

## **Clase 1. Introducción a la Programación**

### **1.1 ¿Qué es y cómo surge la programación?**

La programación es el proceso de crear y diseñar programas de computadora mediante la escritura de código fuente que instruye a una computadora sobre qué hacer para resolver un problema o realizar una tarea específica. El código fuente es escrito en lenguajes de programación que son comprensibles para los programadores y luego se traduce a lenguaje de máquina que la computadora puede ejecutar.

La programación surge de la necesidad de automatizar tareas y realizar cálculos complejos de manera rápida y eficiente. En sus inicios, la programación estaba relacionada principalmente con la codificación de instrucciones en lenguaje de máquina, que era una tarea tediosa y propensa a errores. Sin embargo, a lo largo del tiempo, se desarrollaron lenguajes de programación de alto nivel que facilitaron la tarea de programar y permitieron una mayor abstracción y expresividad en el código.

[\(1141\) HISTORIA DE LA PROGRAMACION EN 5 MINUTOS - YouTube](#)

### **1.2 ¿Qué es un algoritmo?**

En informática, se llama algoritmo a una secuencia de instrucciones u operaciones específicas que permiten controlar determinados procesos. Se trata de conjuntos finitos y ordenados de pasos, que nos conducen a resolver un problema o tomar una decisión.

Por ejemplo, una acción simple y cotidiana como encender la luz de la habitación puede describirse como un conjunto ordenado de pasos, como son:

- 1. ¿La luz está apagada?  
NO: FIN  
SÍ: ve al paso 2

- 2. Presiona el interruptor y vuelve al paso 1.

Un algoritmo sirve para tomar una decisión de manera controlada o para resolver paso a paso un problema. Con ese sentido se utilizan los algoritmos en la matemática y la lógica: muchos de los procedimientos tradicionales de cálculo consisten en aplicar un algoritmo.

En las ciencias de la computación, no obstante, los algoritmos constituyen el esqueleto de los procesos que luego se codificarán y programarán para que sean realizados por el computador. Por esa razón un mismo algoritmo puede ser traducido a distintos lenguajes de programación, dado que se trata de un conjunto de instrucciones de tipo lógico, previas a la programación propiamente dicha.

De hecho, un programa informático puede considerarse como una serie compleja de algoritmos ordenados y codificados mediante un lenguaje de programación, para su posterior ejecución en un sistema informático.

## Características de los algoritmos

Los algoritmos tienen las siguientes características generales:

- **Son secuenciales.** Operan en secuencia: deben procesarse uno a la vez, comenzando por las primeras instrucciones y avanzando linealmente hacia las últimas.
- **Son precisos y específicos.** Las instrucciones que los componen no pueden ser ambiguas o subjetivas, sino directas, fáciles de seguir y lo menos generales posible.
- **Son ordenados.** Deben leerse en un orden específico para que tengan sentido. Descolocar un algoritmo o un elemento del algoritmo puede invalidar a los demás.
- **Son finitos.** Tienen un inicio y un fin determinados.
- **Son definidos.** Un mismo algoritmo debe dar siempre los mismos resultados si es alimentado por los mismos elementos.

## Partes de un algoritmo

Los algoritmos tienen una estructura fija, que se compone de las siguientes tres partes:

- **Input o entrada.** Contiene las instrucciones iniciales, en las que se ingresan los datos que el algoritmo necesita para operar.
- **Proceso o instrucciones.** Está compuesto por las operaciones lógicas que el algoritmo emprenderá con lo recibido del input.
- **Output o salida.** Son los resultados obtenidos luego del proceso, una vez terminada la ejecución del algoritmo.

## Ejemplos de algoritmos

Los siguientes son ejemplos posibles de algoritmo:

### 1. Algoritmo para elegir unos zapatos de fiesta:

INICIO

1) Buscar la sección de zapatos de caballero en la tienda.

2) Tomar un par de zapatos. ¿Son zapatos de fiesta?

