DIANA ALEGRÍA | CLAUDIO BORJA

FUNDAMENTOS DE PROGRAMACIÓN

TEORÍA, PROBLEMAS PROPUESTOS Y RESUELTOS CON ALGORITMOS, DIAGRAMAS DE FLUJO, PRUEBAS DE ESCRITORIO Y CÓDIGO FUENTE EN JAVA

INSTITUTO SUPERIOR TECNOLÓGICO EL LIBERTADOR

dTC DOCTRINA TECH

PRIMERA EDICIÓN

2024

ISBN: 978-9942-7128-1-3

Fundamentos de Programación - Teoría, problemas propuestos y resueltos con algoritmos, diagramas de flujo, pruebas de escritorio y código fuente en Java.

PRIMERA EDICIÓN

Diana Magali Alegría Camino
Claudio Xavier Borja Saltos

INDICE

RESEÑA

La programación combina la lógica, la creatividad y la resolución de problemas, este libro está diseñado para guiar a los lectores a través de los fundamentos de la programación, brindándoles una sólida base teórica y práctica.

Exploraremos los conceptos fundamentales de la programación, desde las estructuras secuenciales, de control y de repetición, cada concepto se explicará de manera detallada y se ilustrará con ejemplos prácticos para facilitar su comprensión.

Una de las características distintivas de este libro es la inclusión de problemas propuestos y resueltos, están cuidadosamente seleccionados para reflejar situaciones del mundo real y desafiar a los lectores a aplicar los conceptos aprendidos, cada problema se abordará utilizando diferentes enfoques a parte de los algoritmos, se proporcionarán diagramas de flujo para cada problema resuelto, se realizarán pruebas de escritorio para cada solución y finalmente, se proporcionará el código fuente en Java para cada problema resuelto.

Este libro no solo se enfoca en los aspectos técnicos de la programación, sino que también aborda la importancia de la planificación, el diseño y la documentación, ya sea que seas un estudiante de informática, un principiante en programación o alguien que busca reforzar sus habilidades, este libro te guiará a través de un recorrido enriquecedor y práctico por los fundamentos de la programación.

CAPITULO 1: INTRODUCCIÓN A LA PROGRAMACIÓN

INTRODUCCIÓN A LOS FUNDAMENTOS DE PROGRAMACIÓN

La programación es una habilidad esencial en el mundo moderno, ya que permite crear herramientas y soluciones tecnológicas que impactan en múltiples áreas, desde la ciencia y la ingeniería hasta la economía y el entretenimiento. Este capítulo introduce los conceptos fundamentales que servirán de base para el aprendizaje de la programación, enfocándose en la comprensión de los principios básicos, el diseño de soluciones mediante algoritmos y el uso de diagramas de flujo.

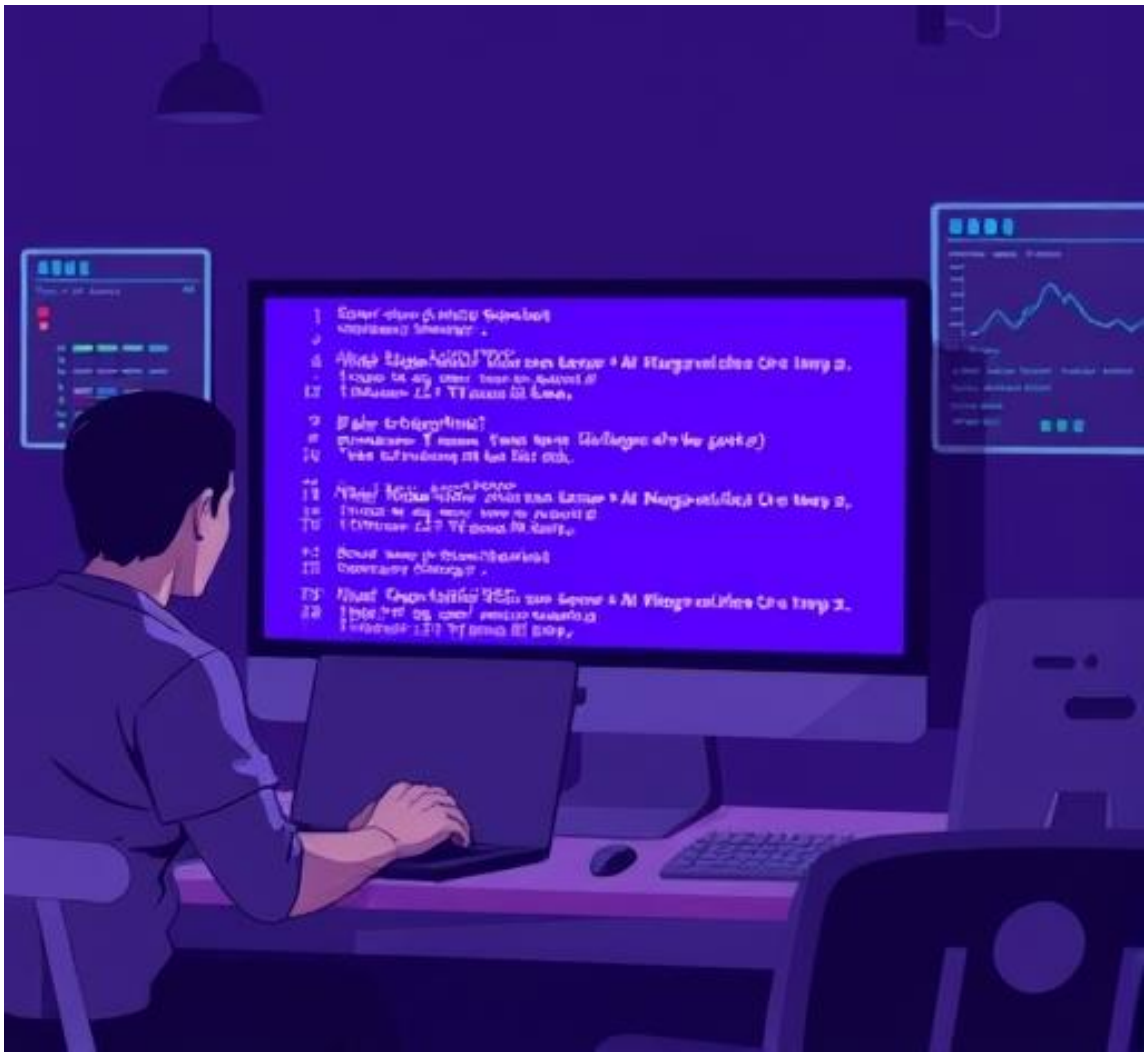


Ilustración 1 - Programación

1.1. ¿Qué es la programación?

La programación es el proceso de diseñar, escribir, probar y mantener el código fuente que ejecutan las computadoras para realizar tareas específicas. Este código está compuesto por una secuencia de instrucciones escritas en un lenguaje de programación que la computadora interpreta y ejecuta.

Características Principales:

1. La programación busca resolver problemas mediante la creación de soluciones lógicas y estructuradas.
2. Permite automatizar procesos que de otro modo serían realizados manualmente.
3. Combina creatividad en el diseño de soluciones con precisión en la escritura del código.

Componentes Fundamentales:

- **Algoritmos:** Son conjuntos ordenados de pasos que describen cómo resolver un problema.
- **Lenguajes de Programación:** Son herramientas que permiten expresar los algoritmos en un formato entendible por la computadora. Ejemplos: Java, Python, C++.
- **Ejecución:** Consiste en convertir el código en acciones que la máquina puede realizar, ya sea a través de compilación o interpretación.

Objetivos de la Programación:

- Crear software que facilite tareas humanas.
- Resolver problemas computacionales mediante el uso de algoritmos.
- Desarrollar sistemas eficientes y seguros.

Ejemplo Simple:

Un ejemplo de un programa básico es el clásico "Hola, mundo", que simplemente muestra un mensaje en pantalla. En Java, este programa se escribe así:

```java
public class HolaMundo {
    public static void main(String[] args) {
        System.out.println("¡Hola, mundo!");
    }
}
```

La programación es esencial en el mundo moderno, ya que impulsa tecnologías como aplicaciones móviles, sistemas operativos, inteligencia artificial, y mucho más. Es una habilidad fundamental en un mundo donde la tecnología juega un papel crucial en nuestras vidas.

1.2. Historia de la programación y lenguajes

La historia de la programación comienza con el desarrollo de las primeras máquinas calculadoras y la necesidad de controlar su funcionamiento mediante instrucciones, uno de los primeros hitos importantes fue el trabajo de Ada Lovelace en el siglo XIX, quien diseñó algoritmos para la Máquina Analítica de Charles Babbage, convirtiéndose en la primera programadora de la historia.

Ilustración 2 - Ada Lovelace

Este evento marcó el inicio de la programación como disciplina, aunque en ese entonces los programas eran concebidos en términos matemáticos y mecánicos, en el siglo XX, la llegada de las computadoras modernas transformó la programación en una actividad esencial.

Durante la década de 1940, John von Neumann introdujo el concepto de arquitectura de programas almacenados, donde las instrucciones y los datos residían en la memoria de la computadora, facilitando la ejecución de procesos complejos.

