, se imprime el valor de max_num, que es el número máximo encontrado en el arreglo.

7) Invertir la lista [7,8,9,10,11,12]

ISBN: 978-631-6557-57-5

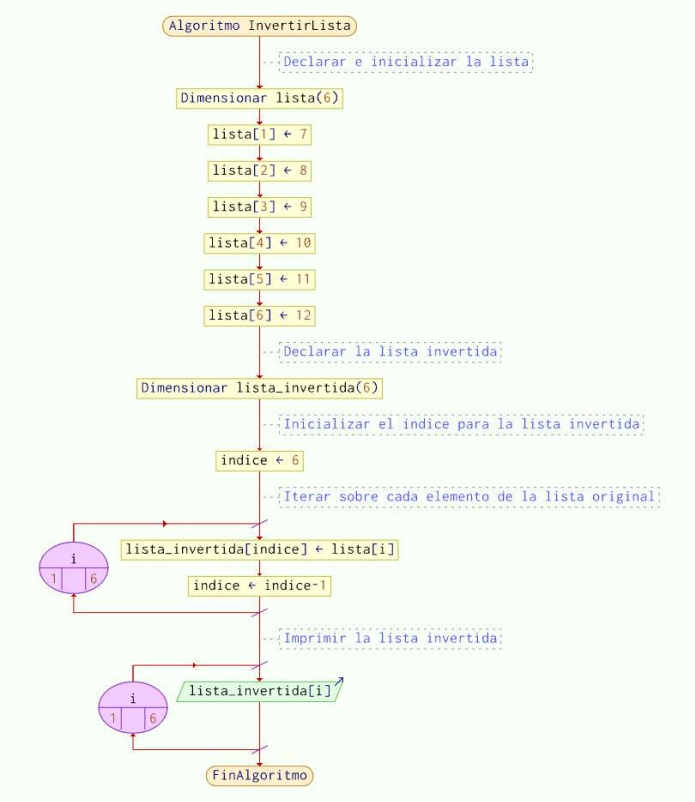
Pseudocódigo

```
Algoritmo InvertirLista
    // Declarar e inicializar la lista
    Dimension lista[6]
    lista[1] ← 7
    lista[2] ← 8
    lista[3] ← 9
    lista[4] ← 10
    lista[5] ← 11
    lista[6] ← 12

    // Declarar la lista invertida
    Dimension lista_invertida[6]

    // Inicializar el índice para la lista invertida
    indice ← 6

    // Iterar sobre cada elemento de la lista original
    Para i ← 1 Hasta 6 Hacer
        lista_invertida[indice] ← lista[i]
        indice ← indice - 1
    Fin Para

    // Imprimir la lista invertida
    Para i ← 1 Hasta 6 Hacer
        Escribir lista_invertida[i]
    Fin Para
FinAlgoritmo
```

- Se declara e inicializa la lista denominada lista con los valores [7, 8, 9, 10, 11, 12].
- Se declara la lista lista_invertida para almacenar los elementos en orden inverso.
- Se inicializa el índice indice en 6, que es la última posición de la lista invertida.
- Se utiliza un bucle Para para recorrer cada elemento de la lista original.
- En cada iteración, el elemento de lista[i] se asigna a la posición indice de lista_invertida.
- Se decrece indice en 1 para la siguiente iteración, moviendo de derecha a izquierda en la lista_invertida.
- Se utiliza un bucle Para para imprimir cada elemento de la lista invertida.

ISBN: 978-631-6557-57-5

Diagrama de flujo



Implementación

En Python

```
lista = [7,8, 9, 10, 11, 12]
lista_invertida = []
for num in lista:
    lista_invertida.insert(0, num)
print(lista_invertida)
```

```
[12, 11, 10, 9, 8, 7]
```

Se crea una lista llamada **lista** que contiene los elementos [7, 8, 9, 10, 11, 12].

Se crea una lista vacía llamada **lista_invertida** que se utilizará para almacenar los elementos de lista en orden inverso.

for num in lista: Este es el inicio de un bucle for que itera sobre cada elemento de la lista denominada **lista**. En cada iteración, la variable **num** toma el valor del siguiente elemento en **lista**.

ISBN: 978-631-6557-57-5

lista_invertida.insert(0, num): Dentro del bucle, se inserta el valor de **num** en la primera posición (índice 0) de lista_invertida. Este método desplaza todos los elementos existentes en lista_invertida hacia la derecha, haciendo espacio para el nuevo elemento al principio de la lista.

Después de que el bucle haya terminado (es decir, después de haber insertado todos los elementos de lista en lista_invertida), se imprime el contenido de lista_invertida.

En JavaScript

```
let lista = [7, 8, 9, 10, 11, 12];
let lista_invertida = [];

for (let num of lista) {
    lista_invertida.unshift(num);
}

console.log(lista_invertida);
```

for (let num of lista) es un bucle for...of que itera sobre cada elemento de la lista lista. En cada iteración, la variable num toma el valor del siguiente elemento en lista.

lista_invertida.unshift(num); inserta el valor de num en la primera posición de lista_invertida. El método unshift desplaza todos los elementos existentes en lista_invertida hacia la derecha, haciendo espacio para el nuevo elemento al principio de la lista.

ISBN: 978-631-6557-57-5

En C++

```cpp
#include <iostream>
#include <vector>

int main() {
    // Declarar e inicializar la lista
    std::vector<int> lista = {7, 8, 9, 10, 11, 12};
    std::vector<int> lista_invertida;

    // Iterar sobre cada elemento de la lista
    for (int num : lista) {
        lista_invertida.insert(lista_invertida.begin(), num);
    }

    // Imprimir la lista invertida
    for (int num : lista_invertida) {
        std::cout << num << " ";
    }

    return 0;
}
```

#include <vector>: Incluye la biblioteca de vectores, que permite utilizar el contenedor std::vector similar a las listas en Python.

std::vector<int> lista = {7, 8, 9, 10, 11, 12}; declara un vector de enteros llamado lista y lo inicializa con los valores [7, 8, 9, 10, 11, 12].

std::vector<int> lista_invertida; declara un vector vacío llamado lista_invertida.

for (int num : lista) es un bucle for-each que itera sobre cada elemento del vector lista.

lista_invertida.insert(lista_invertida.begin(), num); inserta el valor de **num** en la primera posición del vector lista_invertida. El método insert desplaza todos los elementos existentes hacia la derecha, haciendo espacio para el nuevo elemento al principio del vector.

for (int num : lista_invertida) es otro bucle for-each que itera sobre cada elemento del vector lista_invertida.

std::cout << **num** << " "; imprime el valor de num seguido de un espacio.

ISBN: 978-631-6557-57-5

En Java

```java
import java.util.ArrayList;
import java.util.List;

public class InvertirLista {
    public static void main(String[] args) {
        // Declarar e inicializar la lista
        List<Integer> lista = new ArrayList<>();
        lista.add(7);
        lista.add(8);
        lista.add(9);
        lista.add(10);
        lista.add(11);
        lista.add(12);

        List<Integer> lista_invertida = new ArrayList<>();

        // Iterar sobre cada elemento de la lista
        for (int num : lista) {
            lista_invertida.add(0, num);
        }

        // Imprimir la lista invertida
        System.out.println(lista_invertida);
    }
}
```

import java.util.ArrayList;: Importa la clase ArrayList, que permite crear listas dinámicas.

import java.util.List;: Importa la interfaz List, que proporciona métodos para trabajar con listas.

List<Integer> lista = new ArrayList<>(); declara una lista de enteros llamada lista y la inicializa como una nueva ArrayList. El operador <> aquí se conoce como el operador "diamond" y se usa para deducir el tipo genérico del lado derecho a partir del lado izquierdo de la asignación, es decir, Integer.

lista.add(7);, lista.add(8);, etc., añaden los elementos [7, 8, 9, 10, 11, 12] a la lista lista.

List<Integer> lista_invertida = new ArrayList<>(); declara una lista de enteros vacía llamada lista_invertida.

for (int num : lista) es un bucle for-each que itera sobre cada elemento de la lista lista.

lista_invertida.add(0, num); inserta el valor de **num** en la primera posición de lista_invertida. El método add con dos argumentos desplaza todos los elementos existentes hacia la derecha, haciendo espacio para el nuevo elemento al principio de la lista.

8) Formar una lista con los números pares de la lista [7,8, 9, 10, 11, 12]

ISBN: 978-631-6557-57-5

Implementación

En Python

```
lista = [11, 14, 23, 54, 90,101]
pares = []
for num in lista:
    if num % 2 == 0:
        pares.append(num)
print(pares)
```

```
[14, 54, 90]
```

for num in lista:: Este es el inicio de un bucle for que itera sobre cada elemento de la lista denominada **lista**. En cada iteración, la variable **num** toma el valor del siguiente elemento en la lista.

if num % 2 == 0:: Comprueba si el número actual (num) es par. El operador % calcula el residuo de la división de num por 2. Si el residuo es 0, entonces num es par.

pares.append(num): Si **num** es par, se añade a la lista pares utilizando el método append.

En JavaScript

```
let lista = [11, 14, 23, 54, 90, 101];
let pares = [];

for (let num of lista) {
    if (num % 2 === 0) {
        pares.push(num);
    }
}

console.log(pares);
```

let pares = []; se declara una lista vacía llamada pares que se utilizará para almacenar los números pares encontrados en lista.

for (let num of lista) es un bucle for...of que itera sobre cada elemento de la lista **lista**. En cada iteración, la variable num toma el valor del siguiente elemento en la lista.

if (num % 2 === 0) comprueba si el número actual (num) es par. El

ISBN: 978-631-6557-57-5

operador % calcula el residuo de la división de num por 2. Si el residuo es 0, entonces num es par.

pares.push(num); añade num a la lista pares utilizando el método push.En C

En C++

```cpp
#include <iostream>
#include <vector>

int main() {
    // Declarar e inicializar la lista
    std::vector<int> lista = {11, 14, 23, 54, 90, 101};
    std::vector<int> pares;

    // Iterar sobre cada elemento de la lista
    for (int num : lista) {
        if (num % 2 == 0) {
            pares.push_back(num);
        }
    }

    // Imprimir la lista de pares
    std::cout << "[";
    for (size_t i = 0; i < pares.size(); ++i) {
        std::cout << pares[i];
        if (i < pares.size() - 1) {
            std::cout << ", ";
        }
    }
    std::cout << "]" << std::endl;

    return 0;
}
```

for (int num : lista) es un bucle for-each que itera sobre cada elemento del vector lista.

if (num % 2 == 0) comprueba si el número actual (num) es par. El operador % calcula el residuo de la división de num por 2. Si el residuo es 0, entonces num es par.

pares.**push_back**(num); añade num al vector pares utilizando el método push_back.

std::cout << "["; imprime un corchete de apertura.

for (size_t i = 0; i < pares.size(); ++i) es un bucle que itera sobre cada elemento del vector pares.

std::cout << pares[i]; imprime el valor de pares[i].

if (i < pares.size() - 1) comprueba si el elemento actual no es el último en el

ISBN: 978-631-6557-57-5

vector. Si no lo es, imprime una coma y un espacio.

std::cout << "]" << std::endl; imprime un corchete de cierre y un salto de línea.

En Java

```java
import java.util.ArrayList;
import java.util.List;

public class FiltrarPares {
    public static void main(String[] args) {
        // Declarar e inicializar la lista
        List<Integer> lista = new ArrayList<>();
        lista.add(11);
        lista.add(14);
        lista.add(23);
        lista.add(54);
        lista.add(90);
        lista.add(101);

        List<Integer> pares = new ArrayList<>();

        // Iterar sobre cada elemento de la lista
        for (int num : lista) {
            if (num % 2 == 0) {
                pares.add(num);
            }
        }

        // Imprimir la lista de pares
        System.out.println(pares);
    }
}
```

List<Integer> lista = new ArrayList<>(); declara una lista de enteros llamada lista y la inicializa como una nueva ArrayList.

lista.add(11);, lista.add(14);, etc., añaden los elementos 11, 14, 23, 54, 90, 101 a la lista lista.

List<Integer> pares = new ArrayList<>(); declara una lista de enteros vacía llamada pares.

for (int num : lista) es un bucle for-each que itera sobre cada elemento de la lista lista.

if (num % 2 == 0) comprueba si el número actual (num) es par. El operador % calcula el residuo de la división de num por 2. Si el residuo es 0, entonces num es par.

pares.**add**(num); añade num a la lista pares utilizando el método add.

System.out.println(pares); imprime el contenido de pares en la consola. Al llamar implícitamente a pares.**toString**(), se obtiene una representación de cadena de la lista, que se imprime automáticamente.

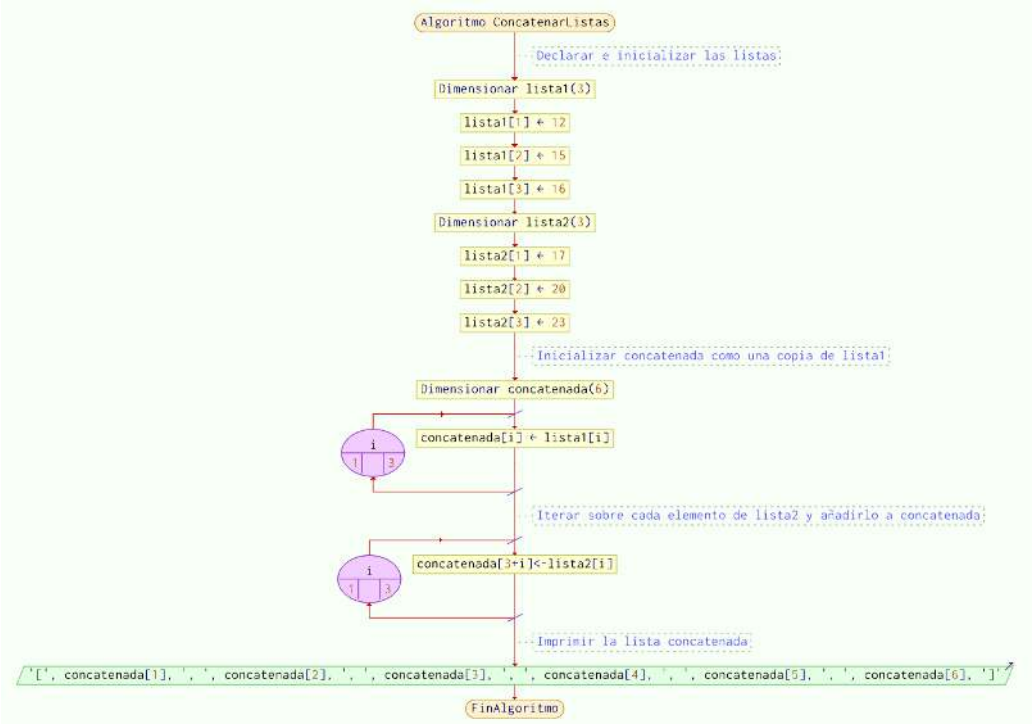
9) Concadenar las listas [12, 15, 16] [17, 20, 23]

ISBN: 978-631-6557-57-5

Pseudocódigo

```
Algoritmo ConcatenarListas
    // Declarar e inicializar las listas
    Dimension lista1[3]
    lista1[1] ← 12
    lista1[2] ← 15
    lista1[3] ← 16

    Dimension lista2[3]
    lista2[1] ← 17
    lista2[2] ← 20
    lista2[3] ← 23

    // Inicializar concatenada como una copia de lista1
    Dimension concatenada[6]
    Para i ← 1 Hasta 3 Hacer
        concatenada[i] ← lista1[i]
    Fin Para

    // Iterar sobre cada elemento de lista2 y añadirlo a concatenada
    Para i ← 1 Hasta 3 Hacer
        concatenada[3 + i] ← lista2[i]
    Fin Para

    // Imprimir la lista concatenada
    Escribir "[", concatenada[1], ", ", concatenada[2], ", ", concatenada[3], ", ", concatenada[4], ", ", concatenada[5], ", ", concatenada[6],
FinAlgoritmo
```

Diagrama de flujo



ISBN: 978-631-6557-57-5

Implementación

En Python

```
lista1 = [12, 15, 16]
lista2 = [17, 20, 23]
concatenada = lista1
for num in lista2:
    concatenada.append(num)
print(concatenada)
```

```
[12, 15, 16, 17, 20, 23]
```

Se crea una nueva lista llamada **concatenada** que inicialmente es una referencia a la lista1. Es importante notar que **concatenada** no es una copia de lista1, sino que ambas variables apuntan al mismo objeto en la memoria. Por lo tanto, cualquier modificación realizada a concatenada también se refleja en lista1.

for num in lista2: Este es el inicio de un bucle for que itera sobre cada elemento de la lista lista2. En cada iteración, la variable **num** toma el valor del siguiente elemento en la lista. concatenada.**append**(num): Dentro del bucle, se añade el valor de num al final de la lista concatenada utilizando el método **append**.

En JavaScript

```
1  let lista1 = [12, 15, 16];
2  let lista2 = [17, 20, 23];
3  let concatenada = lista1;
4
5  for (let num of lista2) {
6      concatenada.push(num);
7  }
8
9  console.log(concatenada);
```

let **concatenada** = lista1; crea una referencia a lista1 llamada concatenada. Al igual que en Python, concatenada no es una copia de lista1, sino que ambas variables apuntan al mismo objeto en la memoria.

for (let num of lista2) es un bucle for...of que itera sobre cada elemento del array lista2. En cada iteración, la variable **num** toma el valor del siguiente elemento en el array.

concatenada.push(num); añade el valor de **num** al final del array concatenado utilizando el método push.

ISBN: 978-631-6557-57-5

En C++

```cpp
#include <iostream>
#include <vector>

int main() {
    // Declarar e inicializar las listas
    std::vector<int> lista1 = {12, 15, 16};
    std::vector<int> lista2 = {17, 20, 23};
    std::vector<int>& concatenada = lista1;

    // Iterar sobre cada elemento de lista2
    for (int num : lista2) {
        concatenada.push_back(num);
    }

    // Imprimir la lista concatenada
    std::cout << "[";
    for (size_t i = 0; i < concatenada.size(); ++i) {
        std::cout << concatenada[i];
        if (i < concatenada.size() - 1) {
            std::cout << ", ";
        }
    }
    std::cout << "]" << std::endl;

    return 0;
}
```

std::vector<int> lista1 = {12, 15, 16}; declara un vector de enteros llamado lista1 y lo inicializa con los valores [12, 15, 16].

std::vector<int> lista2 = {17, 20, 23}; declara un vector de enteros llamado lista2 y lo inicializa con los valores [17, 20, 23].

std::vector<int>& concatenada = lista1; crea una referencia a lista1 llamada concatenada. Esto es similar a la asignación en Python, donde concatenada no es una copia de lista1, sino que ambas variables apuntan al mismo objeto en la memoria.

for (int **num** : lista2) es un bucle for-each que itera sobre cada elemento del vector lista2.

concatenada.**push_back**(num); añade el valor de num al final del vector concatenada utilizando el método push_back.

std::cout << "["; imprime un corchete de apertura.

for (size_t i = 0; i < concatenada.size(); ++i) es un bucle que itera sobre cada elemento del vector concatenada.

std::cout << concatenada[i]; imprime el valor de concatenada[i].

if (i < concatenada.size() - 1) comprueba si el elemento actual no es el último en el vector. Si no lo es, imprime una coma y un espacio.

std::cout << "]" << std::endl; imprime un corchete de cierre y un salto de línea.

ISBN: 978-631-6557-57-5

En Java

```java
import java.util.ArrayList;
import java.util.List;

public class ConcatenarListas {
    public static void main(String[] args) {
        // Declarar e inicializar las listas
        List<Integer> lista1 = new ArrayList<>();
        lista1.add(12);
        lista1.add(15);
        lista1.add(16);

        List<Integer> lista2 = new ArrayList<>();
        lista2.add(17);
        lista2.add(20);
        lista2.add(23);

        List<Integer> concatenada = lista1;

        // Iterar sobre cada elemento de lista2
        for (int num : lista2) {
            concatenada.add(num);
        }

        // Imprimir la lista concatenada
        System.out.println(concatenada);
    }
}
```

List<Integer> lista1 = new ArrayList<>(); declara una lista de enteros llamada lista1 y la inicializa como una nueva ArrayList.

lista1.add(12);, lista1.add(15);, etc., añaden los elementos 12, 15, 16 a la lista lista1.

List<Integer> lista2 = new ArrayList<>(); declara una lista de enteros llamada lista2 y la inicializa como una nueva ArrayList.

lista2.add(17);, lista2.add(20);, etc., añaden los elementos 17, 20, 23 a la lista lista2.

List<Integer> concatenada = lista1; crea una referencia a lista1 llamada concatenada. Al igual que en Python, concatenada no es una copia de lista1, sino que ambas variables apuntan al mismo objeto en la memoria.

for (int num : lista2) es un bucle for-each que itera sobre cada elemento de la lista lista2.

concatenada.**add**(num); añade el valor de num al final de la lista concatenada utilizando el método add.

10) Eliminar los números que aparecen duplicados en la lista [8, 5, 6, 8, 7, 8, 8, 5].

ISBN: 978-631-6557-57-5

Pseudocódigo

```
Algoritmo EliminarDuplicados
    // Declarar e inicializar la lista original
    Dimension lista[8]
    lista[1] ← 8
    lista[2] ← 5
    lista[3] ← 6
    lista[4] ← 8
    lista[5] ← 7
    lista[6] ← 8
    lista[7] ← 8
    lista[8] ← 5

    // Declarar la lista para almacenar los elementos sin duplicados
    Dimension sin_duplicados[8]
    indice_sin_duplicados ← 0

    // Iterar sobre cada elemento de la lista original
    Para i ← 1 Hasta 8 Hacer
        encontrado ← Falso

        // Verificar si el elemento ya está en la lista sin duplicados
        Si indice_sin_duplicados > 0 Entonces
            Para j ← 1 Hasta indice_sin_duplicados Hacer
                Si lista[i] = sin_duplicados[j] Entonces
                    encontrado ← Verdadero
                Fin Si
            Fin Para
        Fin Si

        // Si no se encontró el elemento, agregarlo a la lista sin duplicados
        Si No encontrado Entonces
            indice_sin_duplicados ← indice_sin_duplicados + 1
            sin_duplicados[indice_sin_duplicados] ← lista[i]
        Fin Si
    Fin Para

    // Imprimir la lista sin duplicados
    Escribir "["
    Para k ← 1 Hasta indice_sin_duplicados Hacer
        Escribir sin_duplicados[k]
        Si k < indice_sin_duplicados Entonces
            Escribir ", "
        Fin Si
    Fin Para
    Escribir "]"
FinAlgoritmo
```

Dimension sin_duplicados[8]: Se declara una lista sin_duplicados que almacenará los elementos sin duplicados.

indice_sin_duplicados <- 0: También se inicializa indice_sin_duplicados a 0, que se utilizará para seguir la posición en sin_duplicados.

Para i <- 1 Hasta 8 Hacer: Se inicia un bucle para recorrer cada elemento de lista desde el índice 1 hasta el 8.

Si indice_sin_duplicados > 0 Entonces: Aquí se verifica si el elemento actual (lista[i]) ya está en sin_duplicados. Si indice_sin_duplicados es mayor que 0, se recorre sin_duplicados hasta el índice indice_sin_duplicados. Si se encuentra el elemento en sin_duplicados, se marca encontrado como verdadero.

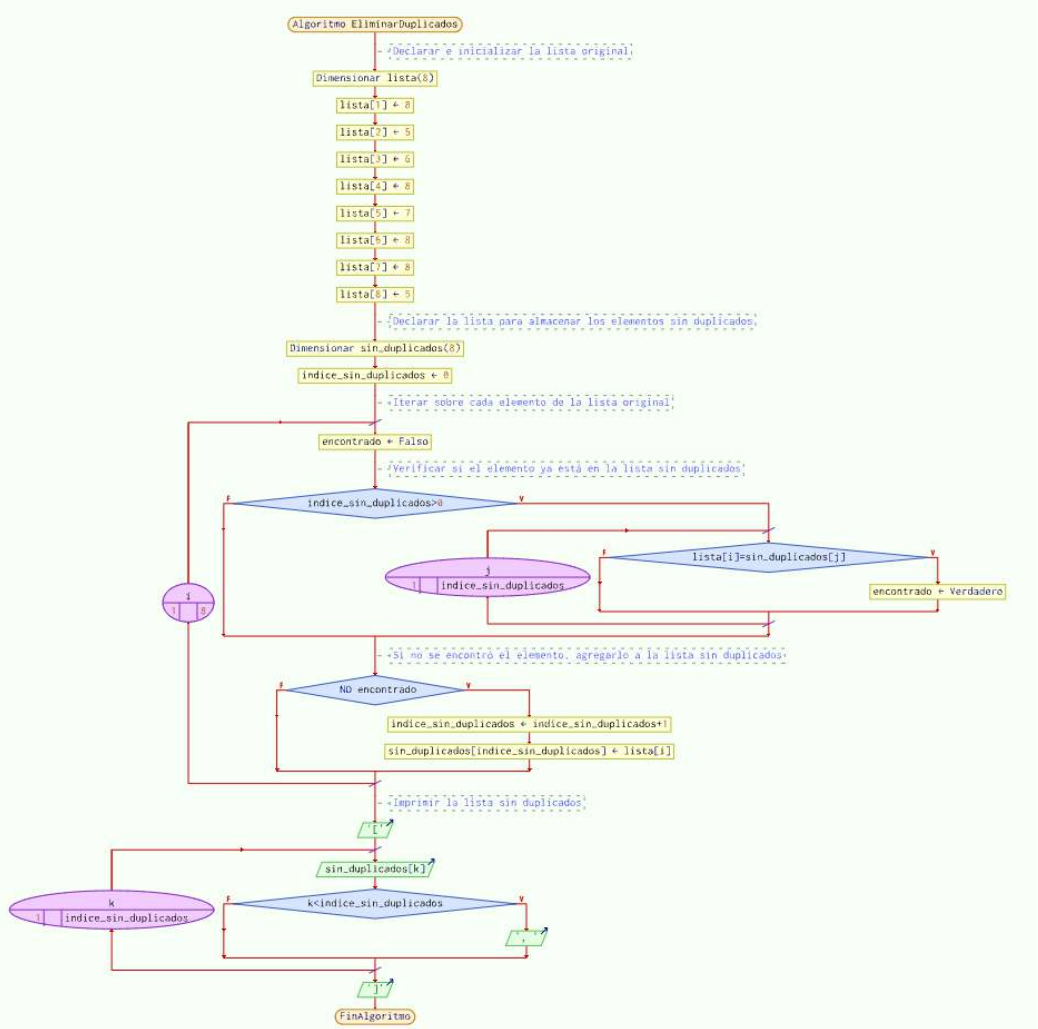
Si No encontrado Entonces: Si **encontrado** es falso (es decir, el elemento no está en sin_duplicados), se incrementa indice_sin_duplicados y se agrega el elemento

ISBN: 978-631-6557-57-5

lista[i] a sin_duplicados en la nueva posición.

Finalmente, se imprime la lista **sin_duplicados**. Se utiliza un bucle para recorrer desde 1 hasta indice_sin_duplicados, escribiendo cada elemento. Se añade una coma y un espacio después de cada elemento, excepto el último.

Diagrama de flujo



ISBN: 978-631-6557-57-5

Implementación
En Pyhton

```
lista = [8, 5, 6, 8, 7, 8, 8, 5]
sin_duplicados = []
for num in lista:
    if num not in sin_duplicados:
        sin_duplicados.append(num)
print(sin_duplicados)
```

```
[8, 5, 6, 7]
```

for num in lista: Se utiliza un bucle for para iterar sobre cada elemento de la lista denominada **lista**. En cada iteración, **num** toma el valor del siguiente elemento de la lista.

if num not in sin_duplicados: Dentro del bucle for, hay una declaración if que comprueba si el número actual (num) no está ya en la lista sin_duplicados.

sin_duplicados.**append**(num): Si num no está en sin_duplicados, se agrega a la lista sin_duplicados, utilizando el método **append** para agregar un elemento al final de la lista.

En JavaScript

```
1  const lista = [8, 5, 6, 8, 7, 8, 8, 5];
2  const sin_duplicados = [];
3
4  for (let num of lista) {
5      if (!sin_duplicados.includes(num)) {
6          sin_duplicados.push(num);
7      }
8  }
9
10 console.log(sin_duplicados);
```

const lista = [8, 5, 6, 8, 7, 8, 8, 5]: Se define una lista constante que contiene los números [8, 5, 6, 8, 7, 8, 8, 5]. Una vez que una variable ha sido asignada utilizando const, su valor no puede cambiarse a otro valor mediante una reasignación.

const sin_duplicados = []: Se inicializa una lista vacía sin_duplicados que se utilizará para almacenar los números únicos de **lista**.

for (let num of lista) {: Se utiliza un bucle for...of para iterar sobre cada elemento de la lista denominada **lista**. En cada iteración, **num** toma el valor del siguiente elemento de la lista.

if (!sin_duplicados.includes(num)) {: Dentro del bucle for, hay una declaración if que comprueba si el número actual (num) no está ya en la lista sin_duplicados. Se utiliza el operador lógico de negación !.

sin_duplicados.**push**(num): Si num no está en sin_duplicados, se agrega a la lista

ISBN: 978-631-6557-57-5

sin_duplicados usando el método **push**.

En C++

```cpp
#include <iostream>
#include <vector>
#include <algorithm>  // Para std::find

int main() {
    std::vector<int> lista = {8, 5, 6, 8, 7, 8, 8, 5};
    std::vector<int> sin_duplicados;

    for (int num : lista) {
        if (std::find(sin_duplicados.begin(), sin_duplicados.end(), num) ==
            sin_duplicados.end()) {
            sin_duplicados.push_back(num);
        }
    }

    for (int num : sin_duplicados) {
        std::cout << num << " ";
    }

    return 0;
}
```

```
8 5 6 7
```

#include <algorithm> se incluye para utilizar el algoritmo std::find

std::vector<int> lista = {8, 5, 6, 8, 7, 8, 8, 5}: Se define un vector lista que contiene los números [8, 5, 6, 8, 7, 8, 8, 5].

std::vector<int> sin_duplicados: Se inicializa un vector vacío sin_duplicados que se utilizará para almacenar los números únicos de lista.

for (int num : lista) {: Se utiliza un bucle for para iterar sobre cada elemento de lista. En cada iteración, **num** toma el valor del siguiente elemento del vector.

std::find(sin_duplicados.begin(), sin_duplicados.end(), num): Busca el valor **num** en el vector sin_duplicados.

Si **num** no se encuentra en sin_duplicados (es decir, std::find devuelve sin_duplicados.end()), se agrega **num** al final de sin_duplicados usando push_back.

Se utiliza otro bucle for para iterar sobre cada elemento de sin_duplicados y se imprime cada número seguido de un espacio.

ISBN: 978-631-6557-57-5

En Java

```java
import java.util.ArrayList;
import java.util.List;

public class EliminarDuplicados {
    public static void main(String[] args) {
        List<Integer> lista = new ArrayList<>();
        lista.add(8);
        lista.add(5);
        lista.add(6);
        lista.add(8);
        lista.add(7);
        lista.add(8);
        lista.add(8);
        lista.add(5);

        List<Integer> sinDuplicados = new ArrayList<>();

        for (Integer num : lista) {
            if (!sinDuplicados.contains(num)) {
                sinDuplicados.add(num);
            }
        }

        System.out.println(sinDuplicados);
    }
}
```

Se utiliza un bucle for-each para iterar sobre cada elemento de lista. En cada iteración, se comprueba si el número actual (num) no está en sinDuplicados utilizando el método **contains**.
Si num no está en sinDuplicados, se añade a la lista sinDuplicados usando el método **add**.

11) Reemplazar todos los números negativos con 0.

Pseudocódigo

```
Proceso ReemplazarNegativosConCero
    Definir vector Como Entero
    Definir i Como Entero
    Definir tamano Como Entero

    // Inicializar el vector
    Dimension vector[6]
    vector[1] ← -8
    vector[2] ← 14
    vector[3] ← -5
    vector[4] ← 20
    vector[5] ← -23
    vector[6] ← 60
    i ← 1
    tamano ← 6

    // Bucle while
    Mientras i ≤ tamano Hacer
        Si vector[i] < 0 Entonces
            vector[i] ← 0
        FinSi
        i ← i + 1
    FinMientras

    // Imprimir el vector modificado
    Para j ← 1 Hasta tamano Hacer
        Escribir vector[j]
    FinPara
FinProceso
```

ISBN: 978-631-6557-57-5

Se define un proceso llamado ReemplazarNegativosConCero.

Se declaran las variables vector, i, y tamano como enteros, vector es el arreglo que contiene los elementos, i es un índice que se usará para recorrer el vector, y tamano almacena la longitud del vector.

Dimension vector[6]: Declara un vector (arreglo) de 6 elementos.

vector[1] <- -8, vector[2] <- 14, etc.: Asigna valores específicos a cada elemento del vector. En PSeInt, los índices de los arreglos comienzan en 1.

i <- 1: Inicializa el índice i en 1, que se utilizará para recorrer el vector.

tamano <- 6: Asigna el valor 6 a la variable tamano, indicando que el vector tiene 6 elementos.

Mientras i <= tamano Hacer: Este bucle mientras se ejecuta mientras el valor de i sea menor o igual a tamano.

Si vector[i] < 0 Entonces: Dentro del bucle, se verifica si el elemento en la posición i del vector es menor que 0.

vector[i] <- 0: Si el elemento es negativo, se asigna 0 a esa posición en el vector.

i <- i + 1: Incrementa el valor de i en 1 para pasar al siguiente elemento del vector.

FinMientras: Termina el bucle mientras.

