Introducción a la programación

Un **ordenador** es un sistema ciertamente complejo que cuenta con unos elementos hardware ciertamente sofisticados y potentes, pero por muy complejo que sea no es capaz de realizar ninguna tarea por sí mismo.

Para que pueda llevar a cabo las numerosas funciones que estamos acostumbrados a ver habitualmente en los ordenadores modernos, como realizar complicados cálculos, procesar imágenes, datos y textos, comunicarse con otros equipos, etc., es necesario la existencia de un programa que transmita una serie de órdenes a bajo nivel al ordenador de modo que, al ser ejecutadas de manera conjunta y ordenada, consigan completar una determinada tarea útil para el usuario.

Un ordenador, es capaz de realizar únicamente tres **tipos de operaciones**:

- Operaciones **aritméticas** básicas.
- Operaciones de tipo **lógico** (comparar dos valores.)
- **Almacenamiento** y **recuperación** información.

Estas tres operaciones convenientemente ligadas entre sí forman lo que llamamos un programa.

1.1. Programas y algoritmos

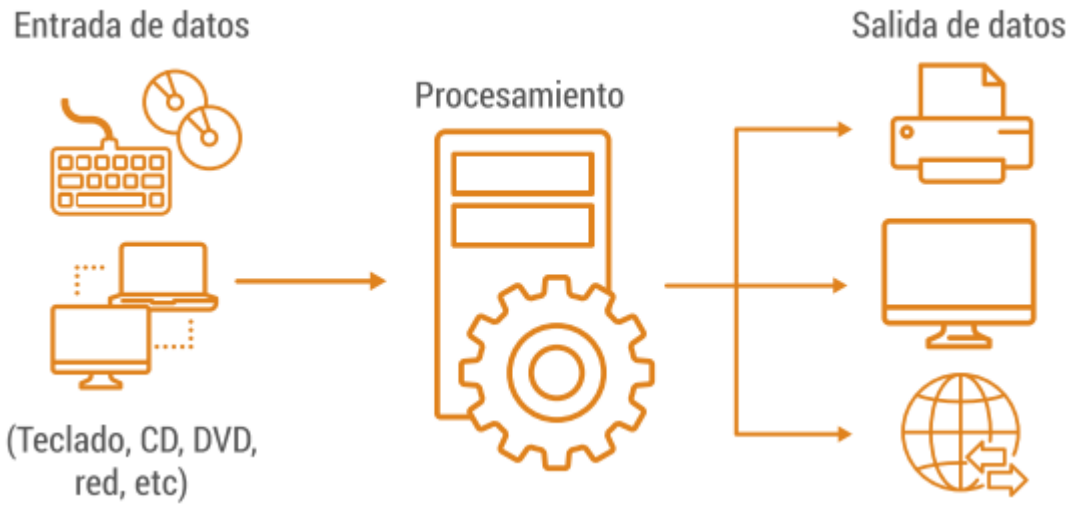
Un **programa** en definitiva es un conjunto de órdenes que ejecuta el ordenador para conseguir un objetivo. Las órdenes se proporcionan a través de un lenguaje de programación (códigos). A estas órdenes escritas en un determinado lenguaje de programación se les llama también instrucciones. De forma general este conjunto de instrucciones toma unos datos de entrada y devuelve unos datos de salida, o resultados.

El ordenador siempre funciona bajo control de un programa, incluso las operaciones más básicas que hace el ordenador, como comunicarse con los dispositivos de entrada/salida, interaccionar con el usuario, gestionar los propios recursos del ordenador, etc., son realizados por un programa llamado sistema operativo, que es el programa más importante que ejecuta un ordenador.

Los **programas** que se ejecutan en un ordenador se encuentran en unidades de almacenamiento permanente, como el disco duro, un disco óptico, o un pendrive,

incluso la memoria ROM, donde se almacena para del núcleo del sistema operativo.

Cuando el ordenador recibe la orden de ejecutar un programa, éste, o parte de él, es cargado en la memoria RAM del ordenador para su ejecución.



Para realizar un programa, los programadores definen un algoritmo. Un algoritmo es la descripción exacta y sin ambigüedades de la secuencia de pasos elementales a aplicar a un proceso para, que a partir de unos datos iniciales, se obtenga la solución buscada a un problema determinado. Un programa es la expresión de un algoritmo en un lenguaje de programación entendible por el ordenador.

1.2. Lenguajes de programación

Los l**enguajes de programación** proporcionan la notación utilizada para la escritura de los programas. Para la escritura de los programas o aplicaciones informáticas actuales, el programador utiliza un lenguaje de programación denominado “de alto nivel”, que le permite escribir las instrucciones siguiendo una notación “entendible” para el programador, no así para el ordenador.

