CIPFP Mislata

Centre Integrat Públic
Formació Professional Superior

Lenguajes de programación

Autor: Joan Puigcerver Ibáñez

Correo electrónico: j.puigcerveribanez@edu.gva.es

Curso: 2022/2023

1.Conceptos básicos

1.1. Informática, binario, digital y software

La palabra **informática** viene de "información" y "automática". La información es un conjunto de datos, y automático es que funciona por sí mismo. De esta forma, la informática es la ciencia que estudia el tratamiento de la información de una mediante un sistema automático, como puede ser un ordenador.

La informática permite automatizar el procesamiento de la información. Toda la información que utiliza un "ordenador" debe estar representada en una secuencia de ceros y unos (dígitos) llamada **código binario**. Esto significa que cada unidad básica de información puede tener dos estados (0, 1). A esta unidad se le conoce como **bit** o *binary digit* (b) y al conjunto de ocho bits se llama **byte** (B). Un byte puede representar 2^8 caracteres, es decir, 256 posibilidades diferentes (Tabla ASCII).

Cuando un dispositivo cualquiera utiliza o almacena información en forma de bits decimos que este dispositivo es digital. Cuando el mecanismo que utiliza no es mediante este código, como un reloj clásico o un termómetro de mercurio, decimos que es analógico.

El **software (software)** es la parte intangible de un sistema informático, equivalente al equipamiento lógico. Todo software está diseñado para realizar una tarea determinada en nuestro sistema. Dentro de este software podemos distinguir dos tipos distintos.

Por un lado, tenemos las **aplicaciones**, software instalado sobre el sistema operativo con la función de realizar una tarea en concreto (editar documento, reproducir música, ...)

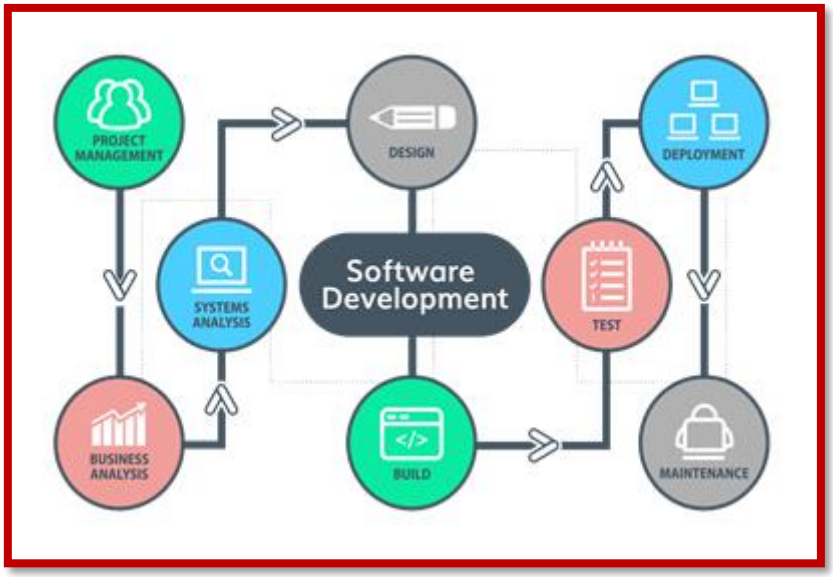
Por un lado, tenemos el **sistema operativo**, un software cuya función es proporcionar una plataforma única para el uso de la máquina, independientemente del **hardware (hardware)** que está presente. El sistema operativo es el encargado de comunicarse con el hardware.

1.2. Programar vs Desarrollar software

Se puede definir **programar** cómo especificar los comandos (instrucciones) y datos en un ordenador (dispositivo) que éste ejecutará para recibir una serie de resultados o probar un cierto comportamiento.

Es importante utilizar la palabra "dispositivo", ya que hoy en día podemos programar desde un ordenador hasta una nevera, pasando por unas zapatillas deportivas o, por qué no, incluso una alfombra si éstos utilizan información digital. Cada vez más elementos de nuestra vida cotidiana aceptan órdenes en forma de pedidos, líneas de código o programas, dentro de lo que se conoce como Internet **de las Cosas** (Internet of Things o **IoT**).

Sin embargo, **desarrollar software** va más allá que programar. Es todo el proceso de analizar, diseñar, probar, documentar y mantener el software, que es mucho mayor al tiempo que se dedica a exclusivamente a *escribir líneas de código*.