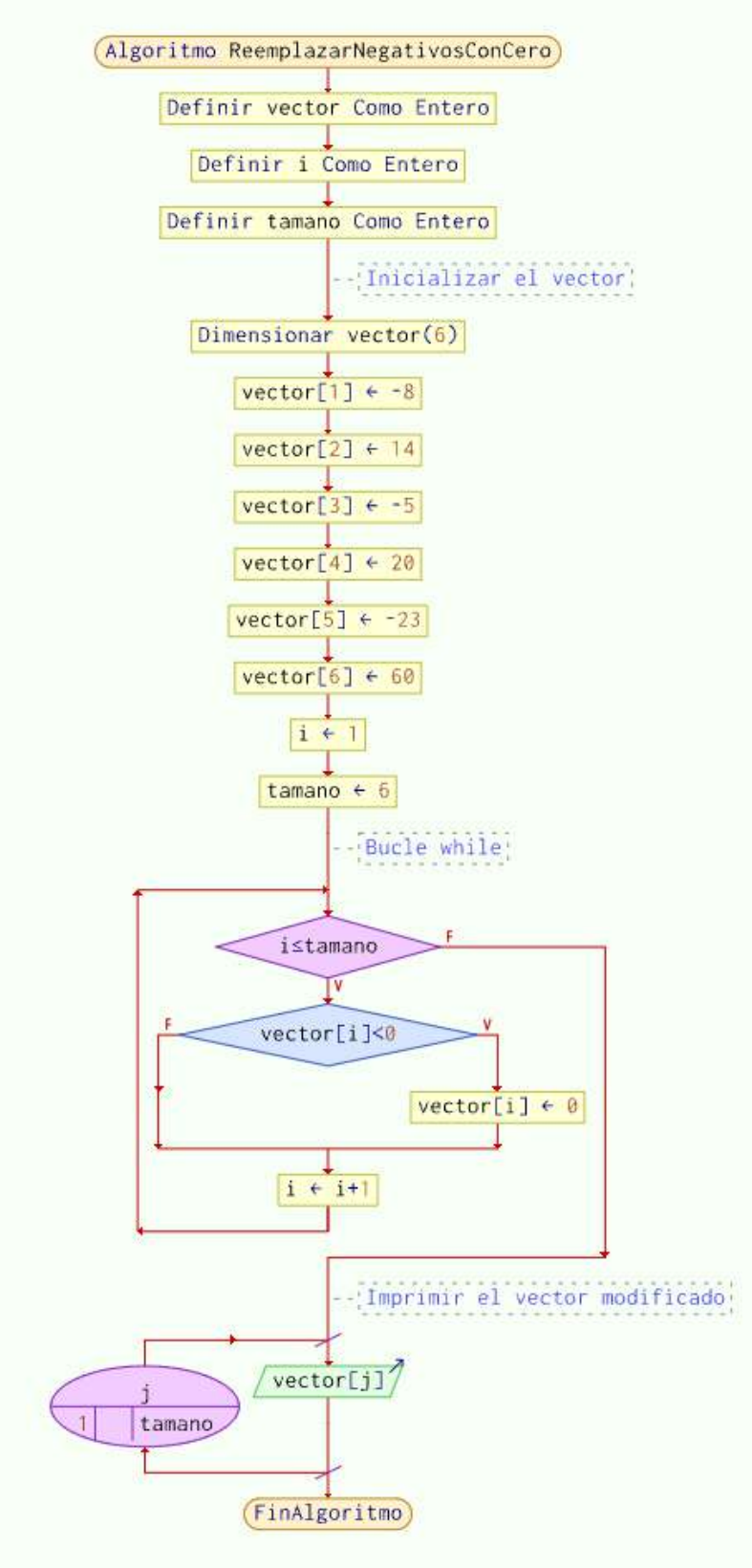
Para j <- 1 Hasta tamano Hacer: Este bucle Para recorre el vector desde el primer índice hasta el último.

Escribir vector[j]: Imprime el valor del elemento en la posición j del vector.

FinPara: Termina el bucle Para.

ISBN: 978-631-6557-57-5

Diagrama de flujo



ISBN: 978-631-6557-57-5

Implementación

En Python

```
vector = [-8, 14, -5, 20, -23, 60]
i = 0
while i < len(vector):
    if vector[i] < 0:
        vector[i] = 0
    i += 1
print(vector)
```

```
[0, 14, 0, 20, 0, 60]
```

Se define un **vector** (lista) llamado vector que contiene seis elementos: [-8, 14, -5, 20, -23, 60].

Se inicializa la variable i en 0. Esta variable se utilizará como índice para recorrer los elementos del vector.

Este bucle while continuará ejecutándose mientras i sea menor que la longitud del vector (len(vector)). len(vector) devuelve el número de elementos en el vector, que en este caso es 6.

Dentro del bucle, se verifica **si** el elemento en la posición **i** del vector es menor que 0. Si vector[i] es negativo, se asigna 0 a esa posición en el vector.

Después de evaluar y (posiblemente) modificar el elemento actual del vector, se incrementa i en 1. Esto asegura que el siguiente ciclo del bucle evalúe el siguiente elemento del vector.

Después de que el bucle while ha terminado (es decir, i es igual a la longitud del vector), se imprime el vector modificado [0, 14, 0, 20, 0, 60] donde todos los elementos negativos han sido reemplazados por 0.

En JavaScript

```
let vector = [-8, 14, -5, 20, -23, 60];
let i = 0;

while (i < vector.length) {
    if (vector[i] < 0) {
        vector[i] = 0;
    }
    i++;
}

console.log(vector);
```

Se define un array llamado vector que contiene los elementos [-8, 14, -5, 20, -23,

ISBN: 978-631-6557-57-5

60].

Se inicializa la variable i en 0, que se utilizará como índice para recorrer los elementos del array.

El bucle while se ejecuta mientras i sea menor que la longitud del array (vector.length).

Dentro del bucle, se verifica si el elemento en la posición i del array es menor que 0.
Si vector[i] es negativo, se asigna 0 a esa posición en el array.

Después de evaluar y (posiblemente) modificar el elemento actual del array, se incrementa i en 1 para pasar al siguiente elemento.

Después de que el bucle while ha terminado (es decir, i es igual a la longitud del array), se imprime el array modificado utilizando console.log.

En C++

```cpp
#include <iostream>
using namespace std;

int main() {
    // Inicializar el vector
    int vector[] = {-8, 14, -5, 20, -23, 60};
    int n = sizeof(vector) / sizeof(vector[0]); // Calcular la longitud del
        vector
    int i = 0;
    // Bucle while
    while (i < n) {
        if (vector[i] < 0) {
            vector[i] = 0;
        }
        i++;
    }
    // Imprimir el vector modificado
    for (int j = 0; j < n; j++) {
        cout << vector[j] << " ";
    }
    cout << endl;

    return 0;
}
```

int vector[] = {-8, 14, -5, 20, -23, 60};: Declara un array llamado vector con los elementos {-8, 14, -5, 20, -23, 60}.

int n = sizeof(vector) / sizeof(vector[0]);: Calcula la longitud del array vector. sizeof(vector) da el tamaño total en bytes del array, y sizeof(vector[0]) da el tamaño en bytes de un elemento del array. Dividiendo estos dos valores obtenemos el número de elementos en el array.

Se inicializa la variable i en 0, que se utilizará como índice para recorrer los

ISBN: 978-631-6557-57-5

elementos del array.

while (i < n): Este bucle while se ejecuta mientras i sea menor que n (la longitud del array).

if (vector[i] < 0): Verifica si el elemento en la posición i del array es menor que 0.

vector[i] = 0;: Si el elemento es negativo, se asigna 0 a esa posición en el array.

i++;: Incrementa el valor de i en 1 para pasar al siguiente elemento.

Se utiliza un bucle for para recorrer el array vector y imprimir cada uno de sus elementos.

cout << vector[j] << " ";: Imprime el elemento en la posición j seguido de un espacio.

cout << endl;: Imprime un salto de línea al final.

En Java

```java
public class Main {
    public static void main(String[] args) {
        // Inicializar el vector
        int[] vector = {-8, 14, -5, 20, -23, 60};
        int i = 0;

        // Bucle while
        while (i < vector.length) {
            if (vector[i] < 0) {
                vector[i] = 0;
            }
            i++;
        }

        // Imprimir el vector modificado
        for (int j = 0; j < vector.length; j++) {
            System.out.print(vector[j] + " ");
        }
    }
}
```

int[] vector = {-8, 14, -5, 20, -23, 60};: Declara un array de enteros llamado vector y lo inicializa con los valores {-8, 14, -5, 20, -23, 60}.

int i = 0;: Inicializa la variable i en 0, que se utilizará como índice para recorrer los elementos del array.

while (i < vector.length): Este bucle while se ejecuta mientras i sea menor que la longitud del array (vector.length).

if (vector[i] < 0): Verifica si el elemento en la posición i del array es menor que 0.

vector[i] = 0;: Si el elemento es negativo, se asigna 0 a esa posición en el array.

i++: Incrementa el valor de i en 1 para pasar al siguiente elemento.

Se utiliza un bucle for para recorrer el array vector y imprimir cada uno de sus

ISBN: 978-631-6557-57-5

elementos.

System.out.print(vector[j] + " ");: Imprime el elemento en la posición j seguido de un espacio.

12) Restar dos vectores elemento a elemento [4,6,7,9] [3,8,5,12]

Pseudocódigo

```
Proceso RestaVectores
    // Declarar los vectores y las variables
    Definir vector1 Como Entero
    Definir vector2 Como Entero
    Definir resta_vectores Como Entero
    Definir i, tamano Como Entero
    // Inicializar los vectores
    Dimension vector1[4]
    Dimension vector2[4]
    Dimension resta_vectores[4]
    vector1[1] ← 4
    vector1[2] ← 6
    vector1[3] ← 7
    vector1[4] ← 9
    vector2[1] ← 3
    vector2[2] ← 8
    vector2[3] ← 5
    vector2[4] ← 12
    i ← 1
    tamano ← 4
    // Bucle while
    Mientras i ≤ tamano Hacer
        resta_vectores[i] ← vector1[i] - vector2[i]
        i ← i + 1
    FinMientras
    // Imprimir el vector de resultados
    Para j ← 1 Hasta tamano Hacer
        Escribir resta_vectores[j]
    FinPara
FinProceso

```

Dimension vector1[4], Dimension vector2[4], Dimension resta_vectores[4]: Son declarados los vectores de tamaño 4, se les asigna valores a cada posición de vector1 y vector2 y se inicializa las variables **i** y **tamano**.

Mientras i <= tamano Hacer: Este bucle mientras se ejecuta mientras i sea menor o igual a tamano.

resta_vectores[i] <- vector1[i] - vector2[i]: Resta el elemento i de vector2 al elemento i de vector1 y almacena el resultado en resta_vectores.

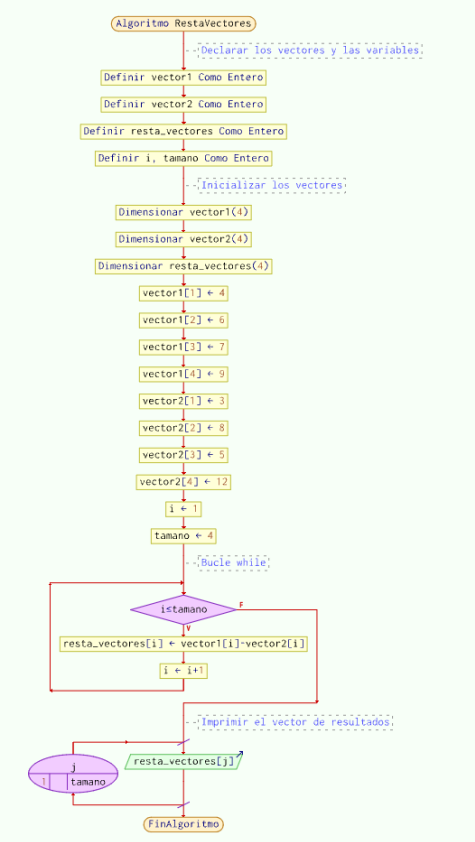
i <- i + 1: Incrementa el valor de i en 1 para pasar al siguiente par de elementos.

Para j <- 1 Hasta tamano Hacer: Este bucle Para recorre el vector resta_vectores desde el primer índice hasta el último.

Escribir resta_vectores[j]: Imprime el valor del elemento en la posición j del vector resta_vectores.

ISBN: 978-631-6557-57-5

Diagrama de flujo



Implementación

En Python

```
vector1 = [4,6,7,9]
vector2 = [3,8,5,12]
resta_vectores = []
i = 0
while i < len(vector1):
    resta_vectores.append(vector1[i] - vector2[i])
    i += 1
print(resta_vectores)
```

```
[1, -2, 2, -3]
```

vector1 es una lista que contiene los elementos [4, 6, 7, 9].

vector2 es otra lista que contiene los elementos [3, 8, 5, 12].

resta_vectores es una lista vacía que se utilizará para almacenar los resultados de la resta de los vectores.

i es una variable inicializada en 0 que se utilizará como índice para recorrer los elementos de los vectores.

while i < len(vector1): Este bucle while se ejecuta mientras i sea menor que la

ISBN: 978-631-6557-57-5

longitud de vector1.

resta_vectores.**append**(vector1[i] - vector2[i]): En cada iteración, se calcula la resta del elemento i de vector1 menos el elemento i de vector2, y se añade el resultado a la lista resta_vectores.

i += 1: Incrementa el valor de i en 1 para pasar al siguiente par de elementos en los vectores.

print(resta_vectores): Imprime la lista resta_vectores, que contiene los resultados de la resta de los elementos correspondientes de vector1 y vector2.

En JavaScript

```
let vector1 = [4, 6, 7, 9];
let vector2 = [3, 8, 5, 12];
let resta_vectores = [];
let i = 0;

while (i < vector1.length) {
    resta_vectores.push(vector1[i] - vector2[i]);
    i++;
}

console.log(resta_vectores);
```

let vector1 = [4, 6, 7, 9];: Declara un array llamado vector1 que contiene los elementos [4, 6, 7, 9].

let vector2 = [3, 8, 5, 12];: Declara otro array llamado vector2 que contiene los elementos [3, 8, 5, 12].

let resta_vectores = [];: Declara un array vacío que se utilizará para almacenar los resultados de la resta de los vectores.

let i = 0;: Inicializa la variable i en 0, que se utilizará como índice para recorrer los elementos de los arrays.

while (i < vector1.length): Este bucle while se ejecuta mientras i sea menor que la longitud del vector1.

resta_vectores.push(vector1[i] - vector2[i]);: En cada iteración, se calcula la resta del elemento i de vector1 menos el elemento i de vector2, y se añade el resultado al array resta_vectores utilizando el método push.

i++;: Incrementa el valor de i en 1 para pasar al siguiente par de elementos en los arrays.

console.log(resta_vectores);: Imprime el array resta_vectores, que contiene los resultados de la resta de los elementos correspondientes de vector1 y vector2.

ISBN: 978-631-6557-57-5

En C++

```cpp
#include <iostream>
#include <vector>
using namespace std;
int main() {
    // Inicializar los vectores
    vector<int> vector1 = {4, 6, 7, 9};
    vector<int> vector2 = {3, 8, 5, 12};
    vector<int> resta_vectores;
    int i = 0;
    // Bucle while
    while (i < vector1.size()) {
        resta_vectores.push_back(vector1[i] - vector2[i]);
        i++;
    }
    // Imprimir el vector de resultados
    for (int j = 0; j < resta_vectores.size(); j++) {
        cout << resta_vectores[j] << " ";
    }
    cout << endl;
    return 0;
}
```

vector<int> vector1 = {4, 6, 7, 9};: Declara un vector de enteros llamado vector1 y lo inicializa con los valores {4, 6, 7, 9}.

vector<int> vector2 = {3, 8, 5, 12};: Declara otro vector de enteros llamado vector2 y lo inicializa con los valores {3, 8, 5, 12}.

vector<int> resta_vectores;: Declara un vector de enteros vacío llamado resta_vectores que se utilizará para almacenar los resultados de la resta de los vectores.

int i = 0;: Inicializa la variable i en 0, que se utilizará como índice para recorrer los elementos de los vectores.

while (i < vector1.size()): Este bucle while se ejecuta mientras i sea menor que la longitud de vector1.

resta_vectores.push_back(vector1[i] - vector2[i]);: En cada iteración, se calcula la resta del elemento i de vector1 menos el elemento i de vector2, y se añade el resultado al vector resta_vectores utilizando el método push_back.

i++;: Incrementa el valor de i en 1 para pasar al siguiente par de elementos en los vectores.

Se utiliza un bucle for para recorrer el vector resta_vectores y imprimir cada uno de sus elementos.

cout << resta_vectores[j] << " ";: Imprime el elemento en la posición j seguido de un espacio.

cout << endl;: Imprime un salto de línea al final.

ISBN: 978-631-6557-57-5

En Java

```java
import java.util.ArrayList;

public class Main {
    public static void main(String[] args) {
        // Inicializar los vectores
        int[] vector1 = {4, 6, 7, 9};
        int[] vector2 = {3, 8, 5, 12};
        ArrayList<Integer> resta_vectores = new ArrayList<>();
        int i = 0;

        // Bucle while
        while (i < vector1.length) {
            resta_vectores.add(vector1[i] - vector2[i]);
            i++;
        }

        // Imprimir el vector de resultados
        System.out.println(resta_vectores);
    }
}
```

import java.util.ArrayList;: Importa la clase ArrayList de la biblioteca estándar de Java, que se utilizará para almacenar los resultados de la resta de los vectores.

int[] vector1 = {4, 6, 7, 9};: Declara un array de enteros llamado vector1 y lo inicializa con los valores {4, 6, 7, 9}.

int[] vector2 = {3, 8, 5, 12};: Declara otro array de enteros llamado vector2 y lo inicializa con los valores {3, 8, 5, 12}.

ArrayList<Integer> resta_vectores = new ArrayList<>();: Declara un ArrayList de enteros vacío llamado resta_vectores que se utilizará para almacenar los resultados de la resta de los vectores.

int i = 0;: Inicializa la variable i en 0, que se utilizará como índice para recorrer los elementos de los arrays.

while (i < vector1.length): Este bucle while se ejecuta mientras i sea menor que la longitud de vector1.

resta_vectores.add(vector1[i] - vector2[i]);: En cada iteración, se calcula la resta del elemento i de vector1 menos el elemento i de vector2, y se añade el resultado al ArrayList resta_vectores utilizando el método add.

i++;: Incrementa el valor de i en 1 para pasar al siguiente par de elementos en los arrays.

13) Sumar los elementos de la diagonal de la siguiente matriz

$$\begin{bmatrix} 11 & 12 & 13 \\ 14 & 15 & 16 \\ 17 & 18 & 19 \end{bmatrix}$$

ISBN: 978-631-6557-57-5

Pseudocódigo

```
Proceso SumaDiagonalPrincipal
    Definir matriz Como Entero
    Definir suma_diagonal, i Como Entero

    // Inicializar la matriz de 3x3
    Dimension matriz[3, 3]

    // Asignar valores a la matriz
    matriz[1,1] ← 11
    matriz[1,2] ← 12
    matriz[1,3] ← 13
    matriz[2,1] ← 14
    matriz[2,2] ← 15
    matriz[2,3] ← 16
    matriz[3,1] ← 17
    matriz[3,2] ← 18
    matriz[3,3] ← 19

    // Inicializar la suma de la diagonal en 0
    suma_diagonal ← 0

    // Bucle para recorrer la diagonal principal
    Para i ← 1 Hasta 3 Con Paso 1 Hacer
        suma_diagonal ← suma_diagonal + matriz[i, i]
    FinPara

    // Imprimir el resultado
    Escribir suma_diagonal
FinProceso
```

Se define un proceso llamado SumaDiagonalPrincipal. Dentro del proceso, se declaran las variables **matriz**, **suma_diagonal** e i. La variable matriz es un arreglo (matriz bidimensional), mientras que suma_diagonal e i son enteros.

Dimension matriz[3, 3]: Esto declara una matriz de 3x3.

Luego, se asignan valores a cada posición de la matriz. En PSeInt, los índices de los arreglos comienzan en 1, por lo que la primera posición es matriz[1,1].

Se inicializa la variable **suma_diagonal** en 0. Esta variable se usará para acumular la suma de los elementos de la diagonal principal.

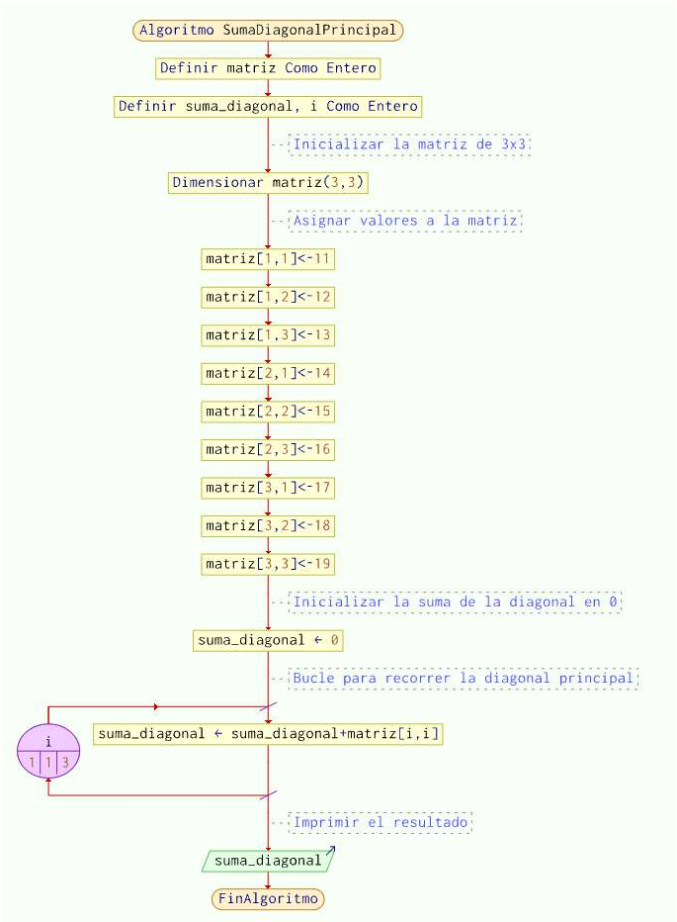
Para i <- 1 Hasta 3 Con Paso 1 Hacer: Este bucle Para recorre los índices de la matriz desde 1 hasta 3.

suma_diagonal <- suma_diagonal + matriz[i, i]: En cada iteración, se suma el valor del elemento en la posición [i, i] de la matriz a suma_diagonal. Esto acumula la suma de los elementos de la diagonal principal.

Se imprime el valor de **suma_diagonal**, que es la suma de los elementos de la diagonal principal de la matriz.

Diagrama de flujo

ISBN: 978-631-6557-57-5



Implementación

En Python

```
matriz = [
    [11, 12, 13],
    [14, 15, 16],
    [17, 18, 19]
]
suma_diagonal = 0
for i in range(len(matriz)):
    suma_diagonal += matriz[i][i]
print(suma_diagonal)
```

```
45
```

Se define una matriz de 3x3, que es una lista de listas, cada sublista representa una fila de la matriz.

Se inicializa la variable **suma_diagonal** en 0. Esta variable se utilizará para acumular la suma de los elementos de la diagonal principal.

Se emplea un bucle for para iterar sobre los índices de las filas de la matriz. range(len(matriz)) genera una secuencia de números desde 0 hasta el número de

ISBN: 978-631-6557-57-5

filas menos uno (en este caso, 0, 1 y 2).

En cada iteración, **i** toma un valor diferente (0, 1 y 2).

Mediante la **matriz**[**i**][**i**] accede al elemento en la posición [i][i] de la matriz, es decir, los elementos de la diagonal principal:

Cuando i = 0, matriz[0][0] es 11.

Cuando i = 1, matriz[1][1] es 15.

Cuando i = 2, matriz[2][2] es 19.

La suma_diagonal += matriz[i][i] agrega cada uno de estos elementos a suma_diagonal. Esta expresión se puede reemplazar por suma_diagonal = suma_diagonal + matriz[i][i].

Al final, se imprime el valor de suma_diagonal, que es la suma de los elementos de la diagonal principal.

En JavaScript

```
let matriz = [
    [11, 12, 13],
    [14, 15, 16],
    [17, 18, 19]
];

let suma_diagonal = 0;
for (let i = 0; i < matriz.length; i++) {
    suma_diagonal += matriz[i][i];
}

console.log(suma_diagonal);
```

Se define una matriz de 3x3 como una lista de listas (arreglo de arreglos). Cada sublista representa una fila de la matriz.

Se inicializa la variable **suma_diagonal** en 0. Esta variable se utilizará para acumular la suma de los elementos de la diagonal principal.

Se usa un bucle for para iterar sobre los índices de las filas de la matriz.

let i = 0; i < matriz.length; i++ define el inicio del bucle en i = 0 y continúa mientras i sea menor que la longitud de la matriz. En cada iteración, i se incrementa en 1.

En cada iteración, **i** toma un valor diferente (0, 1 y 2).

matriz[i][i] accede al elemento en la posición [i][i] de la matriz, es decir, los elementos de la diagonal principal:

Cuando i = 0, matriz[0][0] es 11.

Cuando i = 1, matriz[1][1] es 15.

ISBN: 978-631-6557-57-5

Cuando i = 2, matriz[2][2] es 19.

suma_diagonal += matriz[i][i] agrega cada uno de estos elementos a suma_diagonal.

Mediante console.log(suma_diagonal) se imprime el valor de suma_diagonal, que es la suma de los elementos de la diagonal principal.

En C++

```cpp
#include <iostream>
using namespace std;

int main() {
    // Inicializar la matriz de 3x3
    int matriz[3][3] = {
        {11, 12, 13},
        {14, 15, 16},
        {17, 18, 19}
    };
    // Inicializar la suma de la diagonal en 0
    int suma_diagonal = 0;

    // Bucle para recorrer la diagonal principal
    for (int i = 0; i < 3; i++) {
        suma_diagonal += matriz[i][i];
    }
    // Imprimir el resultado
    cout << suma_diagonal << endl;
    return 0;
}
```

#include <iostream>: Esta línea incluye la biblioteca estándar de entrada y salida en C++, que permite usar cout y cin (Leroy, 2009).

using namespace std;: Esta línea permite usar el espacio de nombres estándar de C++ para evitar escribir std:: antes de cout y cin.

int matriz[3][3]: Declara una matriz de 3x3 de tipo entero. Los valores se inicializan directamente en la declaración de la matriz.

Se declara e inicializa la variable suma_diagonal en 0. Esta variable se usará para acumular la suma de los elementos de la diagonal principal.

for (int i = 0; i < 3; i++): Este bucle for recorre los índices de la matriz desde 0 hasta 2.

suma_diagonal += matriz[i][i]: En cada iteración, se suma el valor del elemento en la posición [i][i] de la matriz a suma_diagonal. Esto acumula la suma de los elementos de la diagonal principal.

cout << suma_diagonal << endl;: Esta línea imprime el valor de **suma_diagonal**, que es la suma de los elementos de la diagonal principal, seguido de un salto de

ISBN: 978-631-6557-57-5

línea.

return 0;: Esta línea indica que el programa ha terminado correctamente.

En Java

```java
public class SumaDiagonalPrincipal {
    public static void main(String[] args) {
        // Inicializar la matriz de 3x3
        int[][] matriz = {
            {11, 12, 13},
            {14, 15, 16},
            {17, 18, 19}
        };

        // Inicializar la suma de la diagonal en 0
        int sumaDiagonal = 0;

        // Bucle para recorrer la diagonal principal
        for (int i = 0; i < matriz.length; i++) {
            sumaDiagonal += matriz[i][i];
        }

        // Imprimir el resultado
        System.out.println(sumaDiagonal);
    }
}
```

public class SumaDiagonalPrincipal: Define una clase pública llamada SumaDiagonalPrincipal.

public static void main(String[] args): Define el método principal main, que es el punto de entrada del programa en Java.

int[][] matriz: Declara una matriz de enteros de 3x3. Los valores se inicializan directamente en la declaración de la matriz.

Se declara e inicializa la variable **sumaDiagonal** en 0. Esta variable se usará para acumular la suma de los elementos de la diagonal principal.

for (int i = 0; i < matriz.length; i++): Este bucle for recorre los **índices** de la matriz desde 0 hasta el tamaño de la matriz menos uno. matriz.length devuelve el número de filas de la matriz.

sumaDiagonal += matriz[i][i]: En cada iteración, se suma el valor del elemento en la posición [i][i] de la matriz a sumaDiagonal. Esto acumula la suma de los elementos de la diagonal principal.

System.out.println(sumaDiagonal);: Esta línea imprime el valor de sumaDiagonal, que es la suma de los elementos de la diagonal principal.

14) Sumar las matrices.

$$\begin{bmatrix} 11 & 12 & 13 \\ 14 & 15 & 16 \\ 17 & 18 & 19 \end{bmatrix} \begin{bmatrix} 2 & 3 & 4 \\ 5 & 6 & 7 \\ 9 & 10 & 11 \end{bmatrix}$$

ISBN: 978-631-6557-57-5

Pseudocódigo

```
Algoritmo SumaMatrices
    Dimension matriz1[3,3], matriz2[3,3], suma[3,3]

    // Inicializar matriz1
    matriz1[1,1] ← 11; matriz1[1,2] ← 12; matriz1[1,3] ← 13
    matriz1[2,1] ← 13; matriz1[2,2] ← 15; matriz1[2,3] ← 16
    matriz1[3,1] ← 17; matriz1[3,2] ← 18; matriz1[3,3] ← 19

    // Inicializar matriz2
    matriz2[1,1] ← 2; matriz2[1,2] ← 3; matriz2[1,3] ← 4
    matriz2[2,1] ← 5; matriz2[2,2] ← 6; matriz2[2,3] ← 7
    matriz2[3,1] ← 9; matriz2[3,2] ← 10; matriz2[3,3] ← 11

    // Sumar las matrices
    Para i ← 1 Hasta 3 Con Paso 1 Hacer
        Para j ← 1 Hasta 3 Con Paso 1 Hacer
            suma[i,j] ← matriz1[i,j] + matriz2[i,j]
        FinPara
    FinPara

    // Imprimir la matriz resultante
    Escribir "La suma de las matrices es:"
    Para i ← 1 Hasta 3 Con Paso 1 Hacer
        Para j ← 1 Hasta 3 Con Paso 1 Hacer
            Escribir Sin Saltar suma[i,j], " "
        FinPara
        Escribir ""
    FinPara
FinAlgoritmo
```

El algoritmo se llama **SumaMatrices**.

Se declaran tres matrices de 3x3: **matriz1**, **matriz2** y **suma**.

Se asignan valores a cada elemento de la **matriz1**. Por ejemplo, matriz1[1,1] <- 11, se asigna el valor 11 a la primera fila, primera columna.

Lo mismo se hace en el caso de la **matriz2**.

Se utilizan dos bucles anidados para recorrer cada elemento de las matrices. El bucle externo (**i**) recorre las filas y el bucle interno (**j**) recorre las columnas. Para cada posición [**i,j**], se suma el elemento correspondiente de matriz1 y matriz2, y se guarda en la misma posición de la matriz suma.

Se utiliza Escribir para mostrar el mensaje.

Se usa dos bucles anidados nuevamente para recorrer la matriz suma.

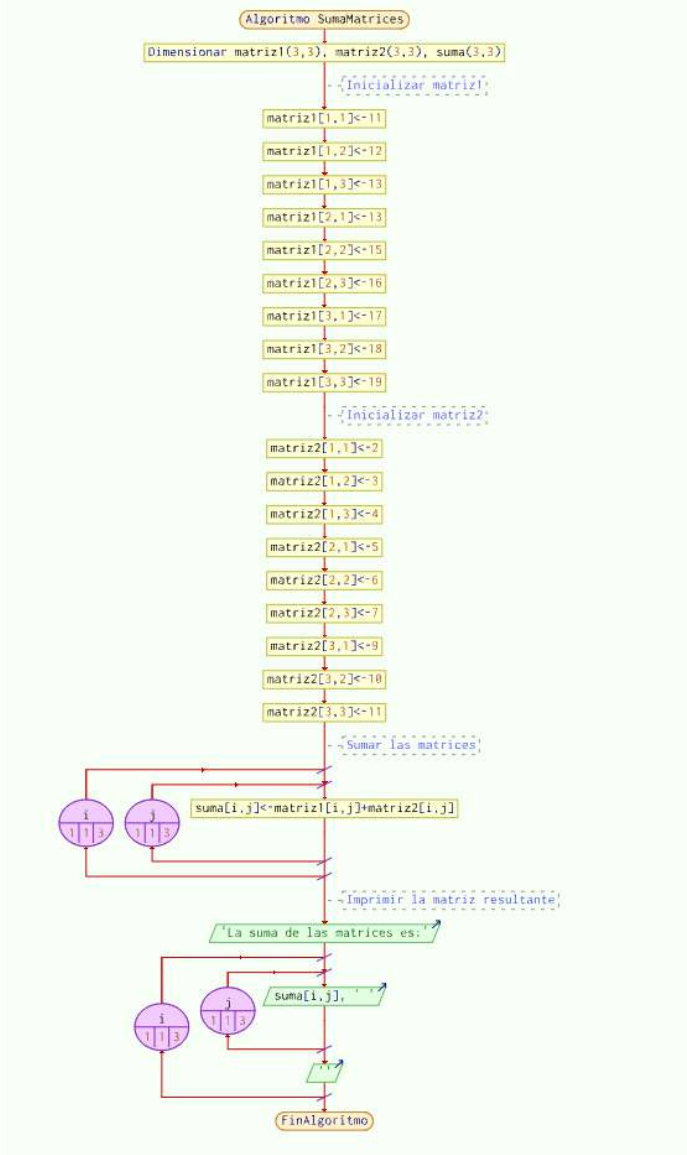
Escribir Sin Saltar" se usa para imprimir los elementos en la misma línea.

Se añade un espacio después de cada elemento para mejor legibilidad.

Después de cada fila (bucle interno), se usa Escribir “ ” vacío para saltar a la siguiente línea.