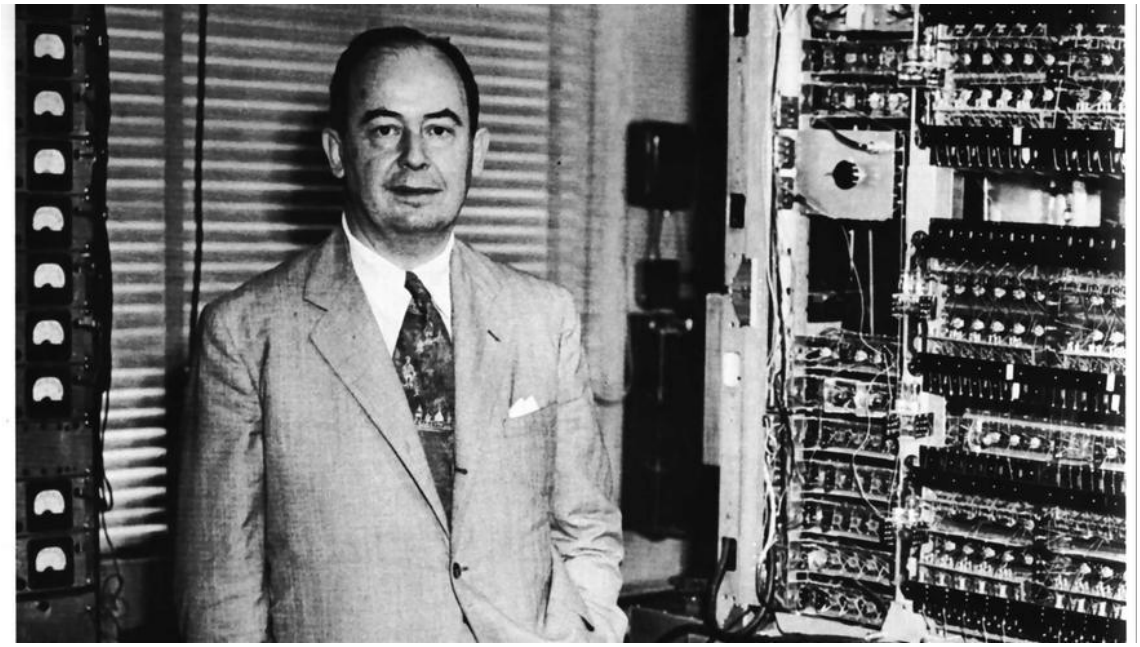


Ilustración 3 - John von Neumann

En este período, se desarrollaron los primeros lenguajes de bajo nivel, como el ensamblador, que permitían a los programadores comunicarse directamente con el hardware utilizando códigos binarios o mnemotécnicos.

En la década de 1950, surgieron los primeros lenguajes de programación de alto nivel, diseñados para ser más accesibles y legibles para los humanos. Fortran (FORmula TRANslation), creado en 1957, fue uno de los primeros lenguajes ampliamente adoptados y estaba orientado a aplicaciones científicas y matemáticas.

Poco después, COBOL (COmmon Business-Oriented Language) se diseñó para satisfacer las necesidades de programación empresarial, sentando las bases de la programación en diversas industrias.

Con el tiempo, aparecieron lenguajes más avanzados como C en los años 70, que combinaba eficiencia con flexibilidad y permitió un control más preciso del hardware. Este lenguaje influyó en muchos lenguajes modernos, como C++, Java y Python, durante los años 80 y 90, la programación orientada a objetos ganó popularidad con lenguajes como Smalltalk, C++ y Java, facilitando la creación de software modular y reutilizable.

En la actualidad, la programación abarca una gran variedad de lenguajes adaptados a necesidades específicas, desde los de propósito general como Python, JavaScript y C#, hasta los especializados en áreas como inteligencia artificial, análisis de datos y desarrollo web.

Esta evolución no solo refleja el avance de la tecnología, sino también la búsqueda constante de herramientas más intuitivas y poderosas que permitan a los desarrolladores abordar problemas cada vez más complejos. La historia de la programación es una muestra del ingenio humano y de cómo el deseo de simplificar tareas y resolver problemas ha llevado a la creación de lenguajes que son, al mismo tiempo, herramientas prácticas y expresiones de creatividad lógica.

A medida que la tecnología avanza, la programación continúa adaptándose, empujando los límites de lo que es posible en el mundo digital.

1.3. Herramientas y entornos de desarrollo

En la programación, contar con las herramientas adecuadas es fundamental para escribir, depurar y ejecutar código de manera eficiente, los entornos de desarrollo y herramientas complementarias no solo simplifican el proceso de programación, sino que también incrementan la productividad y mejoran la calidad del software creado.

Editores de Código

Los editores de código son herramientas esenciales para escribir programas. Proveen características como resaltado de sintaxis, autocompletado y navegación rápida por el código. Algunos de los más utilizados son:

- **Visual Studio Code**: Un editor versátil y extensible que soporta múltiples lenguajes.
- **Sublime Text**: Ligero y rápido, ideal para proyectos pequeños o scripts.
- **Notepad++**: Una opción sencilla para ediciones rápidas de texto y código.

Entornos de Desarrollo Integrado (IDEs)

Un **IDE** (Integrated Development Environment) es una herramienta más completa que combina un editor de código, un compilador o intérprete, herramientas de depuración y otras funciones. Los IDEs están diseñados para facilitar el desarrollo de software en lenguajes específicos. Algunos populares incluyen:

- **IntelliJ IDEA**: Amplia funcionalidad para desarrollo en Java, con herramientas avanzadas de refactorización y depuración.
- **Eclipse**: Un IDE potente y gratuito para múltiples lenguajes, aunque especialmente popular en Java.
- **NetBeans**: Ideal para desarrollo en Java y compatible con varios lenguajes.
- **PyCharm**: Especializado en Python, con soporte avanzado para análisis de datos y desarrollo web.

Compiladores e Intérpretes

Dependiendo del lenguaje, los programas requieren un compilador (convierte el código fuente en código máquina) o un intérprete (ejecuta el código directamente). Ejemplos incluyen:

- **Java Compiler (javac)**: Compila el código Java en bytecode.
- **Python Interpreter**: Ejecuta scripts de Python.
- **GCC (GNU Compiler Collection)**: Una colección de compiladores para lenguajes como C, C++ y Fortran.

Herramientas de Depuración

El proceso de depuración es esencial para encontrar y corregir errores en el código. Algunas herramientas destacadas son:

- **Debuggers integrados en IDEs**: Como el depurador de IntelliJ o Eclipse.
- **GDB (GNU Debugger)**: Una herramienta poderosa para depurar programas escritos en C y C++.
- **Postman**: Útil para probar y depurar APIs.

Herramientas de Diseño y Diagramación

Antes de escribir código, es importante planificar utilizando herramientas de diseño y diagramación:

- **Draw.io** y **Lucidchart**: Para crear diagramas de flujo y modelos de procesos.
- **UMLet**: Especializada en diagramas UML.

1.4. Introducción al lenguaje Java

Java es uno de los lenguajes de programación más populares y ampliamente utilizados en el mundo. Diseñado para ser simple, seguro, y portable, Java se utiliza en una gran variedad de aplicaciones, desde desarrollo de software empresarial y aplicaciones móviles hasta sistemas embebidos y desarrollo web.

Ilustración 4 - Logotipo Java

Java fue creado por **James Gosling** y su equipo en Sun Microsystems en 1995, originalmente concebido como un lenguaje para dispositivos electrónicos, pronto se convirtió en un estándar para aplicaciones multiplataforma gracias a su lema "escribe una vez, ejecuta en cualquier lugar" (*Write Once, Run Anywhere*), con la adquisición de Sun Microsystems por Oracle Corporation en 2010, Java continuó evolucionando y consolidándose como uno de los lenguajes más influyentes en el desarrollo de software.

Java es ideal para principiantes y profesionales debido a su facilidad de aprendizaje y su amplia aplicabilidad, es un lenguaje versátil que te preparará para trabajar en proyectos del mundo real y abordar problemas complejos con soluciones escalables.

A lo largo de este libro, Java será el lenguaje que utilizaremos para implementar algoritmos, diagramas de flujo y soluciones a problemas de programación.

CAPÍTULO 2 - CONCEPTOS BÁSICOS

2.1. Algoritmos: Definición y características

Un **algoritmo** es un conjunto finito de pasos organizados de manera lógica y secuencial que se utilizan para resolver un problema o realizar una tarea específica, en términos simples, un algoritmo es una receta que indica cómo realizar una acción de principio a fin.

Los algoritmos son la base de la programación, ya que todo programa informático es, esencialmente, una implementación de uno o varios algoritmos.

Características de los Algoritmos

Un algoritmo debe cumplir con ciertas características esenciales para ser considerado válido y eficiente:

- **Finitud:** El algoritmo debe tener un número finito de pasos y debe terminar en algún punto. Un algoritmo que no concluye no es útil.
- **Definición:** Cada paso del algoritmo debe estar claramente definido, sin ambigüedades. Esto significa que cualquier persona o máquina que lo lea debe entender exactamente qué hacer en cada paso.
- **Entradas:** Un algoritmo puede requerir datos de entrada para comenzar su ejecución. Estas entradas deben ser bien definidas y conocidas previamente.
- **Salidas:** Todo algoritmo debe producir al menos una salida o resultado. Esta salida es la solución al problema planteado.
- **Eficiencia:** El algoritmo debe utilizar los recursos de manera óptima, como tiempo y memoria. Esto es importante especialmente en sistemas con recursos limitados.
- **Efectividad:** Cada paso del algoritmo debe ser realizable en un tiempo finito utilizando las herramientas disponibles. Los pasos deben ser prácticos y ejecutables.

- **Generalidad:** Un algoritmo debe ser aplicable a una clase general de problemas y no solo a un caso específico. Esto aumenta su utilidad y aplicabilidad.

2.2. Diagramas de flujo: Simbología y reglas

Un diagrama de flujo es una representación gráfica de un algoritmo o proceso que utiliza símbolos conectados por flechas para mostrar el flujo de las operaciones, es una herramienta fundamental en la programación, ya que permite visualizar y analizar la lógica de una solución antes de implementarla en código.