Para que el ordenador pueda entender las órdenes contenidas en un programa cualquiera escrito por el programador en lenguaje de “alto nivel”, es necesario traducir estas instrucciones a otras “de bajo nivel” que puedan ser entendidas por el ordenador. Este código de bajo nivel, conocido como “código máquina”, está compuesto solamente de unos y ceros, es el único que entiende el ordenador y es el que le permite interpretar las órdenes contenidas de los programas para que las pueda ejecutar

Tipos de lenguajes de programación

Según el nivel, es decir, la cercanía de las instrucciones del lenguaje de programación con el lenguaje humano, estos se pueden agrupar en tres tipos:

- Lenguajes de bajo nivel.
- Lenguajes de nivel intermedio.
- Lenguajes de alto nivel.

Lenguajes de bajo nivel

Se trata de lenguajes cuyo juego de instrucciones son entendibles directamente por el hardware. El lenguaje de bajo nivel que utilizan los ordenadores es el conocido como “código máquina”, y está formado por unos y ceros, es decir, código binario, lo que entiende directamente el microprocesador.

Además de la complejidad que supone escribir programas de esta manera, cada tipo de microprocesador dispone de su propio juego de instrucciones o combinaciones de ceros y unos con las que se puede indicar a este las tareas a realizar, por lo que un programa en código máquina solo puede utilizarse en la máquina para la que se programó.

Lenguajes de nivel intermedio

Se les conoce con ese nombre porque están a medio camino entre el código máquina y los lenguajes de alto nivel. El lenguaje de ensamblador fue el primer lenguaje de nivel intermedio en desarrollarse, con el objetivo de sustituir el lenguaje máquina por otro más similar a los utilizados por las personas.

Cada instrucción en ensamblador equivale a una instrucción en lenguaje máquina, utilizando para su escritura palabras nemotécnicas en lugar de cadenas de bits.
El juego de instrucciones del lenguaje ensamblador está formado por palabras abreviadas procedentes del inglés (Ejemplo: MOV A, B).

La programación en lenguaje ensamblador precisa de un amplio conocimiento sobre la constitución, estructura y funcionamiento interno de un ordenador, ya que maneja directamente las posiciones de memoria, registros del procesador y demás elementos físicos.

```
INICIO: ADD B, 1
                MOV A,
                E CMP A,
                B JE FIN
```

```
                    JMP INICIO
FIN : END
```

Al igual que en el caso del código máquina, los programas escritos en ensamblador son dependientes del procesador para el que se han creado.

Aunque fue el primer lenguaje de programación que se empezó a utilizar para ordenadores, actualmente no se utiliza en la creación de programas para usuarios de ordenador, su uso está limitado a la programación de microcontroladores y dispositivos electrónicos.

Lenguajes de alto nivel

Se les llama lenguajes de alto nivel porque el conjunto de órdenes que utilizan son fáciles de entender y aprender. Además no hay incompatibilidades entre un microprocesador y otro, por lo que un programa escrito para un ordenador puede ser utilizado en otro.

Como inconveniente destacable, está la necesidad de traducir los programas escritos en un lenguaje de alto nivel a un lenguaje máquina o ensamblador para que pueda ser ejecutado por la unidad central de proceso, lo que significa disponer necesariamente de un software traductor (ensamblador, compilador o intérprete) para cada tipo de ordenador utilizado. Más adelante, hablaremos de estos paquetes software.

El siguiente listado corresponde a un programa escrito en un lenguaje de programación de alto nivel:

```
int c = 20;
int sum = 0;

for(int i=1; i<=c; i++){
        sum += i;
}

System.out.println(sum);
```

El abanico de lenguajes de programación de alto nivel existentes hoy en día es enorme y no para de crecer. Entre más utilizados en la actualidad tenemos:

- **Java**. Se trata de uno de los lenguajes de programación más utilizados actualmente, apareció a principios de los años 90 y desde entonces su uso no ha hecho más que extenderse. Puede ser utilizado para crear programas para muy diferentes fines, como aplicaciones de escritorio, aplicaciones para Web, incluso programas para dispositivos electrónicos como tabletas o smartphones. Una de las características más interesantes de este lenguaje es que es multiplataforma, lo que significa que un programa compilado en Java puede ser ejecutado en diferentes sistemas operativos.
- **JavaScript**. Aunque de nombre similar a Java, solo se parece a éste en los fundamentos sintácticos. JavaScript es un lenguaje interpretado, utilizado en la creación de scripts en páginas Web, es decir, bloques de código integrados dentro de una página y que son interpretados y ejecutados por el navegador Web al procesar dicha página.
- **Python**: fue creado en 1989 por Guido van Rossum. Guido empezó a implementar este lenguaje como pasatiempo con el objetivo de que fuera fácil de usar y aprender, pero, a la vez, que fuese un lenguaje potente. Es un lenguaje de propósito general que en los últimos años se ha ido haciendo cada vez más popular en áreas como la ciencia de datos o la inteligencia artificial (IA).
- **Ruby**. Es un lenguaje de programación cuya aparición se remonta a mediados de los 90, pero que es últimamente cuando está adquiriendo mayor popularidad. Se trata de un lenguaje dinámico y de código abierto y está enfocado a la productividad. Se aplica especialmente sobre la plataforma on-rails para el desarrollo de aplicaciones para Web.
- **C#**. Se trata de una versión actualizada del lenguaje C que Microsoft creo a finales de los 90 para incorporarlo a la plataforma .NET. Con el se pueden desarrollar aplicaciones Windows, Web y móviles de forma sencilla, al estilo de Visual Basic, pero con la elegancia y precisión de C.

3. Paradigmas de programación

La **complejidad** es el principal factor por el cual el código de los programas se vuelve complejo y difícil de mantener en el tiempo. Cuanta menos complejidad tenga un programa, más fácil será de leer y también de mantener y evolucionar, por ende, menos costes implicará su desarrollo.

Uno de los principales enfoques para lidiar con la complejidad son los **paradigmas de la programación**.

Un paradigma de programación hace referencia a un estilo de programación. No se trata de un lenguaje específico, si no de la forma en la que se programa y estructura el código.

Existen dos principales familias de **paradigmas de programación**:

· Programación **imperativa**

· Programación **declarativa**

3.1. Programación imperativa

La **programación imperativa** es un paradigma a través del cual se especifican en el código todos los pasos que debe realizar el programa para lograr un objetivo. Para ello se utilizan variables, estructuras de control condicional y repetitivo.

La **programación imperativa** se encarga de describir el **cómo** se debe llevar a cabo un proceso paso por paso, definiendo el flujo de control como sentencias que cambian el estado del programa.

Dentro de la programación imperativa existen múltiples enfoques:

· Programación **estructurada**

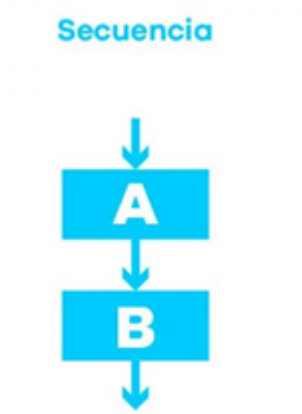
· Programación **orientada a objetos**

3.1.1. Programación estructurada

La **programación estructurada** es un paradigma que busca mejorar la legibilidad y reducir la complejidad de un código por medio del uso de **sentencias de control**.

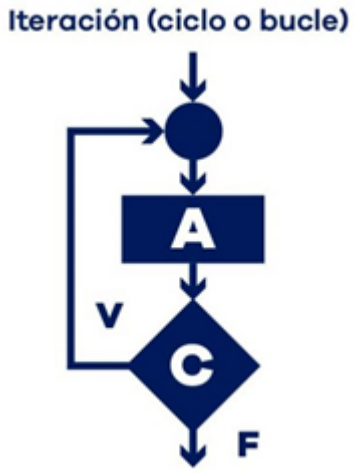
La programación estructurada emplea 3 estructuras para la creación de programas:

· **Secuencia**. La estructura secuencial es la que se da de forma natural en el lenguaje, porque las sentencias se ejecutan en el orden en el que aparecen en el programa, es decir, una detrás de la otra.



· **Selección o condicional**. La estructura condicional se basa en que una sentencia se ejecuta según el valor que se le atribuye a una variable booleana. Una variable booleana es aquella que tiene dos valores posibles. Por tanto, esta estructura se puede ejecutar de dos formas distintas, dependiendo del valor que tenga su variable.

· **Iteración (ciclo o bucle**). La estructura de repetición ejecuta una o un conjunto de sentencias siempre que una variable booleana sea verdadera. Para los bucles o iteraciones, los lenguajes de programación usan las estructuras while y for.



La **programación procedimental** es un tipo de programación estructurada y se basa en el uso de procedimientos o funciones que modifican el estado del programa.

3.1.2 Programación orientada a objetos

La **Programación Orientada a Objetos (POO)** es un paradigma que organiza los programas como objetos, elementos compuestos por variables (estado) y métodos (comportamiento).