ISBN: 978-631-6557-57-5

Diagrama de flujo



Implementación

En Python

```
import numpy as np
matriz1 = np.array([[11, 12, 13], [13, 15,16], [17, 18, 19]])
matriz2 = np.array([[2, 3, 4], [5, 6, 7], [9, 10, 11]])

suma = matriz1 + matriz2
print(suma)
```

```
[[13 15 17]
 [18 21 23]
 [26 28 30]]
```

El código importa la biblioteca Numpy y define dos matrices matriz1 y matriz2,

ISBN: 978-631-6557-57-5

empleando **np.array** que se usa para crear crear arreglos Numpy (también conocidos como matrices). Luego, se suma las dos matrices elemento a elemento y se almacena el resultado, en suma. Finalmente, se imprime la matriz resultante. Se puede hacer el código sin utilizar la biblioteca Numpy.

```
# Definir las matrices como listas de listas
matriz1 = [[11, 12, 13], [13, 15, 16], [17, 18, 19]]
matriz2 = [[2, 3, 4], [5, 6, 7], [9, 10, 11]]

# Inicializar la matriz suma
suma = []

# Sumar las matrices
for i in range(len(matriz1)):
    fila_suma = []
    for j in range(len(matriz1[0])):
        fila_suma.append(matriz1[i][j] + matriz2[i][j])
    suma.append(fila_suma)

# Imprimir el resultado
print("La suma de las matrices es:")
for fila in suma:
    print(fila)
```

suma = []: inicializa una lista vacía llamada suma que almacenará el resultado de la suma de las matrices.

for i in range(len(matriz1)): Inicia un bucle que recorre las filas de la matriz. len(matriz1) da el número de filas.

fila_suma = []: Para cada fila, se crea una nueva lista vacía fila_suma que almacenará la suma de los elementos de esa fila.

for j in range(len(matriz1[0])):

Inicia un bucle interno que recorre las columnas de la matriz. len(matriz1[0]) da el número de columnas.

fila_suma.append(matriz1[i][j] + matriz2[i][j])

Suma los elementos correspondientes de matriz1 y matriz2, y añade el resultado a fila_suma.

suma.append(fila_suma)

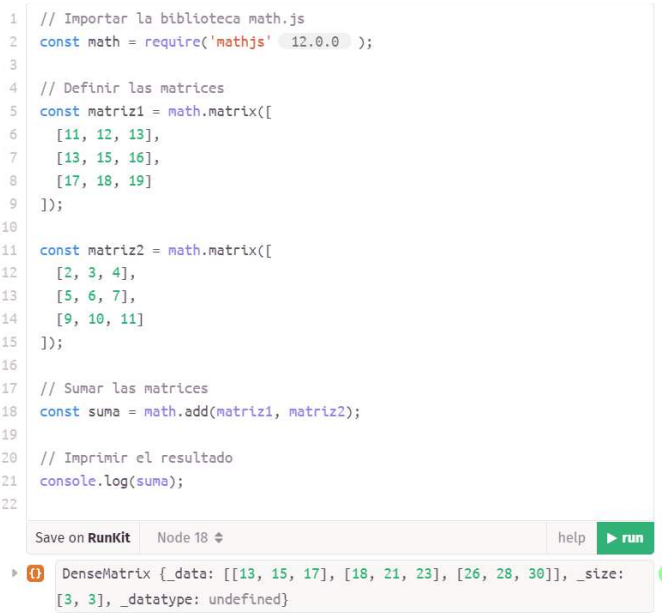
Después de procesar una fila completa, añade fila_suma a la matriz suma.

En JavaScript

Para resolver se va a utilizar RunKit (https://npm.runkit.com/mathjs), que ejecuta

ISBN: 978-631-6557-57-5

en línea un entorno Node.js completo y tiene instalado mathjs.



```
// Importar la biblioteca math.js
const math = require('mathjs' 12.0.0 );

// Definir las matrices
const matriz1 = math.matrix([
  [11, 12, 13],
  [13, 15, 16],
  [17, 18, 19]
]);

const matriz2 = math.matrix([
  [2, 3, 4],
  [5, 6, 7],
  [9, 10, 11]
]);

// Sumar las matrices
const suma = math.add(matriz1, matriz2);

// Imprimir el resultado
console.log(suma);

Save on RunKit   Node 18   help   ▶ run

DenseMatrix {_data: [[13, 15, 17], [18, 21, 23], [26, 28, 30]], _size:
[3, 3], _datatype: undefined}
```

const math = require('mathjs' 12.0.0);: Importa la biblioteca math.js versión 12.0.0 y la asigna a la constante math.

Define **matriz1** y **matriz2** usando la función math.matrix() de math.js, se crea matrices de 3x3.

const suma = math.add(matriz1, matriz2);: Se utiliza la función math.**add**() para sumar la matriz1 y la matriz2, asignando el resultado a **suma**.

console.log(suma);: Imprime el resultado de la suma en la consola.

También se puede hacer el código sin utilizar la biblioteca mathjs.

ISBN: 978-631-6557-57-5

```
 // Definir las matrices
const matriz1 = [
  [11, 12, 13],
  [13, 15, 16],
  [17, 18, 19]
];
const matriz2 = [
  [2, 3, 4],
  [5, 6, 7],
  [9, 10, 11]
];

// Inicializar la matriz resultado
let suma = [];

// Sumar las matrices directamente
for (let i = 0; i < matriz1.length; i++) {
  let fila = [];
  for (let j = 0; j < matriz1[i].length; j++) {
    fila.push(matriz1[i][j] + matriz2[i][j]);
  }
   suma.push(fila);
}

// Imprimir el resultado
console.log("Resultado de la suma de las matrices:");
for (let i = 0; i < suma.length; i++) {
  console.log(suma[i]);
}
```

Se define la matriz1 y matriz2 como arreglos bidimensionales de 3x3.

for (let i = 0; i < matriz1.length; i++) {: Inicia un bucle que recorre cada fila de las matrices, donde i representa el índice de la fila actual.

let fila = [];: Crea un array vacío fila para almacenar la suma de los elementos de la fila actual.

for (let j = 0; j < matriz1[i].length; j++) {: Inicia un bucle interno que recorre cada columna de la fila actual, donde j es el índice de la columna.

fila.push(matriz1[i][j] + matriz2[i][j]);: Suma los elementos correspondientes de matriz1 y matriz2 en la posición [i][j] y añade el resultado a fila.

suma.push(fila);: Cierra los bucles y añade la fila completa al array suma.

for (let i = 0; i < suma.length; i++): Recorre cada fila del array suma.

console.log(suma[i]);: Imprime cada fila en la consola.

ISBN: 978-631-6557-57-5

En C++

```cpp
#include <iostream>
using namespace std;
int main() {
    // Definir las matrices
    int matriz1[3][3] = {
        {11, 12, 13},
        {13, 15, 16},
        {17, 18, 19}
    };
    int matriz2[3][3] = {
        {2, 3, 4},
        {5, 6, 7},
        {9, 10, 11}
    };
    int suma[3][3];
    // Sumar las matrices
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            suma[i][j] = matriz1[i][j] + matriz2[i][j];
        }
    }
```

```cpp
    // Imprimir el resultado
    cout << "Resultado de la suma de las matrices:" << endl;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            cout << suma[i][j] << " ";
        }
        cout << endl;
    }
    return 0;
}
```

La **matriz1** y **matriz2** son matrices de 3x3 definidas como arreglos bidimensionales.

La matriz **suma** se utiliza para almacenar el resultado de la suma elemento a elemento de la matriz1 y matriz2.

Se utilizan dos bucles for anidados para iterar a través de cada elemento de las matrices, sumando los elementos correspondientes y almacenando el resultado en **suma**.

Se utiliza **cout** para imprimir cada elemento de la matriz **suma** en la consola, organizando la salida en forma de matriz.

ISBN: 978-631-6557-57-5

En Java

```java
import java.util.Arrays;

public class MatrixAddition {
    public static void main(String[] args) {
        // Definir las matrices
        int[][] matriz1 = {{11, 12, 13}, {13, 15, 16}, {17, 18, 19}};
        int[][] matriz2 = {{2, 3, 4}, {5, 6, 7}, {9, 10, 11}};

        // Crear una matriz para almacenar la suma
        int[][] suma = new int[3][3];

        // Realizar la suma de matrices
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                suma[i][j] = matriz1[i][j] + matriz2[i][j];
            }
        }

        // Imprimir el resultado
        for (int[] fila : suma) {
            System.out.println(Arrays.toString(fila));
        }
    }
}
```

int[][] suma = new int[3][3];: Se crea una nueva matriz para almacenar el resultado de la suma de la matriz1 y matriz2.

Se usa dos bucles anidados para recorrer cada elemento de las matrices y realizar la suma. El resultado se almacena en la matriz **suma**.

Se utiliza un bucle for-each para recorrer cada fila de la matriz suma y se imprime usando **Arrays.toString**(), que convierte cada fila (que es un array unidimensional) en una representación de String.

15) Realizar la multiplicación de las 2 matrices

$$\begin{bmatrix} 11 & 12 & 13 \\ 14 & 15 & 16 \\ 17 & 18 & 19 \end{bmatrix} \begin{bmatrix} 2 & 3 & 4 \\ 5 & 6 & 7 \\ 9 & 10 & 11 \end{bmatrix}$$

La multiplicación de 2 matrices es conocido como el producto punto, que como resultado da una nueva matriz, que le llamaremos C.

$$\begin{bmatrix} C11 & C12 & C13 \\ C21 & C22 & C23 \\ C31 & C32 & C33 \end{bmatrix}$$

Como ejemplo se calculará C11, C12, C13

C11 = Se multiplica la primera fila de la primera matriz por la primera columna de la segunda matriz y se suman los valores.

C11 = 11x2 + 12x5 +13x9 = 199

C12 = Se multiplica la primera fila de la primera matriz por la segunda columna de la segunda matriz y se suman los valores.

ISBN: 978-631-6557-57-5

C12 = 11x3 + 12x6 + 13x10 = 235

C13 = Se multiplica la primera fila de la primera matriz por la tercera columna de la segunda matriz y se suman los valores.

C13 = 11x4 + 12x7 + 13x11 = 271

De la misma forma se hace para encontrar los otros valores de la matriz resultante.

Pseudocódigo

```
Algoritmo MultiplicacionMatrices
    // Definición de matrices
    Dimension matriz1[3,3]
    Dimension matriz2[3,3]
    Dimension resultado[3,3]

    // Inicialización de matriz1
    // Inicialización de matriz1
    matriz1[1,1] ← 11
    matriz1[1,2] ← 12
    matriz1[1,3] ← 13
    matriz1[2,1] ← 14
    matriz1[2,2] ← 15
    matriz1[2,3] ← 16
    matriz1[3,1] ← 17
    matriz1[3,2] ← 18
    matriz1[3,3] ← 19

    // Inicialización de matriz2
    matriz2[1,1] ← 2
    matriz2[1,2] ← 3
    matriz2[1,3] ← 4
    matriz2[2,1] ← 5
    matriz2[2,2] ← 6
    matriz2[2,3] ← 7
    matriz2[3,1] ← 9
    matriz2[3,2] ← 10
    matriz2[3,3] ← 11
```

```
    matriz2[1,2] ← 3
    matriz2[1,3] ← 4
    matriz2[2,1] ← 5
    matriz2[2,2] ← 6
    matriz2[2,3] ← 7
    matriz2[3,1] ← 9
    matriz2[3,2] ← 10
    matriz2[3,3] ← 11

    // Multiplicación de matrices
    // Multiplicación de matrices
    Para i ← 1 Hasta 3 Con Paso 1 Hacer
        Para j ← 1 Hasta 3 Con Paso 1 Hacer
            resultado[i,j] ← 0
            Para k ← 1 Hasta 3 Con Paso 1 Hacer
                resultado[i,j] ← resultado[i,j] + matriz1[i,k] * matriz2[k,j]
            FinPara
        FinPara
    FinPara

    // Mostrar resultado
    Escribir "Resultado de la multiplicación:"
    Para i ← 1 Hasta 3 Con Paso 1 Hacer
        Para j ← 1 Hasta 3 Con Paso 1 Hacer
            Escribir Sin Saltar resultado[i,j], " "
        FinPara
        Escribir ""
    FinPara
FinAlgoritmo
```

Dimension : se utiliza para declarar el tamaño de las matrices (Matriz1, Matriz2, Resultado).

ISBN: 978-631-6557-57-5

matriz1[i,j] <- valor: Inicializa cada elemento de la matriz1 con los valores dados.

matriz2[i,j] <- valor: Inicializa cada elemento de la matriz2 con los valores dados.

Para **i** <- 1 Hasta 3 Con Paso 1 Hacer: Inicia un bucle para las filas de la matriz resultante.

Para **j** <- 1 Hasta 3 Con Paso 1 Hacer: Inicia un bucle para las columnas de la matriz resultante.

resultado[i,j] <- 0: Inicializa el elemento actual de la matriz resultado a 0.

Para **k** <- 1 Hasta 3 Con Paso 1 Hacer: Inicia un bucle para realizar la suma de productos.

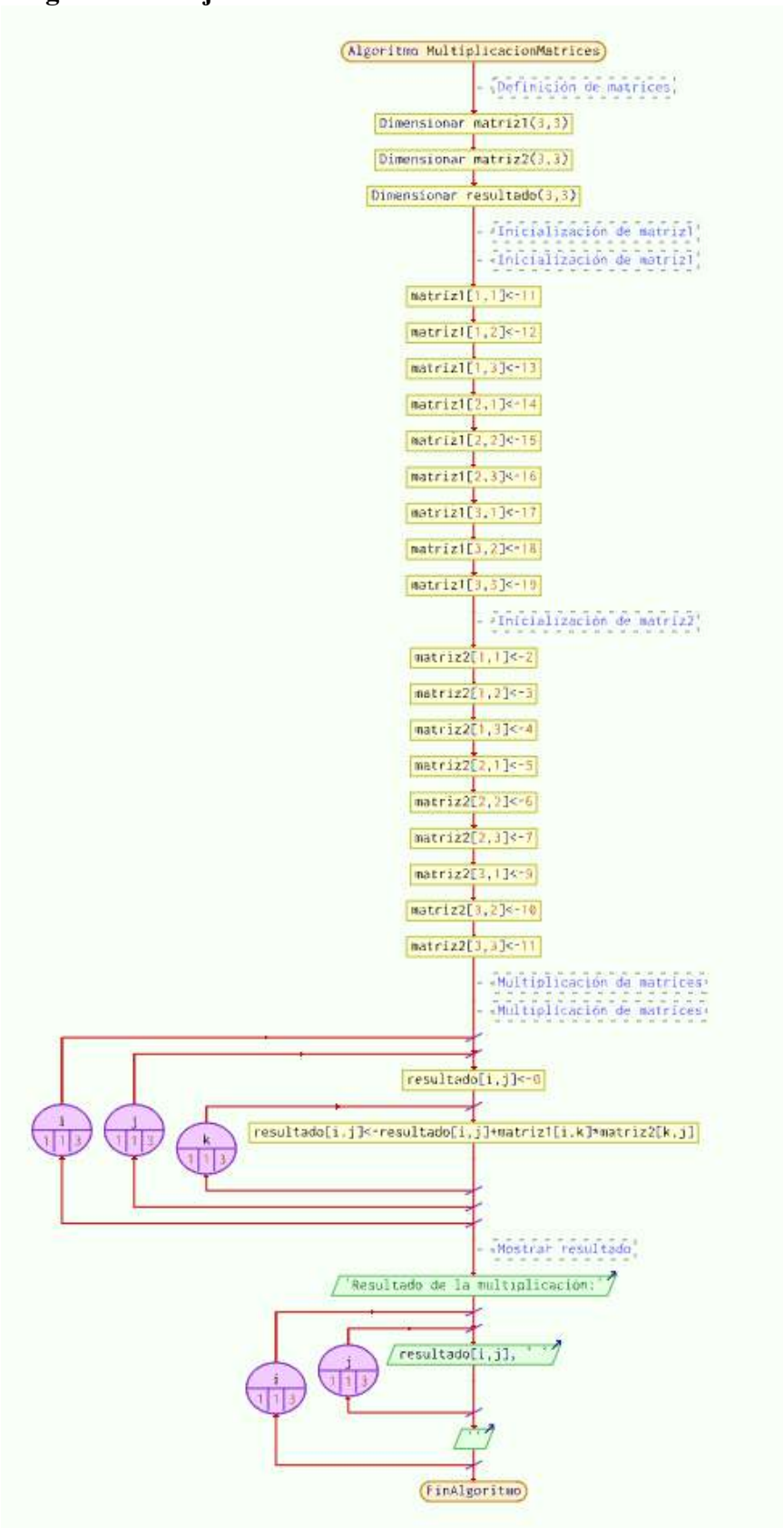
resultado[i,j] <- resultado[i,j] + matriz1[i,k] * matriz2[k,j]: Calcula el producto escalar de la fila i de la matriz1 y la columna j de la matriz2.

Hay dos bucles anidados para recorrer la matriz **resultado**.

```
*** Ejecución Iniciada. ***
Resultado de la multiplicación:
199 235 271
247 292 337
295 349 403
*** Ejecución Finalizada. ***
```

ISBN: 978-631-6557-57-5

Diagrama de flujo



ISBN: 978-631-6557-57-5

Implementación

En Python

```
import numpy as np
matriz1 = np.array([[11, 12, 13], [14, 15, 16], [17, 18, 19]])
matriz2 = np.array([[2, 3, 4], [5, 6, 7], [9, 10, 11]])

multiplicacion = np.dot(matriz1, matriz2)
print(multiplicacion)
```

```
[[199 235 271]
 [247 292 337]
 [295 349 403]]
```

Se crea la **matriz1** y **matriz2** de 3x3 utilizando np.array.

Se utiliza la función **np.dot** para calcular el producto de las matrices (matriz1 y matriz2). Cada elemento de la fila i de la matriz1 se multiplica por el elemento correspondiente de la columna j de la matriz2, y luego se suman los productos obtenidos.

```
# Definir las matrices
matriz1 = [
    [11, 12, 13],
    [14, 15, 16],
    [17, 18, 19]
]

matriz2 = [
    [2, 3, 4],
    [5, 6, 7],
    [9, 10, 11]
]

# Inicializar la matriz de resultado con ceros
resultado = [
    [0, 0, 0],
    [0, 0, 0],
    [0, 0, 0]
]

# Realizar la multiplicación de matrices manualmente
for i in range(len(matriz1)):
    for j in range(len(matriz2[0])):
        for k in range(len(matriz2)):
            resultado[i][j] += matriz1[i][k] * matriz2[k][j]

# Imprimir el resultado
for fila in resultado:
    print(fila)
```

Resolución sin utilizar la librería numpy.

La **matriz** de **resultado** se inicializa con ceros porque durante el proceso de multiplicación de matrices, se acumulan sumas de productos parciales en cada posición de la matriz resultante. Si no se inicializa la matriz con ceros, esos valores acumulados no tendrían un punto de partida adecuado y podrían contener valores indeterminados (o "basura") de memoria. Al inicializar con ceros, se asegura que cada posición de la matriz resultante comienza en cero y recibe correctamente la suma de los productos de los elementos correspondientes de las matrices de entrada.

ISBN: 978-631-6557-57-5

Se utilizan tres bucles for anidados para realizar la multiplicación de matrices. El primer bucle recorre las filas de matriz1. El segundo bucle recorre las columnas de matriz2. El tercer bucle recorre las filas de matriz2 y realiza la multiplicación y suma de productos correspondientes.

Se utiliza un bucle for para imprimir cada fila de la matriz resultado.

En JavaScript

```
// Definir las matrices
const matriz1 = [
    [11, 12, 13],
    [14, 15, 16],
    [17, 18, 19]
];
const matriz2 = [
    [2, 3, 4],
    [5, 6, 7],
    [9, 10, 11]
];
// Inicializar la matriz de resultado con ceros
const resultado = [
    [0, 0, 0],
    [0, 0, 0],
    [0, 0, 0]
];
// Realizar la multiplicación de matrices manualmente
for (let i = 0; i < matriz1.length; i++) {
    for (let j = 0; j < matriz2[0].length; j++) {
        for (let k = 0; k < matriz2.length; k++) {
            resultado[i][j] += matriz1[i][k] * matriz2[k][j];
        }
    }
}
// Imprimir el resultado
console.log("Resultado de la multiplicación de matrices:");
for (let i = 0; i < resultado.length; i++) {
    console.log(resultado[i]);
}
```

const se utiliza para declarar una variable cuyo valor no puede ser reasignado.

for (let i = 0; i < matriz1.length; i++) {: Inicia el primer bucle que recorre las filas de la matriz1.

for (let j = 0; j < matriz2[0].length; j++) {: Inicia el segundo bucle que recorre las columnas de la matriz2.

for (let k = 0; k < matriz2.length; k++) {: Inicia el tercer bucle que se usa para la suma de productos.

resultado[i][j] += matriz1[i][k] * matriz2[k][j];: Realiza la multiplicación y suma para cada elemento de la matriz resultado.

ISBN: 978-631-6557-57-5

for (let i = 0; i < resultado.length; i++) {: Inicia un bucle para recorrer las filas de la matriz resultado.

console.log(resultado[i]);: Imprime cada fila de la matriz resultado en la consola.

Se puede usar la biblioteca.

```
 const math = require('mathjs' 12.0.0 );

// Definir las matrices
const matriz1 = [
    [11, 12, 13],
    [14, 15, 16],
    [17, 18, 19]
];
const matriz2 = [
    [2, 3, 4],
    [5, 6, 7],
    [9, 10, 11]
];

// Multiplicar las matrices
const resultado = math.multiply(matriz1, matriz2);

// Imprimir el resultado
console.log("Resultado de la multiplicación de matrices:");
console.log(resultado);
```

En C++

```
#include <iostream>
using namespace std;
int main() {
    // Definir las matrices
    int matriz1[3][3] = {
        {11, 12, 13},
        {14, 15, 16},
        {17, 18, 19}
    };
    int matriz2[3][3] = {
        {2, 3, 4},
        {5, 6, 7},
        {9, 10, 11}
    };
    // Inicializar la matriz de resultado con ceros
    int resultado[3][3] = {0};

    // Realizar la multiplicación de matrices manualmente
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            for (int k = 0; k < 3; k++) {
                resultado[i][j] += matriz1[i][k] * matriz2[k][j];
            }
        }
    }

    // Imprimir el resultado
    cout << "Resultado de la multiplicación de matrices:" << endl;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            cout << resultado[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}
```

resultado es una matriz de 3x3 inicializada con ceros. Esta matriz almacenará el resultado de la multiplicación.

Se utilizan tres bucles **for** anidados para realizar la multiplicación de matrices. El

ISBN: 978-631-6557-57-5

primer bucle recorre las filas de matriz1. El segundo bucle recorre las columnas de matriz2. El tercer bucle recorre las filas de matriz2 y realiza la multiplicación y suma de productos correspondientes.

resultado[i][j] += **matriz1**[i][k] * **matriz2**[k][j];: Se realiza la multiplicación y suma para cada elemento de la matriz resultado.

for (int i = 0; i < 3; i++) {: Inicia el bucle para imprimir las filas del resultado.

for (int j = 0; j < 3; j++) {: Inicia el bucle para imprimir las columnas del resultado.

cout << resultado[i][j] << " ";: Imprime cada elemento de la matriz resultado.

cout << endl;: Imprime un salto de línea al final de cada fila.

En Java

```java
public class MultiplicacionMatrices {
    public static void main(String[] args) {
        // Definir las matrices
        int[][] matriz1 = {
            {11, 12, 13},
            {14, 15, 16},
            {17, 18, 19}
        };

        int[][] matriz2 = {
            {2, 3, 4},
            {5, 6, 7},
            {9, 10, 11}
        };

        // Inicializar la matriz de resultado con ceros
        int[][] resultado = new int[3][3];

        // Realizar la multiplicación de matrices manualmente
        for (int i = 0; i < matriz1.length; i++) {
            for (int j = 0; j < matriz2[0].length; j++) {
                for (int k = 0; k < matriz2.length; k++) {
                    resultado[i][j] += matriz1[i][k] * matriz2[k][j];
                }
            }
        }

        // Imprimir el resultado
        System.out.println("Resultado de la multiplicación de matrices:");
        for (int i = 0; i < resultado.length; i++) {
            for (int j = 0; j < resultado[0].length; j++) {
                System.out.print(resultado[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

matriz1 y **matriz2** son matrices de 3x3 definidas como arreglos bidimensionales.

resultado es una matriz de 3x3 inicializada con ceros. Esta matriz almacenará el resultado

Se utilizan tres bucles for anidados para realizar la multiplicación de matrices. El primer bucle recorre las filas de la matriz1.El segundo bucle recorre las columnas de la matriz2. El tercer bucle recorre las filas de la matriz2 y realiza la multiplicación y suma de productos correspondientes.

Se utiliza System.out.println y System.out.print para imprimir cada elemento de la matriz resultado.

16) Calcular el determinante de la matriz 3x3.

ISBN: 978-631-6557-57-5

$$\begin{bmatrix} 11 & 12 & 13 \\ 14 & 15 & 16 \\ 17 & 18 & 19 \end{bmatrix}$$

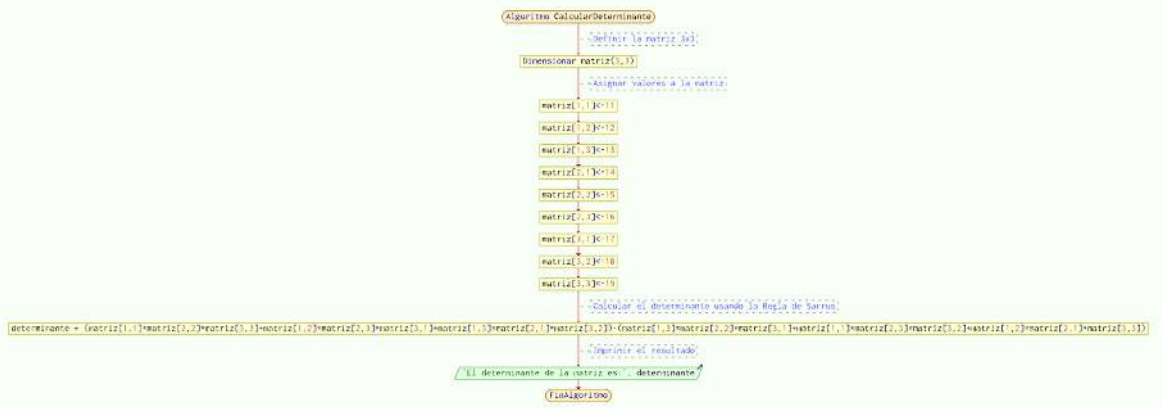
Pseudocódigo

```
Algoritmo CalcularDeterminante
    // Definir la matriz 3x3
    Dimension matriz[3, 3]

    // Asignar valores a la matriz
    matriz[1,1] ← 11; matriz[1,2] ← 12; matriz[1,3] ← 13
    matriz[2,1] ← 14; matriz[2,2] ← 15; matriz[2,3] ← 16
    matriz[3,1] ← 17; matriz[3,2] ← 18; matriz[3,3] ← 19

    // Calcular el determinante usando la Regla de Sarrus
    determinante ← (matriz[1,1] * matriz[2,2] * matriz[3,3] + matriz[1,2] * matriz[2,3]

    // Imprimir el resultado
    Escribir "El determinante de la matriz es:", determinante
FinAlgoritmo
```

En PSeInt el cálculo del determinante se escribe en una sola línea, razón por la que no aparece en su totalidad en la gráfica, a continuación, se escribe la totalidad de la línea.

determinante <- (matriz[1,1] * matriz[2,2] * matriz[3,3] + matriz[1,2] * matriz[2,3] * matriz[3,1] + matriz[1,3] * matriz[2,1] * matriz[3,2])-(matriz[1,3] * matriz[2,2] * matriz[3,1] + matriz[1,1] * matriz[2,3] * matriz[3,2] + matriz[1,2] * matriz[2,1] * matriz[3,3])

Diagrama de flujo



Implementación

En Python

```
import numpy as np
matriz = np.array([[11,12, 13], [14, 15, 16], [17, 18, 19]])

determinante = np.linalg.det(matriz)
print(determinante)
```

```
-2.131628207280298e-14
```

ISBN: 978-631-6557-57-5

Se utiliza la función np.**linalg.det**() para calcular el determinante de la matriz definida. El determinante obtenido es un número muy pequeño y cercano a cero, representado en notación científica. En el contexto de cálculos numéricos, esto indica que el determinante es esencialmente cero, por lo tanto, la matriz es singular, lo que significa que no tiene inversa (Harrison, 2023).

En JavaScript

```javascript
// Importar la librería math.js
const math = require('mathjs' 12.0.0 );

// Definir la matriz
const matriz = [
  [11, 12, 13],
  [14, 15, 16],
  [17, 18, 19]
];

// Calcular el determinante
const determinante = math.det(matriz);

// Imprimir el resultado
console.log(determinante);
```

Se usa la función **`math.`**`det()` de la librería `mathjs` para calcular el determinantde la matriz definida. La función toma la matriz como argumento y devuelve su determinante.

En C++

```cpp
#include <iostream>
using namespace std;
// Función para calcular el determinante de una matriz 3x3
double calcularDeterminante(double matriz[3][3]) {
    return matriz[0][0] * (matriz[1][1] * matriz[2][2] - matriz[1][2] *
        matriz[2][1]) -
           matriz[0][1] * (matriz[1][0] * matriz[2][2] - matriz[1][2] *
               matriz[2][0]) +
           matriz[0][2] * (matriz[1][0] * matriz[2][1] - matriz[1][1] *
               matriz[2][0]);
}
int main() {
    // Definir la matriz
    double matriz[3][3] = {
        {11, 12, 13},
        {14, 15, 16},
        {17, 18, 19}
    };
    // Calcular el determinante
    double determinante = calcularDeterminante(matriz);
    // Imprimir el resultado
    cout << "Determinante: " << determinante << endl;
    return 0;
}
```

ISBN: 978-631-6557-57-5

La función **CalcularDeterminante** calcula el determinante de una matriz de 3x3 utilizando la fórmula de determinante específica para matrices de 3x3.

En Java

```java
public class DeterminanteMatriz {

    // Función para calcular el determinante de una matriz 3x3
    public static double calcularDeterminante(double[][] matriz) {
        return matriz[0][0] * (matriz[1][1] * matriz[2][2] - matriz[1][2] * matriz[2][1]) -
               matriz[0][1] * (matriz[1][0] * matriz[2][2] - matriz[1][2] * matriz[2][0]) +
               matriz[0][2] * (matriz[1][0] * matriz[2][1] - matriz[1][1] * matriz[2][0]);
    }

    public static void main(String[] args) {
        // Definir la matriz
        double[][] matriz = {
            {11, 12, 13},
            {14, 15, 16},
            {17, 18, 19}
        };

        // Calcular el determinante
        double determinante = calcularDeterminante(matriz);

        // Imprimir el resultado
        System.out.println("Determinante: " + determinante);
    }
}

```

Se define una clase pública llamada **DeterminanteMatriz**, y se establece el método **calcularDeterminante**. Se utiliza la regla de Sarrus para implementar la fórmula del determinante.

17) Calcular la matriz inversa de la siguiente matriz

$$\begin{bmatrix} 21 & 22 \\ 23 & 24 \end{bmatrix}$$

Pseudocódigo

```
Algoritmo CalcularInversaMatriz2x2
    // Definir la matriz 2x2
    Dimension matriz[2, 2]
    Dimension inversa[2, 2]
    Definir determinante Como Real

    // Asignar valores a la matriz
    matriz[1,1] ← 21
    matriz[1,2] ← 22
    matriz[2,1] ← 23
    matriz[2,2] ← 24

    // Calcular el determinante de la matriz
    determinante ← (matriz[1,1] * matriz[2,2]) - (matriz[1,2] * matriz[2,1])

    // Verificar si el determinante es 0
    Si determinante = 0 Entonces
        Escribir "La matriz no tiene inversa porque su determinante es 0."
    FinSi

```

ISBN: 978-631-6557-57-5

```
// Calcular la inversa utilizando la fórmula para matrices 2x2
inversa[1,1] ← matriz[2,2] / determinante
inversa[1,2] ← -matriz[1,2] / determinante
inversa[2,1] ← -matriz[2,1] / determinante
inversa[2,2] ← matriz[1,1] / determinante

// Imprimir la matriz inversa
Escribir "La matriz inversa es:"
Escribir "[[", inversa[1,1], ", ", inversa[1,2], "],"
Escribir " [", inversa[2,1], ", ", inversa[2,2], "]]"
FinAlgoritmo

```

Se crea dos matrices de 2x2, una para la matriz original y otra para almacenar su inversa, luego se llena la matriz original con valores numéricos específicos (en este caso, 21, 22, 23 y 24), para a continuación calcular el determinante, utilizando la fórmula estándar para calcular el determinante de una matriz de 2x2.

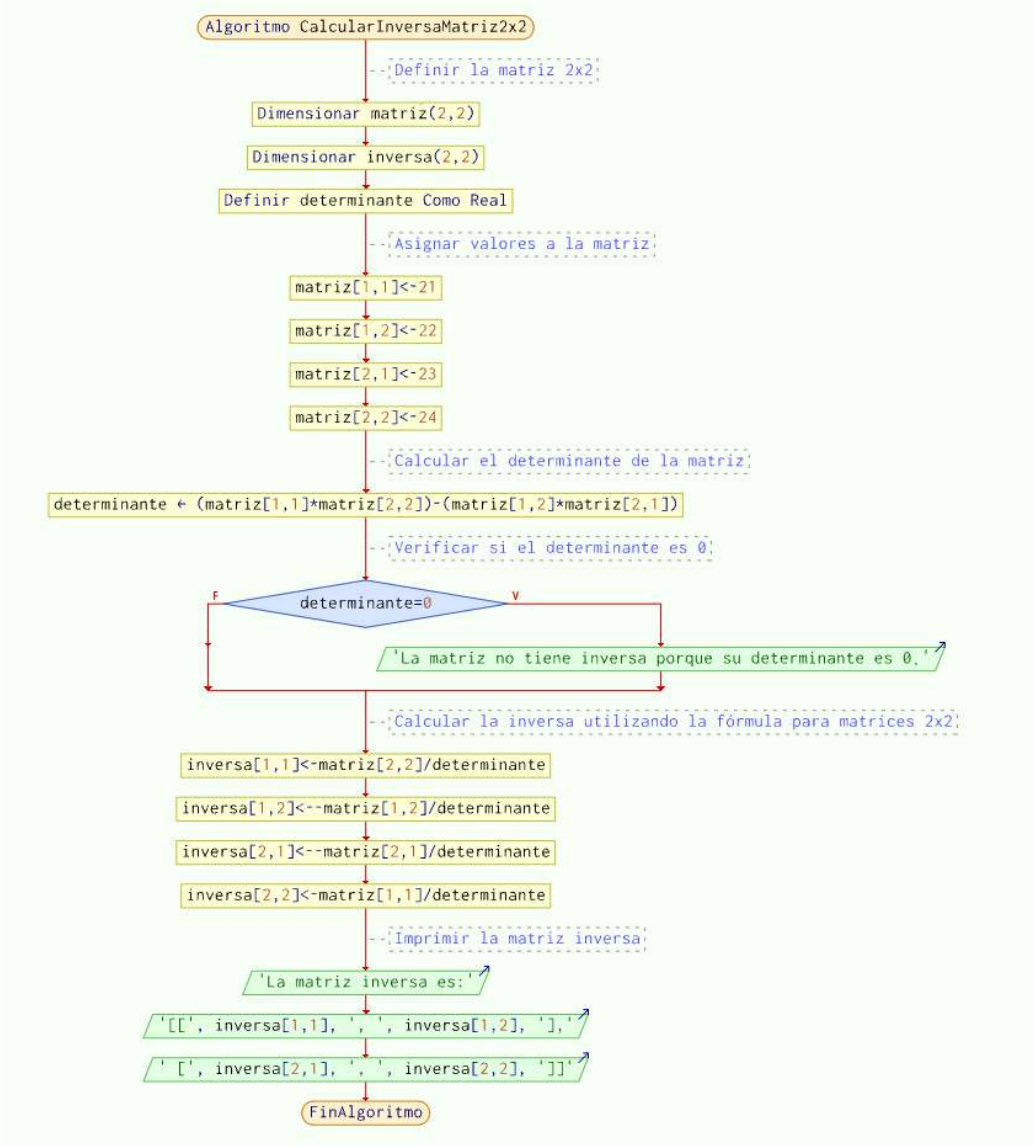
Se verifica si el determinante es cero, porque si es el caso la matriz no tiene inversa y el algoritmo termina.

Si el determinante es distinto de cero, se utiliza la fórmula específica para calcular la inversa de una matriz de 2x2 y se almacena los resultados en la matriz inversa.

Por ultimo se imprime la inversa en un formato legible.

ISBN: 978-631-6557-57-5

Diagrama de flujo



Implementación

En Python