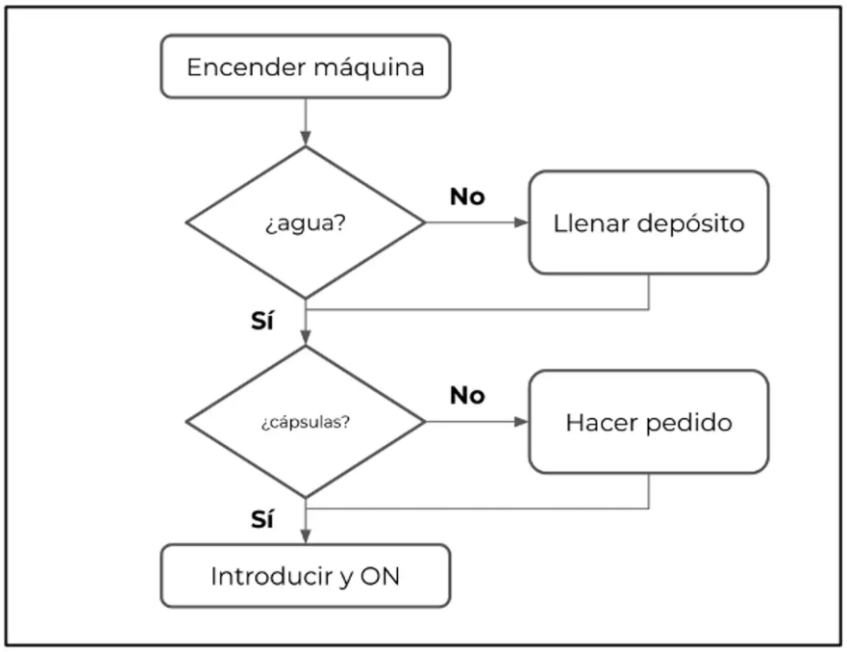


Ilustración 5 -Diagrama de Flujo
Fuente: https://datos.nexos.com.mx/que-es-un-algoritmo/

Al ser intuitivos y fáciles de interpretar, los diagramas de flujo son útiles tanto para programadores como para personas sin conocimientos técnicos que necesiten comprender un proceso.

Los diagramas de flujo emplean una simbología estándar que representa diferentes tipos de acciones o decisiones dentro de un proceso.

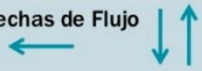
símbolo	Función	Símbolo	Función
Terminal 	Indicar el inicio y fin del diagrama	Teclado 	Introducir datos manualmente por el teclado
Entrada/salida 	Entrada o salida simple de información	Decisión 	Indica operaciones lógicas o de comparación y tienen dos salidas dependiendo del resultado.
Proceso 	Realizar cualquier operación o calculo con la información		
Salida a Impresora 	Salida de informacion a la impresora	Conectores 	Une dos partes del diagrama a la misma o diferente página
Salida a Pantalla 	Mostrar información de salida a la pantalla	Flechas de Flujo 	Indica la direccion del flujo de la información

Ilustración 6 - Elementos de un diagrama de flujo

Fuente: https://formacion.intef.es/representacin_de_un_algoritmo

El diseño de un diagrama de flujo debe seguir reglas básicas para garantizar su eficacia. Primero, debe tener un único punto de inicio y uno de fin, lo que asegura que el proceso sea bien delimitado, las flechas deben indicar el flujo del proceso de manera lógica, generalmente de arriba hacia abajo o de izquierda a derecha.

Las decisiones deben estar claramente marcadas con opciones posibles y conectadas a las acciones correspondientes, evitando ambigüedades. Además, es crucial mantener un diseño limpio y ordenado, especialmente en procesos complejos, para facilitar su lectura y análisis.

La importancia de los diagramas de flujo radica en su capacidad para simplificar la comunicación y el diseño de soluciones, son herramientas ideales para identificar errores o inconsistencias en la lógica antes de escribir código, lo que ahorra tiempo y recursos. de trabajo.

2.3. Introducción a las pruebas de escritorio

Las pruebas de escritorio, también conocidas como trazas de ejecución, son una técnica utilizada para verificar la lógica de un algoritmo o programa antes de ejecutarlo en una computadora. Consisten en

simular manualmente la ejecución del código, paso a paso, utilizando ejemplos concretos de datos de entrada y registrando los cambios en las variables y las salidas del programa.

Este enfoque ayuda a identificar errores lógicos y comprender el comportamiento del algoritmo en diferentes escenarios.

PRUEBA DE ESCRITORIO

	4		
h	i	j	
4	1	4	****
	2	3	***
	3	2	**
	4	1	*

```
Inicio
      Escriba "ingresar altura del
      triangulo"
      Lea h
      Para i=1 hasta h
            a=(h+1)-j
            Para j=1 hasta a
                  Escriba "*"
            Fin para
      Fin para
Fin
```

Ilustración 7 - Prueba de escritorio

El objetivo principal de las pruebas de escritorio es validar la lógica del programa, asegurarse de que cumple con los requisitos y anticiparse a posibles errores. Al simular manualmente la ejecución, es posible detectar problemas como condiciones incorrectas en estructuras de control, errores en cálculos o bucles infinitos, que de otro modo podrían no ser evidentes hasta que el programa esté en producción.

Realizar una prueba de escritorio implica seguir un enfoque estructurado, primero, se seleccionan datos de entrada representativos que incluyan casos comunes, límites y excepciones. Luego, se avanza paso a paso por el algoritmo, anotando los valores de las variables en cada etapa en una tabla organizada.

Esta tabla, conocida como tabla de trazabilidad, incluye columnas para las variables del programa, condiciones evaluadas, y las salidas generadas en cada paso.

	Test 1	Test 2	Test 3	Test 4
Req 1		x		
Req 2			x	
Req 3	x		x	
Req 4				x

Ilustración 8 - Tabla de trazabilidad

Las pruebas de escritorio son especialmente útiles en etapas iniciales del desarrollo de software, cuando los algoritmos están en fase de diseño. También son valiosas en educación, ya que ayudan a los estudiantes a comprender cómo se comportan las estructuras de control y los bucles al ejecutarse. Aunque no sustituyen las pruebas automatizadas, son un complemento eficaz para asegurar la calidad del código.

CAPITULO 3 - ELEMENTOS DE LA PROGRAMACIÓN

3. Estructuras Básicas

Las estructuras básicas son los componentes fundamentales de cualquier programa, representan las herramientas esenciales que permiten a un programador definir cómo se manejarán los datos, realizar cálculos y controlar el flujo de ejecución.

Comprender estas estructuras es crucial para desarrollar soluciones claras y eficientes, ya que constituyen la base sobre la cual se construyen programas más complejos.

Las estructuras básicas son el primer paso para comprender la lógica de programación, proveen los elementos esenciales para construir algoritmos que realicen cálculos, manipulen datos y presenten resultados, dominar estos conceptos es crucial antes de avanzar hacia estructuras de control y repetición más complejas.

En este capítulo, se estudiarán en detalle los tipos de datos, operadores y técnicas de entrada/salida, acompañados de ejercicios prácticos que te permitirán aplicarlos en situaciones reales. Esto sentará una base sólida para construir programas eficientes y funcionales.

3.1. Variables y tipos de datos

Las **variables** y los **tipos de datos** son conceptos esenciales en programación. Una variable es un espacio de memoria reservado para almacenar información que puede cambiar durante la ejecución del programa. Por otro lado, los **tipos de datos** definen el tipo de información que se puede almacenar en estas variables, como números, texto, valores booleanos, entre otros.

Comprender estos conceptos es fundamental, ya que todas las operaciones que realiza un programa giran en torno al manejo de datos.

Variables

Una variable tiene tres elementos principales:

- Nombre: Identificador único que se utiliza para referirse a la variable.
- Tipo de dato: Define qué tipo de valor puede almacenar.
- Valor: Representa la información actual almacenada en la variable.

En Java, las variables deben declararse antes de ser utilizadas. La declaración sigue el siguiente formato:

```
tipo nombre = valor;
```

```
int edad = 25;  // Declaración de una variable de tipo entero
```

Las variables en Java tienen un ámbito (*scope*), que determina dónde pueden ser utilizadas dentro del código. Existen diferentes tipos de variables:

- **Locales**: Definidas dentro de un método y accesibles solo en ese método.
- **De instancia**: Declaradas dentro de una clase, pero fuera de los métodos. Se asocian a un objeto específico.
- **Estáticas**: Declaradas con la palabra clave `static`, son compartidas por todas las instancias de la clase.

Tipos de Datos

Los tipos de datos en Java se clasifican en primitivos y no primitivos. Cada uno cumple un rol específico en el manejo de datos.

1. Tipos de Datos Primitivos

Los datos primitivos son los tipos más básicos y representan valores simples. Existen ocho tipos en Java:

TIPO	TAMAÑO	VALOR POR DEFECTO	EJEMPLO
BYTE	8 bits	0	byte b = 100;
SHORT	16 bits	0	short s = 32000;
INT	32 bits	0	int i = 2147483647;
LONG	64 bits	0L	long l = 922337203685477L;
FLOAT	32 bits	0.0f	float f = 3.14f;
DOUBLE	64 bits	0.0d	double d = 3.1415926535;
CHAR	16 bits	'\u0000'	char c = 'A';
BOOLEAN	1 bit	false	boolean b = true;

Los tipos float y double se utilizan para números decimales. float requiere el sufijo f y long el sufijo L al asignar valores.

2. Tipos de Datos No Primitivos

Los tipos no primitivos son más complejos y se basan en clases. Ejemplos comunes son:

- **String:** Almacena texto.
- **Arreglos:** Almacenan colecciones de elementos del mismo tipo.
- **Objetos:** Instancias de clases definidas por el programador.

Conversiones de Tipos de Datos

A veces, es necesario convertir un tipo de dato a otro. Java permite conversiones de manera explícita o implícita:

- **Conversión implícita (ampliación):** Cuando un tipo más pequeño se convierte en uno más grande automáticamente.

```
int numero = 10;
double numeroDecimal = numero;  // Conversión automática
```

- **Conversión explícita (reducción)**: Requiere el uso de un cast para evitar pérdida de datos.

```
double decimal = 5.8;
int entero = (int) decimal;  // Conversión forzada
```

Ejemplo Práctico

El siguiente programa solicita al usuario ingresar dos números y calcula su suma, demostrando el uso de variables y tipos de datos:

```java
import java.util.Scanner;

public class Suma {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Declaración de variables
        System.out.println("Ingrese el primer número:");
        int numero1 = scanner.nextInt();

        System.out.println("Ingrese el segundo número:");
        int numero2 = scanner.nextInt();

        // Cálculo y salida
        int suma = numero1 + numero2;
        System.out.println("La suma es: " + suma);
    }
}
```

Dominar las variables y los tipos de datos es esencial para escribir programas eficientes y libres de errores. Elegir el tipo correcto garantiza un uso óptimo de los recursos y facilita el diseño de algoritmos robustos.