Sí: ir al paso 3 – NO: volver al paso 2

3) ¿Hay de la talla adecuada?

Sí: ir al paso 4 – NO: volver al paso 2

4) ¿El precio es adecuado?

Sí: ir al paso 5 – NO: volver al paso 2

5) Comprar el par de zapatos.

FIN

### 2. Algoritmo para calcular el área de un triángulo rectángulo:

INICIO

- Hallar las medidas de la base (b) y altura (h)
- Multiplicar base por altura ( $b \times h$ )
- Dividir entre 2 el resultado  $(b \times h) / 2$

FIN

## 1.3 Lenguajes de Programación

Un lenguaje de programación es un lenguaje formal (o artificial, es decir, un lenguaje con reglas gramaticales bien definidas) que proporciona a una persona, en este caso el programador, la capacidad y habilidad de escribir (o programar) una serie de instrucciones o secuencias de órdenes en forma de algoritmos con el fin de controlar el comportamiento físico o

lógico de un sistema informático, para que de esa manera se puedan obtener diversas clases de datos o ejecutar determinadas tareas. A todo este conjunto de **órdenes escritas mediante un lenguaje de programación se le denomina** programa informático.

#### Lenguajes de Bajo Nivel:

- Los lenguajes de bajo nivel están más cercanos al lenguaje de máquina y, por lo tanto, son más comprensibles para la computadora. Estos lenguajes proporcionan un mayor control sobre el hardware de la máquina y permiten acceder directamente a sus recursos.
- El lenguaje ensamblador es un ejemplo típico de lenguaje de bajo nivel. En el ensamblador, las instrucciones se representan mediante mnemónicos que representan operaciones específicas del hardware, como movimientos de datos o operaciones aritméticas.
- Los lenguajes de bajo nivel suelen ser específicos de la arquitectura del procesador y no son portables entre diferentes plataformas. Cada tipo de CPU tiene su propio conjunto de instrucciones de bajo nivel.

#### Lenguajes de Alto Nivel:

Los lenguajes de alto nivel están más alejados del lenguaje de máquina y son más comprensibles para los programadores humanos. Estos lenguajes proporcionan abstracciones más poderosas y simplifican la tarea de escribir programas complejos.

Ejemplos de lenguajes de alto nivel incluyen Java, Python, C++, C#, Ruby, entre otros. Estos lenguajes permiten a los programadores expresar ideas y algoritmos de una manera más natural y cercana al lenguaje humano.

Los lenguajes de alto nivel son más portables, lo que significa que el mismo código puede ejecutarse en diferentes plataformas con pocos o ningún cambio, siempre que haya un intérprete o compilador adecuado para cada plataforma.

#### Relación entre Lenguajes de Bajo Nivel y Alto Nivel:

La relación entre los lenguajes de bajo nivel y alto nivel está dada por el proceso de compilación o interpretación. Los lenguajes de alto nivel

deben ser traducidos a lenguaje de máquina para que la computadora pueda ejecutarlos.

Cuando un programa escrito en un lenguaje de alto nivel se compila, se traduce a lenguaje de máquina o código intermedio (bytecode) específico de la plataforma. Esto permite que la computadora entienda y ejecute las instrucciones.

La traducción del código de alto nivel a bajo nivel está a cargo del compilador o intérprete del lenguaje. El compilador traduce el código completo antes de la ejecución, mientras que el intérprete realiza la traducción línea por línea durante la ejecución.

En resumen, los lenguajes de bajo nivel se encuentran más cerca del lenguaje de máquina y son más comprensibles para la computadora, mientras que los lenguajes de alto nivel son más cercanos al lenguaje humano y permiten una programación más expresiva y portabilidad entre diferentes plataformas. El proceso de compilación o interpretación es lo que permite que los programas escritos en lenguajes de alto nivel sean ejecutados por la computadora.