```
import numpy as np
matriz = np.array([[21, 22], [23, 24]])

inversa = np.linalg.inv(matriz)
print(inversa)
```

```
[[-12.   11. ]
 [ 11.5 -10.5]]
```

Se crea una matriz de 2x2 y se calcula la inversa de esta matriz usando np.**linalg.inv**().

ISBN: 978-631-6557-57-5

En JavaScript

```javascript
// Importar la librería math.js
const math = require('mathjs' 12.0.0 );

// Definir la matriz
const matriz = [
  [21, 22],
  [23, 24]
];

// Calcular la inversa
const inversa = math.inv(matriz);

// Imprimir el resultado
console.log(inversa);
```

Se calcula la inversa de la matriz usando la función **inv()** de math.js y se asigna el resultado a la constante **inversa**.

En C++

```cpp
#include <iostream>
using namespace std;

// Función para calcular el determinante de una matriz 2x2
double calcularDeterminante(double matriz[2][2]) {
    return (matriz[0][0] * matriz[1][1]) - (matriz[0][1] * matriz[1][0]);
}

// Función para calcular la inversa de una matriz 2x2
void calcularInversa(double matriz[2][2], double inversa[2][2]) {
    double determinante = calcularDeterminante(matriz);

    if (determinante == 0) {
        cout << "La matriz no tiene inversa porque su determinante es 0." <<
            endl;
        return;
    }

    // Calcular la inversa utilizando la fórmula para matrices 2x2
    inversa[0][0] = matriz[1][1] / determinante;
    inversa[0][1] = -matriz[0][1] / determinante;
    inversa[1][0] = -matriz[1][0] / determinante;
    inversa[1][1] = matriz[0][0] / determinante;
}
```

```cpp
int main() {
    // Definir la matriz
    double matriz[2][2] = {
        {21, 22},
        {23, 24}
    };

    // Matriz para almacenar la inversa
    double inversa[2][2];

    // Calcular la inversa
    calcularInversa(matriz, inversa);

    // Imprimir la matriz inversa
    cout << "La matriz inversa es: " << endl;
    cout << "[[" << inversa[0][0] << ", " << inversa[0][1] << "], " << endl;
    cout << " [" << inversa[1][0] << ", " << inversa[1][1] << "]]" << endl;

    return 0;
}
```

ISBN: 978-631-6557-57-5

Se calcula el determinante para una matriz de 2 x 2, luego se calcula la inversa de la matriz, verificando primero si el determinante es cero (matriz no invertible) y luego aplicado la fórmula para la inversa de una matriz 2x2.

En Java

```java
public class InversaMatriz {

    // Función para calcular el determinante de una matriz 2x2
    public static double calcularDeterminante(double[][] matriz) {
        return (matriz[0][0] * matriz[1][1]) - (matriz[0][1] * matriz[1][0]);
    }

    // Función para calcular la inversa de una matriz 2x2
    public static double[][] calcularInversa(double[][] matriz) {
        double determinante = calcularDeterminante(matriz);

        if (determinante == 0) {
            throw new ArithmeticException("La matriz no tiene inversa porque su determinante es 0.");
        }

        double[][] inversa = new double[2][2];

        // Calcular la inversa utilizando la fórmula para matrices 2x2
        inversa[0][0] = matriz[1][1] / determinante;
        inversa[0][1] = -matriz[0][1] / determinante;
        inversa[1][0] = -matriz[1][0] / determinante;
        inversa[1][1] = matriz[0][0] / determinante;

        return inversa;
    }
```

```java
    public static void main(String[] args) {
        // Definir la matriz
        double[][] matriz = {
            {21, 22},
            {23, 24}
        };

        try {
            // Calcular la inversa
            double[][] inversa = calcularInversa(matriz);

            // Imprimir la matriz inversa
            System.out.println("La matriz inversa es:");
            System.out.println("[[" + inversa[0][0] + ", " + inversa[0][1] + "],");
            System.out.println(" [" + inversa[1][0] + ", " + inversa[1][1] + "]]");
        } catch (ArithmeticException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Se calcula el determinante de una matriz 2x2 usando la fórmula ad - bc. Antes de calcular la inversa, se revisa si el determinante es cero. Luego se crea una matriz de 2 x 2 para almacenar la inversa, para a continuación calcular los elementos de la matriz inversa usando las fórmulas para matrices 2x2.

Se define la matriz 2x2 con la cual se va hacer el cálculo y luego se procede a ensayar la inversa de la matriz.

En Java, el bloque catch se utiliza para manejar excepciones. Las excepciones son eventos que ocurren durante la ejecución de un programa y que pueden interrumpir el flujo normal de las instrucciones, como errores al intentar dividir por cero, intentar acceder a un índice fuera de los límites de un array, o intentar abrir un archivo que no existe.

ISBN: 978-631-6557-57-5

18) Multiplicar cada elemento de la matriz 2x2 por 10.

$$\begin{bmatrix} 21 & 22 \\ 23 & 24 \end{bmatrix}$$

Pseudocódigo

```
Algoritmo MultiplicarMatrizPor10
    // Definir la matriz 2x2
    Dimension matriz[2, 2]
    Dimension matrizResultado[2, 2]

    // Asignar valores a la matriz
    matriz[1,1] ← 21
    matriz[1,2] ← 22
    matriz[2,1] ← 23
    matriz[2,2] ← 24

    // Multiplicar cada elemento de la matriz por 10
    Para i ← 1 Hasta 2 Hacer
        Para j ← 1 Hasta 2 Hacer
            matrizResultado[i,j] ← matriz[i,j] * 10
        FinPara
    FinPara

    // Imprimir la matriz resultante
    Escribir "La matriz resultante es:"
    Escribir "[[", matrizResultado[1,1], ", ", matrizResultado[1,2], "],"
    Escribir " [", matrizResultado[2,1], ", ", matrizResultado[2,2], "]]"
FinAlgoritmo

```

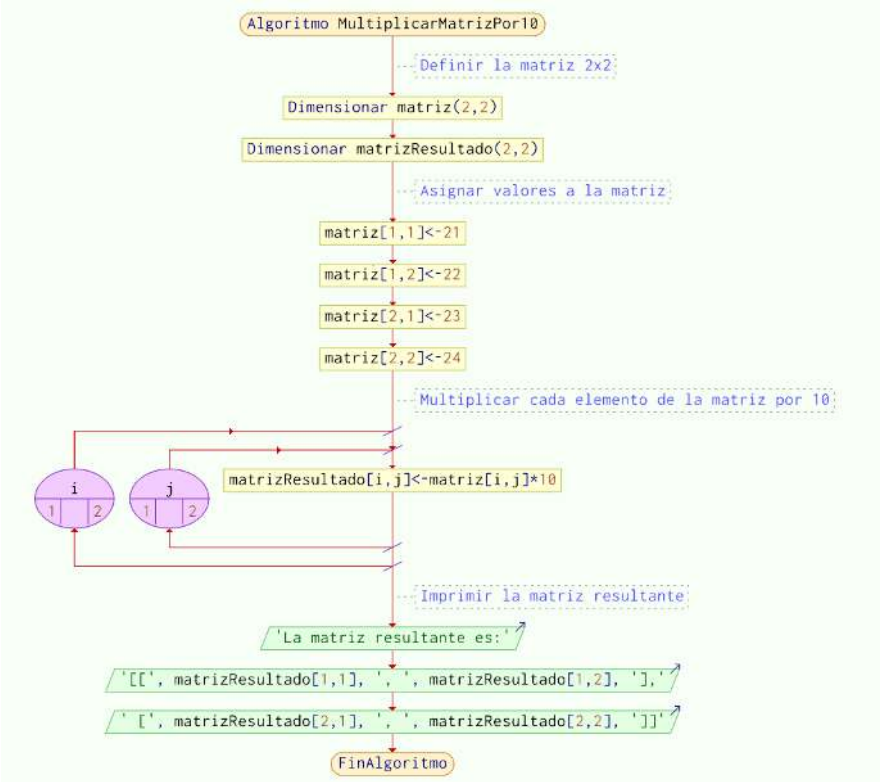
Se declara una matriz 2x2 para la matriz original y otra para almacenar el resultado de la multiplicación, luego se asigna los valores específicos a la matriz.

Se utiliza dos bucles Para anidados para recorrer cada elemento de la matriz y multiplicarlo por 10, almacenando el resultado en **matrizResultado**.

Por último, se imprime la matriz resultante.

ISBN: 978-631-6557-57-5

Diagrama de flujo



Implementación

En Python

```
import numpy as np

# Definir la matriz
matriz = np.array([[21, 22], [23, 24]])

# Multiplicar cada elemento de la matriz por 10
matriz_resultado = matriz * 10

# Imprimir la matriz resultante
print("La matriz resultante es:")
print(matriz_resultado)
```

```
La matriz resultante es:
[[210 220]
 [230 240]]
```

Se importa la biblioteca **NumPy**, que se utiliza para trabajar con matrices y realizar operaciones matemáticas, luego se define la matriz 2x2 utilizando un array de NumPy. Se multiplica cada elemento de la matriz por 10, NumPy permite realizar esta operación de manera vectorizada, lo que simplifica el código.

ISBN: 978-631-6557-57-5

En JavaScript

```javascript
// Definir la matriz
const matriz = [
  [21, 22],
  [23, 24]
];

// Definir la matriz resultante
const matrizResultado = [];

// Multiplicar cada elemento de la matriz por 10
for (let i = 0; i < matriz.length; i++) {
  matrizResultado[i] = [];
  for (let j = 0; j < matriz[i].length; j++) {
    matrizResultado[i][j] = matriz[i][j] * 10;
  }
}

// Imprimir la matriz resultante
console.log("La matriz resultante es:");
console.log(matrizResultado);
```

Se declara una constante llamada **matriz** y se le asigna un arreglo bidimensional (matriz), que contiene dos filas y dos columnas, con los valores 21, 22, 23 y 24 en las posiciones correspondientes.

Se declara otra constante llamada **matrizResultado** y se inicializa como un arreglo vacío, esta matriz se utilizará para almacenar los resultados de las operaciones.

Se inicia un bucle for que iterará sobre las filas de la matriz denominada **matriz**, donde **i** es un contador que se utiliza para acceder a cada fila, mediante matriz.length se representa el número total de filas en la matriz.

En cada iteración del bucle externo, se crea una nueva fila en **matrizResultado** para almacenar los resultados de la fila correspondiente en **matriz**.

Se inicia un bucle for anidado interno que iterará sobre las columnas de la fila actual, donde **j** es un contador que se utiliza para acceder a cada columna y mediante matriz[**i**].**length** se representa el número de columnas en la fila actual de **matriz**.

Se multiplica el elemento en la posición [**i**][**j**] de la matriz original por 10 y se asigna el resultado a la misma posición en la matriz **matrizResultado**.

ISBN: 978-631-6557-57-5

En C++

```cpp
#include <iostream>
using namespace std;
int main() {
    // Definir la matriz 2x2
    int matriz[2][2] = {
        {21, 22},
        {23, 24}
    };
    // Definir la matriz resultante
    int matrizResultado[2][2];

    // Multiplicar cada elemento de la matriz por 10
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 2; j++) {
            matrizResultado[i][j] = matriz[i][j] * 10;
        }
    }
    // Imprimir la matriz resultante
    cout << "La matriz resultante es:" << endl;
    cout << "[[" << matrizResultado[0][0] << ", " << matrizResultado[0][1] <<
        "], " << endl;
    cout << " [" << matrizResultado[1][0] << ", " << matrizResultado[1][1] <<
        "]]" << endl;
    return 0;
}
```

int matriz[2][2] declara una matriz de enteros llamada matriz con 2 filas y 2 columnas, se inicializa cada elemento de la matriz con los valores {{21, 22}, {23, 24}}.

Se declara una nueva matriz de enteros llamada **matrizResultado** con las mismas dimensiones que matriz para almacenar el resultado de la operación.

for (int i=0; i<2; i++): Este bucle externo itera sobre las filas de la matriz.

for (int j = 0; j < 2; j++): Este bucle interno itera sobre las columnas de la matriz.

matrizResultado[**i**][**j**] = matriz[**i**][**j**] * 10;: En cada iteración, se multiplica el elemento correspondiente de matriz por 10 y se almacena el resultado en la misma posición de matrizResultado.

ISBN: 978-631-6557-57-5

En Java

```java
public class MultiplicarMatriz {

    public static void main(String[] args) {
        // Definir la matriz 2x2
        int[][] matriz = {
            {21, 22},
            {23, 24}
        };

        // Definir la matriz resultante
        int[][] matrizResultado = new int[2][2];

        // Multiplicar cada elemento de la matriz por 10
        for (int i = 0; i < 2; i++) {
            for (int j = 0; j < 2; j++) {
                matrizResultado[i][j] = matriz[i][j] * 10;
            }
        }

        // Imprimir la matriz resultante
        System.out.println("La matriz resultante es:");
        System.out.println("[[" + matrizResultado[0][0] + ", " + matrizResultado[0][1] + "],");
        System.out.println(" [" + matrizResultado[1][0] + ", " + matrizResultado[1][1] + "]]");
    }
}
```

Se declara una matriz de enteros bidimensional llamada **matriz**. Esta matriz tiene 2 filas y 2 columnas y se inicializa la con los valores dados: 21, 22, 23 y 24.

Se declara otra matriz bidimensional llamada **matrizResultado** del mismo tamaño que **matriz**. Esta matriz se utilizará para almacenar el resultado de la multiplicación.

Se utilizan dos bucles anidados para recorrer todos los elementos de la matriz **matriz**. La variable i controla el índice de las filas y la variable j controla el índice de las columnas. En cada iteración, se multiplica el elemento correspondiente de **matriz** por 10 y se almacena el resultado en la misma posición de **matrizResultado**.

Se utiliza System.out.println para imprimir en la consola la matriz **matrizResultado** en un formato legible. Cada línea de la matriz se imprime por separado, concatenando los elementos con comas y encerrando cada fila entre corchetes.

19) Crear una matriz de 3x3 de unos.

ISBN: 978-631-6557-57-5

Pseudocódigo

```
Proceso CrearMatrizDeUnos
    // Definir el tamaño de la matriz
    Definir filas, columnas Como Entero
    filas ← 3
    columnas ← 3

    // Definir la matriz
    Dimension matrizUnos[filas, columnas]

    // Llenar la matriz con unos
    Para i ← 1 Hasta filas Hacer
        Para j ← 1 Hasta columnas Hacer
            matrizUnos[i, j] ← 1
        FinPara
    FinPara

    // Imprimir la matriz
    Para i ← 1 Hasta filas Hacer
        Para j ← 1 Hasta columnas Hacer
            Escribir Sin Saltar matrizUnos[i, j], " "
        FinPara
        Escribir ""
    FinPara
FinProceso
```

Se declaran dos variables de tipo entero, **filas** y **columnas**, que almacenarán las dimensiones de la matriz 3x3.

Se declara una matriz bidimensional llamada **matrizunos** con las dimensiones especificadas por las variables filas y columnas. Esta matriz almacenará los valores numéricos.

Se inicia un bucle Para que iterará desde 1 hasta el valor de **filas** (en este caso, 3). La variable i se utilizará como índice para recorrer las filas de la matriz.

Dentro del bucle externo, se inicia otro bucle Para que iterará desde 1 hasta el valor de **columnas** (también 3). La variable j se utilizará como índice para recorrer las columnas de la matriz.

En cada iteración de los bucles, se asigna el valor 1 a la posición [i, j] de la matriz **matrizunos**, esto llena la matriz con unos.

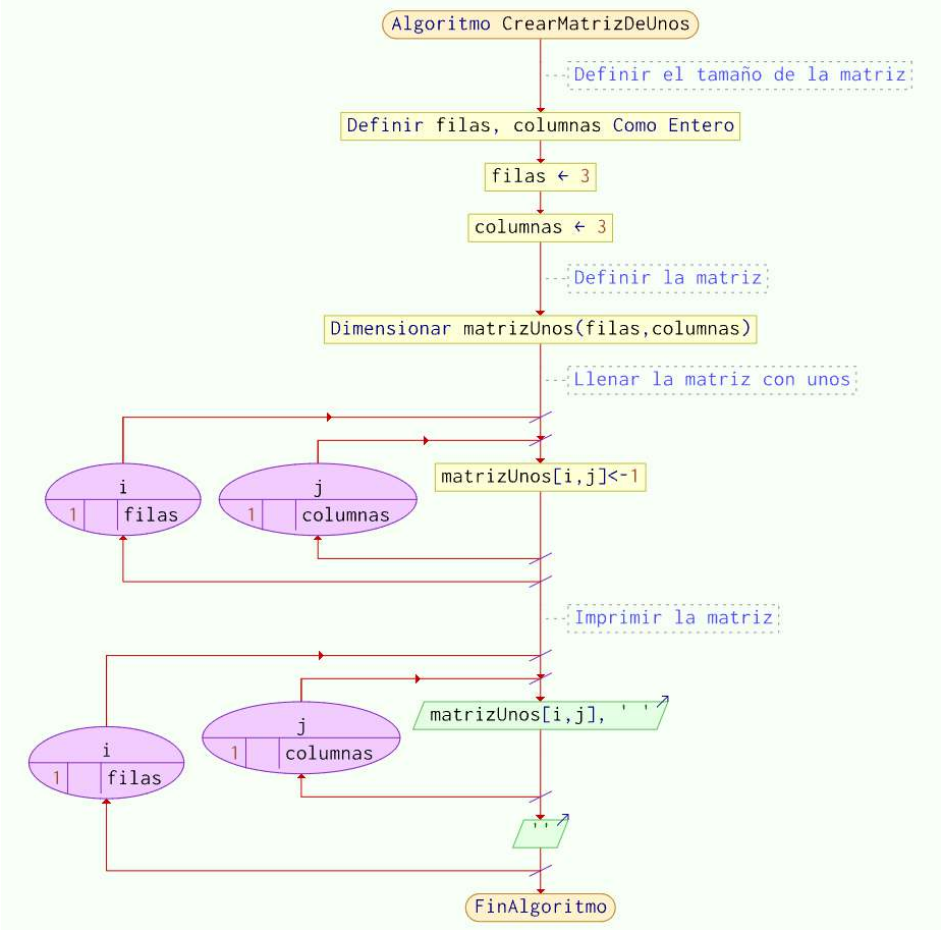
Para imprimir se inicia un nuevo bucle externo Para para recorrer las filas de la matriz nuevamente.

Dentro del bucle externo, se inicia otro bucle Para para recorrer las columnas.

Se imprime el valor almacenado en la posición [i, j] de la matriz, seguido de un espacio, sin saltar a una nueva línea. Esto permite imprimir los elementos de la matriz en una misma línea.

ISBN: 978-631-6557-57-5

Diagrama de flujo



Implementación

En Python

```
import numpy as np
matriz_unos = np.ones((3, 3))
print(matriz_unos)
```

```
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

Se crea una matriz de unos utilizando la función np.ones.

ISBN: 978-631-6557-57-5

En JavaScript

```javascript
// Crear una matriz de 3x3 llena de unos
const filas = 3;
const columnas = 3;
const matrizUnos = [];

// Llenar la matriz con unos
for (let i = 0; i < filas; i++) {
  matrizUnos[i] = [];
  for (let j = 0; j < columnas; j++) {
    matrizUnos[i][j] = 1;
  }
}

// Imprimir la matriz
console.log(matrizUnos);
```

Se declara una constante llamada **filas** y se le asigna el valor 3, esta constante representará el número de filas de la matriz.

Se declara una constante **columnas** con el valor **3**, representando el número de columnas de la matriz.

Se declara un arreglo vacío llamado **matrizUnos**, este arreglo se utilizará para almacenar la matriz de unos que se va a crear.

for (let i = 0; i < filas; i++): Este bucle iterará sobre las filas de la matriz. La variable `i` se utilizará como índice de fila, comenzando desde 0 hasta filas - 1.

matrizUnos[i] = [];: En cada iteración, se crea una nueva fila en la matriz matrizUnos. Esto es necesario porque los arreglos en JavaScript son dinámicos y se puede agregar elementos sobre la marcha.

for (let j = 0; j < columnas; j++): Este bucle anidado iterará sobre las columnas de cada fila. La variable j se utilizará como índice de columna.

matrizUnos[i][j] = 1;: En cada iteración, se asigna el valor 1 a la posición correspondiente de la matriz matrizUnos, esto hace que la matriz se llene con unos.

ISBN: 978-631-6557-57-5

En C++

```cpp
#include <iostream>
using namespace std;
int main() {
    // Definir el tamaño de la matriz
    const int filas = 3;
    const int columnas = 3;
    // Definir la matriz
    int matrizUnos[filas][columnas];

    // Llenar la matriz con unos
    for (int i = 0; i < filas; i++) {
        for (int j = 0; j < columnas; j++) {
            matrizUnos[i][j] = 1;
        }
    }
    // Imprimir la matriz
    cout << "La matriz resultante es:" << endl;
    for (int i = 0; i < filas; i++) {
        for (int j = 0; j < columnas; j++) {
            cout << matrizUnos[i][j] << " ";
        }
        cout << endl;
    }
    return 0;
}
```

Se declaran dos constantes enteras, **filas** y **columnas**, y se les asigna el valor 3. Las constantes no pueden cambiar su valor durante la ejecución del programa.

Se declara una matriz bidimensional de enteros llamada **matrizUnos**. El tamaño de la matriz se define por las constantes filas y columnas, por lo que será una matriz de 3x3.

Se utilizan dos bucles anidados para recorrer todos los elementos de la matriz. En cada iteración de los bucles, se asigna el valor 1 a la posición correspondiente de la matriz, llenándola así con unos.

Para imprimir la matriz se utilizan dos bucles anidados para recorrer todos los elementos de la matriz nuevamente. En cada iteración, se imprime el valor del elemento actual de la matriz, seguido de un espacio. Al final de cada fila, se imprime un salto de línea (**endl**) para pasar a la siguiente fila.

ISBN: 978-631-6557-57-5

En Java

```java
public class MatrizUnos {
    public static void main(String[] args) {
        // Definir el tamaño de la matriz
        int filas = 3;
        int columnas = 3;

        // Definir la matriz
        int[][] matrizUnos = new int[filas][columnas];

        // Llenar la matriz con unos
        for (int i = 0; i < filas; i++) {
            for (int j = 0; j < columnas; j++) {
                matrizUnos[i][j] = 1;
            }
        }

        // Imprimir la matriz
        System.out.println("La matriz resultante es:");
        for (int i = 0; i < filas; i++) {
            for (int j = 0; j < columnas; j++) {
                System.out.print(matrizUnos[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

Se declaran dos variables enteras, **filas** y **columnas**, y se les asigna el valor 3. Estas variables determinarán las dimensiones de la matriz.

Se declara una matriz bidimensional de enteros llamada **matrizUnos**. El tamaño de la matriz se define por las variables filas y columnas, por lo que será una matriz de 3x3. La palabra clave new se utiliza para reservar la memoria necesaria para almacenar los elementos de la matriz.

Se utilizan dos bucles anidados para recorrer todos los elementos de la matriz. En cada iteración de los bucles, se asigna el valor 1 a la posición correspondiente de la matriz, llenándola así con unos.

Se utilizan dos bucles anidados para recorrer todos los elementos de la matriz nuevamente. En cada iteración, se imprime el valor del elemento actual de la matriz, seguido de un espacio. Al final de cada fila, se imprime un salto de línea (println) para pasar a la siguiente fila.

20) Encontrar el valor máximo de la matriz 3x3.

$$\begin{bmatrix} 8 & 16 & 21 \\ 12 & 9 & 14 \\ 22 & 17 & 10 \end{bmatrix}$$

Pseudocódigo

Se declara una matriz (también conocida como arreglo bidimensional) llamada **matriz**

ISBN: 978-631-6557-57-5

con dimensiones de 3 filas y 3 columnas. Esto significa que la matriz podrá almacenar un total de 9 elementos.

```
Proceso EncontrarMaximo
    // Definir la matriz
    Dimension matriz[3, 3]
    // Asignar valores a la matriz
    matriz[1,1] ← 8
    matriz[1,2] ← 16
    matriz[1,3] ← 21
    matriz[2,1] ← 12
    matriz[2,2] ← 9
    matriz[2,3] ← 14
    matriz[3,1] ← 22
    matriz[3,2] ← 17
    matriz[3,3] ← 10
    // Inicializar la variable maximo
    Definir maximo Como Entero
    maximo ← matriz[1,1]
    // Encontrar el valor máximo en la matriz
    Para i ← 1 Hasta 3 Hacer
        Para j ← 1 Hasta 3 Hacer
            Si matriz[i,j] > maximo Entonces
                maximo ← matriz[i,j]
            FinSi
        FinPara
    FinPara
    // Imprimir el valor máximo
    Escribir "El valor máximo es: ", maximo
FinProceso
```

Se declara una variable llamada **maximo** de tipo entero y se le asigna inicialmente el valor de la posición matriz[1,1]. Esta variable sirve para almacenar el valor máximo encontrado en la matriz durante la ejecución del algoritmo.

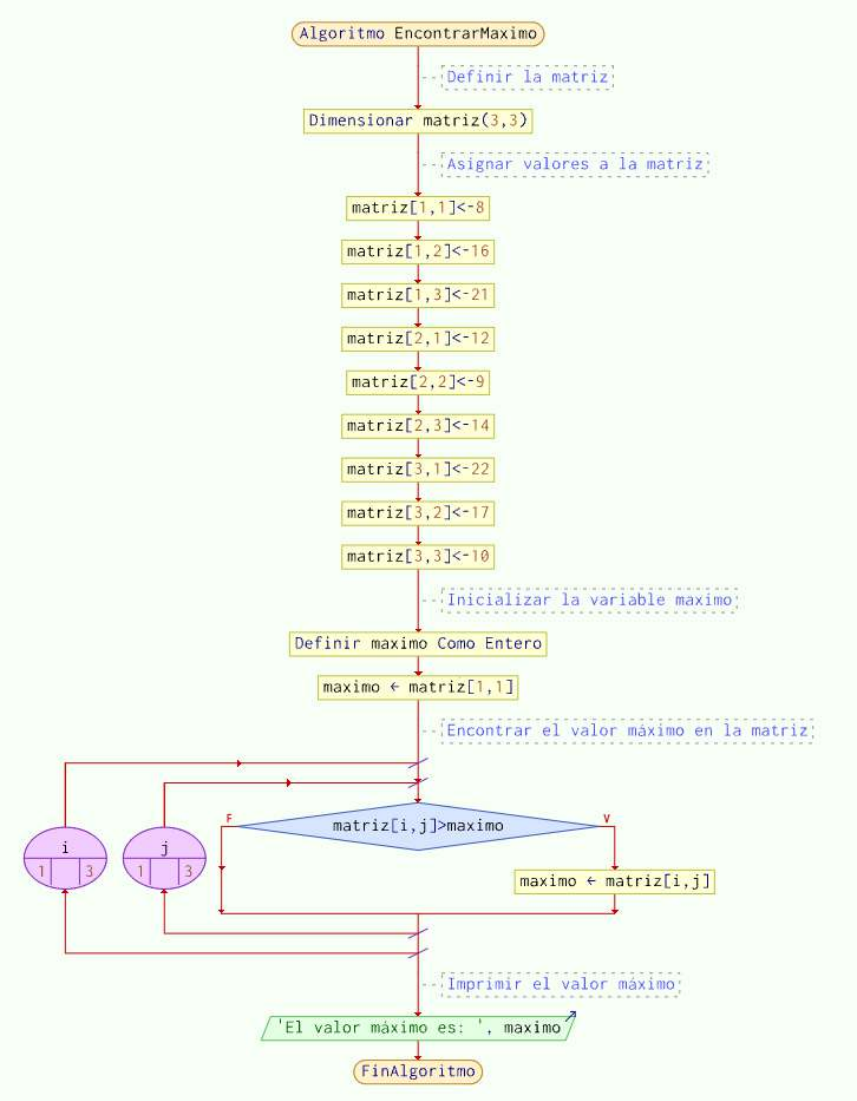
Se emplean dos bucles anidados (uno dentro del otro) para recorrer todas las posiciones de la matriz, la variable **i** controla las filas de la matriz y la variable **j** controla las columnas de la matriz.

Dentro de los bucles, se compara el valor actual de la matriz (matriz[**i**,**j**]) con el valor almacenado en la variable **maximo**.

Si el valor actual de la matriz es mayor que el valor máximo actual, entonces se actualiza el valor de **maximo** con el valor actual de la matriz.

ISBN: 978-631-6557-57-5

Diagrama de flujo



Implementación

En Python

```
import numpy as np
matriz = np.array([[8, 16, 21], [12, 9, 14], [22, 17, 10]])
maximo = np.max(matriz)
print(maximo)
```

```
22
```

Se utiliza la función max de numpy para encontrar el número mayor de la matriz.

Se puede hacer el programa también de tal forma que no use np.max.

ISBN: 978-631-6557-57-5

```python
import numpy as np

# Definir la matriz
matriz = np.array([[8, 16, 21], [12, 9, 14], [22, 17, 10]])

# Inicializar la variable maximo
maximo = matriz[0, 0]

# Encontrar el valor máximo en la matriz
for i in range(matriz.shape[0]):
    for j in range(matriz.shape[1]):
        if matriz[i, j] > maximo:
            maximo = matriz[i, j]

# Imprimir el valor máximo
print(maximo)
```

Se crea una matriz de números utilizando la función **np.array**. Los números se organizan en filas y columnas dentro de listas anidadas. La matriz resultante tiene 3 filas y 3 columnas.

Se crea una variable llamada **maximo** y se le asigna el valor del elemento ubicado en la primera fila y primera columna de la matriz (es decir, el número 8). Este valor se utilizará como punto de partida para comparar con los demás elementos de la matriz.

Se implementa dos bucles anidados para recorrer todos los elementos de la matriz, el primer bucle (for i in range(matriz.shape[0])) itera sobre las filas de la matriz, el segundo bucle (for j in range(matriz.shape[1])) itera sobre las columnas de la matriz.

Dentro de los bucles, se compara el valor actual del elemento (matriz[i, j]) con el valor actual de **maximo**. Si el valor actual es mayor que **maximo**, entonces se actualiza el valor de **maximo** con el valor actual.

En JavaScript

```javascript
// Definir la matriz
const matriz = [
  [8, 16, 21],
  [12, 9, 14],
  [22, 17, 10]
];

// Encontrar el valor máximo en la matriz utilizando Math.max y flat
const maximo = Math.max(...matriz.flat());

// Imprimir el valor máximo
console.log("El valor máximo es: " + maximo);
```

const matriz = [...]; declara una constante llamada matriz y le asigna un arreglo (o matriz) de números. La palabra clave const indica que el valor de esta constante no podrá ser cambiado después de su inicialización. Dentro del arreglo **matriz**, se definen tres sub-

ISBN: 978-631-6557-57-5

arreglos, cada uno representando una fila de la matriz. Estos sub-arreglos contienen los valores numéricos de la matriz.

matriz.flat() convierte la matriz bidimensional en un arreglo unidimensional, es decir, "aplana" la matriz. El operador de propagación (**...**) descompone el arreglo resultante de matriz.flat() en una lista de argumentos individuales.

Math.**max**() encuentra el valor máximo entre todos los argumentos.

Tambien se puede hacer sin utilizar Math.max y matriz.flat.

```
// Definir la matriz
const matriz = [
  [8, 16, 21],
  [12, 9, 14],
  [22, 17, 10]
];

// Inicializar la variable maximo
let maximo = matriz[0][0];

// Encontrar el valor máximo en la matriz
for (let i = 0; i < matriz.length; i++) {
  for (let j = 0; j < matriz[i].length; j++) {
    if (matriz[i][j] > maximo) {
      maximo = matriz[i][j];
    }
  }
}

// Imprimir el valor máximo
console.log("El valor máximo es: " + maximo);

```

let maximo = matriz[0][0]; declara una variable llamada **maximo** y le asigna el valor del primer elemento de la matriz (fila 0, columna 0). Esta variable se utilizará para almacenar el valor máximo encontrado durante la búsqueda.

Se inicia un bucle for que iterará sobre las filas de la matriz. La variable **i** se utiliza como índice de fila.

También se inicia un bucle for anidado que iterará sobre las columnas de la matriz para cada fila. La variable **j** se utiliza como índice de columna.

if (matriz[i][j] > maximo) verifica si el valor actual de la matriz (en la posición [**i**][**j**]) es mayor que el valor actual de maximo.

maximo = matriz[i][j] si la condición del if es verdadera, se actualiza el valor de

ISBN: 978-631-6557-57-5

maximo con el valor actual de la matriz.