3.2. Operadores: Aritméticos, relacionales y lógicos

Los operadores son símbolos que permiten realizar operaciones sobre uno o más valores en un programa. En Java, los operadores se clasifican en varias categorías según su propósito, entre las que destacan los operadores aritméticos, relacionales y lógicos.

Estos operadores son fundamentales para implementar cálculos, tomar decisiones y controlar el flujo de ejecución de un programa.

Operadores Aritméticos

Los operadores aritméticos se utilizan para realizar operaciones matemáticas sobre valores numéricos, son los más básicos y se aplican a variables de tipo numérico como int, float y double.

Operador	Significado	Ejemplo	Resultado
+	Suma	5 + 3	8
-	Resta	5 - 3	2
*	Multiplicación	5 * 3	15
/	División	6 / 2	3
%	Módulo (residuo)	5 % 2	1

Ejemplo:

```
int a = 10, b = 3;
int suma = a + b;     // 13
int resta = a - b;    // 7
int producto = a * b; // 30
int division = a / b; // 3
int residuo = a % b; // 1
System.out.println("Suma: " + suma);
```

Operadores Relacionales

Los operadores relacionales comparan dos valores y producen un resultado de tipo booleano (true o false). Se utilizan principalmente en estructuras de control como if, while, o for.

Operador	Significado	Ejemplo	Resultado
0	Igual a	5 == 5	true
!=	Diferente de	5 != 3	true
>	Mayor que	5 > 3	true
<	Menor que	5 < 3	false
>=	Mayor o igual que	5 >= 5	true
<=	Menor o igual que	3 <= 5	true

Ejemplo:

```
int x = 10, y = 20;
boolean esMayor = x > y;       // false
boolean sonIguales = x == y;   // false
System.out.println("Es mayor: " + esMayor);
System.out.println("Son iguales: " + sonIguales);
```

Operadores Lógicos

Los operadores lógicos trabajan con valores booleanos y permiten combinar o modificar condiciones. Se utilizan para construir expresiones lógicas más complejas.

Operador	*Significado*	*Ejemplo*	*Resultado*
&&	Y lógico (AND)	(5 > 3) && (3 > 1)	true
||	O lógico (OR)	(5 > 3) || (3 < 1)	true
!	Negación lógica (NOT)	!(5 > 3)	false

Ejemplo:

```
boolean cond1 = (5 > 3);      // true
boolean cond2 = (3 < 1);      // false

boolean resultadoAND = cond1 && cond2; // false
boolean resultadoOR = cond1 || cond2;  // true
boolean resultadoNOT = !cond1;         // false

System.out.println("Resultado AND: " + resultadoAND);
System.out.println("Resultado OR: " + resultadoOR);
System.out.println("Resultado NOT: " + resultadoNOT);
```

Ejemplo Práctico: Uso de Operadores Combinados

El siguiente programa utiliza operadores aritméticos, relacionales y lógicos para determinar si un número es divisible entre dos valores dados.

```java
import java.util.Scanner;

public class Divisibilidad {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Entrada de datos
        System.out.println("Ingrese un número:");
        int numero = scanner.nextInt();

        System.out.println("Ingrese el divisor 1:");
        int divisor1 = scanner.nextInt();

        System.out.println("Ingrese el divisor 2:");
        int divisor2 = scanner.nextInt();

        // Operadores aritméticos y relacionales
        boolean divisiblePor1 = (numero % divisor1 == 0);
        boolean divisiblePor2 = (numero % divisor2 == 0);

        // Operadores lógicos
        boolean divisiblePorAmbos = divisiblePor1 && divisiblePor2;

        // Resultados
        System.out.println("Es divisible por " + divisor1 + ": " + divisiblePor1);
        System.out.println("Es divisible por " + divisor2 + ": " + divisiblePor2);
        System.out.println("Es divisible por ambos: " + divisiblePorAmbos);
    }
}
```

Los operadores son esenciales para cualquier lenguaje de programación, ya que permiten realizar cálculos, tomar decisiones y construir expresiones complejas.

El conocimiento profundo de los operadores aritméticos, relacionales y lógicos te permitirá desarrollar programas eficientes y resolver problemas de manera estructurada.

Estos operadores serán fundamentales en las siguientes secciones, donde se abordarán las estructuras de control y los ciclos.

3.3. Entrada y salida de datos

Java utiliza la clase Scanner del paquete java.util para capturar datos de entrada desde el teclado. Esta clase proporciona métodos para leer diferentes tipos de datos, como enteros, decimales, cadenas de texto y más.

Pasos para Capturar Datos:

1. Importar la clase Scanner.

```
import java.util.Scanner;
```

2. Crear un objeto de tipo Scanner asociado a la entrada estándar (System.in).

```
Scanner scanner = new Scanner(System.in);
```

3. Utilizar los métodos de la clase `Scanner` para leer datos.

MÉTODO	DESCRIPCIÓN	EJEMPLO
NEXTINT()	Lee un número entero	scanner.nextInt();
NEXTDOUBLE()	Lee un número decimal	scanner.nextDouble();
NEXTLINE()	Lee una línea completa de texto	scanner.nextLine();
NEXT()	Lee una palabra (sin espacios)	scanner.next();
NEXTBOOLEAN()	Lee un valor booleano (true o false)	scanner.nextBoolean();

Ejemplo Básico de Entrada:

```
import java.util.Scanner;

public class EntradaEjemplo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Ingrese su nombre: ");
        String nombre = scanner.nextLine();

        System.out.print("Ingrese su edad: ");
        int edad = scanner.nextInt();

        System.out.println("Hola " + nombre + ", tienes " + edad + " años.");
    }
}
```

Salida de Datos

Java utiliza la clase System para manejar la salida estándar (generalmente la consola). Los métodos más comunes son:

- **System.out.print():** Muestra un mensaje sin un salto de línea al final.
- **System.out.println():** Muestra un mensaje y añade un salto de línea al final.
- **System.out.printf():** Permite formatear la salida con mayor control.

Ejemplos de Salida:

print() y println():

```
System.out.print("Este es un mensaje");
System.out.println(" en la misma línea.");
System.out.println("Este está en otra línea.");
```

```
Este es un mensaje en la misma línea.
Este está en otra línea.
```

printf():

```
double precio = 10.5;
System.out.printf("El precio es %.2f dólares.\n", precio);
```

```
El precio es 10.50 dólares.
```

En el ejemplo, %.2f indica que el número decimal debe mostrarse con dos cifras después del punto.

Ejemplo Práctico: Cálculo de Promedio

El siguiente programa solicita al usuario tres calificaciones, calcula el promedio y muestra el resultado:

```java
import java.util.Scanner;

public class CalculoPromedio {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Entrada de datos
        System.out.print("Ingrese la primera calificación: ");
        double calificacion1 = scanner.nextDouble();

        System.out.print("Ingrese la segunda calificación: ");
        double calificacion2 = scanner.nextDouble();

        System.out.print("Ingrese la tercera calificación: ");
        double calificacion3 = scanner.nextDouble();

        // Cálculo
        double promedio = (calificacion1 + calificacion2 + calificacion3) / 3;

        // Salida de datos
        System.out.printf("El promedio de las calificaciones es: %.2f\n", promedio);
    }
}
```

La entrada y salida de datos son esenciales para que un programa sea interactivo y útil. Permiten al usuario proporcionar información al sistema y recibir resultados de manera clara y comprensible, el manejo correcto de estos procesos mejora la experiencia del usuario y asegura que los datos se procesen adecuadamente.

En los siguientes capítulos, estas técnicas se combinarán con estructuras de control y repetición para crear programas más dinámicos y funcionales.

3.4. Ejemplos prácticos y problemas resueltos

A continuación, se presentan ejercicios prácticos enfocados en la utilización de variables, tipos de datos, operadores, y entrada/salida de datos en Java. Estos problemas están diseñados para reforzar los conceptos básicos y desarrollar habilidades prácticas.

Escribe un programa que solicite dos números al usuario y muestre su suma.

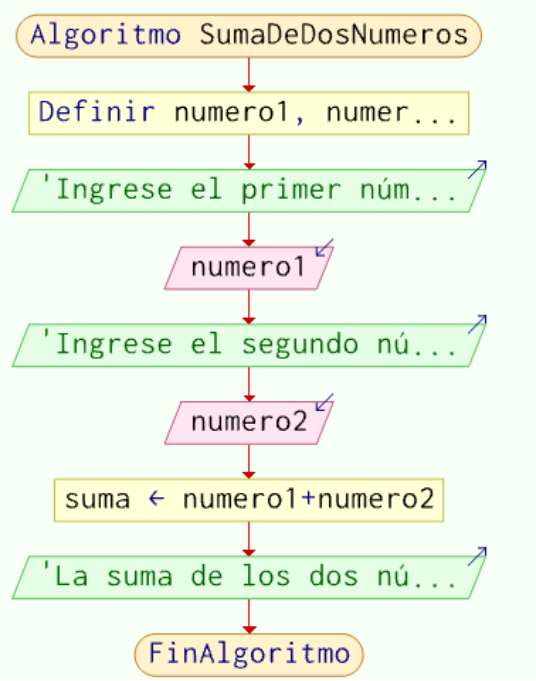
Algoritmo

Inicio.
Solicitar al usuario el primer número.
Solicitar al usuario el segundo número.
Calcular la suma de ambos números.
Mostrar el resultado de la suma.
Fin.

Pseudocódigo

```
Algoritmo SumaDeDosNumeros
    Definir numero1, numero2, suma Como Real
    Escribir "Ingrese el primer número:"
    Leer numero1
    Escribir "Ingrese el segundo número:"
    Leer numero2
    suma <- numero1 + numero2
    Escribir "La suma de los dos números es: ", suma
FinAlgoritmo
```

Diagrama de Flujo



Código Fuente en Java

```java
import java.util.Scanner;

public class SumaDeDosNumeros {
    public static void main(String[] args) {
        // Crear un objeto Scanner para leer los datos de entrada
        Scanner scanner = new Scanner(System.in);

        // Solicitar el primer número
        System.out.print("Ingrese el primer número: ");
        double numero1 = scanner.nextDouble();

        // Solicitar el segundo número
        System.out.print("Ingrese el segundo número: ");
        double numero2 = scanner.nextDouble();

        // Calcular la suma
        double suma = numero1 + numero2;

        // Mostrar el resultado
        System.out.println("La suma de los dos números es: " + suma);
    }
}
```

Calcula el área de un rectángulo leyendo su base y altura.