## **1.4 Diagramas de Flujo**

El diagrama de flujo o flujograma o diagrama de actividades es la representación gráfica de un algoritmo o proceso. Se utiliza en disciplinas como programación, economía, procesos industriales y psicología cognitiva.

En Lenguaje Unificado de Modelado (UML), es un diagrama de actividades que representa los flujos de trabajo paso a paso.

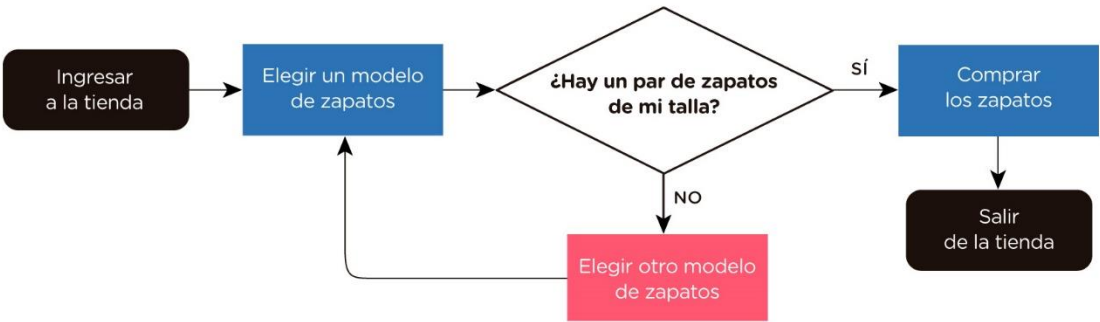
Estos diagramas utilizan símbolos con significados definidos que representan los pasos del algoritmo, y representan el flujo de ejecución mediante flechas que conectan los puntos de inicio y de fin del proceso.

# Diagrama de flujo y sus principales símbolos

SÍMBOLO	NOMBRE	FUNCIÓN
	Inicio / final	Representa el inicio y el final de un proceso
	Línea de flujo	Indica el orden de la ejecución de las operaciones
	Proceso	Representa cualquier tipo de operación
	Decisión	Permite analizar una situación según si su respuesta es verdadera o falsa, sí o no
	Entrada / salida	Representa la lectura de datos en la entrada y la impresión

## Diagrama de flujo

Ejemplo de diagrama de flujo de un proceso de compra



## 1.5 Paradigmas de programación y ejemplos:

Un paradigma de programación consiste en un método para llevar a cabo cálculos y la forma en la que deben estructurarse y organizarse las tareas que debe realizar un programa.

Un paradigma de programación está delimitado en el tiempo en cuanto a aceptación y uso, porque nuevos paradigmas aportan nuevas o mejores soluciones que lo sustituyen parcial o totalmente.

1. **Programación Secuencial:** Es el paradigma más básico y lineal. Se basa en la ejecución secuencial de instrucciones, donde cada instrucción se ejecuta en orden, una después de la otra. Este enfoque es directo y fácil de entender, pero puede volverse complejo rápidamente a medida que crece el tamaño del programa.
2. **Programación Estructurada:** Surgió como una evolución de la programación secuencial en la década de 1960. Este paradigma introduce estructuras de control, como bucles y condicionales, para mejorar la organización del código. Con la programación estructurada, los programas pueden ser más legibles y mantenibles.
3. **Programación Orientada a Objetos (POO):** La POO fue desarrollada en la década de 1960 y se popularizó en la década de 1980. Este paradigma se basa en la idea de modelar problemas del mundo real como objetos con atributos y métodos. La POO promueve la reutilización de código, la modularidad y una visión más clara del diseño del programa.
4. **Programación Funcional:** Aunque los fundamentos de la programación funcional se remontan a la década de 1930 con el cálculo lambda, se popularizó en la década de 1950. La programación funcional se basa en el uso de funciones como unidades fundamentales para resolver problemas. Este paradigma evita el uso de variables mutables y se enfoca en la inmutabilidad y en la ausencia de efectos secundarios.
5. **Programación Lógica:** Desarrollada en la década de 1970, la programación lógica se basa en la lógica matemática y en el uso de reglas para establecer relaciones y llegar a conclusiones. Los programas lógicos están formados por hechos y reglas lógicas, y el proceso de inferencia es utilizado para obtener resultados.
6. **Programación Concurrente:** La programación concurrente trata de la ejecución simultánea de varias tareas o procesos. Surgió en la década de 1960 para aprovechar el poder de las computadoras con múltiples núcleos o

procesadores. La programación concurrente se enfoca en la gestión de hilos o procesos para lograr la concurrencia y coordinación entre ellos.