En C++

```cpp
#include <iostream>
#include <algorithm> // Para std::max_element
using namespace std;
int main() {
    // Definir la matriz
    int matriz[3][3] = {
        {8, 16, 21},
        {12, 9, 14},
        {22, 17, 10}
    };
    // Inicializar la variable maximo
    int maximo = matriz[0][0];
    // Encontrar el valor máximo en la matriz
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if (matriz[i][j] > maximo) {
                maximo = matriz[i][j];
            }
        }
    }
    // Imprimir el valor máximo
    cout << "El valor máximo es: " << maximo << endl;
    return 0;
}
```

La cabecera algorithm, que proporciona una variedad de algoritmos estándar, como std::max_element para encontrar el elemento máximo en un rango. Aunque este algoritmo no se utiliza explícitamente en este código, podría ser útil en otras situaciones.

int matriz[3][3] = { ... };: Declara una matriz de enteros llamada matriz con 3 filas y 3 columnas. Los valores dentro de las llaves inicializan cada elemento de la matriz.

int maximo = matriz[0][0];: Declara una variable entera maximo y la inicializa con el valor del primer elemento de la matriz (fila 0, columna 0). Esta variable se utilizará para almacenar el valor máximo encontrado durante la búsqueda.

El bucle externo itera sobre las filas de la matriz, mientras el bucle interno itera sobre las columnas de la matriz para cada fila.

if (matriz[i][j] > maximo) { ... }: Si el elemento actual matriz[**i**][**j**] es mayor que el valor actual de maximo, se actualiza maximo con el nuevo valor.

ISBN: 978-631-6557-57-5

En Java

```java
public class EncontrarMaximo {
    public static void main(String[] args) {
        // Definir la matriz
        int[][] matriz = {
            {8, 16, 21},
            {12, 9, 14},
            {22, 17, 10}
        };

        // Inicializar la variable maximo
        int maximo = matriz[0][0];

        // Encontrar el valor máximo en la matriz
        for (int i = 0; i < matriz.length; i++) {
            for (int j = 0; j < matriz[i].length; j++) {
                if (matriz[i][j] > maximo) {
                    maximo = matriz[i][j];
                }
            }
        }

        // Imprimir el valor máximo
        System.out.println("El valor máximo es: " + maximo);
    }
}
```

3.4.2 Ejercicios propuestos

1) Crear un vector de 5 elementos

2) Contar elementos mayores que el número 5 dado la lista [2,7,5,9].

3) Multiplicar todos los elementos de la lista [2,3,4,5,6,7]

4) Encontrar el menor elemento de la lista [4,6,7,3,10,14]

5) Sumar los elementos de las listas [1,2] [3,4]

6) Iterar en la lista edades = {"Tania": 25, "Luis": 30, "Karla": 30} *y como salida se tenga:*

   Tania tiene 25 años

   Juan tiene 35 años

   Karla tiene 30 años

7) Dada la lista [2,4,5,8,10,14] g*enerar una lista formada por los cuadrados de cada uno de los elementos de la lista.*

8) Encontrar los números pares de la lista [2,5,7,8,9,10,11,14] *y construir una nueva lista.*

9) Contar cuantos elementos impares existe en la lista [2,3,5,6,8,9,10,13,15].

ISBN: 978-631-6557-57-5

10) Indicar cuantas veces se repite el *número 7 en la lista [2,7,4,7,7,8,12,14,7,20].*

11) Generar una lista con las factoriales de los elementos de la siguiente lista [2,4,5,7,8,9].

12) Encontrar el índice del número 5 de la lista [3,6,7,5,8,9].

13) Calcular la media de la lista [2,3,4,5,6,7,8,9].

14) Contar el n*úmero de veces que el 8 aparece en la lista [6,7,8,9,10,8,8,11,16,8,18].*

15) Se tiene una lista [32.0, 86.0, 107.6, 149.0, 179.6, 248.0] en que los elementos representan la temperatura en grados Fahrenheit convertir los valores a grados Celsius.

16) Obtener los elementos comunes de dos listas *[1,3,4,6] [3,5,6,8].*

17) Sumar dos vectores elemento a elemento. [6,8,12,*7] [4,5,16,3]*

18) Determinar si el vector [5,6,7,8,9], esta ordenado de forma ascendente.

19) Sumar *los siguientes vectores [2,3][8,-4]*

20) *Calcular el producto punto de los vectores [5,6,7][-2,3,4]*

21) Contar cuantos elementos de la lista [1,3,5,6,7] son mayores que el promedio.

22) Mover todos los *ceros del vector [1,0,4,0,5,0,0] al inicio.*

23) Crear una matriz de 2x2 e imprimir.

24) Restar las matrices.

$$\begin{bmatrix} 11 & 12 & 13 \\ 14 & 15 & 16 \\ 17 & 18 & 19 \end{bmatrix} \begin{bmatrix} 2 & 3 & 4 \\ 5 & 6 & 7 \\ 9 & 10 & 11 \end{bmatrix}$$

25) Transponer la matriz.

ISBN: 978-631-6557-57-5

$$\begin{bmatrix} 11 & 12 & 13 \\ 14 & 15 & 16 \\ 17 & 18 & 19 \end{bmatrix}$$

26) Dividir cada elemento de la matriz por 10.

$$\begin{bmatrix} 21 & 22 \\ 23 & 24 \end{bmatrix}$$

27) Crear una matriz de 3x3 de ceros.

28) Encontrar el valor mínimo de la matriz 3x3.

$$\begin{bmatrix} 8 & 16 & 21 \\ 12 & 9 & 14 \\ 22 & 17 & 10 \end{bmatrix}$$

29) Calcular la suma de todos los elementos de una matriz 3x3.

$$\begin{bmatrix} 8 & 16 & 21 \\ 12 & 9 & 14 \\ 22 & 17 & 10 \end{bmatrix}$$

ISBN: 978-631-6557-57-5

ISBN: 978-631-6557-57-5

CAPÍTULO 4

4 PROGRAMACION MODULAR

4.1 La programación modular

La programación modular es un enfoque utilizado en el desarrollo de software que implica dividir un programa en módulos o unidades más pequeñas e independientes. Este enfoque ofrece múltiples beneficios, como facilitar la comprensión del código, simplificar el proceso de depuración y mantenimiento, y permitir la reutilización de código.

4.1.1 Módulo

Un módulo en el contexto de la programación es una unidad lógica de código que tiene una funcionalidad específica y que puede ser utilizada de manera independiente del resto del programa. La idea de un módulo se basa en el principio de la división del trabajo en la programación, lo que facilita la organización, comprensión, mantenimiento y reutilización del código.

a) **Características de un Módulo**

Independencia: Cada módulo debe ser capaz de funcionar de manera independiente, lo que significa que puede realizar su tarea específica sin necesidad de información o recursos externos no especificados en su interfaz.

Encapsulación: Los módulos encapsulan comportamientos y datos, exponiendo solo aquellas partes necesarias para el uso de dicho módulo (su interfaz) y ocultando los detalles de implementación internos. Esto significa que el resto del programa no necesita conocer los detalles internos del módulo para poder utilizarlo.

Interfaz clara: Un módulo tiene una interfaz definida que especifica cómo pueden otros componentes del programa interactuar con él. Esta interfaz incluye funciones, procedimientos, o métodos que el módulo expone, así como los parámetros que estos esperan y los valores que retornan.

ISBN: 978-631-6557-57-5

b) **Tipos de Módulos**

Funciones y Procedimientos: Son bloques de código que realizan una tarea específica. La diferencia entre ellos suele depender del lenguaje de programación; por ejemplo, en algunos lenguajes, las funciones retornan un valor mientras que los procedimientos no.

Clases: En la programación orientada a objetos, una clase se puede considerar un módulo que encapsula datos (atributos) y comportamientos (métodos) relacionados. Las clases permiten crear instancias (objetos) que mantienen su propio estado y comportamiento.

Archivos y Paquetes: Un archivo que contiene código (funciones, clases, variables) puede ser tratado como un módulo, especialmente si el lenguaje de programación permite importar este archivo para usar su contenido. De manera similar, un paquete, que es una colección de módulos relacionados, también puede considerarse un módulo a una escala mayor.

c) **Importancia de la Modularidad**

La modularidad permite a los desarrolladores dividir programas complejos en partes más pequeñas y manejables, facilitando el desarrollo, las pruebas y la depuración. Los módulos bien diseñados tienen las siguientes ventajas:

Mejora en la organización del código: Al agrupar funcionalidades relacionadas, los módulos ayudan a mantener el código organizado y claro.

Reutilización: Los módulos pueden diseñarse para ser reutilizables en diferentes partes de un programa o incluso en diferentes proyectos, lo que reduce el esfuerzo de desarrollo y mejora la consistencia del código.

Facilidad de mantenimiento: La independencia de los módulos facilita la actualización y el mantenimiento del código, debido a que los cambios en un módulo suelen tener un impacto limitado en el resto del programa.

Desarrollo en paralelo: La modularidad permite que diferentes equipos trabajen en diferentes módulos de manera simultánea, agilizando el proceso de desarrollo.

4.1.2 Cohesión y acoplamiento

Cohesión y acoplamiento son dos conceptos a tener en cuenta en el diseño de software, especialmente cuando se habla de programación modular. Estos conceptos ayudan a evaluar y mejorar la calidad del diseño de un sistema de software, enfocándose

ISBN: 978-631-6557-57-5

en cómo los diferentes componentes del sistema (módulos, clases, funciones) están organizados y cómo interactúan entre sí.

1. **Cohesión**

Un módulo altamente cohesivo es aquel en el que todas las partes están estrechamente relacionadas y enfocadas en realizar una única tarea o conjunto de tareas relacionadas. Esto significa que el módulo tiene una única responsabilidad bien definida y no realiza trabajos que sean irrelevantes para su propósito principal.

a) Niveles de Cohesión

Existen diferentes niveles de cohesión, desde la cohesión funcional (la más alta), cohesión lógica y temporal (las más bajas).

La cohesión funcional se considera el grado más alto de cohesión en el diseño de software y es el objetivo ideal cuando se estructuran módulos o componentes dentro de un sistema. Este nivel de cohesión se alcanza cuando todas las tareas o funciones dentro de un módulo están orientadas hacia la realización de una única actividad específica, garantizando que el módulo sea autónomo y esté dedicado completamente a una sola función o propósito.

La cohesión lógica se presenta cuando los elementos de un módulo se agrupan porque son lógicamente categorizables de la misma manera, aunque no realicen la misma función. En este tipo de cohesión, las funciones se incluyen en el mismo módulo porque comparten algún tipo de característica lógica, pero no necesariamente porque contribuyan a una única tarea o propósito específico.

La cohesión temporal se presenta cuando los elementos de un módulo están agrupados por el momento o el contexto en el que se llevan a cabo, en lugar de por la tarea que realizan. Esto significa que las funciones se realizan en la misma secuencia temporal o están relacionadas por el tiempo durante el cual se ejecutan.

b) Beneficios de la Alta Cohesión

Facilidad de mantenimiento: Los módulos con alta cohesión son más fáciles de entender, modificar y depurar, ya que todo el código relevante para una tarea específica

ISBN: 978-631-6557-57-5

se encuentra en un solo lugar.

Reutilización: Los módulos altamente cohesivos tienen más probabilidades de ser reutilizables en diferentes partes de un programa o en diferentes proyectos, ya que su funcionalidad es clara y bien definida.

2. Acoplamiento

El acoplamiento se refiere a la interdependencia entre módulos. Bajo acoplamiento significa que los módulos son relativamente independientes unos de otros y se comunican a través de interfaces bien definidas, reduciendo las interacciones directas. El objetivo es minimizar cómo los cambios en un módulo afectan a otros módulos, lo que facilita la gestión del código y mejora la modularidad.

a) Tipos de Acoplamiento

El acoplamiento puede variar desde el acoplamiento débil (ideal) hasta el acoplamiento fuerte (menos deseable). El acoplamiento débil se logra limitando la información compartida y las dependencias entre módulos, mientras que el acoplamiento fuerte ocurre cuando los módulos dependen estrechamente de los detalles internos de otros módulos.

b) Beneficios del Bajo Acoplamiento

Flexibilidad: Los sistemas con bajo acoplamiento son más flexibles y fáciles de modificar. Cambiar la implementación de un módulo es más sencillo cuando este no está fuertemente conectado con otros módulos.

Facilidad de pruebas: Es más fácil probar módulos en aislamiento cuando están débilmente acoplados, debido a que se reducen las dependencias que necesitan ser simuladas o configuradas en el entorno de prueba.

3. Diseño de Software

En la práctica, lograr alta cohesión y bajo acoplamiento requiere un diseño cuidadoso y considerado. Los diseñadores de software deben esforzarse por:

ISBN: 978-631-6557-57-5

- Definir claramente las responsabilidades de cada módulo.
- Limitar las interacciones entre módulos a interfaces bien definidas.
- Evitar que los módulos compartan detalles internos innecesarios.

Un buen diseño modular, que promueva alta cohesión dentro de los módulos y bajo acoplamiento entre ellos, resulta en un software más robusto, mantenible y escalable. Estos principios no solo son aplicables al diseño de sistemas grandes y complejos, sino que también son relevantes para la organización de código incluso en pequeños proyectos de programación.

4.1.3 Diseño de módulos

El diseño de módulos está relacionado a que una buena modularización puede significar la diferencia entre un sistema fácil de mantener y uno que no lo es. Veamos en detalle cómo se puede llevar a cabo este proceso, centrándonos en la identificación de módulos y la definición de interfaces de módulo.

a) Identificación de Módulos

La identificación de módulos es el primer paso hacia un diseño modular eficaz. Este proceso implica analizar y descomponer el sistema en componentes o módulos más pequeños, cada uno con una funcionalidad específica. Algunas estrategias para identificar módulos incluyen:

Identificar Tareas Repetitivas: Uno de los indicadores más claros de que una funcionalidad puede ser modularizada es si esa tarea se realiza en múltiples lugares del programa. Por ejemplo, si varias partes de tu aplicación necesitan acceder a una base de datos, esto puede encapsularse en un módulo de acceso a datos.

Agrupar Funcionalidades Relacionadas: Observa las funcionalidades que están conceptualmente relacionadas y agrúpalas en un módulo. Por ejemplo, todas las operaciones relacionadas con el manejo de usuarios (crear, actualizar, borrar usuarios) pueden agruparse en un módulo de gestión de usuarios.

Principio de Responsabilidad Única: Cada módulo debe tener una y solo una razón para cambiar. Esto significa que un módulo debe encargarse de una sola parte de la

ISBN: 978-631-6557-57-5

funcionalidad del software, y esta responsabilidad debe estar completamente encapsulada por el módulo.

Independencia Conceptual: Las tareas que son conceptualmente independientes del resto del programa también son buenos candidatos para la modularización. Esto significa que la tarea tiene una clara definición y límites que la separan del resto de las funcionalidades del sistema.

b) Interfaces de Módulo

Una vez identificados los módulos, se debe definir interfaces claras para ellos. La interfaz de un módulo es el punto de contacto entre el módulo y el resto del sistema, y define cómo se comunican entre sí. Una interfaz clara y bien definida es esencial por varias razones:

Abstracción: La interfaz proporciona una abstracción del módulo, escondiendo los detalles de implementación y exponiendo solo lo que es necesario para el uso del módulo. Esto permite cambiar la implementación interna del módulo sin afectar a quienes lo usan, siempre y cuando la interfaz permanezca constante.

Comunicación: La interfaz especifica cómo otros módulos o partes del programa pueden interactuar con el módulo, incluyendo los parámetros que se pueden pasar y los valores que se pueden esperar a cambio. Esto incluye métodos o funciones accesibles, tipos de datos esperados, y cualquier otro protocolo de comunicación necesario.

Documentación: Una interfaz bien definida sirve como documentación, facilitando a otros desarrolladores entender cómo usar el módulo sin necesidad de adentrarse en los detalles de su implementación.

Contrato: La interfaz actúa como un contrato entre el módulo y el resto del sistema, especificando lo que el módulo promete hacer. Esto es crucial para el mantenimiento y la prueba del software, debido a que proporciona una base clara para las pruebas de integración y unidad.

4.1.4 Implementación de módulos

La implementación de módulos en la programación es un paso crucial para estructurar programas de manera efectiva, facilitando tanto la organización del código como su reutilización y mantenimiento. Dependiendo del paradigma de programación,

ISBN: 978-631-6557-57-5

los módulos pueden implementarse a través de funciones y procedimientos en la programación procedimental, o mediante clases y objetos en la programación orientada a objetos.

Funciones y Procedimientos

Las funciones y los procedimientos son bloques de código que se ejecutan para realizar una tarea específica. La principal diferencia entre ambos radica en que las funciones retornan un valor como resultado de su ejecución, mientras que los procedimientos realizan una acción sin retornar un valor directamente.

Sintaxis para Definir Funciones

La sintaxis para definir funciones varía entre lenguajes de programación, pero generalmente incluye el nombre de la función, parámetros (si los hay), y el código que define la operación a realizar.

Por ejemplo, en **Python**, una función se define así:

```
def multiplicar(a, b):
    return a * b
```

En **JavaScript** la función tendría esta forma:

```
function multiplicar(a, b) {
    return a * b;
}
```

En **C++,** se definiría de manera similar, especificando el tipo de dato que retorna:

```
int multiplicar(int a, int b) {
    return a * b;
}
```

En C++, no se utiliza una palabra clave específica como def o function para definir funciones. Esto se debe a que C++ es un lenguaje de programación que sigue una sintaxis y estructura derivada de C, que también carece de una palabra clave específica para la definición de funciones (Leroy, 2009).

En C++, las funciones se definen especificando directamente el tipo de retorno, el nombre de la función y sus parámetros. La sintaxis se basa en la declaración explícita del tipo de datos, lo que hace innecesario el uso de una palabra clave adicional para la definición de funciones.

int es el tipo de retorno de la función.

ISBN: 978-631-6557-57-5

multiplicar es el nombre de la función.

int a, int b son los parámetros de la función.

Esta forma de definir funciones es directa y concisa, aprovechando la fuerte tipificación del lenguaje y su sintaxis estructurada.

Y en **Java**, la función sería:

```
public static int multiplicar(int a, int b) {
   return a * b;
}
```

En Java, las funciones se definen dentro de una clase y se les llama métodos (Ogihara, 2018). No existe una palabra clave específica como def en Python o function en JavaScript porque Java sigue un paradigma orientado a objetos más estricto.

Para definir un método en Java, se utilizan varias palabras clave que indican el contexto y el comportamiento del método:

a) **Modificador de acceso**: public, private, protected o sin modificador (paquete). Esto define la visibilidad del método.
b) **Tipo de retorno**: El tipo de dato que el método va a devolver. Si no devuelve nada, se usa void.
c) **Nombre del método**: El nombre que se le da al método.
d) **Parámetros**: La lista de parámetros que el método acepta, encerrada entre paréntesis.

En el caso del ejemplo:

public es el modificador de acceso.

static indica que el método pertenece a la clase en lugar de a instancias de la clase.

int es el tipo de retorno.

multiplicar es el nombre del método.

int a, int b son los parámetros del método.

Llamada a Funciones

Para utilizar una función, simplemente se "llama" o "invoca" especificando su nombre y proporcionando los valores de los argumentos requeridos, si los hay. Por ejemplo:

En **Python**

ISBN: 978-631-6557-57-5

```
resultado = multiplicar(7, 4)
```

En **JavaScript**

```
let resultado = multiplicar(7, 4);
```

En **C++**

```
int resultado = multiplicar(7, 4);
```

En **Java**

```
int resultado = multiplicar(7, 4);
```

Clases y Objetos

En la programación orientada a objetos (POO), las clases actúan como plantillas para crear objetos. Una clase define los atributos (datos) y métodos (funciones asociadas) que tendrán los objetos creados a partir de ella. De esta manera, las clases pueden considerarse como módulos que agrupan datos y comportamientos relacionados.

Definición de Clases

La definición de una clase incluye el nombre de la clase, sus atributos y métodos. Por ejemplo, en Java, una clase "Persona" podría definirse así:

```
public class Persona {
   String nombre;
   int edad;
   public Persona(String nombre, int edad) {
      this.nombre = nombre;
      this.edad = edad;
   }
   public void presentarse() {
      System.out.println("Hola, mi nombre es " + nombre + " y tengo " + edad + " años.");
   }
}
```

ISBN: 978-631-6557-57-5

La clase Persona tiene dos atributos:

nombre: Un atributo de tipo String que almacena el nombre de la persona.

edad: Un atributo de tipo int que almacena la edad de la persona.

Estos atributos se declaran dentro de la clase, fuera de cualquier método, y son accesibles desde cualquier método de la clase.

Los **métodos** son las acciones o comportamientos que un objeto puede realizar (Ogihara, 2018). La clase Persona tiene un método:

presentarse(): Este método es de tipo void, lo que significa que no devuelve ningún valor. Su función es imprimir un mensaje en la consola presentando a la persona, utilizando los valores de los **atributos** nombre y edad.

Además del método presentarse(), la clase Persona tiene un **constructor**. El constructor es un método especial que se llama automáticamente cuando se crea un nuevo objeto de la clase. En este caso, el constructor tiene dos parámetros:

nombre: Un parámetro de tipo String que se utiliza para inicializar el atributo nombre del objeto.

edad: Un parámetro de tipo int que se utiliza para inicializar el atributo edad del objeto.

La palabra clave this se utiliza para referirse a los atributos del objeto que se está creando.

Creación y Uso de Objetos

Para utilizar una clase, se crea una instancia de ella, es decir, un objeto. Luego, se puede acceder a sus atributos y llamar a sus métodos. Siguiendo el ejemplo anterior en Java:

```
Persona persona = new Persona("Gisel", 30);
persona.presentarse();  // Imprime: Hola, mi nombre es Gisel y tengo 30 años.
```

Persona persona: Se declara una variable llamada persona de tipo Persona. Esta variable actuará como una referencia a un objeto de la clase Persona.

new Persona("Gisel", 30): Se crea un nuevo objeto de la clase Persona. Al crear este objeto, se invoca al constructor de la clase Persona y se pasan los valores "Gisel" y 30 como argumentos. Estos valores se asignan a los atributos nombre y edad del nuevo objeto, respectivamente.

ISBN: 978-631-6557-57-5

En la línea persona.presentarse(); **persona** es la variable que hace referencia a un objeto de la clase Persona y **presentarse()** es el nombre de un método que pertenece a la clase Persona.

El programa completo quedaría de la siguiente forma:

```
public class Persona {
   String nombre;
   int edad;
   public Persona(String nombre, int edad) {
      this.nombre = nombre;
      this.edad = edad;
   }
   public void presentarse() {
      System.out.println("Hola, mi nombre es " + nombre + " y tengo " + edad + " años.");
   }
public static void main(String[] args) {
      // Crear una instancia de Persona
Persona persona = new Persona("Gisel", 30);
persona.presentarse();  // Imprime: Hola, mi nombre es Gisel y tengo 30 años.
     }
}
```

```
Hola, mi nombre es Gisel y tengo 30 años.
```

La línea que no se ha conceptualizado en este código es, public static void main(String[] args) { donde:

public: Indica que el método main es accesible desde cualquier parte del programa.

static: Significa que el método pertenece a la clase y no a una instancia específica de la clase. Esto permite que el método sea llamado sin crear un objeto de la clase.

void: Indica que el método main no devuelve ningún valor.

main: Es el nombre del método, que es un estándar establecido en Java.

(String[] args): Define un parámetro llamado args que es un arreglo de cadenas de

ISBN: 978-631-6557-57-5

caracteres. Este arreglo se utiliza para pasar argumentos a la aplicación desde la línea de comandos cuando se ejecuta.

4.1.5 Beneficios de la programación modular

La programación modular ofrece numerosos beneficios que hacen que el desarrollo, mantenimiento y prueba de software sean más eficientes y manejables. Estos beneficios son el resultado directo de la capacidad de la programación modular para dividir programas complejos en unidades más pequeñas, cohesivas y manejables.

a) Facilidad de Mantenimiento

El mantenimiento del software es una tarea constante y necesaria que implica corregir errores, actualizar funcionalidades y mejorar el rendimiento. La modularización facilita estos procesos de varias maneras:

b) Independencia de los Módulos

Dado que cada módulo es independiente y se centra en una funcionalidad específica, los cambios realizados en un módulo tienen menos probabilidades de impactar en otros módulos. Esto aísla los cambios y reduce el riesgo de efectos secundarios no deseados, lo que facilita la actualización y corrección de errores.

c) Comprensión Mejorada

Los módulos bien definidos son más fáciles de entender individualmente que un programa monolítico. Esto significa que los desarrolladores pueden identificar y localizar problemas más rápidamente, así como implementar mejoras sin tener que comprender todo el sistema en detalle.

d) Facilita las Modificaciones

Al tener una estructura modular, es posible actualizar o modificar partes del software para cumplir con nuevos requisitos o tecnologías sin necesidad de reescribir el sistema completo. Esto es especialmente útil en el entorno actual de rápido avance tecnológico.

e) Reutilización de Código

Uno de los principios fundamentales de la ingeniería de software es DRY (Don't Repeat Yourself), que busca minimizar la duplicación de código. La programación modular es clave para lograr esto:

ISBN: 978-631-6557-57-5

f) Modularidad como Unidades de Reutilización

Los módulos que encapsulan funcionalidades de manera efectiva pueden ser reutilizados en diferentes partes de un programa, o incluso entre varios proyectos. Esto no solo ahorra tiempo y recursos al desarrollar nuevas aplicaciones, sino que también promueve la consistencia y reduce la probabilidad de errores.

g) Bibliotecas y Frameworks

La reutilización de código se facilita mediante la creación de bibliotecas de módulos o frameworks, que pueden ser compartidos y utilizados por la comunidad de desarrolladores. Esto enriquece el ecosistema de desarrollo y proporciona una base sólida sobre la cual construir nuevas soluciones.

h) Facilidad de Pruebas

Las pruebas son una parte esencial del desarrollo de software que garantiza la calidad y el correcto funcionamiento de los programas. La programación modular mejora significativamente el proceso de pruebas:

i) Pruebas Unitarias

Los módulos pueden ser probados de forma independiente usando pruebas unitarias, lo cual es más eficiente que probar un sistema completo de una vez. Esto permite a los desarrolladores identificar y corregir errores en etapas tempranas del desarrollo.

j) Automatización de Pruebas

La independencia de los módulos facilita la automatización de las pruebas, debido a que se pueden diseñar suites de pruebas específicas para cada módulo sin preocuparse por el resto del sistema. La automatización de pruebas ahorra tiempo y mejora la confiabilidad de las pruebas.

k) Integración Continua

En sistemas modulares, es más fácil implementar prácticas de integración continua, donde las pruebas automáticas se ejecutan regularmente a medida que se agregan cambios al código. Esto ayuda a mantener la calidad del software a lo largo de su ciclo de vida.

ISBN: 978-631-6557-57-5

4.1.6 Prácticas de programación modular

La programación modular no solo organiza mejor el código y facilita su mantenimiento, sino que también impulsa prácticas más avanzadas como el desarrollo de bibliotecas y la simplificación de la depuración. Vamos a explorar estas prácticas en detalle.

1. Bibliotecas

a) Desarrollo de Bibliotecas

Una biblioteca en programación es una colección de módulos, funciones, clases, y otros recursos que proporcionan una funcionalidad específica y que pueden ser reutilizados en diferentes proyectos. El desarrollo de bibliotecas se basa en identificar conjuntos de funcionalidades que son comunes a varios proyectos o que resuelven problemas generales.

b) Beneficios de las Bibliotecas

Reutilización de Código: Las bibliotecas permiten a los desarrolladores reutilizar código probado y fiable, reduciendo el tiempo de desarrollo y minimizando la duplicación de esfuerzos.

Estándares de Calidad: Al centralizar soluciones a problemas comunes, las bibliotecas ayudan a estandarizar la calidad y el enfoque de resolución de problemas dentro de una organización o comunidad.

Fomenta la Colaboración: Las bibliotecas pueden ser desarrolladas y mejoradas por múltiples desarrolladores, promoviendo la colaboración y el intercambio de conocimientos.

c) Estrategias para el Desarrollo de Bibliotecas

Identificar Necesidades Comunes: Observar los problemas que se repiten a través de diferentes proyectos y considerar abstractar estas soluciones en módulos reutilizables.

Diseño Modular: Diseñar las bibliotecas con principios de modularidad en mente, asegurando que cada módulo tenga una responsabilidad clara y limitada.

Documentación: Proporcionar documentación clara y ejemplos de uso para cada módulo, facilitando su adopción por otros desarrolladores.

A cotinuación se tiene 2 ejemplos de biliotecas de Python.

ISBN: 978-631-6557-57-5

math proporciona funciones matemáticas estándar como: math.sin(), math.cos(), math.tan(), math.exp(), math.log(), math.log10(), math.sqrt() (raíz cuadrada), math.pow() (potencia), math.pi (π), math.e (base del logaritmo natural), etc.

```
import math
def raiz_cuadrada(a):
    """Calcula la raíz cuadrada de un número."""
    return math.sqrt(a)

print(raiz_cuadrada(64))
```

Usar """ """ para documentar funciones, clases o módulos de forma formal y generar documentación clara para otros desarrolladores.
Usar **help**(raíz_cuadrada) o raíz_cuadrada.__**doc**__, para ver la documentación.

NumPy es una de las bibliotecas más populares y fundamentales en Python para realizar cálculos numéricos. Está diseñada para trabajar con arreglos multidimensionales (llamados ndarrays) y proporciona una gran cantidad de funciones para operar con esos arreglos de manera eficiente

```
import numpy as np

# Definir matrices como arrays de NumPy
matriz_a = np.array([[10, 21, 33], [4, 15, 6], [7, 8, 9]])
matriz_b = np.array([[9, 8, 7], [6, 5, 4], [3, 22, 1]])

# Sumar matrices directamente
suma = matriz_a + matriz_b

# Imprimir resultado
print("Suma de las matrices:")
print(suma)
```

```
Suma de las matrices:
[[19 29 40]
 [10 20 10]
 [10 30 10]]
```

np es el alias de numpy

np.array es una función de la biblioteca NumPy que se utiliza para crear un objeto denominado ndarray (n-dimensional array o arreglo multidimensional). Este objeto es el

ISBN: 978-631-6557-57-5

núcleo de NumPy y se utiliza para almacenar datos de manera eficiente, similar a las listas en Python, pero con un rendimiento mucho más rápido y con soporte para operaciones matemáticas vectorizadas.

Pandas es una poderosa biblioteca para la manipulación de datos, análisis y estructuras de datos como DataFrame y Series.

```
import pandas as pd
data = {'Nombre': ['Juan', 'Ana', 'Pedro'], 'Edad': [28, 22, 35]}
df = pd.DataFrame(data)
print(df)
```

```
  Nombre  Edad
0   Juan    28
1    Ana    22
2  Pedro    35
```

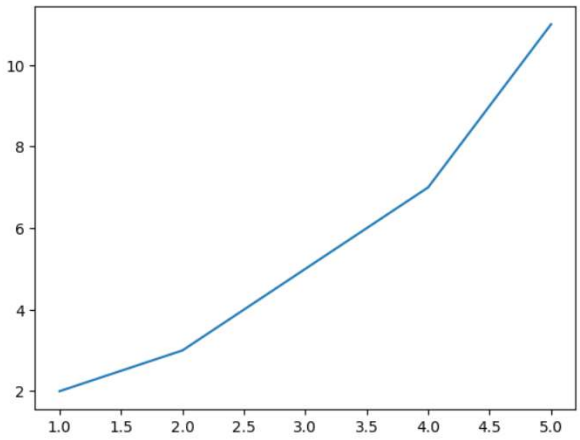
Aquí estamos creando un diccionario de datos de Python llamado **data**, que contiene dos claves: **Nombre** que tiene como valor una lista con tres nombres y **Edad** que tiene como valor una lista con tres edades. Este, diccionario representa una pequeña tabla con dos columnas: una para los nombres y otra para las edades.

Se crea un DataFrame de Pandas usando el diccionario data. Un DataFrame es una estructura bidimensional (como una tabla) donde los datos se organizan en filas y columnas. La clave **Nombre** se convierte en una columna del DataFrame. La clave **Edad** se convierte en otra columna del DataFrame.El resultado es una tabla donde cada fila contiene un nombre y su edad correspondiente.

Matplotlib es una biblioteca popular en Python para crear gráficos y visualizaciones de datos, y **pyplot** es un conjunto de funciones que facilita la creación de gráficos de manera sencilla y rápida.

```
import matplotlib.pyplot as plt
x = [1, 2, 3, 4, 5]
y = [2, 3, 5, 7, 11]
plt.plot(x, y)
plt.show()
```

ISBN: 978-631-6557-57-5



plt.plot(x, y): Esta línea genera un gráfico de líneas utilizando los valores de las listas x y y. La función plt.plot() dibuja una línea conectando los puntos definidos por las coordenadas (x, y)

plt.show(): Abre una ventana emergente con el gráfico generado. Este es el paso que hace visible el gráfico para que se pueda ver en la interfaz gráfica.

2. Depuración

La depuración es el proceso de identificar y corregir errores en el código. La modularización del software puede simplificar enormemente la depuración por varias razones.

a) Aislamiento de Problemas

Pruebas Unitarias: Al dividir el programa en módulos, puedes probar cada módulo de forma independiente con pruebas unitarias. Esto ayuda a localizar rápidamente el origen de un error.

Simplificación de la Depuración: Cuando encuentras un error, la modularidad permite aislar y examinar las partes específicas del programa donde podría residir el problema, sin tener que considerar el sistema en su totalidad.

b) Estrategias de Depuración en Programación Modular

Uso de Interfaces Claras: Asegurarse de que las interacciones entre módulos sean a través de interfaces bien definidas. Esto reduce la complejidad y hace más predecible el comportamiento del sistema.

Registro y Monitoreo: Implementar registros (logging) y monitoreo a nivel de

ISBN: 978-631-6557-57-5

módulo, lo que puede proporcionar información valiosa sobre el comportamiento del sistema en tiempo real y facilitar la identificación de errores.

4.1.7 Funciones y subprogramas

Dentro del ámbito de la programación, la organización del código en bloques reutilizables es fundamental para crear programas eficientes, legibles y mantenibles. Dos conceptos clave en este enfoque son las funciones y los subprogramas. Aunque están relacionados, tienen características distintas que los hacen útiles en diferentes contextos.