Algoritmo

Inicio.

Solicitar al usuario la base del rectángulo.

Solicitar al usuario la altura del rectángulo.

Calcular el área como area=base×altura

Mostrar el resultado del área.

Fin.

Pseudocódigo

```
Algoritmo CalculoAreaRectangulo
    Definir base, altura, area Como Real
    Escribir "Ingrese la base del rectángulo:"
    Leer base
    Escribir "Ingrese la altura del rectángulo:"
    Leer altura
    area <- base * altura
    Escribir "El área del rectángulo es: ", area
FinAlgoritmo
```

Diagrama de flujo

Código Fuente en Java

```java
import java.util.Scanner;

public class CalculoAreaRectangulo {
    public static void main(String[] args) {
        // Crear un objeto Scanner para la entrada de datos
        Scanner scanner = new Scanner(System.in);

        // Solicitar la base del rectángulo
        System.out.print("Ingrese la base del rectángulo: ");
        double base = scanner.nextDouble();

        // Solicitar la altura del rectángulo
        System.out.print("Ingrese la altura del rectángulo: ");
        double altura = scanner.nextDouble();

        // Calcular el área
        double area = base * altura;

        // Mostrar el resultado
        System.out.println("El área del rectángulo es: " + area);
    }
}
```

Pide al usuario el radio de un círculo y calcula su perímetro.

Algoritmo

Inicio.

Solicitar al usuario el radio del círculo.

Calcular el perímetro usando la fórmula P=2πr

Mostrar el resultado del perímetro.

Fin.

Pseudocódigo

Algoritmo CalculoPerimetroCirculo

Definir radio, perimetro Como Real

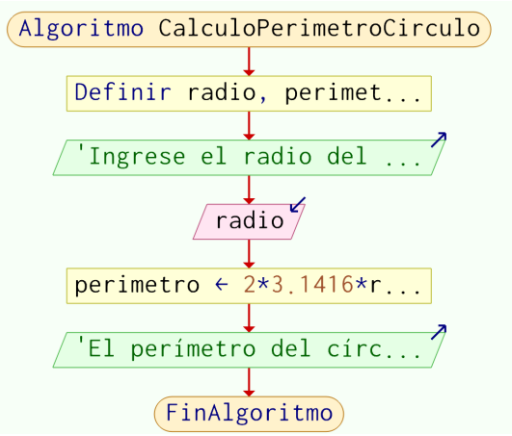
Constante PI <- 3.1416

Escribir "Ingrese el radio del círculo:"

Leer radio

```
    perimetro <- 2 * PI * radio
    Escribir "El perímetro del círculo es: ", perimetro
FinAlgoritmo
```

Diagrama de flujo



Código Fuente

```java
import java.util.Scanner;

public class CalculoPerimetroCirculo {
    public static void main(String[] args) {
        // Crear un objeto Scanner para la entrada de datos
        Scanner scanner = new Scanner(System.in);

        // Solicitar el radio del círculo
        System.out.print("Ingrese el radio del círculo: ");
        double radio = scanner.nextDouble();

        // Calcular el perímetro
        double perimetro = 2 * Math.PI * radio;

        // Mostrar el resultado
        System.out.println("El perímetro del círculo es: " + perimetro);
    }
}
```

Ejercicios Propuestos

- Solicita tres números al usuario y calcula su promedio.
- Convierte una temperatura dada en Celsius a Fahrenheit.
- Implementa una calculadora que permita sumar, restar, multiplicar y dividir dos números ingresados por el usuario.
- Lee el nombre y apellido del usuario y muestra su nombre completo.
- Solicita una palabra al usuario y muestra cuántos caracteres tiene.
- Pide al usuario una palabra y muestra la primera letra.
- Solicita una oración y muestra la misma en mayúsculas, minúsculas y con los espacios eliminados.
- Solicita al usuario su edad en años y calcula cuántos días ha vivido aproximadamente.
- Calcula la velocidad promedio de un objeto leyendo la distancia recorrida y el tiempo empleado.
- Pide al usuario las horas trabajadas y su salario por hora, y calcula su salario semanal.
- Convierte una cantidad de dinero dada en dólares a euros (suponiendo una tasa de conversión fija).
- Lee dos valores, intercambia sus contenidos y muestra el resultado.

UNIDAD 4 - CONTROL DE FLUJO

4.1. Introducción a las estructuras de control

En programación, las estructuras de control son fundamentales para dirigir el flujo de ejecución de un programa, estas estructuras permiten que el programa tome decisiones, ejecute bloques de código de manera repetitiva o altere su comportamiento en función de ciertas condiciones.

Las estructuras de control son esenciales porque:

- **Hacen los programas dinámicos:** Permiten que los programas respondan a diferentes entradas y escenarios.
- **Facilitan la solución de problemas:** Ayudan a dividir problemas en subprocesos lógicos.

Sin las estructuras de control, un programa seguiría una secuencia lineal de instrucciones, lo que limitaría severamente su funcionalidad.

4.2. Condicionales

Las estructuras condicionales son herramientas fundamentales en programación que permiten ejecutar diferentes bloques de código dependiendo de si se cumple o no una condición específica.

Estas estructuras son esenciales para tomar decisiones y hacer que el programa responda de manera dinámica a diferentes situaciones.

4.2.1. IF Y IF-ELSE

IF

Es la estructura condicional más simple, evalúa una condición y, si es verdadera, ejecuta el bloque de código asociado.

Sintaxis:

```
if (condición) {
    // Código a ejecutar si la condición es verdadera
}
```

Ejemplo:

```
int numero = 10;
if (numero > 5) {
    System.out.println("El número es mayor que 5.");
}
```

IF-ELSE

Permite definir un bloque alternativo de código que se ejecuta si la condición es falsa.

Sintaxis:

```
if (condición) {
    // Código si la condición es verdadera
} else {
    // Código si la condición es falsa
}
```

Ejemplo:

```
int numero = 3;
if (numero > 5) {
    System.out.println("El número es mayor que 5.");
} else {
    System.out.println("El número es menor o igual a 5.");
}
```

IF-ELSE Anidado

Se utiliza para evaluar múltiples condiciones en orden jerárquico.

Sintaxis:

```
if (condición1) {
    // Código si condición1 es verdadera
} else if (condición2) {
    // Código si condición2 es verdadera
} else {
    // Código si ninguna condición anterior es verdadera
}
```

Ejemplo:

```
int numero = 7;
if (numero > 10) {
    System.out.println("El número es mayor que 10.");
} else if (numero > 5) {
    System.out.println("El número está entre 6 y 10.");
} else {
    System.out.println("El número es 5 o menor.");
}
```

4.2.2. Switch-case

Es una alternativa más legible al uso de múltiples IF-ELSE IF cuando se necesita evaluar una variable contra múltiples valores posibles.

Sintaxis:

```
switch (variable) {
    case valor1:
        // Código si variable == valor1
        break;
    case valor2:
        // Código si variable == valor2
        break;
    default:
        // Código si ninguno de los casos anteriores coincide
        break;
}
```

Ejemplo:

```
char letra = 'A';
switch (letra) {
    case 'A':
        System.out.println("La letra es A.");
        break;
    case 'B':
        System.out.println("La letra es B.");
        break;
    default:
        System.out.println("La letra no es A ni B.");
        break;
}
```

Ejemplo Práctico

Determinar si un número ingresado es positivo, negativo o cero.

Código en Java:

```java
import java.util.Scanner;

public class CondicionalesEjemplo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Entrada de datos
        System.out.print("Ingrese un número: ");
        int numero = scanner.nextInt();

        // Condicionales
        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.");
        }
    }
}
```

Ejemplos prácticos y problemas resueltos

A continuación, se presentan ejercicios prácticos enfocados en la utilización de condicionales.

Escribe un programa que solicite un número e indique si es positivo.

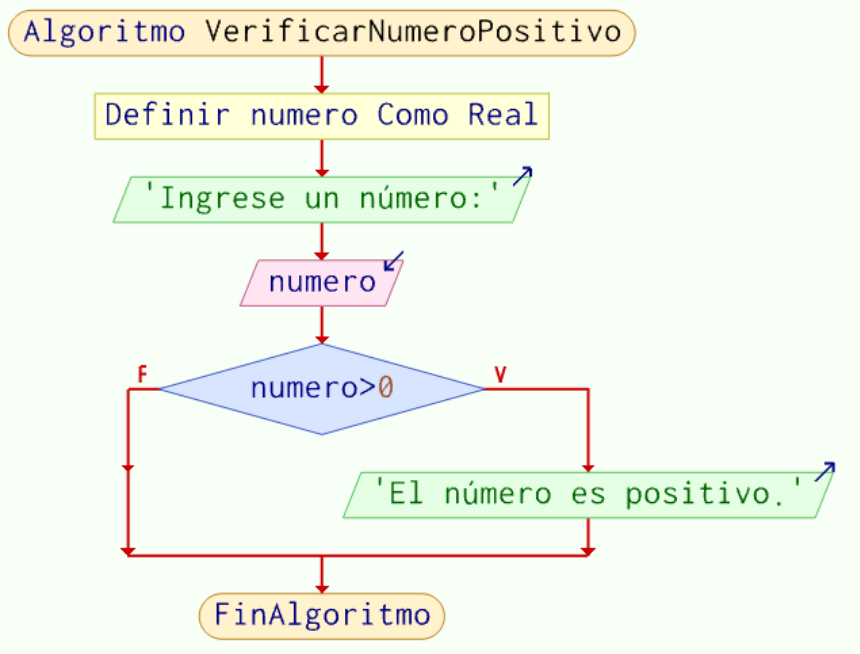
Algoritmo

Inicio.
Solicitar al usuario que ingrese un número.
Leer el número ingresado.
Si el número es mayor que 0, entonces:
    Mostrar "El número es positivo".
Fin.