7. **Programación Paralela:** La programación paralela es un paradigma que busca dividir una tarea en sub-tareas que se ejecutan simultáneamente en múltiples procesadores o núcleos. Surgió como resultado del aumento en la potencia de procesamiento y permite mejorar el rendimiento y la velocidad de ejecución de ciertos algoritmos y programas.
8. **Programación basada en Reglas (Rule-based programming):** Este paradigma se centra en definir reglas lógicas y patrones que guían el comportamiento del programa. Las reglas son evaluadas para tomar decisiones y resolver problemas.
9. **Programación basada en Aspectos (Aspect-Oriented Programming, AOP):** La AOP es un paradigma que permite separar las preocupaciones transversales (aspectos) del código principal. Los aspectos encapsulan funcionalidades que se aplican a diferentes partes del programa, mejorando la modularidad y la reutilización.
10. **Programación Orientada a Eventos (Event-Driven Programming):** En este paradigma, el flujo de ejecución del programa está determinado por eventos que ocurren en el sistema (como clics de botones o señales de red). El programa responde a estos eventos ejecutando las acciones asociadas a ellos.

Estos son solo algunos de los paradigmas de programación que existen. Cada uno de ellos tiene sus ventajas y desventajas, y la elección del paradigma adecuado dependerá del tipo de problema a resolver, las características del proyecto y las preferencias del programador. A lo largo de la historia de la computación, han surgido y evolucionado diversos paradigmas para abordar diferentes tipos de problemas y necesidades de desarrollo.

ejemplos de lenguajes de programación asociados con cada paradigma y algunos lenguajes que son multiparadigmas:

1. <b>Programación Secuencial:</b>
------------------------------------

- |   |
|---|
| <ul style="list-style-type: none"><li>• Lenguaje ensamblador (Assembly)</li></ul> |
|---|

2. <b>Programación Estructurada:</b>
--------------------------------------

- |  |
|--|
| <ul style="list-style-type: none"><li>• C</li><li>• Pascal</li></ul> |
|--|

3. <b>Programación Orientada a Objetos (POO):</b>
---



- Java
- Python
- C++
- C#
- Ruby

#### 4. **Programación Funcional:**

- Haskell
- Lisp
- Scheme
- F#
- Erlang

#### 5. **Programación Lógica:**

- Prolog
- Datalog

#### 6. **Programación Concurrente:**

- Go (Golang)
- Erlang
- Java (con el uso de hilos)

#### 7. **Programación Paralela:**

- OpenMP (no es un lenguaje, sino una API para paralelismo)
- MPI (Message Passing Interface) (también una API para paralelismo)

#### 8. **Programación basada en Reglas (Rule-based programming):**

- CLIPS
- Jess

#### 9. **Programación basada en Aspectos (Aspect-Oriented Programming, AOP):**

- AspectJ
- Spring AOP (Java con soporte para AOP)

#### 10. **Programación Orientada a Eventos (Event-Driven Programming):**

- JavaScript (en el contexto de navegadores web)
- C# (con aplicaciones de interfaz gráfica usando Windows Forms o WPF)

**Lenguajes Multiparadigmas:** Algunos lenguajes de programación son multiparadigmas, lo que significa que admiten múltiples enfoques de programación. Pueden combinar características de varios paradigmas, permitiendo a los programadores utilizar el enfoque más adecuado para cada parte del programa. Ejemplos de lenguajes multiparadigmas incluyen:

1. **Python:** Admite programación imperativa, orientada a objetos y funcional.
2. **JavaScript:** Es principalmente un lenguaje orientado a objetos y funcional, pero también admite programación imperativa.
3. **C++:** Es principalmente un lenguaje orientado a objetos, pero también admite programación imperativa y funcional.
4. **C#:** Es principalmente un lenguaje orientado a objetos, pero también admite programación funcional.
5. **Scala:** Es un lenguaje multiparadigma que combina programación orientada a objetos y funcional.