2.Lenguajes de programación

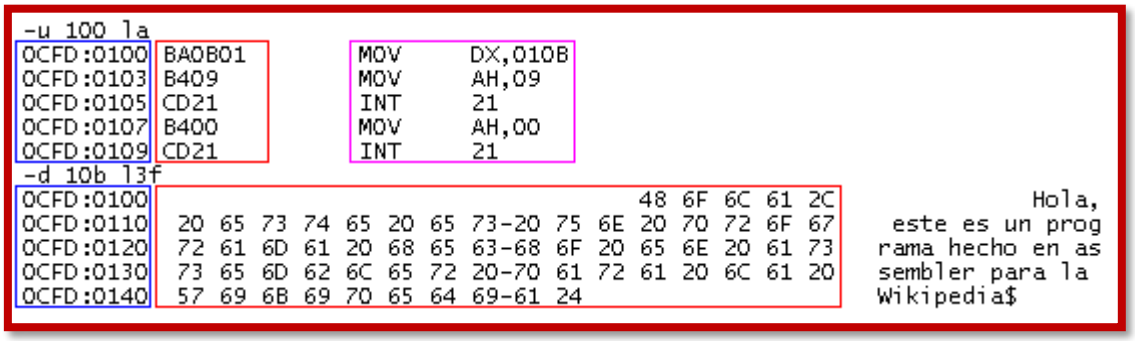
2.1. Según el nivel de abstracción

Los seres humanos nos comunicamos mediante **lenguajes naturales** que se han ido originando y evolucionando mediante siglos. Estos lenguajes son muy complejos, con muchos matices y ambigüedades. Incluso cuando nosotros nos comunicamos, podemos llegar a malentendidos o confundirnos por culpa del lenguaje.

Por el contrario, si queremos dar instrucciones a una máquina, necesitamos hacerlo de una forma que sea clara, concisa y que sólo tenga una única interpretación. A raíz de esta necesidad, se han creado los **lenguajes de programación**.

El lenguaje de programación más básico, pero a la vez más difícil de utilizar, y de entender es el **lenguaje máquina**. Este lenguaje es un sistema de códigos que son directamente interpretables por un procesador concreto que realizan diferentes operaciones (suma, resta, mover datos de un registro a otro...).

Estos códigos que el procesador entiende están en binario y, por tanto, son difíciles de utilizar para los seres humanos. Para facilitar esta tarea, existe el **lenguaje de bajo nivel o ensamblador**, que es una relación entre los diferentes códigos del lenguaje máquina y palabras o valores fácilmente recordables (ADD en lugar de 0110). Sin embargo, son igualmente difíciles de utilizar.



Para que la tarea de desarrollar código sea más amigable, fácil y flexible, se han ido creando distintos **lenguajes de programación de alto nivel**, cada uno con diferentes características, funcionalidades y propósitos. Estos lenguajes se asemejan más al lenguaje natural, pero son claros y no permiten ambigüedades. Además, también tienen una alta abstracción respecto al hardware de la plataforma donde se desea ejecutar.

```
public void processData()
{
  do
  {
    int data = getData();
    if(data < 0)
      performOperation1(data);
    else
      performOperation2(data);
  }
  while(hasMoreData());
}
```

2.2. Según el tipo de ejecución

Hemos comentado que los desarrolladores prefieren utilizar lenguajes de alto nivel para desarrollar software, pero, en cambio, los dispositivos sólo entienden el lenguaje máquina diseñado para éste. Para poder ejecutar el código desarrollado mediante lenguajes de alto nivel, es necesario traducirlo antes a lenguaje que el dispositivo sea capaz de entender.

Según cómo se haga esta traducción, podemos clasificar los lenguajes de programación en dos tipos:

- **Lenguajes interpretados**: El código fuente va que se traduce instrucción por instrucción a medida que se ejecuta el código. El **intérprete** traduce cada instrucción a lenguaje máquina y éste se ejecuta.

Algunos ejemplos de lenguajes interpretados son: Python, Ruby, PHP (lenguajes de programación de scripting y servidores web), JavaScript (cliente web), Bash, PowerShell (scripting) u otros como Perl, MATLAB o Mathematica.

- **Lenguajes compilados**: es aquel cuyo código fuente, escrito en un lenguaje de alto nivel, es traducido por un **compilador** a un archivo **ejecutable** comprensible para la máquina en determinada plataforma. Este ejecutable puede ejecutar el programa cuantas veces sea necesario sin tener que repetir el proceso.

Algunos ejemplos de lenguajes compilados son: C, C++, Objective C, Rust, Pascal, Haskell o VisualBasic.

Interpretados vs Compilados

	Interpretados	**Compilados**
Tiempo ejecución	Son más lentos porque deben traducir mientras ejecutan el código	Al ser traducidos de antemano, son más rápidos al ejecutarse
Detección errores de sintaxis	Sólo pueden encontrar errores en el código que se ejecuta	Pueden encontrar errores de sintaxis en todo el código
Optimización	No pueden aplicar optimizaciones	Pueden aplicar optimizaciones en el código
Portabilidad	Se puede ejecutar siempre que exista un intérprete	El ejecutable sólo puede ser ejecutado en la arquitectura compilada
Tiempo desarrollo	Es más rápido que probar, no hace falta compilarlo entero	Es más lento de probar, ya que es necesario compilarlo entero para poder ejecutarlo (*)

(*): Muchos lenguajes de programación compilados disponen de intérpretes para agilizar el proceso de desarrollo y prueba.