Pseudocódigo

```
Algoritmo VerificarNumeroPositivo
    Definir numero Como Real
    Escribir "Ingrese un número:"
    Leer numero
    Si numero > 0 Entonces
        Escribir "El número es positivo."
    Fin Si
FinAlgoritmo
```

Diagrama de Flujo



Código Fuente en Java

```java
import java.util.Scanner;

public class VerificarNumeroPositivo {
    public static void main(String[] args) {
        // Crear un objeto Scanner para la entrada de datos
        Scanner scanner = new Scanner(System.in);

        // Solicitar al usuario que ingrese un número
        System.out.print("Ingrese un número: ");
        double numero = scanner.nextDouble();

        // Verificar si el número es positivo
        if (numero > 0) {
            System.out.println("El número es positivo.");
        }
    }
}
```

Solicita la edad del usuario y determina si es mayor de 18 años.

Algoritmo

Inicio.

Solicitar al usuario que ingrese su edad.

Leer la edad ingresada.

Si la edad es mayor o igual a 18, entonces:

    Mostrar "Eres mayor de edad."

Si no, entonces:

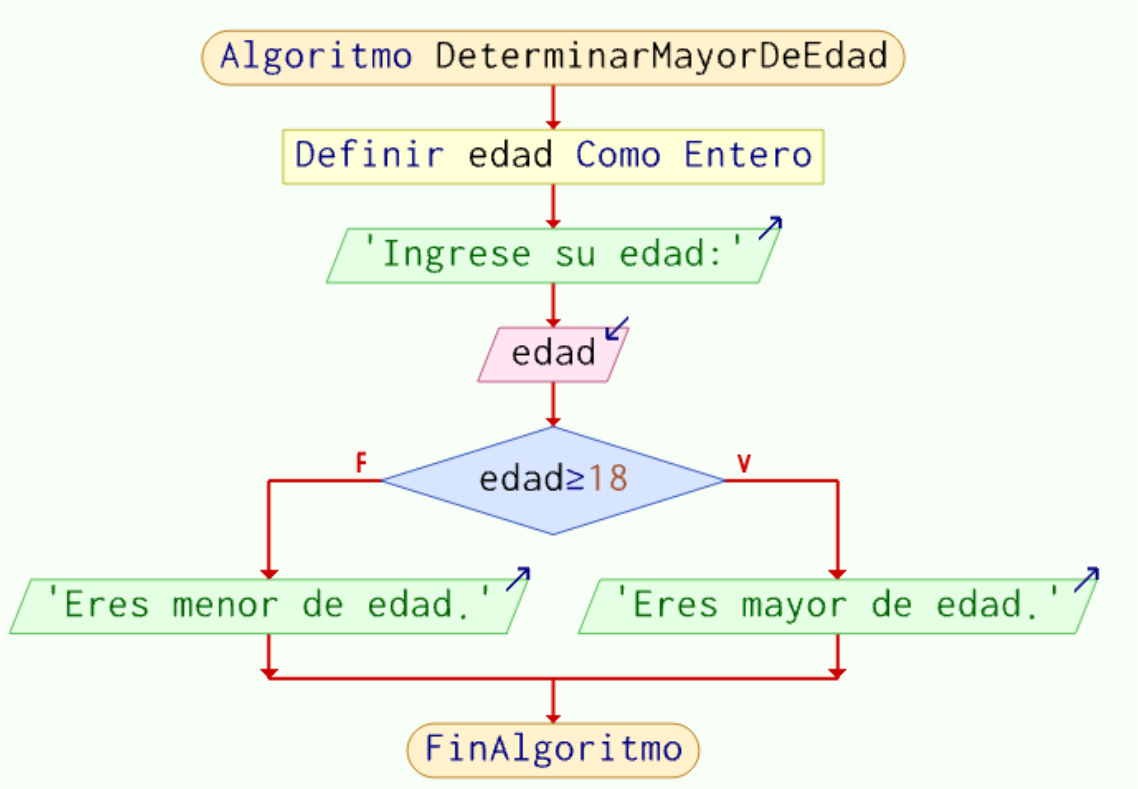
    Mostrar "Eres menor de edad."

Fin.

Pseudocódigo

```
Algoritmo DeterminarMayorDeEdad
    Definir edad Como Entero
    Escribir "Ingrese su edad:"
    Leer edad
    Si edad >= 18 Entonces
        Escribir "Eres mayor de edad."
    Sino
        Escribir "Eres menor de edad."
    Fin Si
FinAlgoritmo
```

Diagrama de Flujo



Código Fuente en Java

```java
import java.util.Scanner;

public class DeterminarMayorDeEdad {
    public static void main(String[] args) {
        // Crear un objeto Scanner para la entrada de datos
        Scanner scanner = new Scanner(System.in);

        // Solicitar al usuario que ingrese su edad
        System.out.print("Ingrese su edad: ");
        int edad = scanner.nextInt();

        // Verificar si es mayor o menor de edad
        if (edad >= 18) {
            System.out.println("Eres mayor de edad.");
        } else {
            System.out.println("Eres menor de edad.");
        }
    }
}
```

Solicita una calificación y muestra un mensaje si es mayor o igual a 7.

Algoritmo

Inicio.

Solicitar al usuario que ingrese una calificación.

Leer la calificación ingresada.

Si la calificación es mayor o igual a 7, entonces:

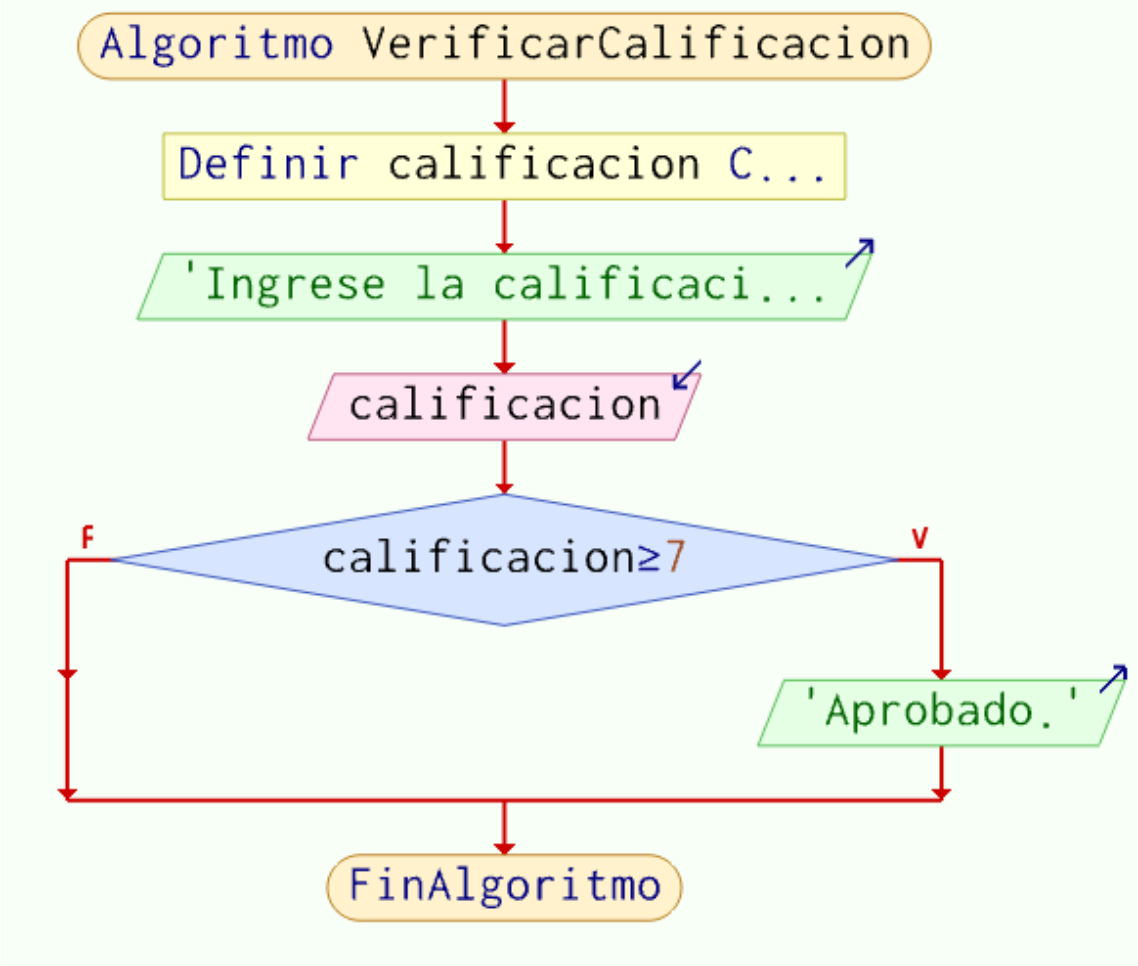
    Mostrar "Aprobado".

Fin.

Pseudocódigo

```
Algoritmo VerificarCalificacion
    Definir calificacion Como Real
    Escribir "Ingrese la calificación:"
    Leer calificacion
    Si calificacion >= 7 Entonces
        Escribir "Aprobado."
    Fin Si
FinAlgoritmo
```

Diagrama de Flujo



Código Fuente en Java

```java
import java.util.Scanner;

public class VerificarCalificacion {
    public static void main(String[] args) {
        // Crear un objeto Scanner para la entrada de datos
        Scanner scanner = new Scanner(System.in);

        // Solicitar al usuario que ingrese una calificación
        System.out.print("Ingrese la calificación: ");
        double calificacion = scanner.nextDouble();

        // Verificar si la calificación es mayor o igual a 7
        if (calificacion >= 7) {
            System.out.println("Aprobado.");
        }
    }
}
```

Ejercicios Propuestos

- Pregunta si el cliente tiene un cupón y, si lo tiene, muestra un mensaje que diga "Descuento aplicado".
- Solicita un número y muestra si es positivo o negativo.
- Pide al usuario un número e indica si es par o impar.
- Solicita dos números y determina cuál es mayor o si son iguales.
- Solicita una calificación y muestra si el estudiante aprobó (≥6) o reprobó.
- Pide al usuario su edad y muestra "Permitido" si es mayor de 18 y "No permitido" en caso contrario.
- Pide una edad e indica si es un niño (0-12), adolescente (13-17), adulto (18-64) o adulto mayor (65+).
- Solicita una calificación y muestra su equivalente en letras (A, B, C, D o F).