1. Función

Una función es un bloque de código diseñado para llevar a cabo una tarea específica. Lo que la distingue principalmente es su capacidad para retornar un valor después de ejecutar su tarea. Las funciones aceptan datos de entrada, conocidos como parámetros o argumentos, que influencian su ejecución. Tras completar su operación, una función puede devolver un resultado al contexto desde donde fue llamada.

a) Características de las Funciones

Reutilización: Una vez definida, una función puede ser llamada múltiples veces desde diferentes partes del programa.

Abstracción: La función encapsula operaciones complejas detrás de una interfaz simple, lo que permite a los desarrolladores no preocuparse por los detalles internos de cómo realiza su tarea.

Modularidad: Las funciones ayudan a dividir el programa en pequeños bloques, mejorando la organización del código y facilitando tanto el mantenimiento como la depuración.

2. Subprograma

El término subprograma es más general y se refiere a cualquier segmento de código que realiza una operación o conjunto de operaciones dentro de un programa. Los subprogramas incluyen tanto a las funciones como a los procedimientos.

a) Diferencias entre Funciones y Procedimientos

Funciones: Como se mencionó, retornan un valor y generalmente se usan cuando se

ISBN: 978-631-6557-57-5

necesita realizar cálculos o transformar datos de entrada en algún tipo de salida.

Procedimientos: No retornan un valor directamente. En su lugar, realizan una tarea, como modificar el estado de un programa o realizar una operación de salida, como imprimir un mensaje en la pantalla. En algunos lenguajes de programación, esta distinción es clara y se define mediante sintaxis específica. Por ejemplo, en Pascal, se usa "function" para definir una función y "procedure" para definir un procedimiento.

Ejemplo en **Python**

```
def dividir(a, b):  # Función que retorna un valor
    return a / b

def imprimir_mensaje(mensaje):  # Procedimiento que realiza una tarea, pero no retorna un valor
    print(mensaje)

resultado = dividir(5, 3)  # Llamada a función
imprimir_mensaje("Hola, Mundo")  # Llamada a procedimiento
```

Entender la diferencia entre funciones y subprogramas es importante para la estructuración lógica de un programa. La elección entre usar una función o un procedimiento depende de la necesidad específica del momento en el desarrollo del software:

- Si se requiere un resultado específico basado en entradas, se debe optar por una función.
- Si se necesita realizar acciones que afectan el programa o el entorno sin necesidad de retornar un valor, un procedimiento es la opción adecuada.

3. Sintaxis y estructura

La sintaxis para declarar funciones varía entre los diferentes lenguajes de programación, pero existen algunos principios comunes que se aplican en general. La declaración de una función típicamente incluye un nombre, una lista de parámetros, el tipo de retorno (en lenguajes con tipado estático), y un bloque de código que define la operación que realiza la función. Vamos a detallar cómo se declara una función en

ISBN: 978-631-6557-57-5

algunos lenguajes populares de programación.

En **Python**

Python es un lenguaje de programación de tipado dinámico, lo que significa que no es necesario declarar el tipo de los parámetros o el valor de retorno en la definición de una función (Singh et al., 2023).

```
def nombre_funcion(parametro1, parametro2):
    # Bloque de código
    return valor_retorno
```

Ejemplo:

```
def restar(a, b):
    return a - b
```

En este ejemplo," restar" es el nombre de la función, "a" y "b" son los parámetros, y "return a b" indica que la función retorna la resta de "a" y "b".

En **Java**

Java requiere que se especifiquen los tipos de los parámetros y el tipo de retorno al declarar una función. En Java, las funciones suelen ser métodos de una clase.

```
public TipoRetorno nombreFuncion(TipoParametro1 parametro1, TipoParametro2 parametro2) {
    // Bloque de código
    return valorRetorno;
}
```

Ejemplo:

```
public int restar(int a, int b) {
    return a - b;
}
```

Aquí, "int" antes de "restar" indica que la función retorna un valor de tipo entero.

En **C#**

C# es similar a Java en cuanto a la necesidad de especificar los tipos de datos

ISBN: 978-631-6557-57-5

(Nakov, 2013).

```
public TipoRetorno NombreFuncion(TipoParametro1 parametro1, TipoParametro2 parametro2) {
   // Bloque de código
   return valorRetorno;
}
```

Ejemplo:

```
public int Restar(int a, int b) {
   return a - b;
}
```

En **JavaScript**

JavaScript, al igual que Python, es un lenguaje de programación de tipado dinámico. Las funciones se pueden declarar de varias maneras, pero la forma más básica es:

```
function nombreFuncion(parametro1, parametro2) {
   // Bloque de código
   return valorRetorno;
}
```

Ejemplo:

```
function restar(a, b) {
   return a - b;
}
```

En **C++**

Mientras que JavaScript es un lenguaje de tipado dinámico, C++ es un lenguaje de tipado estático. Esto significa que en C++ se debe declarar explícitamente el tipo de dato de las variables, los parámetros de las funciones y el valor de retorno.

Declaración de funciones en C++

La sintaxis básica para declarar una función en C++ es la siguiente:

```
tipo_de_retorno nombre_funcion(tipo_parametro1 parametro1, tipo_parametro2 parametro2, ...) {
```

ISBN: 978-631-6557-57-5

```
    // Bloque de código
    return valor_de_retorno;
}
```

Ejemplo:

```
int restar(int a, int b) {
    return a - b;
}
```

4. Importancia de los Nombres Descriptivos y Tipos

Nombres Descriptivos: El nombre de una función debe ser claro y descriptivo para indicar qué hace la función. Por ejemplo, "calcularAreaCirculo" es más descriptivo que simplemente "area".

Parámetros: Los parámetros permiten a las funciones recibir datos de entrada para su ejecución. Es importante nombrarlos de manera clara y, en lenguajes de tipado estático, especificar su tipo.

Tipo de Retorno: En lenguajes de tipado estático, es necesario declarar el tipo de dato que la función retornará. Esto proporciona información valiosa sobre lo que se puede esperar de la función y ayuda a evitar errores en tiempo de compilación.

Al seguir estas prácticas, se mejora la legibilidad y mantenibilidad del código, facilitando tanto el desarrollo como la depuración de software.

5. Parámetros y argumentos

Los parámetros y argumentos son conceptos fundamentales en la programación que permiten a las funciones recibir datos desde fuera, procesarlos y, potencialmente, retornar un resultado. Entender cómo se pasan datos a las funciones y la diferencia entre pasar por valor y por referencia es decisivo para el diseño efectivo de funciones y para la manipulación adecuada de los datos dentro de un programa.

Parámetros: Son las variables listadas como parte de la definición de la función. Actúan como placeholders (marcadores de posición) que reciben valores (o referencias a valores) cuando se llama a la función. Los parámetros definen qué tipo de datos espera recibir la función y cuántos.

Argumentos: Son los valores reales o referencias a datos que se pasan a la función en el momento de llamarla. Los argumentos deben coincidir con los parámetros de la función en número y, en lenguajes de tipado estático, en tipo.

ISBN: 978-631-6557-57-5

6. Pasar Datos por Valor

Cuando se pasa un argumento a una función por valor, lo que se transfiere es una copia del valor del argumento. Esto significa que cualquier modificación hecha al parámetro dentro de la función no afectará al dato original en el contexto desde donde se llamó la función.

Ventajas: Seguridad de los datos, debido a que las operaciones dentro de la función no pueden alterar los datos originales.

Desventajas: La duplicación de datos puede aumentar el uso de memoria, especialmente con estructuras de datos grandes.

Ejemplo en C (pasar por valor):

```
void agregarSeis(int numero) {
   numero += 6;
   // Aquí, numero es una copia del argumento pasado, y modificarlo no afecta el valor original fuera de esta función.
}
```

7. Pasar Datos por Referencia

Pasar un argumento por referencia significa pasar una dirección de memoria donde reside el valor, permitiendo a la función acceder directamente al dato original y modificarlo si es necesario.

Ventajas: Permite modificar el valor original y es más eficiente en términos de memoria con datos grandes, debido a que no se crean copias.

Desventajas: Riesgo de efectos secundarios, debido a que las modificaciones en la función afectan al dato original.

Ejemplo en C++ (pasar por referencia):

```
void agregarSeis(int &numero) {
   numero += 6;
   // Aquí, numero es una referencia al argumento pasado, y modificarlo afectará el valor original.
}
```

ISBN: 978-631-6557-57-5

8. Consideraciones de Uso

La elección entre pasar por valor o por referencia depende de varios factores:

Intención de la Función: Si la función necesita modificar el valor original o trabajar con estructuras de datos grandes, pasar por referencia puede ser más adecuado.

Seguridad de los Datos: Para garantizar que los datos no se modifiquen accidentalmente, pasar por valor es preferible.

Eficiencia: Pasar grandes estructuras de datos por valor puede ser costoso en términos de memoria y rendimiento. En estos casos, pasar por referencia es más eficiente.

9. Lenguajes de Programación

El comportamiento predeterminado (pasar por valor o por referencia) varía entre lenguajes de programación. Por ejemplo, en Java, los tipos primitivos se pasan por valor, mientras que los objetos se pasan como referencias a valores. En Python, todo se pasa por asignación, pero debido a la naturaleza de los tipos de datos de Python, el efecto puede parecerse tanto al paso por valor como al paso por referencia, dependiendo de si el tipo de dato es mutable o inmutable.

Entender cómo funcionan los parámetros y argumentos, y la diferencia entre pasar por valor y por referencia, es esencial para la manipulación efectiva de datos en funciones y para el diseño general de programas robustos y eficientes.

10. Valores de retorno

Los valores de retorno son un aspecto fundamental de las funciones en la programación, permitiendo que una función envíe un resultado de vuelta al contexto desde el que fue llamada. Esta capacidad de retorno hace a las funciones herramientas versátiles y poderosas para el procesamiento de datos, la ejecución de cálculos, y la toma de decisiones dentro de un programa. Vamos a explorar con más detalle cómo funcionan los valores de retorno y su importancia.

Un valor de retorno es el dato que una función "devuelve" después de completar sus operaciones. Cuando una función retorna un valor, ese valor se puede asignar a una variable, utilizar en una expresión o pasar como argumento a otra función. No todas las funciones deben retornar un valor explícito; algunas pueden realizarse por sus efectos secundarios (como modificar un archivo o imprimir en pantalla), sin necesidad de retornar

ISBN: 978-631-6557-57-5

un dato específico.

a) Cómo Funcionan los Valores de Retorno

Declaración de la Función: En lenguajes de programación de tipado estático, generalmente se especifica el tipo de dato que la función va a retornar en su declaración. En lenguajes de tipado dinámico, no es necesario especificar el tipo, pero la función aún puede retornar un valor.

Uso de la Palabra Clave "return": La mayoría de los lenguajes de programación utilizan la palabra clave return seguida del valor o de la expresión que la función debe retornar. Cuando la ejecución del código llega a la instrucción return, la función finaliza y devuelve el valor especificado.

Ejemplos de Uso

Retorno de un Valor Simple

En **Python**

```
def multiplicar(a, b):
    return a * b
resultado = multiplicar(6, 3)
print(resultado)  # Imprime: 18
```

En este ejemplo, la función multiplicar retorna el producto de `a` y `b`. El valor retornado se asigna a la variable 'resultado', que luego se imprime.

Retorno de Múltiples Valores

Algunos lenguajes permiten retornar múltiples valores desde una función, comúnmente utilizando tuplas o estructuras similares.

Em **Python**

```
def calcular(a, b):
    suma = a + b
    resta = a - b
    return suma, resta
```

ISBN: 978-631-6557-57-5

```
resultado_suma, resultado_resta = calcular(12, 6)
print(resultado_suma, resultado_resta)  # Imprime: 18 6
```

Este código retorna dos valores: la suma y la resta de los argumentos.

Utilización de los Valores de Retorno

Los valores retornados por una función pueden ser utilizados de varias maneras:

Asignación a Variables: Como se mostró en los ejemplos, el valor retornado se puede asignar a una variable para su uso posterior.

En Expresiones: Los valores de retorno pueden usarse directamente en expresiones, por ejemplo, resultado = suma(12, 4) * 10.

Como Argumentos: Los valores de retorno pueden pasarse como argumentos a otras funciones, en cadenas de llamadas de función.

Los valores de retorno son importantes para la modularidad y la reusabilidad del código, permitiendo que las funciones se comuniquen entre sí y con el resto del programa. Facilitan la abstracción de la lógica en bloques discretos que realizan tareas específicas y devuelven resultados, lo que contribuye a la claridad, mantenibilidad y eficiencia del código. La habilidad para retornar valores hace a las funciones herramientas flexibles y poderosas en la construcción de programas complejos.

11. Ámbito de variables

El ámbito de las variables, también conocido como alcance, define en qué partes de un programa está disponible una variable, es decir, dónde puede ser leída o modificada. Comprender la diferencia entre ámbito local y global y cómo estos afectan el acceso a los datos dentro de las funciones es fundamental para escribir código claro y evitar errores difíciles de detectar. Vamos a explorar estos conceptos con más detalle.

a) Ámbito Local

El ámbito local se refiere al contexto dentro de una función o bloque de código específico donde se ha declarado una variable. Las variables definidas dentro de una función tienen un ámbito local; es decir, solo existen y pueden ser accedidas o

ISBN: 978-631-6557-57-5

modificadas dentro de esa función.

Características del Ámbito Local:

- Las variables locales se crean al entrar en la función y se destruyen al salir de ella.
- Una función puede acceder a variables locales definidas dentro de ella, pero no a las variables locales de otras funciones.
- Si una variable local tiene el mismo nombre que una variable global, la variable local "oculta" a la variable global dentro de su ámbito.

Ejemplo en **Python**:

```
def miFuncion():
    variableLocal = "Estoy dentro de miFuncion"
    print(variableLocal)  # Acceso válido
     Para ejecutar
miFuncion()
     Para tratar de imprimir
print(variableLocal)  #  Error: variableLocal no está definida en este ámbito
```

b) Ámbito Global

El ámbito global se refiere a las variables definidas en el nivel principal del programa, fuera de cualquier función o bloque específico. Estas variables pueden ser accedidas desde cualquier parte del programa, incluidas todas las funciones definidas.

Características del Ámbito Global:

- Las variables globales están disponibles a lo largo de todo el programa desde el punto en que son definidas.
- Se debe tener cuidado al modificar variables globales dentro de funciones, debido a que esto puede llevar a efectos secundarios no deseados y hacer que el código sea más difícil de entender y mantener.

Ejemplo en **Python**:

```
variableGlobal = "Estoy fuera de cualquier función"
```

ISBN: 978-631-6557-57-5

```
def miFuncion():
    print(variableGlobal)  # Acceso válido
```

Para ejecutar:

```
miFuncion()
```

Para imprimir:

```
print(variableGlobal)  # Acceso válido
```

En algunos lenguajes, como Python, si intentas modificar una variable global dentro de una función sin declararla como tal, se creará una nueva variable local con el mismo nombre. Para modificar la variable global efectivamente, se debe utilizar la palabra clave `global`.

Ejemplo en Python:

```
def miFuncion():
    global variableGlobal
    variableGlobal = 10

miFuncion()  # no muestra nada
print(variableGlobal)  # Imprime: 10
```

El ámbito de las variables juega un papel fundamental en la estructuración de un programa:

Seguridad: El ámbito local permite encapsular variables dentro de funciones, reduciendo el riesgo de colisiones de nombres y efectos secundarios no deseados.

Legibilidad: Un buen uso del ámbito de variables ayuda a mantener el código organizado y claro, facilitando su comprensión y mantenimiento.

Eficiencia: La destrucción de variables locales tras la ejecución de una función ayuda a la gestión eficiente de la memoria.

Entender cómo funcionan los ámbitos local y global es esencial para cualquier programador, ya que afecta directamente cómo se estructuran los datos y se diseñan las interacciones dentro de un programa.

12. Funciones recursivas

ISBN: 978-631-6557-57-5

Las funciones recursivas son un concepto fascinante y poderoso en la programación, permitiendo a los desarrolladores escribir soluciones concisas y elegantes para problemas que de otro modo requerirían complejas estructuras de bucle. La recursión ocurre cuando una función se llama a sí misma directamente o indirectamente, permitiendo que el problema se resuelva en términos de una versión más pequeña de sí mismo.

Una función recursiva tiene dos características fundamentales:

a. Caso Base: Una condición que detiene la recursión, evitando que la función se llame a sí misma infinitamente. Este caso base devuelve un valor sin realizar una llamada recursiva.

b. Llamada Recursiva: La función se llama a sí misma con un conjunto de parámetros que se acercan al caso base.

Ejemplo 1: Factorial de un Número

El cálculo del factorial de un número (n!) es un ejemplo clásico de recursión. Matemáticamente, el factorial de un número n es el producto de todos los enteros positivos menores o iguales a n, y se define como:

n! = n x (n - 1) x (n - 2) x ….. x 2 x 1

0! = 1(por definición)

En términos de recursión:

n! = n x (n-1)!

0! = 1 (caso base)

Implementación en **Python**:

Ejemplo 1: Factorial

```
def factorial(n):
    if n == 0:  # Caso base
        return 1
    else:  # Llamada recursiva
        return n * factorial(n-1)
```

Ejemplo 2: Fibonacci

ISBN: 978-631-6557-57-5

La secuencia de Fibonacci es otro ejemplo donde la recursión es útil. La secuencia comienza con dos números, 0 y 1, y cada número subsiguiente es la suma de los dos anteriores.

En términos de recursión, el enésimo número de Fibonacci se define como:

fib(n) = fib(n-1) + fib(n-2)

fib(0) = 0 y fib(1) = 1 (casos base)

Implementación en **Python**:

```
def fibonacci(n):
    if n == 0:  # Caso base 1
        return 0
    elif n == 1:  # Caso base 2
        return 1
    else:  # Llamada recursiva
        return fibonacci(n-1) + fibonacci(n-2)
```

Ventajas:

Claridad y Elegancia: Las soluciones recursivas pueden ser más claras y fáciles de entender que sus equivalentes iterativos, especialmente para problemas que se prestan naturalmente a la recursión.

Menos Código: A menudo, la recursión puede lograr en pocas líneas lo que requeriría muchas más líneas con bucles y estructuras de control adicionales.

Desventajas:

Eficiencia: La recursión puede ser menos eficiente en términos de tiempo y espacio. Cada llamada a la función añade una nueva capa a la pila de llamadas, lo que puede llevar a un consumo significativo de memoria y, en casos extremos, a desbordamientos de pila.

Complejidad: En algunos casos, las soluciones recursivas pueden ser difíciles de seguir y entender, especialmente para quienes no están familiarizados con la recursión

ISBN: 978-631-6557-57-5

o cuando el caso base no está bien definido.

4.2 Ejercicios de aplicación

4.2.1 Ejercicios resueltos

1. Escribir una función para completar un saludo con el nombre de la persona.

Pseudocódigo

```
Algoritmo Saludar
    Definir nombre Como Cadena
    Definir mensaje Como Cadena

    Escribir "Ingrese un nombre:"
    Leer nombre

    mensaje ← "Hola " + nombre + " ¿cómo estás?"

    Escribir mensaje
FinAlgoritmo
```

Definir nombre Como Cadena: Declara una variable llamada nombre para almacenar el nombre que se ingresará.

Definir mensaje Como Cadena: Declara una variable llamada mensaje para almacenar el saludo final.

Escribir "Ingrese un nombre:": Muestra un mensaje en pantalla solicitando al usuario que ingrese un nombre.

Leer nombre: Lee el nombre ingresado por el usuario y lo almacena en la variable nombre.

mensaje <- "Hola " + nombre + " ¿cómo estás?": Concatena las cadenas y el valor de la variable nombre para formar el saludo y lo asigna a la variable mensaje.

Escribir mensaje: Imprime en pantalla el valor de la variable mensaje, mostrando el saludo personalizado.

```
PSeInt - Ejecutando proceso SALUDAR
*** Ejecución Iniciada. ***
Ingrese un nombre:
> Silvanita
Hola Silvanita ¿cómo estás?
*** Ejecución Finalizada. ***
```

ISBN: 978-631-6557-57-5

Diagrama de flujo

En **Python**

```
[1]  def saludar(nombre):
         return "Hola" + nombre + " ¿cómo estás?"
```

```
# Invocar la función y pasarle un argumento
mensaje = saludar(" Silvanita. ")
print(mensaje) # Imprime "Hola Silvanita. ¿cómo estás?"
```

```
Hola Silvanita.  ¿cómo estás?
```

La función saludar toma un argumento llamado nombre. Cuando esta función se invoca, retorna un saludo que concatena la palabra "Hola" con el argumento nombre y la frase " ¿cómo estás?".

Luego, la función se invoca pasando el argumento "Silvanita. ", y el resultado retornado se asigna a la variable **mensaje**. Finalmente, se imprime el contenido de la variable mensaje, que resulta en el mensaje "Hola Silvanita. ¿cómo estás?" que se muestra en la consola.

ISBN: 978-631-6557-57-5

En **JavaScript**

En JavaScript, la función saludar se define de manera similar a Python. Sin embargo, para imprimir en la consola, usamos console.log en lugar de print. Además, utilizamos let para declarar la variable mensaje.

Las funciones son fundamentales para la abstracción y la reutilización de código, permitiendo crear bloques de construcción que puedes combinar de maneras complejas para construir programas más grandes y poderosos.

En **C++**

```cpp
#include <iostream>
#include <string>

std::string saludar(const std::string& nombre) {
    return "Hola " + nombre + " ¿cómo estás?";
}

int main() {
    std::string mensaje = saludar("Silvanita.");
    std::cout << mensaje << std::endl; // Imprime "Hola
        Silvanita. ¿cómo estás?"
    return 0;
}
```

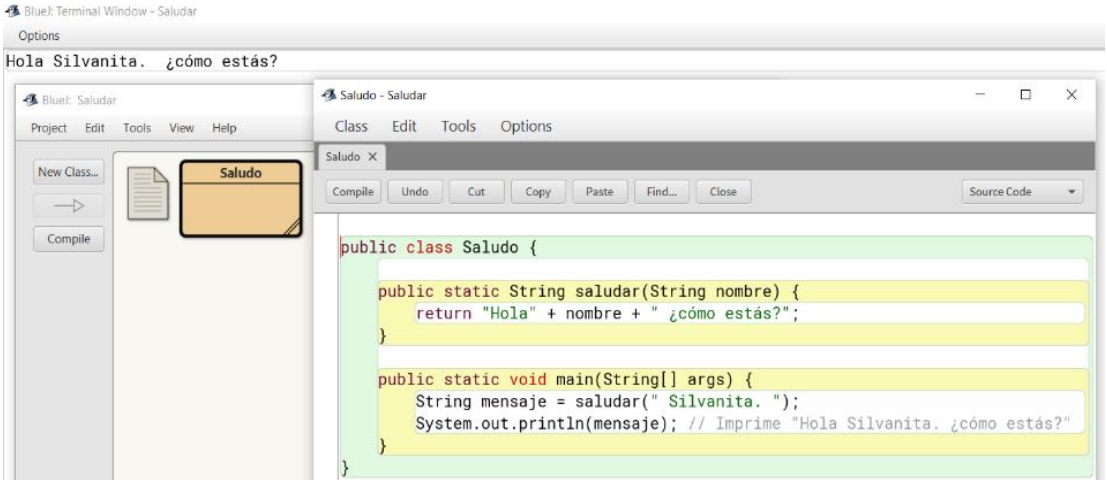
La biblioteca <iostream> se utiliza para la entrada y salida estándar, y la biblioteca <string> se usa para manejar cadenas de texto.

La función saludar toma un argumento nombre de tipo const std::string& para asegurar que no se modifique la cadena original y evitar copias innecesarias.

En la función main, se llama a saludar con el nombre y se imprime el mensaje.

ISBN: 978-631-6557-57-5

En **Java**



Se muestra el entorno de desarrollo integrado (IDE) que ejecuta código en Java. El código consiste en una clase **Saludo** que contiene dos métodos:

1. **saludar:** Es un método static que recibe un parámetro **nombre** de tipo `String` y retorna una cadena de texto que incluye el saludo "Hola" seguido del **nombre** proporcionado y la pregunta " ¿cómo estás?".
2. **main:** Este es el método principal (main) de cualquier programa de Java, el punto de entrada del programa. Dentro de él, se llama al método saludar con el argumento "Silvanita. ". El resultado de esta llamada se almacena en la variable **mensaje**. Luego se imprime **mensaje** utilizando System.out.println(mensaje), lo que da como resultado el texto "Hola Silvanita. ¿cómo estás?" en la consola del IDE.

El comentario después de // en la última línea del método main explica que el código imprime el mensaje "Hola Silvanita. ¿cómo estás?".

2. En Java, desarrolle un método que reciba dos números como parámetros y retorne la suma de ambos. Incluir comentarios en tu código que expliquen cada paso.

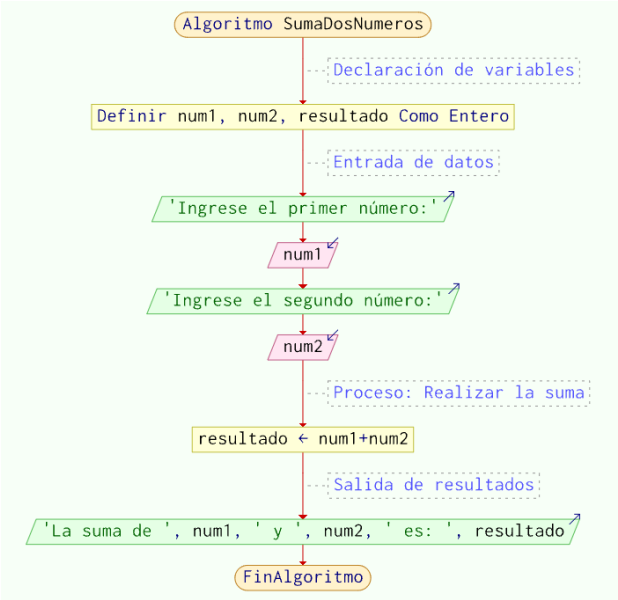
Pseudocódigo

ISBN: 978-631-6557-57-5

```
Algoritmo SumaDosNumeros

    // Declaración de variables
    Definir num1, num2, resultado Como Entero

    // Entrada de datos
    Escribir "Ingrese el primer número:"
    Leer num1
    Escribir "Ingrese el segundo número:"
    Leer num2

    // Proceso: Realizar la suma
    resultado ← num1 + num2

    // Salida de resultados
    Escribir "La suma de ", num1, " y ", num2, " es: ", resultado

FinAlgoritmo
```

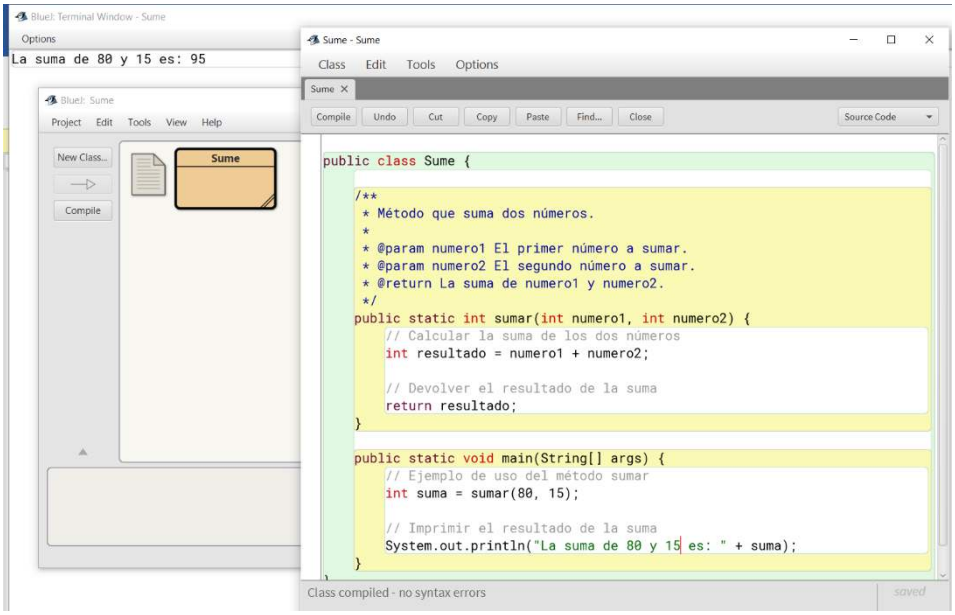
```
PSeInt - Ejecutando proceso SUMADOSNUME...
*** Ejecución Iniciada. ***
Ingrese el primer número:
> 80
Ingrese el segundo número:
> 15
La suma de 80 y 15 es: 95
*** Ejecución Finalizada. ***
```

Diagrama de flujo



ISBN: 978-631-6557-57-5

En **Java**



-Las etiquetas @param se utilizan en los comentarios de la documentación en Java, conocidos como JavaDoc, en este ejemplo proporcionan información sobre los parámetros numero1 y numero2 que se pasan al método sumar.

-Se define una clase Sume que contiene el método sumar.

-El método sumar es static, lo que significa que puede ser llamado sin necesidad de crear una instancia de la clase Sume.

-sumar toma dos parámetros enteros (int), numero1 y numero2.

-Dentro del método sumar, se realiza la suma de los dos parámetros y se almacena el resultado en una variable local llamada resultado.

-Luego, sumar devuelve el resultado de esta operación.

-En el método main, que es el punto de entrada del programa, se llama al método sumar con dos números ejemplo, 80 y 15.

-Finalmente, se imprime el resultado de la suma, que en este caso será 95.

3. Hacer los módulos en Python para que el usuario ingrese su nombre y el programa devueva un saludo basado en el nombre ingresado.

ISBN: 978-631-6557-57-5

```python
# Módulo principal
def principal():
    nombre = obtener_nombre()
    saludo = generar_saludo(nombre)
    imprimir_saludo(saludo)

# Módulo para obtener el nombre
def obtener_nombre():
    return input("Ingresa tu nombre: ")

# Módulo para generar un saludo
def generar_saludo(nombre):
    return f"Hola, {nombre} Bienvenido."

# Módulo para imprimir el saludo
def imprimir_saludo(saludo):
    print(saludo)

# Ejecuta el programa
if __name__ == "__main__":
    principal()
```

En este código tenemos la línea if __name__ == "__main__":

__name__ es una variable especial. Su valor cambia dependiendo de cómo se ejecute el archivo: Cuando se ejecuta un archivo directamente (es decir, desde la terminal o un entorno de desarrollo), el valor de __name__ es "__main__".

__main__" es una cadena reservada en Python, la utiliza como identificador para el archivo principal que se está ejecutando directamente.

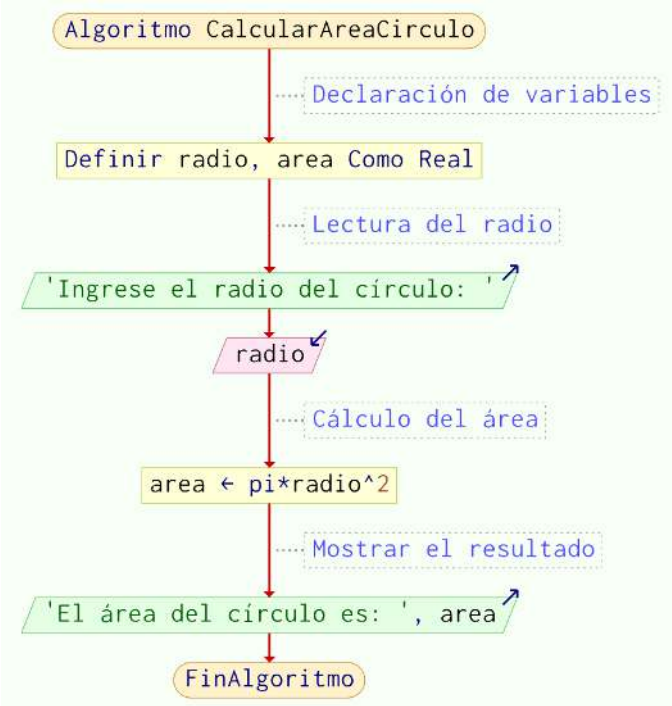
4. Calcular el área del circulo

Pseudocódigo

ISBN: 978-631-6557-57-5

```
Algoritmo CalcularAreaCirculo

    // Declaración de variables
    Definir radio, area Como Real

    // Lectura del radio
    Escribir "Ingrese el radio del círculo: "
    Leer radio

    // Cálculo del área
    area ← pi * radio↑2

    // Mostrar el resultado
    Escribir "El área del círculo es: ", area

FinAlgoritmo
```

Diagrama de flujo



Implementación

ISBN: 978-631-6557-57-5

```python
import math

def area_circulo(radio):
    return math.pi * radio**2

# Ejemplo de uso
radio = 10
print("El área del círculo es:", area_circulo(radio))
```

```
El área del círculo es: 314.1592653589793
```

Se importa la biblioteca math que proporciona funciones matemáticas comunes, como el valor de pi (math.pi), operaciones trigonométricas, logarítmicas, etc.

Se define la función llamada def area_circulo(radio) la cual calcula el área del círculo (área = π * radio^2).

Cuando llamamos a la función, se le proporciona un valor para el radio.