Los elementos básicos de la programación orientada a objetos son las **clases** y los **objetos**. Las clases actúan como plantillas que especifican los atributos y métodos que tendrán los objetos.

Los **objetos son** las **instancias** particulares que se crean a partir de una clase. Ejemplo: a partir de la clase vehículo se pueden crear muchos objetos vehículo, uno para representar un Toyota, otro para representar un Ford, uno puede estar en marcha, otro estacionado, etc.

Otros **mecanismos** de la programación orientada a objetos son:

· Encapsulación

· Herencia

· Abstracción

· Polimorfismo

· Serialización

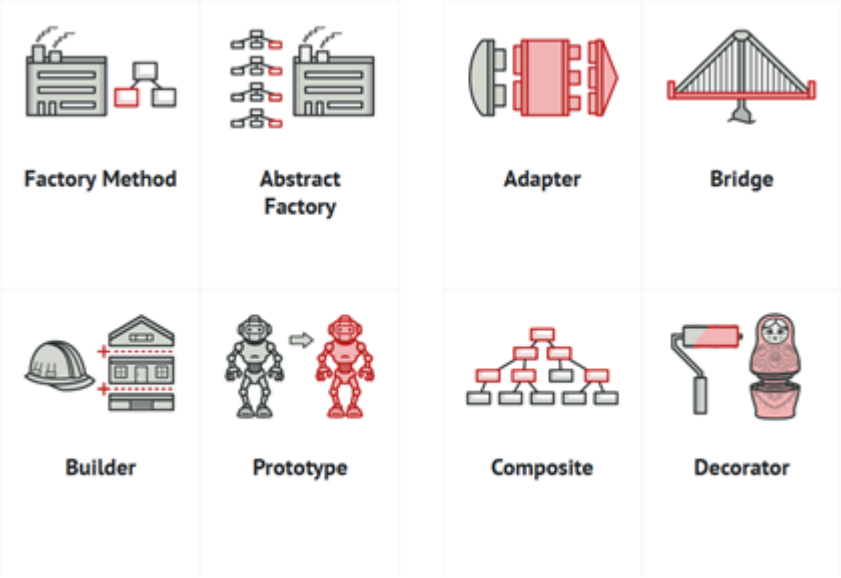
· Asociación

· Composición

La programación orientada a objetos se utiliza especialmente en el desarrollo de programas complejos y grandes **sistemas de software** de forma **escalable**.

Dentro de la programación orientada a objetos se emplea el uso de patrones de diseño con el fin de promover el uso de **buenas prácticas** y la creación de código de calidad.

Los **patrones de diseño** son soluciones a problemas comunes en software, no son código en sí, son formas de estructurar los elementos de la programación orientada a objetos (clases, objetos, abstracción).



Fuente: https://refactoring.guru/design-patterns/catalog

3.2. Programación declarativa

De forma paralela a la programación imperativa se desarrolló el paradigma declarativo. La **programación declarativa** define la lógica del programa, pero no detalla el flujo de control.

En la **programación declarativa** el programador define qué es lo que espera del programa sin definir cómo debe ser implementado, es decir, sin programar los pasos necesarios para obtenerlo. Este paradigma se enfoca en qué es necesario obtener en lugar de especificar el cómo obtenerlo.

Dentro del paradigma de la programación declarativa destacan algunos subparadigmas como:

· Programación funcional

· Programación lógica

3.2.1 Programación funcional

Entre los elementos sintácticos de los lenguajes de programación se encuentran las funciones. Una función es un bloque de sentencias para ejecutar una determinada tarea de forma reutilizable a lo largo de un código.

La **programación funcional** es un paradigma que consta de llamadas a funciones concatenadas en las que cada parte del programa se interpreta como una función.

De esta forma, dentro de la programación funcional las funciones pueden adoptar distintas estructuras.

Un ejemplo de uso de la programación funcional es la parametrización del comportamiento, la cual consiste en poder utilizar como parámetro de una función otra función. Siendo posible también utilizar una función como resultado o retorno de otra función.

La **programación funcional** se emplea a menudo en conjunto a la programación orientada a objetos, permitiendo la creación de aplicaciones flexibles y escalables.

Características de la programación funcional:

1. Inmutabilidad de los datos
2. Transparencia referencial
3. Modularidad
4. Mantenibilidad
5. Funciones puras

1 - Inmutabilidad de los datos

Se debe poder crear nuevas variables y estructuras de datos **sin tener que modificar** las ya existentes.

Al evitar los datos existentes y crear copias se evitan posibles **efectos secundarios**.