- Solicita un número e indica si está en el rango de 1-10, 11-20, 21-30 o fuera de esos rangos.
- Pide una temperatura y muestra si está "Fría" (<15°C), "Agradable" (15-25°C) o "Caliente" (>25°C).
- Solicita el monto de una compra e indica el descuento aplicable: 10% si es menos de $100, 15% si es entre $100 y $500, y 20% si supera $500.
- Solicita un número y muestra si está en el rango de 1 a 100. Si no lo está, indica "Fuera de rango".
- Pide el peso y la altura de una persona y calcula su índice de masa corporal (IMC). Luego, clasifica el resultado como "Bajo peso", "Normal", "Sobrepeso" u "Obesidad".
- Solicita dos números y una operación (+, -, *, /). Usa condicionales para realizar la operación correspondiente.
- Solicita una letra y muestra si es vocal, consonante o no es una letra.
- Pide las horas trabajadas en una semana y calcula el salario. Si trabajó más de 40 horas, calcula las horas extra con un 50% adicional.

UNIDAD 5: ESTRUCTURAS DE REPETICIÓN

5. Ciclos de Repetición

Los ciclos de repetición son estructuras fundamentales en programación que permiten ejecutar un bloque de código varias veces, ya sea mientras se cumple una condición o un número específico de veces. Estas estructuras son esenciales para realizar tareas repetitivas y automatizar procesos.

5.1. Introducción a los ciclos

Un ciclo de repetición ejecuta un conjunto de instrucciones de manera continua hasta que una condición específica deja de cumplirse. Esto elimina la necesidad de escribir manualmente varias líneas de código para realizar tareas repetitivas.

Partes de un ciclo:

- Inicio: Punto donde comienza la ejecución del ciclo.
- Condición: Criterio que determina si el ciclo continúa o termina.
- Incremento/Decremento: Ajuste del valor que controla la condición.
- Cuerpo del ciclo: Código que se ejecuta en cada iteración.

5.2. WHILE: Concepto y aplicaciones

El ciclo WHILE ejecuta un bloque de código repetidamente mientras una condición sea verdadera. Es ideal para situaciones donde no se conoce de antemano cuántas veces debe ejecutarse el ciclo.

Sintaxis:

```
while (condición) {
    // Código a ejecutar mientras la condición sea verdadera
}
```

Ejemplo: Imprimir los números del 1 al 5.

```
int i = 1;
while (i <= 5) {
    System.out.println(i);
    i++;
}
```

5.3. Do-While: Comparación con While

El ciclo do-while es similar a while, pero garantiza que el bloque de código se ejecute al menos una vez, ya que la condición se evalúa al final de la iteración.

```
do {
    // Código a ejecutar al menos una vez
} while (condición);
```

Ejemplo: Solicitar una contraseña hasta que sea correcta.

```
String contraseña;
do {
    System.out.print("Ingrese su contraseña: ");
    contraseña = scanner.nextLine();
} while (!contraseña.equals("1234"));
```

Aspecto	While	Do-While
Evaluación	Al inicio del ciclo.	Al final del ciclo.
Ejecución inicial	Puede no ejecutarse nunca.	Se ejecuta al menos una vez.

5.4. For: Ciclo controlado por contador

El ciclo for es ideal para iteraciones controladas por un contador. Se utiliza cuando se conoce de antemano el número de repeticiones necesarias.

Sintaxis:

```
for (inicialización; condición; incremento/decremento) {
    // Código a ejecutar
}
```

Ejemplo: Imprimir los números del 1 al 5.

```
for (int i = 1; i <= 5; i++) {
    System.out.println(i);
}
```

5.5. Anidación de ciclos

La anidación de ciclos ocurre cuando un ciclo se encuentra dentro del cuerpo de otro ciclo, es útil para trabajar con estructuras multidimensionales, como matrices.

Ejemplo: Mostrar una tabla de multiplicar.

```
for (int i = 1; i <= 3; i++) {
    for (int j = 1; j <= 3; j++) {
        System.out.println(i + " x " + j + " = " + (i * j));
    }
}
```

Resumen de los Ciclos en Java

Tipo	Condición	Uso Común
`while`	Al inicio del ciclo	Cuando no se conoce el número de iteraciones.
`do-while`	Al final del ciclo	Garantizar al menos una ejecución.
`for`	Al inicio del ciclo	Iteraciones controladas por un contador.

Los ciclos permiten automatizar tareas repetitivas de manera eficiente y son esenciales para resolver problemas de programación. Cada tipo tiene su propio caso de uso, y elegir el adecuado depende del problema específico.

Ejemplos prácticos y problemas resueltos

Suma de números consecutivos: Calcula la suma de los números del 1 al 100.

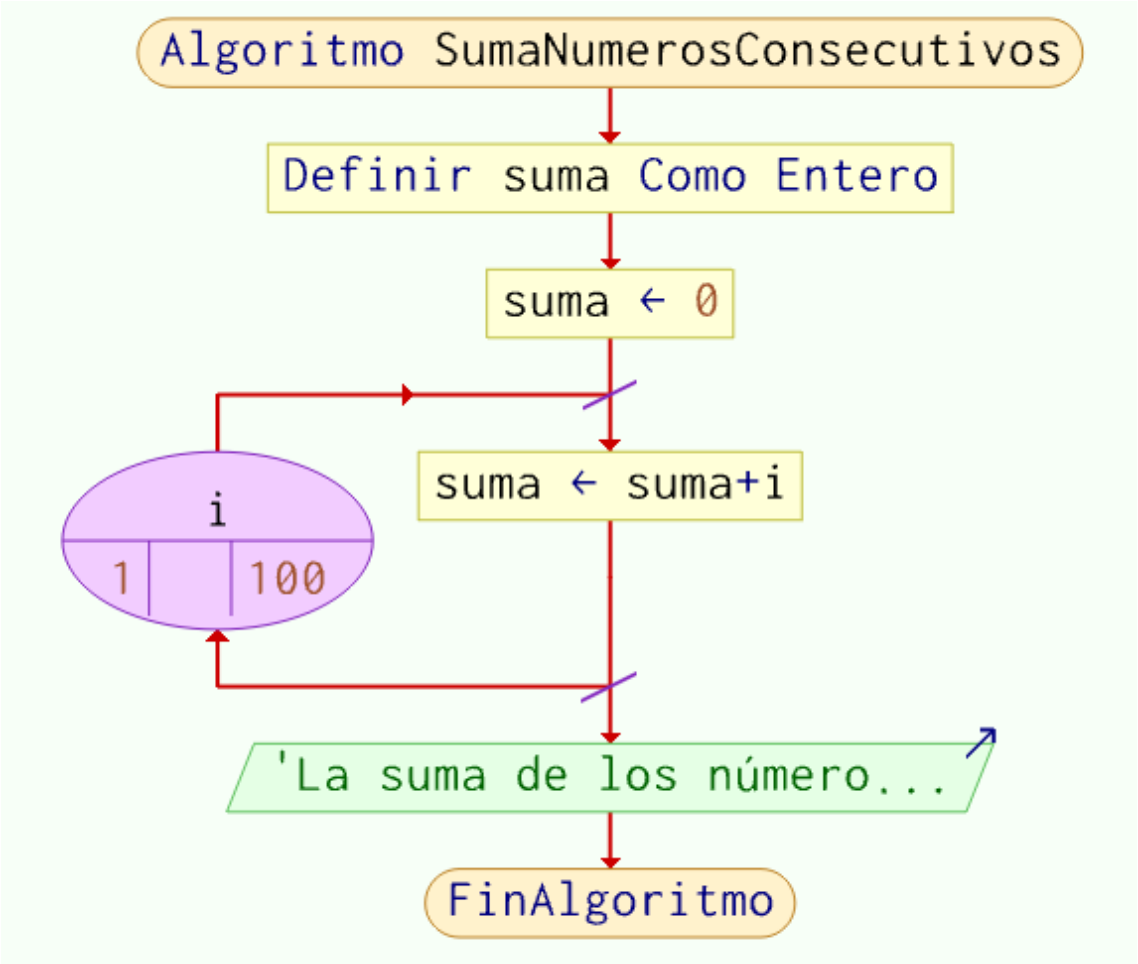
Algoritmo

```
Algoritmo SumaNumerosConsecutivos
    Definir suma Como Entero
    suma <- 0
    Para i <- 1 Hasta 100 Hacer
        suma <- suma + i
    Fin Para
    Escribir "La suma de los números del 1 al 100 es: ", suma
FinAlgoritmo
```

Pseudocódigo

```
Algoritmo SumaNumerosConsecutivos
    Definir suma Como Entero
    suma <- 0
    Para i <- 1 Hasta 100 Hacer
        suma <- suma + i
    Fin Para
    Escribir "La suma de los números del 1 al 100 es: ", suma
FinAlgoritmo
```

Diagramas de flujo



Código Fuente

Usando un ciclo for:

```java
public class SumaNumerosConsecutivos {
    public static void main(String[] args) {
        int suma = 0;

        // Ciclo para sumar los números del 1 al 100
        for (int i = 1; i <= 100; i++) {
            suma += i;
        }

        // Mostrar el resultado
        System.out.println("La suma de los números del 1 al 100 es: " + suma);
    }
}
```

Usando un ciclo while:

```java
public class SumaNumerosConsecutivosWhile {
    public static void main(String[] args) {
        int suma = 0;
        int i = 1;

        // Ciclo while para sumar los números del 1 al 100
        while (i <= 100) {
            suma += i;
            i++;
        }

        // Mostrar el resultado
        System.out.println("La suma de los números del 1 al 100 es: " + suma);
    }
}
```

Usando un ciclo do-while:

```java
public class SumaNumerosConsecutivosDoWhile {
    public static void main(String[] args) {
        int suma = 0;
        int i = 1;

        // Ciclo do-while para sumar los números del 1 al 100
        do {
            suma += i;
            i++;
        } while (i <= 100);

        // Mostrar el resultado
        System.out.println("La suma de los números del 1 al 100 es: " + suma);
    }
}
```

Ejercicios Propuestos

Realiza los siguientes ejercicios con todos los ciclos de repetición.