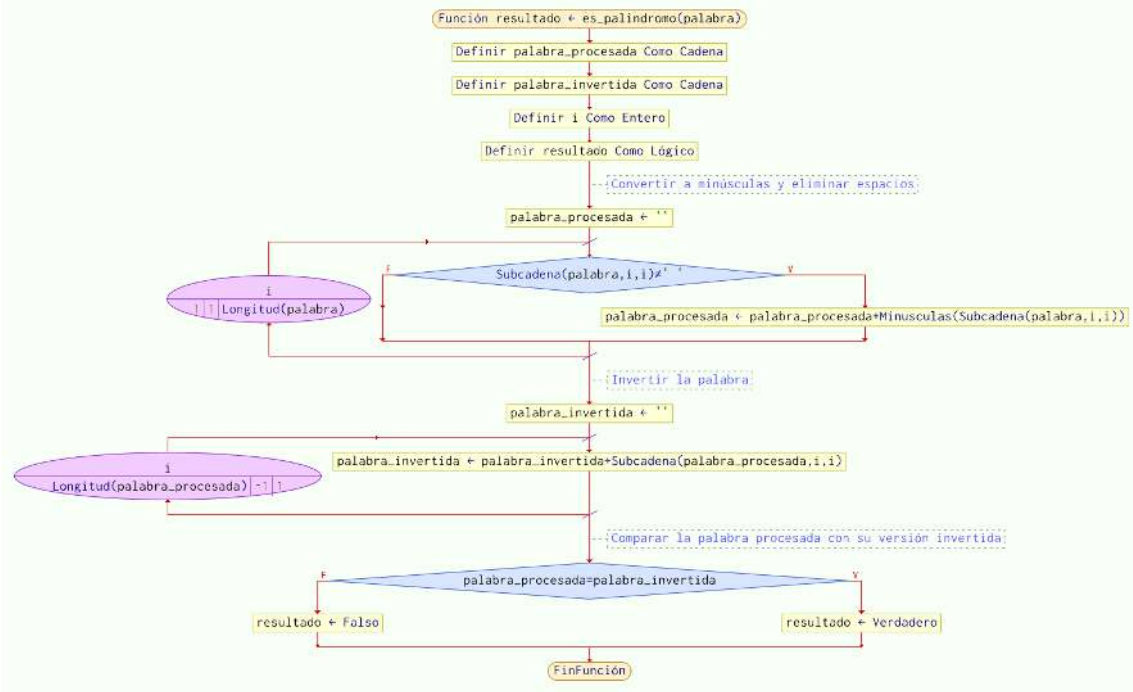
5. Realizar una función que determine si una palabra es un palíndromo (se lee igual de izquierda a derecha que de derecha a izquierda).

Pseudocódigo

```
Funcion resultado ← es_palindromo(palabra)
    Definir palabra_procesada Como Cadena
    Definir palabra_invertida Como Cadena
    Definir i Como Entero
    Definir resultado Como Logico
    // Convertir a minúsculas y eliminar espacios
    palabra_procesada ← ""
    Para i ← 1 Hasta Longitud(palabra) Con Paso 1 Hacer
        Si Subcadena(palabra, i, i) ≠ " " Entonces
            palabra_procesada ← palabra_procesada + Minusculas(Subcadena(palabra, i, i))
        FinSi
    FinPara
    // Invertir la palabra
    palabra_invertida ← ""
    Para i ← Longitud(palabra_procesada) Hasta 1 Con Paso -1 Hacer
        palabra_invertida ← palabra_invertida + Subcadena(palabra_procesada, i, i)
    FinPara
    // Comparar la palabra procesada con su versión invertida
    Si palabra_procesada = palabra_invertida Entonces
        resultado ← Verdadero
    Sino
        resultado ← Falso
    FinSi
FinFuncion
Algoritmo EjemploPalindromo
    Definir palabra Como Cadena
    palabra ← "Ana"
    Escribir "¿Es palíndromo? ", es_palindromo(palabra)
FinAlgoritmo
```

ISBN: 978-631-6557-57-5

Diagrama de flujo



Implementación

```
def es_palindromo(palabra):
    palabra = palabra.lower().replace(" ", "")
    return palabra == palabra[::-1]
```

```
# Ejemplo
palabra = "Ana"
print("¿Es palíndromo?", es_palindromo(palabra))
```

```
¿Es palíndromo? True
```

La función llamada es_palindromo toma una palabra como entrada y determina si la palabra es un palíndromo.

palabra.lower(): Convierte todos los caracteres de la palabra a minúsculas. Esto es importante para que la comparación no se vea afectada por mayúsculas y minúsculas.

replace(" ", ""): Elimina todos los espacios en blanco de la palabra. Esto permite que la función también funcione con frases y no solo con palabras individuales.

ISBN: 978-631-6557-57-5

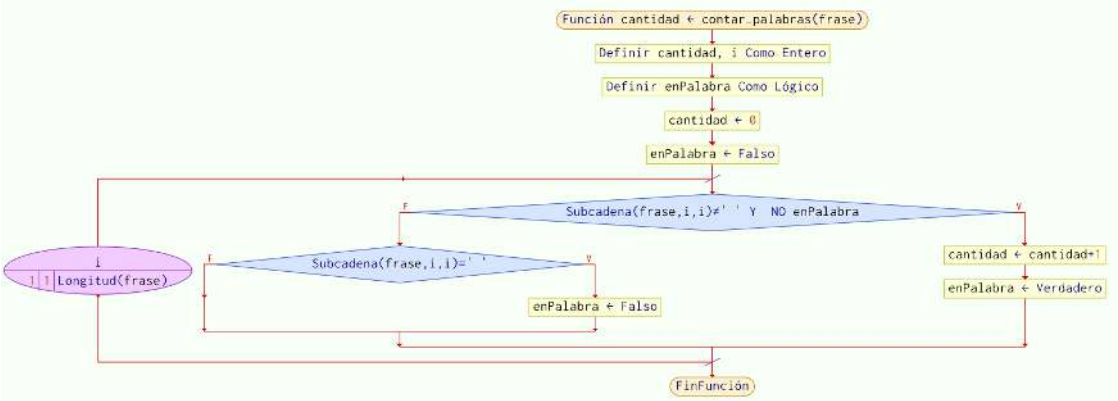
return palabra == palabra[::-1]: Esta línea compara la palabra original con su versión invertida y devuelve True si son iguales, lo que significa que la palabra es un palíndromo, o False si son diferentes.

6. Realizar una función que cuente cuántas palabras hay en una frase.

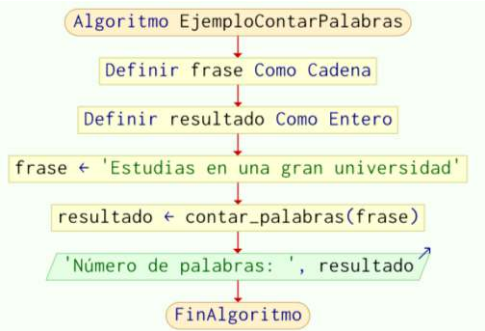
Pseudocódigo

```
Funcion cantidad ← contar_palabras(frase)
    Definir cantidad, i Como Entero
    Definir enPalabra Como Logico

    cantidad ← 0
    enPalabra ← Falso

    Para i ← 1 Hasta Longitud(frase) Con Paso 1 Hacer
        Si Subcadena(frase, i, i) ≠ " " Y NO enPalabra Entonces
            cantidad ← cantidad + 1
            enPalabra ← Verdadero
        SiNo
            Si Subcadena(frase, i, i) = " " Entonces
                enPalabra ← Falso
            FinSi
        FinSi
    FinPara
FinFuncion

Algoritmo EjemploContarPalabras
    Definir frase Como Cadena
    Definir resultado Como Entero

    frase ← "Estudias en una gran universidad"
    resultado ← contar_palabras(frase)

    Escribir "Número de palabras: ", resultado
FinAlgoritmo
```

Diagrama de flujo



ISBN: 978-631-6557-57-5



Implementación

```
def contar_palabras(frase):
    return len(frase.split())
```

```
# Ejemplo
frase = "Estudias en una gran universidad"
print("Número de palabras:", contar_palabras(frase))
```

```
Número de palabras: 5
```

Se define la función contar_palabras, la cual será llamada en el ejemplo. Dentro de los paréntesis, se coloca el parámetro de la función. En este caso, el parámetro es frase. Esto significa que cuando se llame a la función, se tendrá que proporcionarle una frase como entrada.

Mediante la palabra clave retun se devuelve un valor desde la función.

La función len que esta integrada en Python cuenta el número de elementos en un objeto.

El método split divide una cadena de texto (la frase) en una lista de palabras, utilizando los espacios en blanco como separadores por defecto.

7. Diseñar un algoritmo que ofrezca un menú con opciones para realizar sumas, restas, multiplicaciones y divisiones. El usuario debe poder elegir la operación deseada y luego ingresar dos números para realizar la operación seleccionada. Implementar en Python.

Pseudocódigo

```
Algoritmo Calculadora_Basica
  definir opción, numero1, numero2, resultado Como Real
  Repetir
    Escribir "Seleccione la operación a realizar:"
```

ISBN: 978-631-6557-57-5

```
Escribir "1. Suma"
Escribir "2. Resta"
Escribir "3. Multiplicación"
Escribir "4. División"
Escribir "5. Salir"
Leer opción
Si opción < 1 o opción > 5 Entonces
   Escribir "Opción no válida, por favor intente de nuevo."
SiNo
     Si opción >= 1 y opción <= 4 Entonces
   Escribir "Ingrese el primer número:"
   Leer numero1
   Escribir "Ingrese el segundo número:"
   Leer numero2
FinSi
Según opción Hacer
    1:
      resultado <- numero1 + numero2
      Escribir "El resultado de la suma es: ", resultado
    2:
      resultado <- numero1 - numero2
      Escribir "El resultado de la resta es: ", resultado
    3:
      resultado <- numero1 * numero2
      Escribir "El resultado de la multiplicación es: ", resultado
    4:
      Si numero2 = 0 Entonces
         Escribir "Error: No se puede dividir por cero."
      Sino
         resultado <- numero1 / numero2
         Escribir "El resultado de la división es: ", resultado
```

ISBN: 978-631-6557-57-5

```
            FinSi
        5:
            Escribir "Gracias por usar la calculadora."
    FinSegun
  FinSi
 Hasta Que opción = 5
FinAlgoritmo
```

Interpretación del pseudocódigo:

Definición de Variables: Las variables opción, numero1, numero2, y resultado son definidas como de tipo Real, lo que implica que pueden contener números decimales.

Ciclo Repetitivo: El algoritmo utiliza un bucle Repetir que sigue ejecutándose hasta que el usuario decida salir seleccionando la opción 5 (Hasta Que opción = 5).

Menú de Opciones:

- Se muestra al usuario un menú con las siguientes opciones: Suma (1), Resta (2), Multiplicación (3), División (4) y Salir (5).
- El usuario debe ingresar su elección, que se lee y almacena en la variable opción.

Validación de la Opción:

- Se verifica si la opción ingresada es válida (entre 1 y 5). Si no es válida, se muestra un mensaje de error y se presenta de nuevo el menú.
- Si la opción es válida y está entre 1 y 4, el programa solicita al usuario que ingrese dos números (numero1 y numero2), que se utilizarán para realizar la operación matemática elegida.

Ejecución de la Operación Seleccionada:

- Utilizando la estructura de control Según, el programa selecciona la operación matemática basada en la opción ingresada:
    - Suma (1): Se suma numero1 y numero2 y se muestra el resultado.
    - Resta (2): Se resta numero1 de numero2 y se muestra el resultado.
    - Multiplicación (3): Se multiplica numero1 por numero2 y se muestra el resultado.
    - División (4): Antes de dividir, se verifica que numero2 no sea cero para evitar

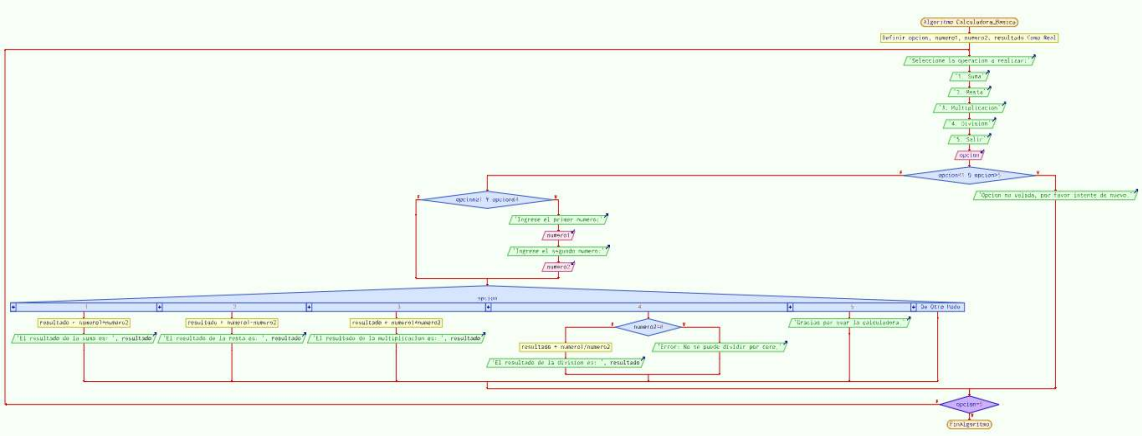
ISBN: 978-631-6557-57-5

un error matemático. Si numero2 es cero, se muestra un mensaje de error; de lo contrario, se realiza la división y se muestra el resultado.

- Salir (5): Se muestra un mensaje de agradecimiento por usar la calculadora y se termina el bucle.

Fin del Programa: El algoritmo termina cuando el usuario selecciona la opción 5 (Salir).

Diagrama de flujo



ISBN: 978-631-6557-57-5

Implementación de la calculadora en Python

```
Seleccione la operación a realizar:
1. Suma
2. Resta
3. Multiplicación
4. División
5. Salir
1
Ingrese el primer número:
2
Ingrese el segundo número:
3
El resultado de la suma es:  5.0
Seleccione la operación a realizar:
1. Suma
2. Resta
3. Multiplicación
4. División
5. Salir
```

```
        numero1 = float(input())
        print("Ingrese el segundo número:")
        numero2 = float(input())

        # Realizar la operación seleccionada
        if opcion == 1:
            resultado = numero1 + numero2
            print("El resultado de la suma es: ", resultado)
        elif opcion == 2:
            resultado = numero1 - numero2
            print("El resultado de la resta es: ", resultado)
        elif opcion == 3:
            resultado = numero1 * numero2
            print("El resultado de la multiplicación es: ", resultado)
        elif opcion == 4:
            if numero2 == 0:
                print("Error: No se puede dividir por cero.")
            else:
                resultado = numero1 / numero2
                print("El resultado de la división es: ", resultado)

# Llamar a la función para iniciar la calculadora
calculadora_basica()
```

8. Hacer una caluladora que sume, reste, multiplique, divida, realice el exponencial, la raiza cuadrada y calcule el seno de un ángulo en grados. Usar funciones para cada operación matemática y la librería math.

ISBN: 978-631-6557-57-5

```python
import math
def sumar(a, b):
    """Sumar dos números."""
    return a + b
def restar(a, b):
    """Resta dos números."""
    return a - b

def multiplicar(a, b):
    """Multiplica dos números."""
    return a * b

def dividir(a, b):
    """Divide dos números."""
    if b != 0:
        return a / b
    else:
        return "No se puede dividir entre 0"

def raiz_cuadrada(a):
    """Calcula la raíz cuadrada de un número."""
    return math.sqrt(a)

def exponencial(a, b):
    """Calcula el exponencial de un número."""
    return math.pow(a,b)
```

```python
def seno_en_grados():
    """
    Solicita un ángulo en grados, calcula su seno y lo devuelve.
    """
    angulo_grados = float(input("Introduce el ángulo en grados: "))
    angulo_radianes = math.radians(angulo_grados)  # Convertimos grados a radianes
    resultado = math.sin(angulo_radianes)  # Calculamos el seno
    print(f"Seno de {angulo_grados}°: {resultado}")
    return resultado

# Forma de utilizar
if __name__ == "__main__":
    print("Operaciones disponibles:")
    print("1. Sumar")
    print("2. Restar")
    print("3. Multiplicar")
    print("4. Dividir")
    print("5. Raiz Cuadrada")
    print("6. Exponencial")
    print("7. Seno de un ángulo en grados")

    # Solicitar al usuario que elija una opción
    opcion = int(input("Elige una opción (1-7): "))
```

ISBN: 978-631-6557-57-5

```
opcion = int(input("Elige una opción (1-7): "))

if opcion == 1:
    a = float(input("Introduce el primer número: "))
    b = float(input("Introduce el segundo número: "))
    print(f"Resultado: {a + b}")  # Falta la función sumar, puedes agregarla
elif opcion == 2:
    a = float(input("Introduce el primer número: "))
    b = float(input("Introduce el segundo número: "))
    print(f"Resultado: {restar(a, b)}")
elif opcion == 3:
    a = float(input("Introduce el primer número: "))
    b = float(input("Introduce el segundo número: "))
    print(f"Resultado: {multiplicar(a, b)}")
elif opcion == 4:
    a = float(input("Introduce el primer número: "))
    b = float(input("Introduce el segundo número: "))
    print(f"Resultado: {dividir(a, b)}")
elif opcion == 5:
    a = float(input("Introduce el número: "))
    print(f"Resultado: {raiz_cuadrada(a)}")
elif opcion == 6:
    a = float(input("Introduce la base: "))
    b = float(input("Introduce el exponente: "))
    print(f"Resultado: {exponencial(a, b)}")
elif opcion == 7:
    seno_en_grados()
else:
    print("Opción no válida.")
```

```
Operaciones disponibles:
1. Sumar
2. Restar
3. Multiplicar
4. Dividir
5. Raiz Cuadrada
6. Exponencial
7. Seno de un ángulo en grados
Elige una opción (1-7): 7
Introduce el ángulo en grados: 45
Seno de 45.0°: 0.7071067811865475
```

La calculadora importa el módulo math para realizar operaciones matemáticas avanzadas. Define funciones básicas (sumar, restar, multiplicar, dividir) que operan con dos números. Incluye funciones más complejas como raíz cuadrada, exponencial y seno en grados. Muestra un menú al usuario con 7 opciones diferentes para elegir la operación deseada. Solicita los números necesarios según la operación seleccionada (uno o dos números). Procesa la operación elegida y muestra el resultado, incluyendo manejo de

ISBN: 978-631-6557-57-5

errores.

9. Hacer el pseudocódigo, diagrama de flujo y programar en Python, JavaScript para el caso en el que se solicita al usuario que indique cuántos números de la secuencia de Fibonacci desea generar y luego mostrar esos números.

Pseudocódigo

```
Algoritmo Fibonacci
    Definir num, i Como Entero
    Definir a, b, siguiente Como Entero
    Escribir "Ingrese cuantos números de la secuencia de Fibonacci desea generar:"
    Leer num
    Si num <= 0 Entonces
        Escribir "Ingrese un número mayor que 0."
    SiNo
        a <- 0
        b <- 1
      Si num = 1 Entonces
            Escribir "Fibonacci: ", a
       SiNo
            Escribir "Fibonacci: ", a, ", ", b
            Para i Desde 3 Hasta num Con Paso 1 Hacer
                siguiente <- a + b
                Escribir ", ", siguiente
                a <- b
                b <- siguiente
            FinPara
        FinSi
    FinSi
FinAlgoritmo
```

A continuación, una breve descripción del pseudocódigo:

Definición de Variables: Se define num para almacenar la cantidad de números

ISBN: 978-631-6557-57-5

Fibonacci que el usuario desea. a, b, y siguiente se utilizan para calcular la secuencia de Fibonacci.

Entrada del Usuario: Se solicita al usuario que indique cuántos números de la secuencia de Fibonacci desea ver.

Validación de Entrada: Se verifica si el número ingresado es menor o igual a cero, y en ese caso, se le solicita ingresar un número válido.

Inicialización de Variables: Se inicializan a y b con los dos primeros números de la secuencia de Fibonacci, que son 0 y 1, respectivamente.

Generación de la Secuencia:

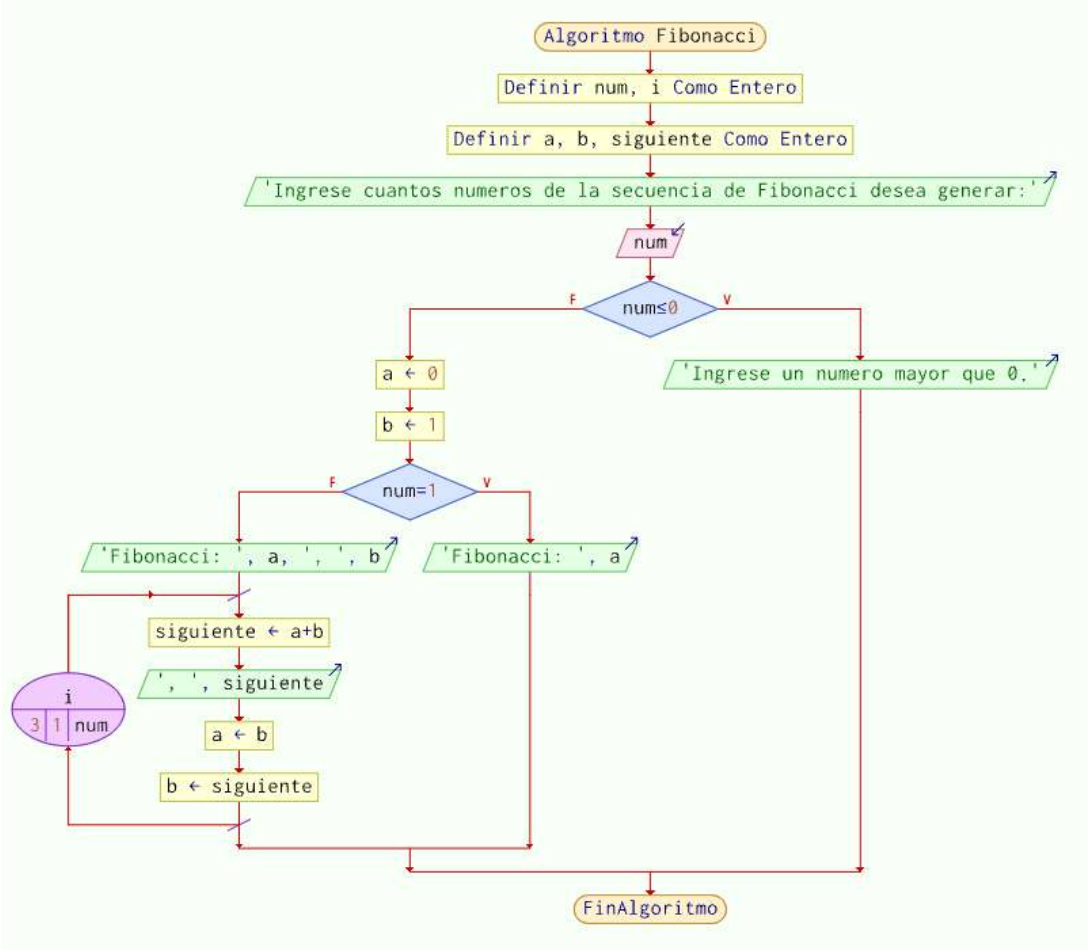
Si el usuario solo quiere el primer número, se muestra solo a.

Si el número solicitado es mayor que 1, se muestra a y b como los dos primeros números.

Luego, se usa un bucle Para para calcular los siguientes números de la secuencia hasta alcanzar el número solicitado por el usuario. En cada iteración, se calcula el siguiente número sumando a y b, se imprime este número, y se actualizan a y b para las siguientes iteraciones.

ISBN: 978-631-6557-57-5

Diagrama de flujo



El diagrama de flujo representa un algoritmo para generar números de la secuencia de Fibonacci. Inicia definiendo las variables: num, i, a, b, siguiente. Luego se solicita al usuario que ingrese cuántos números de la secuencia desea generar. Si el número ingresado es menor o igual a cero, se le pide que ingrese un número mayor que cero. Si el número ingresado es uno, se muestra el primer número de la secuencia: 0.

Para un número ingresado mayor a uno, inicializa a en 0, b en 1, y en un bucle, calcula el siguiente número sumando a y b, muestra el valor de siguiente, y actualiza los valores de a y b con los valores de b y siguiente respectivamente. Este bucle se repite hasta que se hayan generado la cantidad de números de la secuencia Dde Fibonacci deseada. El flujo termina con la etiqueta FinAlgoritmo.

Implementación

ISBN: 978-631-6557-57-5

En Phyton

El código en Python define una función llamada fibonacci_secuencia que toma un parámetro num, que representa el número de términos de la secuencia de Fibonacci que el usuario desea generar.

```
def fibonacci_secuencia(num):
    if num <= 0:
        print("Ingrese un número mayor que 0.")
        return

    a, b = 0, 1
    if num == 1:
        print(f"Fibonacci: {a}")
    else:
        print(f"Fibonacci: {a}, {b}", end="")
        for i in range(3, num + 1):
            siguiente = a + b
            print(f", {siguiente}", end="")
            a, b = b, siguiente

fibonacci_secuencia(10)
```

```
Fibonacci: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34
```

Aquí está lo que hace cada parte del código:

1) Verificación de la entrada: Primero, el código verifica si el número ingresado (num) es menor o igual a cero. Si lo es, imprime un mensaje solicitando al usuario que ingrese un número mayor que cero y termina la ejecución de la función.

2) Variables iniciales: Si el número ingresado es válido, inicializa dos variables, a y b, que representan los dos primeros números de la secuencia de Fibonacci. a es inicializado en 0 y b en 1.

3) Caso especial para num = 1: Si el usuario solo desea el primer número de la secuencia, el programa simplemente imprime 0.

4) Generación de la secuencia: Si el usuario desea más de uno, el código entra en un bucle for que comienza en 3 (porque los dos primeros números ya están definidos) y continúa hasta el número deseado. Dentro del bucle:

- Calcula el siguiente número de la secuencia sumando a y b.
- Imprime el número siguiente, con una coma delante para separarlo de los anteriores.
- Actualiza el valor de a para que sea igual a b y el valor de b para que sea igual

ISBN: 978-631-6557-57-5

al siguiente número calculado. Esto prepara las variables para el próximo ciclo del bucle.

El resultado es una lista de números de la secuencia de Fibonacci, impresos en la misma línea y separados por comas.

Por ejemplo, si llamamos a la función con el argumento 10, imprimirá los primeros 10 números de la secuencia de Fibonacci, como se ha demostrado en el resultado del código anterior.

En JavaScript

main.js

```javascript
function fibonacciSecuencia(num) {
    if (num <= 0) {
        console.log("Ingrese un número mayor que 0.");
        return;
    }

    let a = 0, b = 1, siguiente;
    if (num === 1) {
        console.log("Fibonacci: " + a);
    } else {
        process.stdout.write("Fibonacci: " + a + ", " + b);
        for (let i = 3; i <= num; i++) {
            siguiente = a + b;
            process.stdout.write(", " + siguiente);
            a = b;
            b = siguiente;
        }
    }
}

// Call the function with an example value
fibonacciSecuencia(8);
```

Run | Output

```
node /tmp/7tTNGWQBrN.js
Fibonacci: 0, 1, 1, 2, 3, 5, 8, 13
```

El código JavaScript define una función llamada fibonacciSecuencia que genera la secuencia de Fibonacci hasta el número de términos especificado por el usuario. Aquí está la descripción paso a paso del código:

1) Definición de la función: fibonacciSecuencia es una función que acepta un parámetro num, que indica cuántos términos de la secuencia de Fibonacci se deben

ISBN: 978-631-6557-57-5

generar.

2) Comprobación de entrada válida: Si num es menor o igual a cero, se imprime un mensaje al usuario pidiéndole que ingrese un número mayor que cero y se termina la función usando return.

3) Inicialización de variables: Se inicializan tres variables: a, b y siguiente. Las variables a y b se inicializan en 0 y 1, respectivamente, que son los dos primeros números de la secuencia de Fibonacci.

4) Manejo del primer término: Si el usuario pide solo el primer término de la secuencia (num === 1), se imprime 0.

5) Generación e impresión de la secuencia:

- Si el usuario solicita más de un término, se imprime 0 y 1 como los dos primeros números de la secuencia.
- Se utiliza un bucle for que comienza en 3 y continúa hasta el número deseado (num). Dentro de este bucle:
  - Se calcula el siguiente número en la secuencia sumando a y b.
  - Se imprime el siguiente número, precedido por una coma para separarlo de los números anteriores.
  - Las variables a y b se actualizan para la siguiente iteración del bucle: a toma el valor de b, y b toma el valor de siguiente.

6) Ejecución de la función: Al final del código, la función fibonacciSecuencia se invoca con el valor 8, que imprimirá los primeros ocho términos de la secuencia de Fibonacci en la consola.

El código utiliza console.log para imprimir mensajes completos y process.stdout.write para imprimir sin un salto de línea automático, permitiendo que todos los números de la secuencia se muestren en una sola línea en la consola.

10. Realizar una función en Python que reciba una lista y que al elemento que se encuentre en la posición 1 de la lista le incremente en 2.

Pseudocódigo

ISBN: 978-631-6557-57-5

```
Funcion incrementar_valor(lista Por Referencia, indice)
    lista[indice] ← lista[indice] + 2
FinFuncion

Algoritmo EjemploIncrementarValor
    Dimension valor[3]
    valor[1] ← 6
    valor[2] ← 5
    valor[3] ← 8

    incrementar_valor(valor, 2)

    Escribir "El valor incrementado es: ", valor[2]
FinAlgoritmo
```

Diagrama de flujo

Implementación

```
def incrementar_valor(lista):
    lista[1] += 2  # Modifica el valor por referencia
```

```
# Ejemplo
valor = [6,5,8]
incrementar_valor(valor)
print("El valor incrementado es:", valor[1])
```

```
El valor incrementado es: 7
```

La función incrementar_valor(lista), toma una lista como entrada, le suma 2 al segundo elemento de la lista (índice 1). Al modificar un elemento de una lista dentro de una función, se modifica la lista original, debido a que las listas son mutables en Python.

En el ejemplo se define una lista de valores, luego se llama a la función

ISBN: 978-631-6557-57-5

incrementar_valor() con la lista como argumento y se imprime el segundo elemento de la lista modificada, mostrando que su valor ha aumentado en 2.

11. Crear una función en C++ que intercambie dos variables usando punteros (paso por referencia).

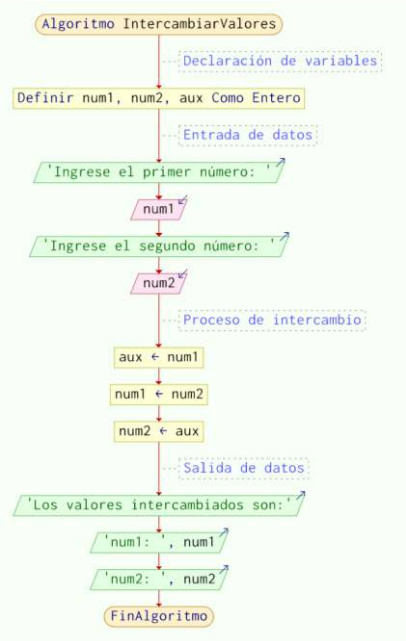
Pseudocódigo

```
Algoritmo IntercambiarValores
    // Declaración de variables
    Definir num1, num2, aux Como Entero

    // Entrada de datos
    Escribir "Ingrese el primer número: "
    Leer num1
    Escribir "Ingrese el segundo número: "
    Leer num2

    // Proceso de intercambio
    aux ← num1
    num1 ← num2
    num2 ← aux

    // Salida de datos
    Escribir "Los valores intercambiados son:"
    Escribir "num1: ", num1
    Escribir "num2: ", num2
FinAlgoritmo
```

```
PSeInt - Ejecutando proceso INTERCAMBIARV...
Ingrese el primer número:
> 10
Ingrese el segundo número:
> 20
Los valores intercambiados son:
num1: 20
num2: 10
*** Ejecución Finalizada. ***
```

Diagrama de flujo

ISBN: 978-631-6557-57-5



Implementación

```cpp
#include <iostream>
using namespace std;

void intercambiar(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

int main() {
    int x = 10, y = 20;
    intercambiar(&x, &y);  // Pasar las direcciones de memoria
    cout << "x = " << x << ", y = " << y << endl;
    return 0;
}
```

```
x = 20, y = 10
```

#include <iostream>: Esta línea incluye la biblioteca estándar de entrada/salida de C++, que proporciona funciones como cin y cout para interactuar con el usuario.

using namespace std;: Esta línea nos permite utilizar los nombres de las funciones y objetos de la biblioteca estándar sin tener que escribir std:: antes de cada uno.

void intercambiar(int* a, int* b) { ... }: Esta línea define una función llamada intercambiar que toma dos punteros enteros como argumentos. Un puntero es una variable que almacena la dirección de memoria de otro valor. En este caso, a y b apuntarán a las direcciones de memoria de las variables x e y, respectivamente. La función no devuelve ningún valor (por eso es void).

int temp = *a;: Se crea una variable temporal temp y se le asigna el valor al que apunta

ISBN: 978-631-6557-57-5

el puntero a (es decir, el valor de x).

*a = *b;: Se asigna el valor al que apunta el puntero b (el valor de y) al valor al que apunta el puntero a (cambia el valor de x).

*b = temp;: Se asigna el valor almacenado en temp (el valor original de x) al valor al que apunta el puntero b (cambia el valor de y).

int main() { ... }: Esta es la función principal del programa, donde comienza la ejecución.

int x = 10, y = 20;: Se declaran y se inicializan dos variables enteras, x y y.

intercambiar(&x, &y);: Se llama a la función intercambiar y se le pasan las direcciones de memoria de las variables x y y utilizando el operador de dirección (&).

cout << "x = " << x << ", y = " << y << endl;: Se imprime en la consola el nuevo valor de x y y después de haber sido intercambiados.

12. Crear una función en JavaScript que modifique un objeto pasado por referencia.

En PSeInt, no se puede crear un objeto como en los lenguajes orientados a objetos caso JavaScript, por lo que se ovia presentar el pseudocódigo.

```
function actualizarNombre(obj) {
    obj.nombre = "Alfonso"; // Modifica el nombre por referencia
}

const persona = { nombre: "Victoria" };
actualizarNombre(persona);
console.log("Nombre actualizado:", persona.nombre);
```

```
Nombre actualizado: Alfonso
```

Se define una función llamada actualizarNombre que toma un objeto como parámetro. Este objeto es al que se le va a cambiar el nombre. Dentro de la función, se accede a la propiedad nombre del objeto que se pasó como parámetro y se le asigna el valor "Alfonso". Esto modifica directamente el objeto original, debido a que los objetos en JavaScript se pasan por referencia.

Se crea un objeto constante llamado persona (const persona) con una propiedad nombre cuyo valor inicial es "Victoria". El uso de const indica que no se puede reasignar la variable persona a otro objeto, pero sí se puede modificar las propiedades dentro del objeto.

Se llama a la función actualizarNombre y se pasa el objeto persona como argumento. Esto hará que la función modifique la propiedad nombre del objeto persona a

ISBN: 978-631-6557-57-5

"Alfonso".

console.log("Nombre actualizado:", persona.nombre);: Se imprime en la consola el valor actualizado de la propiedad nombre del objeto persona. Como la función actualizarNombre modificó el objeto original, el resultado será "Alfonso".