Favorece el uso del **paralelismo**, evitando que una ejecución altere el estado de otra ejecución que tiene lugar simultáneamente.

2 - Transparencia referencial

Una función es **referencialmente transparente** si siempre devuelve el mismo resultado cuando es invocada con los mismos valores de entrada en sus argumentos.

Siempre devuelve el mismo resultado para la misma entrada, independientemente de que se ejecute múltiples veces y en diferentes lugares de un programa.

3 - Modularidad

La **modularidad** es una técnica para diseñar software en pequeños módulos o fragmentos independientes, cada uno con todo lo necesario para ser ejecutado y proporcionar una **funcionalidad**.

La modularidad incrementa la **productividad** debido a que los módulos pueden ser programados más rápido y con capacidad para ser reutilizados en diferentes programas.

Alternativamente, los programas modulares pueden ser **probados** y **depurados** de manera más sencilla y rápida.

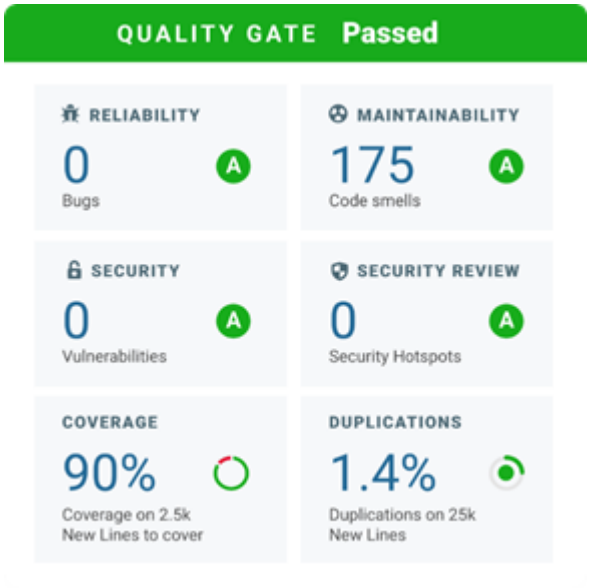
4 - Mantenibilidad

La **mantenibilidad** de un software hace referencia a la facilidad con la que se puede evolucionar y corregir fallos en el mismo.

Cuando un software es muy complejo, introducir nuevos cambios y mejorar las funcionalidades existentes se vuelve cada vez más difícil y costoso.

La **deuda técnica** es una técnica que mide la entropía del software, es decir, el coste acumulado por utilizar código que no cumple con los estándares de calidad y que no mantiene una coherencia con otras partes del programa.

En el largo plazo, una deuda técnica alta puede hacer que los costes de mantenimiento de un software se disparen.



Fuente: https://sonarcloud.io/

5 - Funciones puras

Las **funciones puras** son también conocidas como funciones libres de efectos secundarios o funciones sin estado.

Estas funciones cumplen con las siguientes condiciones:

· Retorna el mismo resultado para los mismos argumentos de entrada siempre.

· No tiene efectos secundarios: no modifica datos en base de datos, no altera el estado de variables externas, etc.

Un ejemplo de función pura puede ser una función suma, que recibe dos parámetros y devuelve la suma de ambos sin modificar ninguno de ellos en el proceso.

Por el contrario, una función impura es aquella que tiene efectos secundarios, por ejemplo, modificando los argumentos que recibe en la entrada. Las funciones utilizadas en el paradigma imperativo son generalmente impuras ya que modifican el estado del programa o toman valores de consola diferentes en cada lectura.

6 - Funciones de orden superior

Las **funciones de orden superior** toman una o más funciones como argumentos en la entrada e incluso pueden retornar una función.

Las funciones de orden superior dan lugar a la **parametrización del comportamiento**, ya que los funciones representan un comportamiento y puede ser

pasado como parámetro a su vez a otras funciones.

Ejemplo de programación funcional en Java:

```
Optional<Car> carOpt = this.carService.findById(id);
return carOpt.map(
                car -> ResponseEntity.ok(car))
        .orElseGet(
                () -> ResponseEntity.notFound().build()
        );
```

Ejemplo de controlador REST que devuelve un objeto Coche o una respuesta HTTP *Not Found* en función de si existe en la base de datos o no.

3.2.2 Programación lógica

El paradigma de la **programación lógica** utiliza un enfoque declarativo basado en la lógica formal.

Se compone de **hechos** y **reglas** en un determinado ámbito de dominio o problema en lugar de utilizar instrucciones. Por ejemplo:

> A es verdadero si B, C y D son verdaderos.

Dentro de los lenguajes del paradigma de la programación lógica destaca **Prolog**, abreviatura de *programming in logic*.