- Números pares: Imprime todos los números pares entre 1 y 50.
- Promedio de calificaciones: Solicita al usuario ingresar 5 calificaciones y calcula el promedio.
- Contar dígitos: Solicita un número y muestra cuántos dígitos tiene.
- Suma de números ingresados: Permite al usuario ingresar números hasta que decida detenerse, luego muestra la suma total.
- Fibonacci: Imprime los primeros 10 términos de la serie de Fibonacci.
- Contar números positivos y negativos: Solicita al usuario ingresar números hasta que decida parar y cuenta cuántos son positivos y cuántos negativos.
- Tabla de potencias: Solicita un número base y muestra las potencias desde el 1 hasta el 5.
- Números primos: Determina si un número ingresado por el usuario es primo.
- Imprimir caracteres: Solicita una cadena de texto y muestra cada carácter por separado.
- Suma de números pares e impares: Solicita números consecutivos del 1 al 20 y calcula la suma de los pares y la suma de los impares.
- Inverso de un número: Pide un número entero positivo y muestra su inverso.
- Patrón de asteriscos: Imprime un triángulo de asteriscos con 5 niveles.
- Cálculo de promedio dinámico: Solicita números al usuario hasta que ingrese 0, luego calcula el promedio de los números ingresados.

- Contador de vocales: Solicita una palabra y cuenta cuántas vocales contiene.
- Número perfecto: Determina si un número ingresado es perfecto (la suma de sus divisores propios es igual al número).
- Suma de números múltiplos de 3 y 5: Calcula la suma de todos los números entre 1 y 100 que sean múltiplos de 3 o 5.
- Generador de números aleatorios: Genera 10 números aleatorios entre 1 y 100 e imprime el mayor y el menor.

Apéndices

A. Glosario de términos de programación

A

- **Algoritmo**: Secuencia finita de pasos organizados lógicamente para resolver un problema o realizar una tarea específica.
- **API (Application Programming Interface)**: Conjunto de funciones y procedimientos que permiten interactuar con bibliotecas, software o servicios externos.
- **Argumento**: Valor que se pasa a una función o método al invocarlo.
- **Array**: Estructura de datos que almacena una colección de elementos, generalmente del mismo tipo, en posiciones indexadas.

B

- **Bug**: Error o defecto en un programa que causa un comportamiento no deseado o incorrecto.
- **Booleano**: Tipo de dato que solo puede tener dos valores: true o false.
- **Buffer**: Espacio de memoria temporal utilizado para almacenar datos mientras se transfieren entre dos lugares.

C

- **Clase**: Plantilla para crear objetos en programación orientada a objetos. Define atributos y métodos comunes.
- **Compilador**: Programa que traduce código fuente escrito en un lenguaje de programación a un lenguaje de máquina ejecutable.

- **Condicional**: Estructura de control que ejecuta diferentes bloques de código según se cumpla o no una condición.
- **Constructor**: Método especial de una clase que se llama al crear un objeto y se utiliza para inicializarlo.
- **Constante**: Valor que no puede cambiar durante la ejecución de un programa.

D

- **Depuración (Debugging)**: Proceso de encontrar y corregir errores en un programa.
- **Diagrama de flujo**: Representación gráfica de un proceso o algoritmo utilizando símbolos estándar.
- **Directiva**: Instrucción que le indica al compilador o intérprete cómo procesar el código fuente.

E

- **Ejecución**: Proceso en el que un programa realiza las instrucciones especificadas en su código.
- **Estructura de control**: Bloques de código que modifican el flujo de ejecución, como condicionales o bucles.
- **Excepción**: Evento inesperado que ocurre durante la ejecución de un programa, generalmente causado por errores.

F

- **Función**: Bloque de código reutilizable que realiza una tarea específica y puede devolver un valor.

- **Framework**: Conjunto de herramientas y bibliotecas que facilitan el desarrollo de aplicaciones.

I

- **IDE (Integrated Development Environment)**: Entorno de desarrollo que incluye un editor de código, depurador, compilador y otras herramientas.
- **Iteración**: Ejecución repetida de un bloque de código, generalmente dentro de un bucle.

L

- **Librería**: Conjunto de funciones y procedimientos predefinidos que pueden ser utilizados en un programa.
- **Lenguaje de programación**: Conjunto de reglas y sintaxis que permiten a los programadores escribir instrucciones comprensibles por una computadora.

M

- **Método**: Función asociada a una clase o a un objeto.
- **Memoria**: Espacio utilizado por un programa para almacenar datos temporales o permanentes durante su ejecución.

O

- **Objeto**: Instancia de una clase que combina datos y métodos para representar una entidad en programación orientada a objetos.
- **Operador**: Símbolo que realiza operaciones matemáticas, lógicas o relacionales en valores o variables.

P

- **Pseudocódigo**: Representación simplificada y legible de un algoritmo, sin seguir la sintaxis estricta de un lenguaje de programación.
- **Polimorfismo**: Propiedad de la programación orientada a objetos que permite a un método comportarse de diferentes maneras dependiendo del objeto que lo invoque.
- **Puntero**: Variable que almacena la dirección de memoria de otro valor.

R

- **Recursión**: Técnica en la que una función se llama a sí misma para resolver subproblemas similares al problema original.
- **Return**: Palabra clave utilizada para devolver un valor desde una función o método.

S

- **Sintaxis**: Conjunto de reglas que definen cómo escribir correctamente instrucciones en un lenguaje de programación.

- **Stack**: Estructura de datos que sigue el principio LIFO (último en entrar, primero en salir).
- **String**: Tipo de dato que representa una secuencia de caracteres.

T

- **Tipo de dato**: Clasificación de datos en un programa, como entero, flotante, booleano o cadena.
- **Thread**: Unidad básica de ejecución en un programa que permite realizar tareas simultáneamente.

V

- **Variable**: Espacio de memoria con un nombre asociado que almacena un valor que puede cambiar durante la ejecución del programa.
- **Valor**: Información contenida dentro de una variable.

W

- **While**: Ciclo de repetición que ejecuta un bloque de código mientras una condición sea verdadera.

B. Referencias y recursos recomendados

A continuación, se presenta una lista de referencias y recursos recomendados para profundizar en los conceptos de programación, resolver dudas, y mejorar habilidades prácticas.

LIBROS RECOMENDADOS

1. **"Clean Code: A Handbook of Agile Software Craftsmanship"** - Robert C. Martin
    - Enfocado en escribir código limpio, legible y mantenible.
2. **"Introduction to Algorithms"** - Thomas H. Cormen et al.
    - Un texto esencial para entender algoritmos y estructuras de datos.
3. **"Java: The Complete Reference"** - Herbert Schildt
    - Una guía completa sobre el lenguaje de programación Java.
4. **"Python Crash Course"** - Eric Matthes
    - Ideal para principiantes en Python, con ejemplos prácticos.
5. **"Eloquent JavaScript"** - Marijn Haverbeke
    - Una introducción moderna y profunda a JavaScript.

CURSOS Y PLATAFORMAS ONLINE

1. **freeCodeCamp**
    - Ofrece cursos gratuitos en HTML, CSS, JavaScript, Python y más, con proyectos prácticos.
2. **Codecademy**
    - Cursos interactivos en varios lenguajes de programación.
3. **Coursera**
    - Cursos universitarios en programación y ciencia de datos, muchos ofrecidos gratuitamente.

4. **EdX**
    - Ofrece cursos de programación y tecnología impartidos por universidades reconocidas.
5. **Khan Academy**
    - Introducción a conceptos básicos de programación y matemáticas computacionales.

COMUNIDADES Y FOROS

1. **Stack Overflow**
    - Comunidad de programadores para resolver dudas específicas sobre código.
2. **Reddit**
    - Discusiones, noticias y recursos sobre programación.
3. **GitHub Discussions**
    - Foro para discutir proyectos y resolver dudas en repositorios de código.
4. **HackerRank**
    - Plataforma para practicar algoritmos y problemas de programación competitiva.

DOCUMENTACIÓN OFICIAL

1. **Java Documentation**: https://docs.oracle.com/en/java/
    - Referencia oficial para el lenguaje Java.
2. **Python Documentation**: https://docs.python.org/3/
    - Guía oficial para el lenguaje Python.
3. **JavaScript MDN**: https://developer.mozilla.org/en-US/docs/Web/JavaScript
    - Documentación oficial de JavaScript por Mozilla.
4. **C++ Reference**: https://en.cppreference.com/
    - Guía para programadores de C++.
5. **W3Schools**: https://www.w3schools.com/

- Tutoriales básicos en varios lenguajes y tecnologías web.

CANALES DE YOUTUBE

1. **Programming with Mosh**
   - Cursos claros sobre varios lenguajes de programación.
2. **Traversy Media**
   - Tutoriales de desarrollo web y herramientas modernas.
3. **The Net Ninja**
   - Guías paso a paso para aprender desarrollo web.
4. **Derek Banas**
   - Explicaciones detalladas y rápidas sobre lenguajes y conceptos de programación.
5. **Computerphile**
   - Conceptos teóricos y aplicaciones prácticas de la informática.

HERRAMIENTAS RECOMENDADAS

1. **Visual Studio Code**: Editor de código ligero y extensible.
2. **IntelliJ IDEA**: IDE ideal para proyectos en Java.
3. **Replit**: IDE en línea para ejecutar programas en múltiples lenguajes.
4. **Git**: Control de versiones para gestionar cambios en el código.
5. **Draw.io**: Herramienta gratuita para diagramas de flujo.

EJERCICIOS Y PROBLEMAS

1. **LeetCode**
   - Problemas de programación competitiva y entrevistas técnicas.

2. **Codewars**
    - Resuelve problemas de programación y mejora tus habilidades con desafíos.
3. **Project Euler**
    - Retos matemáticos y de programación para pensar lógicamente.
4. **GeeksforGeeks**
    - Ejercicios y tutoriales prácticos de estructuras de datos y algoritmos.
5. **Rosetta Code**
    - Ejemplos de soluciones en múltiples lenguajes de programación.