13. Hacer una función en Java que modifique un array pasado por referencia.

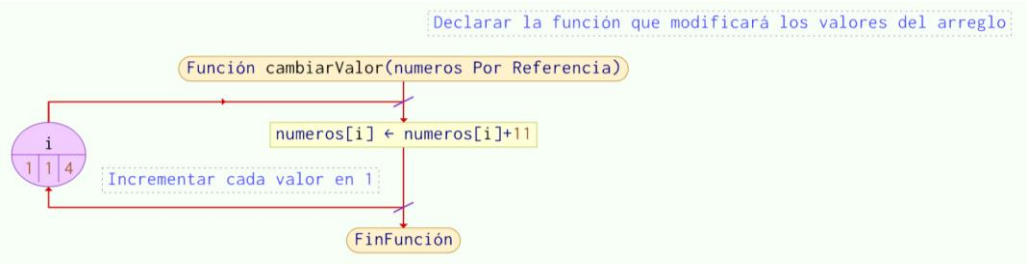
Pseudocódigo

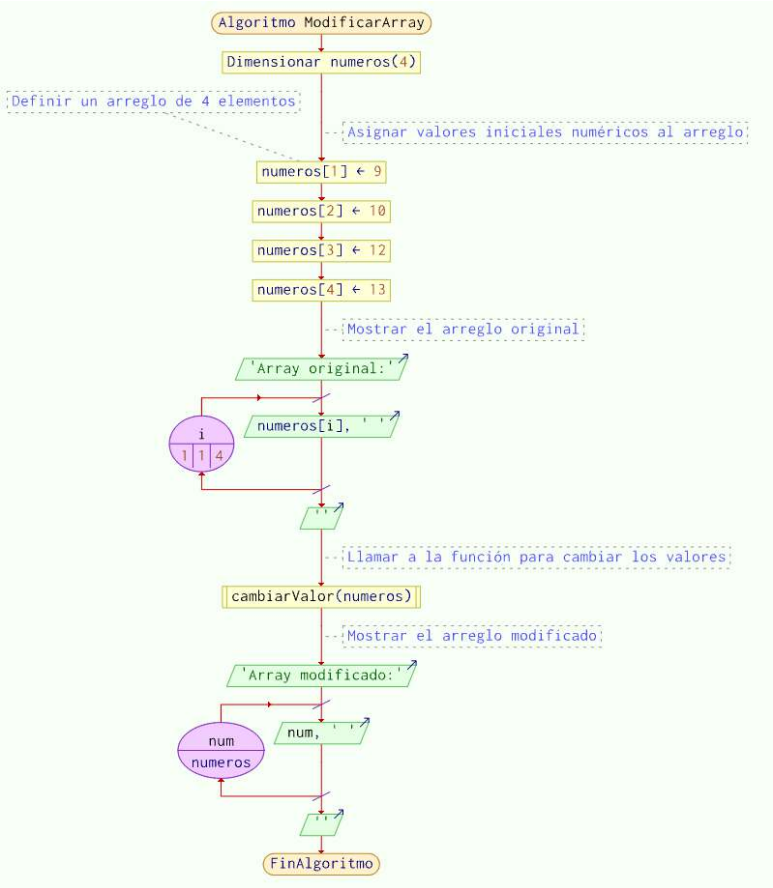
```
// Declarar la función que modificará los valores del arreglo
Funcion cambiarValor(numeros Por Referencia)
    Para i ← 1 Hasta 4 Con Paso 1 Hacer
        numeros[i] ← numeros[i] + 11  // Incrementar cada valor en 1
    FinPara
FinFuncion
Algoritmo ModificarArray
    Dimension numeros[4]  // Definir un arreglo de 4 elementos
 // Asignar valores iniciales numéricos al arreglo
    numeros[1] ← 9
    numeros[2] ← 10
    numeros[3] ← 12
    numeros[4] ← 13
   // Mostrar el arreglo original
    Escribir "Array original:"
    Para i ← 1 Hasta 4 Con Paso 1 Hacer
        Escribir Sin Saltar numeros[i], " "
    FinPara
    Escribir ""
  // Llamar a la función para cambiar los valores
    cambiarValor(numeros)
  // Mostrar el arreglo modificado
    Escribir "Array modificado:"
    Para cada num en numeros Hacer
        Escribir Sin Saltar num, " "
    FinPara
    Escribir ""
FinAlgoritmo
```

```
PSeInt - Ejecutando proceso MODIFICARARRAY
*** Ejecución Iniciada. ***
Array original:
9 10 12 13
Array modificado:
20 21 23 24
*** Ejecución Finalizada. ***
```

Diagrama de flujo

ISBN: 978-631-6557-57-5





Implementación

```java
public class Ejemplo {
    public static void cambiarValor(int[] numeros) {
        for (int i = 0; i < numeros.length; i++) {
            numeros[i] += 11; // Modificar cada valor del array por referencia
        }
    }

    public static void main(String[] args) {
        int[] numeros = {9, 10, 12, 13};
        cambiarValor(numeros);
        for (int num : numeros) {
            System.out.print(num + " "); // Imprime: 20 21 23 24
        }
    }
}
```

```
20 21 23 24
```

ISBN: 978-631-6557-57-5

public class Ejemplo {: Esta línea define una clase pública llamada Ejemplo, que es la estructura básica de un programa en Java. Todas las demás partes del código se ubicarán dentro de esta clase.

public static void cambiarValor(int[] numeros) {: Esta línea define un método estático (pertenece a la clase, no a un objeto) llamado cambiarValor que recibe un arreglo de enteros como parámetro. Este método no devuelve ningún valor (void).

for (int i = 0; i < numeros.length; i++) {: Este es un bucle for que itera sobre cada elemento del arreglo numeros. La variable i se utiliza como índice para acceder a cada elemento.

numeros[i] += 11;: En cada iteración del bucle, se suma 11 al valor actual del elemento en la posición i del arreglo. Esto modifica directamente el arreglo original, debido que los arreglos en Java se pasan por referencia.

public static void main(String[] args) {: Este es el método principal de la clase, donde comienza la ejecución del programa.

int[] numeros = {9, 10, 12, 13};: Se declara un arreglo de enteros llamado numeros y se inicializa con los valores 9, 10, 12 y 13.

cambiarValor(numeros);: Se llama al método cambiarValor y se pasa el arreglo numeros como argumento. Esto hará que todos los valores del arreglo se incrementen en 11.

for (int num : numeros) {: Este es un bucle for-each que itera sobre cada elemento del arreglo numeros y asigna el valor actual a la variable num.

System.out.print(num + " ");: Se imprime el valor de num seguido de un espacio en blanco.

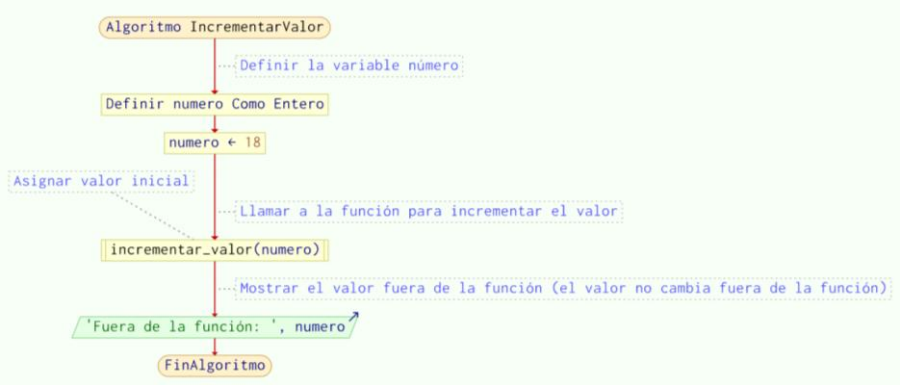
14. Desarrollar una función en Python que reciba un número, lo incremente y muestre el valor dentro y fuera de la función.

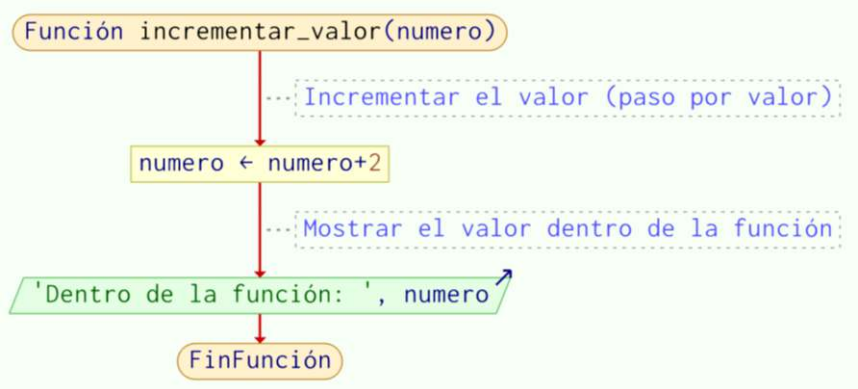
ISBN: 978-631-6557-57-5

Pseudocódigo

```
Proceso IncrementarValor
    // Definir la variable número
    Definir numero Como Entero
    numero ← 18  // Asignar valor inicial

    // Llamar a la función para incrementar el valor
    incrementar_valor(numero)

    // Mostrar el valor fuera de la función (el valor no cambia fuera de la función)
    Escribir "Fuera de la función: ", numero
FinProceso

SubProceso incrementar_valor(numero)
    // Incrementar el valor (paso por valor)
    numero ← numero + 2

    // Mostrar el valor dentro de la función
    Escribir "Dentro de la función: ", numero
FinSubProceso
```

Diagrama de flujo





ISBN: 978-631-6557-57-5

Implementación

```
def incrementar_valor(numero):
    numero += 2  # Incrementa el número (paso por valor)
    print("Dentro de la función:", numero)
```

```
# Ejemplo
numero = 18
incrementar_valor(numero)
print("Fuera de la función:", numero)  # El valor no cambia fuera de la función
```

```
Dentro de la función: 20
Fuera de la función: 18
```

def incrementar_valor(numero):: Esta línea define una función llamada incrementar_valor que toma un número como parámetro.

numero += 2: Dentro de la función, se incrementa el valor del número en 2 unidades. Sin embargo, es importante notar que, en Python, los números son inmutables. Esto significa que cuando se pasa un número a una función, se crea una copia de ese número y cualquier modificación se realiza sobre esa copia. Por lo tanto, el valor original del número fuera de la función no se modifica.

print("Dentro de la función:", numero): Esta línea imprime un mensaje en la consola mostrando el valor del número dentro de la función.

numero = 18: Se crea una variable llamada numero y se le asigna el valor 18.

incrementar_valor(numero): Se llama a la función incrementar_valor pasando el valor de numero como argumento.

print("Fuera de la función:", numero): Se imprime el valor de numero nuevamente, pero esta vez fuera de la función. Como ya se mencionó, el valor de numero no ha cambiado porque los números son inmutables en Python.

15. Desarrollar una función en C++ que reciba una estructura Punto y modifique sus valores, mostrando el resultado dentro y fuera de la función.

ISBN: 978-631-6557-57-5

```cpp
#include <iostream>
using namespace std;

struct Punto {
    int x, y;
};

void moverPunto(Punto p) {
    p.x += 7; // Modifica los valores de x e y localmente
    p.y += 8;
    cout << "Dentro de la función: (" << p.x << ", " << p.y << ")" << endl;
}

int main() {
    Punto punto = {20, 30};
    moverPunto(punto);
    cout << "Fuera de la función: (" << punto.x << ", " << punto.y << ")" << endl
        ; // El valor original no cambia
    return 0;
}
```

```
Dentro de la función: (27, 38)
Fuera de la función: (20, 30)
```

Se define una estructura llamada Punto. Una estructura es un tipo de dato compuesto que agrupa variables de diferentes tipos bajo un mismo nombre. En este caso, la estructura Punto tiene dos miembros enteros: x y y, que representan las coordenadas de un punto en un plano cartesiano.

La función moverPunto toma como parámetro un objeto de tipo Punto llamado p. Dentro de la función, se incrementa en 7 la coordenada x y en 8 la coordenada y del punto recibido. Luego, se imprime en pantalla las nuevas coordenadas.

La función main es la función principal del programa. Aquí es donde comienza la ejecución. Se crea un objeto de tipo Punto llamado punto y se inicializan sus coordenadas con los valores 20 y 30.

Se llama a la función moverPunto pasando como argumento el objeto punto. Se imprime en pantalla las coordenadas del objeto punto después de la llamada a la función.

16. Hacer una función en Java que reciba un array, modifique una copia local de ese array y muestre el valor dentro y fuera de la función.

ISBN: 978-631-6557-57-5

```java
public class Array {
    public static void modificarArray(int[] numeros) {
        numeros = new int[]{10, 20, 30}; // Modifica la referencia localmente
        System.out.println("Dentro de la función: ");
        for (int num : numeros) {
            System.out.print(num + " ");
        }
        System.out.println();
    }

    public static void main(String[] args) {
        int[] numeros = {11, 22, 33};
        modificarArray(numeros);
        System.out.print("Fuera de la función: ");
        for (int num : numeros) {
            System.out.print(num + " "); // El valor original no cambia
        }
    }
}
```

```
Dentro de la función:
10 20 30
Fuera de la función: 11 22 33
```

Se define una clase llamada Array, esta clase contiene dos métodos principales: modificarArray y main.

En el método modificarArray, se recibe un arreglo de enteros llamado números y se crea un nuevo arreglo dentro del método y se asigna a la variable numeros. Esto significa que la variable numeros dentro del método ahora apunta a un arreglo completamente diferente, con los valores 10, 20 y 30. Luego se imprime el contenido del nuevo arreglo dentro de la función.

El punto de entrada es el método principal main donde comienza la ejecución del programa. Aquí se crea un arreglo de enteros llamado numeros con los valores 11, 22 y 33, luego se llama al método modificarArray pasando el arreglo numeros como argumento. Después de la llamada al método, se imprime el contenido del arreglo numeros nuevamente.

17. Escribe una función llamada obtener_datos que retorne una lista con tres elementos: un nombre, una edad y una profesión. Luego, utiliza esta función para obtener los valores de la lista y desestructurarlos en tres variables: nombre, edad y profesion. Finalmente, imprime cada una de estas variables en un formato claro.

```python
def obtener_datos():
    return ["Andrés", 27, "Ingeniero"]  # Retorna una lista

nombre, edad, profesion = obtener_datos()

# Imprimir los valores
print(f"Nombre: {nombre}")  # Imprime: Juan
print(f"Edad: {edad}")  # Imprime: 25
print(f"Profesión: {profesion}")  # Imprime: Ingeniero
```

ISBN: 978-631-6557-57-5

```
Nombre: Andrés
Edad: 27
Profesión: Ingeniero
```

Se define una función llamada obtener_datos(). Esta función no recibe ningún argumento y su tarea es devolver un conjunto de datos.

Dentro de la función, se crea una lista que contiene tres elementos: un nombre (string), una edad (entero) y una profesión (string). Esta lista es la que se devuelve al llamar a la función.

La línea nombre, edad, profesion = obtener_datos() hace uso de la asignación múltiple en Python. Lo que ocurre aquí es lo siguiente:

Se llama a la función obtener_datos(), la cual devuelve una lista con tres elementos.

Los tres elementos de la lista se asignan a las variables nombre, edad y profesión respectivamente. Es como si estuviéramos desempaquetando la lista en estas tres variables.

18. Crear una función llamada obtenerValoresAsync que retorne una promesa que resuelva dos valores (por ejemplo, 1 y 2) utilizando Promise.all. Luego, utilizar .then() para manejar la promesa y mostrar los dos valores obtenidos en la consola.

```
function obtenerValoresAsync() {
    return Promise.all([Promise.resolve(10), Promise.resolve(20)]); // Retorna una
        Promesa con múltiples valores
}

obtenerValoresAsync().then(([v1, v2]) => {
    console.log("Valor 1:", v1);  // Imprime: Valor 1: 10
    console.log("Valor 2:", v2);  // Imprime: Valor 2: 20
});
```

```
Valor 1: 10
Valor 2: 20
```

La función obtenerValoresAsync() se encarga de obtener dos valores de forma asíncrona y devolverlos como una promesa.

La función Promise.all() toma un arreglo de promesas como argumento y devuelve una nueva promesa.

Promise.resolve(10) y Promise.resolve(20): Estas son promesas que se resuelven inmediatamente con los valores 10 y 20, respectivamente. Al pasarlas a Promise.all(),

ISBN: 978-631-6557-57-5

estamos creando una nueva promesa que se resolverá cuando todas las promesas dentro del arreglo se hayan resuelto.

La promesa resultante se resolverá con un arreglo que contiene los valores resueltos de cada promesa individual. En este caso, el arreglo será [10, 20].

El código del bloque then() se ejecuta cuando la promesa devuelta por obtenerValoresAsync() se resuelve.

([v1, v2]) => { ... }: Esta sintaxis se llama desestructuración. Permite asignar los valores del arreglo devuelto por la promesa a las variables v1 y v2 de forma directa.

console.log("Valor 1:", v1);: Imprime en la consola el valor de v1, que será 1.

console.log("Valor 2:", v2);: Imprime en la consola el valor de v2, que será 2.

19. Determinar una constante global llamada PI que almacene el valor de 3.1416. Luego, crear una función llamada calcular_area que reciba como parámetro el radio de un círculo y utilice la constante PI para calcular y devolver el área del círculo. Finalmente, imprime el resultado de la función para un radio de 16.

```
PI = 3.1416  # Constante global

def calcular_area_circulo(radio):
    return PI * radio ** 2  # Uso de la constante global

print(calcular_area_circulo(8))  # Imprime el área
```

```
201.0624
```

Se crea una variable llamada PI y se le asigna el valor de 3.1416. Este valor es una aproximación del número pi (π), una constante matemática fundamental que relaciona la circunferencia de un círculo con su diámetro. Al definir PI fuera de cualquier función, se convierte en una constante global. Esto significa que puede ser utilizada desde cualquier parte del programa.

La línea def calcular_area_circulo(radio) define una función llamada calcular_area_circulo que toma un parámetro llamado radio. Esta función se encargará de calcular el área de un círculo.

Dentro de la función se ecnuentra return PI * radio ** 2, se utiliza la fórmula matemática para calcular el área de un círculo: Área = π * radio^2. Aquí se utiliza la constante global PI definida anteriormente.

20. Hacer un arreglo global de números enteros llamado numeros en C++ que

ISBN: 978-631-6557-57-5

contenga los valores {100, 200, 300, 400}. Luego, crea una función llamada imprimirNumeros que recorra el arreglo global y muestre en la consola todos los números separados por un espacio. Finalmente, llama a esta función desde la función main para que se muestren los valores del arreglo en la consola.

```cpp
#include <iostream>

int numeros[] = {100, 200, 300, 400};  // Array global

void imprimirNumeros() {
    for (int numero : numeros) {
        std::cout << numero << " ";  // Imprime: 100 200 300 400
    }
}

int main() {
    imprimirNumeros();
    return 0;
}
```

```
100 200 300 400
```

#include <iostream>: Esta línea le indica al compilador que incluya la librería iostream.

int numeros[] = {100, 200, 300, 400};: Aquí se declara un arreglo (o matriz) de números enteros llamado numeros. Un arreglo es una estructura de datos que almacena una colección de elementos del mismo tipo (en este caso, enteros). El arreglo numeros contiene cuatro elementos: 100, 200, 300 y 400. Al declararlo fuera de cualquier función, se convierte en un arreglo global, lo que significa que puede ser accedido desde cualquier parte del programa.

void imprimirNumeros(): Esta función se encarga de imprimir los números almacenados en el arreglo numeros.

for (int numero : numeros): Este es un ciclo for que itera sobre cada elemento del arreglo numeros. En cada iteración, la variable numero toma el valor del siguiente elemento del arreglo.

std::cout << numero << " ";: Esta línea imprime el valor actual de numero seguido de un espacio. En otras palabras, imprime cada número del arreglo en una misma línea, separados por espacios.

int main(): Esta es la función principal de cualquier programa en C++. Es el punto de entrada de la ejecución del programa.

imprimirNumeros();: Esta línea llama a la función imprimirNumeros() para que imprima los números del arreglo.

return 0: Esta línea indica que el programa se ejecutó correctamente y termina.

ISBN: 978-631-6557-57-5

21. Hacer una función recursiva llamada potencia que calcule la potencia de un número. La función debe recibir dos parámetros: base y exponente. Si el exponente es 0, la función debe retornar 1 (caso base). En cualquier otro caso, debe retornar el resultado de multiplicar la base por la llamada recursiva a potencia con el exponente decrementado en 1. Finalmente, muestra el resultado de la función para una base de 5 y un exponente de 3.

```python
def potencia(base, exponente):
    if exponente == 0:
        return 1  # Caso base
    else:
        return base * potencia(base, exponente - 1)  # Llamada recursiva

print(potencia(5, 3))  # Imprime: 125
```

```
125
```

La línea def potencia(base, exponente) define una nueva función llamada potencia. Esta función toma dos argumentos: base que es el número que se va a multiplicar por sí mismo y exponente que es el número de veces que se va a multiplicar la base.

La condición if exponente == 0 verifica si el exponente es igual a cero. Si el exponente es cero, la función devuelve 1 mediante return 1. Esto se debe a que cualquier número elevado a la potencia de 0 es igual a 1. Este es el caso base de la recursión.

Si el exponente no es cero, se ejecuta else.

En return base * potencia(base, exponente - 1) es donde ocurre la recursión. La función se llama a sí misma, donde base mantiene el mismo valor de la base y exponente – 1 se decrementa en 1.

22. Hacer una función que invierta la palabra reconocer, utilizando C++.

```cpp
#include <iostream>
#include <string>

std::string invertir_cadena(const std::string& cadena) {
    if (cadena.empty()) return "";  // Caso base
    return cadena.back() + invertir_cadena(cadena.substr(0, cadena.size() - 1));
        // Llamada recursiva
}

int main() {
    std::cout << invertir_cadena("reconocer");  // Imprime: aloh
    return 0;
}
```

```
reconocer
```

La cabecera <string> proporciona la clase std::string para trabajar con cadenas de caracteres.

ISBN: 978-631-6557-57-5

Esta línea std::string invertir_cadena(const std::string& cadena), define una nueva función llamada invertir_cadena. Esta función toma un argumento de tipo std::string llamado cadena y devuelve una std::string. El parámetro const std::string& indica que la función recibe una referencia constante a la cadena, lo que significa que no puede modificarla dentro de la función.

La condición if (cadena.empty()) verifica si la cadena cadena está vacía (es decir, si tiene una longitud de 0). Si la cadena está vacía, la función return devuelve una cadena vacía (""). Este es el caso base de la recursión.

La expresión cadena.back() obtiene el último carácter de la cadena denominada cadena.

La expresión cadena.substr(0, cadena.size() - 1) crea una nueva cadena que es una subcadena de cadena, comenzando desde el índice 0 hasta el índice cadena.size() - 1, excluyendo el último carácter. En otras palabras, crea una nueva cadena que es cadena sin su último carácter.

Esta línea return cadena.back() + invertir_cadena(cadena.substr(0, cadena.size() - 1)) devuelve una nueva cadena que es la concatenación del último carácter de cadena con el resultado de llamar recursivamente a invertir_cadena con la subcadena que no incluye el último carácter.

La función principal del programa es int main().

Mediante std::cout << invertir_cadena("reconocer") se llama a la función invertir_cadena con la cadena "reconocer" como argumento y se imprime el resultado en la consola.

4.2.2 Ejercicios propuestos

1. Realizar una función quedado un numero verifique si es par o impar.
2. Crear una función que dada la temperatura en grados Farenheit convierta a grados Celcius
3. Hacer una función que encuentre el número mayor de tres números ingresados por teclado.
4. Escribir una función que sume los dígitos de un número dado.
5. Crear una función que calcule el promedio de la lista de números [8,12,4,38,52]
6. Realizar una función en JavaScript que agregue elementos a un array pasado por referencia. Por ejemplo, se tiene el array [7,8,9,10] y debe agragarle el 11.

ISBN: 978-631-6557-57-5

7. Hacer una función en Java que modifique un objeto de tipo Persona pasado por referencia. Por ejemplo, se tiene una persona llamado Leonel que inicialmente esta puesto la edad 4 años y se debe hacer que a esa edad le sume 2 años, de tal forma que imprima la edad actualizada 6.
8. Desarrollar una función en C++ que reciba un número, lo multiplique por 2 y muestre el valor dentro y fuera de la función.
9. Crear una función en Python que reciba una tupla (inmutable), intente modificar su contenido y muestre el valor dentro y fuera de la función.
10. Define un método llamado obtenerLambda que retorne una función lambda. La función lambda debe recibir un número entero como parámetro y devolver el doble de ese número. Utiliza el tipo Function de la biblioteca java.util.function para definir la función lambda.
11. Hacer una función en Java que reciba una cadena, modifique su contenido y muestre el valor dentro y fuera de la función.
12. Realizar una función anónima en JavaScript que reciba un número, lo incremente y muestre el valor dentro y fuera de la función.
13. Escribir una función llamada obtener_usuario que retorne un diccionario con tres claves: "nombre", "edad", "estatura" . Luego, utilizar esta función para obtener el diccionario y guarda su resultado en una variable llamada usuario. Finalmente, imprimir el valor de cada clave del diccionario de manera individual.
14. Definir una clase llamada Persona con un constructor (__init__) que tome dos parámetros: nombre y edad. Luego, crea una función llamada crear_persona que retorne un objeto de la clase Persona con los valores "Carlos" para el nombre y 40 para la edad. Utiliza esta función para crear una instancia de Persona, guárdala en una variable llamada persona e imprime los atributos nombre y edad del objeto.
15. Definir un método llamado obtenerOptional en Java que retorne un objeto de tipo Optional<String>. El método debe devolver una cadena de texto ("Hola") utilizando la clase Optional para envolver el valor. Luego, demuestra cómo utilizar el valor devuelto por el método obtenerOptional para imprimir el contenido del Optional en la consola.
16. Realizar una clase llamada Contar que contenga una variable global contador de

ISBN: 978-631-6557-57-5

tipo entero, inicializada en 0. Crear un método estático llamado incrementar que incremente el valor de contador en 1 cada vez que se llame. En el método main, llamar al método incrementar y luego mostrar el valor de contador en la consola.

17. Crear una constante global llamada MAX_USUARIOS de tipo int con un valor de 100. Realizar una función llamada mostrarMaximo que imprima en la consola el valor de MAX_USUARIOS con el mensaje "Max usuarios: ". Luego, llama a esta función desde el método main para mostrar el resultado.
18. Definir un array global llamado nombres que contenga los elementos "María", "Patricio" y "Gisel". Desarrollar una función llamada agregarNombre que reciba un nombre como parámetro y lo añada al array nombres. Luego, llamar a la función agregarNombre con el nombre "Xavier" y muestra el contenido actualizado del array en la consola.
19. Realizar una función recursiva en C++ llamada conteo_regresivo que reciba un número entero n como parámetro. La función debe imprimir en la consola el valor de n en cada llamada hasta que n sea menor o igual a 0. Cuando n sea menor o igual a 0, debe imprimir el mensaje "Despegue!" (este es el caso base). Luego, utilizar la función conteo_regresivo desde la función main con un valor inicial de 5 para mostrar la cuenta regresiva.
20. Escribe una función recursiva en Python llamada mcd que calcule el máximo común divisor (MCD) de dos números enteros a y b. La función debe seguir el algoritmo de Euclides, donde:
21. Si b es 0, la función debe retornar a (este es el caso base).
22. Si b no es 0, la función debe llamar recursivamente a mcd con los parámetros b y a % b.

ISBN: 978-631-6557-57-5

BIBLIOGRAFÍA

Annable, C. (2019). *JavaScript Mastery: A Step-by-Step Beginner's Guide to Learning JavaScript Programming*. Chloe Annable.

Annable, C. (2024). *Mastering Python Programming: A Comprehensive Beginner's Guide with Step-by-Step Instructions and Practical Exercises*. Chloe Annable.

Barry, P. (2023). *Head First Python. A Learner's Guide to the Fundamentals of Python Programming*. O'Reilly.

Barzee, R. (2017). *Programming Fundamentals in JavaScript*. Maia L.L.C.

Benjumea, V., & Roldan, M. (2011). *Fundamentos De Programación Con C++*. Universidad de Malaga.

Bhimavarapu, U., & Hemanth, J. (2023). *Learning Professional Python*. Chapman and Hall.

Budd, T. (2009). *Exploring Python*. McGraw-Hill Science.

Cerrada, J., & Collado, M. (2010). *Fundamentos de programación*. Editorial Universitaria Ramón Areces.

Chinnathambi, K. (2024). *Absolute Beginner's Guide to Algorithms: A Practical Introduction to Data Structures and Algorithms in JavaScript*. Pearson.

Easttom, C. (2001). *Advanced JavaScript*. Wordware Publishing.

Easttom, C. (2003). *C++ Programming Fundamentals*. Charles River Media.

Easttom, C. (2006). *Programming Language Fundamentals by Example*. Auerbach Publications.

Friesen, J. (2024). *Learn Java Fundamentals: A Primer for Java Development and Programming*. Apress.

Gold, M. (2024). *Epic Python Coding: Interactive Coding Adventures for Kids*. Leanpub.

Halterman, R. (2014). *Fundamentals of Programming C++*. Southern Adventist University.

Harrison, J. (2023). *Python Programming for Beginners: The Complete Python Coding Crash Course*. Independently Published.

Hazrat, R. (2023). *A Course in Python: The Core of the Language*. Springer.

Hood, C., & Kölling, M. (2023). *Python for Beginners: The definitive beginner's guide to the realm of coding, introducing you to the programming language Python*.

Independent.

Joyanes, L. (2020). *Fundamentos de programación: Algoritmos, estructura de datos y objetos*. McGraw-Hill.

Learning, T. (2000). *Java Programming Fundamentals*. Thomson Learning.

Leroy, K. (2009). *Programming Fundamentals: A Modular Structured Approach Using C++*. University Press of Florida.

Malhotra, D., & Malhotra, N. (2023). *C++ Programming Fundamentals*. Mercury Learning and Information.

Matuszek, D. (2022). *Quick Python 3*. CRC Press/Chapman & Hall.

Mejia, P., Gonzalez, R., & Ortega, S. (2024). *Exception Handling: Fundamentals and Programming*. Springer.

Meyers, J. (2023). *Python Programming Bible*. PublishDrive.

Nair, P. (2009). *Java Programming Fundamentals: Problem Solving Through Object Oriented Analysis and Design*. CRC Press.

Nakov, S. (2013). *Fundamentals of Computer Programming with C#*. Svetlin Nakov.

Ogihara, M. (2018). *Fundamentals of Java Programming*. Springer International Publishing.

Ormaza, J. (2020). *Aprende lógica de programación*. Amazon.

Sharan, K., & Davis, A. (2021). *Beginning Java 17 Fundamentals: Object-Oriented Programming in Java 17*. Apress.

Shehzad, M. (2024). *Fundamentals of Programming*. Toronto Academic Press.

Singh, K., Jagjit, J., & Balamurugan, B. (2023). *Python for Beginners*. CRC Press/Chapman & Hall.

ISBN: 978-631-6557-57-5

ISBN: 978-631-6557-57-5

ISBN: 978-631-6557-57-5

DE LOS AUTORES

PATRICIO XAVIER MORENO VALLEJO

Ing. Patricio Xavier Moreno Vallejo M.S.

ESTUDIOS

- Master of Science Major – Computer Science and Applications, Virginia Polytechnic Institute and State University, Estados Unidos de América
- Ingeniero en Sistemas Informáticos, Escuela Superior Politécnica de Chimborazo, Ecuador

EXPERIENCIA

- Docente en la Escuela Superior Politécnica de Chimborazo desde Octubre del 2018 hasta la actualidad.
- Senior Developer en Top-Tech Advisors desde Agosto – 2018 hasta Septiembre – 2018.
- Teaching Assistant – Docente en Virginia Polytechnic Institute and State University (Virginia Tech) desde Agosto – 2016 hasta Mayo – 2017.
- Web Developer en Organization of American States (OEA) - Department of Information and Technology Services desde Enero – 2016 hasta Mayo – 2016.
- Program Developer en EasySoft S.A. desde Septiembre – 2014 hasta Septiembre – 2015.
- Desarrollador de Sistemas en Clínica Riobamba desde Mayo – 2013 hasta Mayo – 2014.
- Desarrollador Web en la Escuela Superior Politécnica de Chimborazo desde Febrero – 2013 hasta Julio – 2013.

ISBN: 978-631-6557-57-5

GISEL KATERINE BASTIDAS GUACHO

Ing. Gisel Katerine Bastidas Guacho M.S.

ESTUDIOS

- Master of Science in Computer Science, University of California, Riverside, Estados Unidos.
- Ingeniera en Sistemas Informáticos, Escuela Superior Politécnica de Chimborazo, Ecuador

EXPERIENCIA

- Docente en la Escuela Superior Politécnica de Chimborazo desde Octubre del 2019 hasta la actualidad.
- Ingeniero Senior de Business Intelligence en TopTech Advisors - Produbanco desde agosto-2018 hasta septiembre-2019.
- Analista de Tecnologías de la Información y Comunicaciones en Ministerio del Interior desde marzo-2015 hasta octubre-2015.
- Analista de TICs en CNT EP - Corporación Nacional de Telecomunicaciones desde junio-2014 hasta marzo-2015.
- Desarrolladora de Software en Clínica Riobamba desde julio-2013 hasta junio-2014.
- Desarrolladora de Software – Prácticas pre-profesionales en Escuela Superior Politécnica de Chimborazo desde febrero-2013 hasta julio-2013.
- Soporte Técnico en Scytel - CNE desde enero-2013 hasta febrero-2013.

ISBN: 978-631-6557-57-5

MARIA ELENA VALLEJO SANAGUANO

Ing. María Elena Vallejo Sanaguano MSc.

ESTUDIOS

- Magister en Gerencia Educativa – Universidad Estatal de Bolívar – Ecuador
- Diploma Superior en Gestión y Planificación Educativa
- Master en Informática Aplicada – Escuela Superior Politécnica de Chimborazo - Ecuador
- Ingeniera Mecánica – Escuela Superior Politécnica de Chimborazo – Ecuador

EXPERIENCIA

- Docente Escuela Superior Politécnica de Chimborazo desde 1995

PUERTO MADERO
EDITORIAL

ISBN 978-631-6557-57-5