Estos son solo algunos ejemplos, y hay muchos otros lenguajes de programación con diferentes capacidades y características en cuanto a paradigmas. La elección del lenguaje de programación dependerá de los requisitos del proyecto y de las preferencias del programador. Es importante destacar que un lenguaje multiparadigma permite a los desarrolladores aprovechar lo mejor de diferentes enfoques de programación para abordar problemas de manera más eficiente y elegante.

Hay varios paradigmas de programación, cada uno con su enfoque y estilo de desarrollo. Los principales paradigmas de programación son:

1. **\*\*Programación Imperativa\*\*:** Este paradigma se enfoca en describir cómo lograr un resultado a través de una secuencia de instrucciones. Los programas imperativos modifican el estado del programa a medida que se ejecutan las instrucciones.

Ejemplo en Python (imperativo):

# Cálculo del factorial de un número usando un bucle while

```
def factorial(n):
```

```
    result = 1
```

```
    while n > 0:
```

```
        result *= n
```

```
        n -= 1
```

```
    return result
```

2. **\*\*Programación Orientada a Objetos (POO)\*\***: Como se describió anteriormente, en la POO se modelan conceptos del mundo real como objetos con atributos y métodos. Los objetos interactúan entre sí para resolver problemas.

Ejemplo en Java (POO):

// Definición de una clase "Persona" con atributos y métodos

```
class Persona {
```

```
    String nombre;
```

```
    int edad;
```

```
    void saludar() {
```

```
        System.out.println("Hola, soy " + nombre + " y tengo " + edad + " años.");
```

```
    }
```

```
}
```

// Creación de objetos y llamada a métodos

```
Persona persona1 = new Persona();
```

```
persona1.nombre = "Juan";
```

```
persona1.edad = 25;
```

```
persona1.saludar(); // Salida: "Hola, soy Juan y tengo 25 años."
```

3. **\*\*Programación Funcional\*\***: Se centra en el uso de funciones como unidades fundamentales para resolver problemas. Las funciones son tratadas como ciudadanos de primera clase, lo que permite pasar funciones como argumentos y devolver funciones como resultados.

Ejemplo en JavaScript (funcional):

// Función que calcula el doble de un número usando funciones de flecha (arrow functions)

```
const doble = (num) => num * 2;
```

// Uso de la función

```
const resultado = doble(5); // resultado es 10
```

4. **\*\*Programación Lógica\*\***: Se basa en reglas lógicas y en la inferencia para llegar a una solución. Los programas lógicos establecen relaciones entre hechos y reglas y buscan respuestas a través de la unificación y la resolución.

Ejemplo en Prolog (lógico):

% Definición de hechos y reglas

hombre(juan).

hombre(pedro).

padre(pedro, juan).

padre(pedro, ana).

% Consulta que busca quiénes son los hijos de pedro

?- padre(pedro, X).

% Respuesta: X = juan ; X = ana

Estos son solo algunos ejemplos de los paradigmas de programación que existen. Cada paradigma tiene sus ventajas y desventajas, y es importante elegir el adecuado según los requerimientos y características del proyecto que se está desarrollando.

Es importante tener en cuenta que la elección del paradigma de programación dependerá del problema a resolver, las necesidades del proyecto y las preferencias del programador. Cada paradigma tiene sus ventajas y desventajas, y es fundamental comprender sus características para utilizar el enfoque más adecuado en cada situación.