Lenguajes intermedios o bytecode

Algunos lenguajes de programación más modernos han combinado los dos modos de traducción para intentar aprovechar al máximo las ventajas de cada método. El **lenguaje intermedio o bytecode** es un lenguaje más abstracto respecto al hardware que el código máquina, pero que ha generado un archivo **binario** mediante el proceso de compilación de un lenguaje de alto nivel. El bytecode es ejecutado en el dispositivo mediante un intérprete.

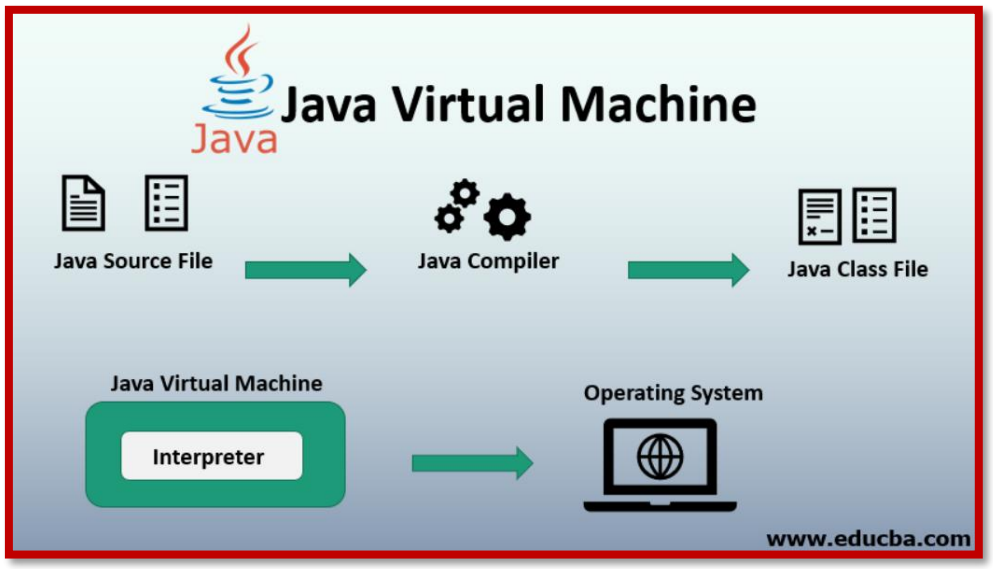
Las ventajas de estos métodos son:

- El bytecode ha sido compilado, por tanto, se han podido producir optimizaciones y se han encontrado errores de sintaxis.
- El bytecode es un archivo ejecutable y, por tanto, no es necesario recompilarlo cada vez.
- El bytecode es portable y no depende de ninguna arquitectura.

Ejemplos de este tipo de lenguaje son: Java, Kotlin (compatible con Java), C#, ...

Java

Un ejemplo de utilización de lenguajes intermedios es **Java**. El proceso de ejecución de un código fuente Java es el siguiente:



- **Código fuente**: Contiene el programa a ejecutar en lenguaje **Java** (alto nivel). Se utiliza la extensión **.java**.
- **Compilador**: Se compila el código fuente en bytecode con el compilador de Java, que se instala mediante **Java Development Kit (JDK)**.
- **Bytecode**: Contiene el programa en lenguaje intermedio en binario. Se utiliza la extensión **.class**.
- **JVM**: **Java Virtual Machine** interpreta el bytecode y lo traduce a lenguaje máquina, entendido por nuestro sistema operativo. JVM **se** instala mediante **Java Runtime Enviorment (JRE)** (viene incluido en JDK).

Código fuente:

```
joapuiib@coltrane:~/java$ cat HolaMon.java
public class HolaMon {
    public static void main(String[] args){
        System.out.println("Hola món!");
    }
}
```

Compilar:

```
joapuiib@coltrane:~/java$ ls
HolaMon.java
joapuiib@coltrane:~/java$ javac HolaMon.java
joapuiib@coltrane:~/java$ ls
HolaMon.class  HolaMon.java
joapuiib@coltrane:~/java$
```

Ejecutar:

```
joapuiib@coltrane:~/java$ java HolaMon
Hola món!
```

2.3. Según el paradigma

Los lenguajes de programación también pueden ser clasificados según su paradigma, es decir, su forma de trabajar y cómo están estructurados.

- **Paradigma imperativo**: Los lenguajes imperativos se basan en conocer el estado de la máquina y modificarlo mediante instrucciones. Se llaman imperativos porque utilizamos órdenes para decir al dispositivo qué debe hacer y cómo hacerlo. La mayoría de arquitecturas de ordenadores siguen una filosofía imperativa, por tanto, la traducción de un lenguaje imperativo a código máquina es más sencilla que la de los lenguajes declarativos.
    - Casi todos los lenguajes son imperativos: C, Java, Python, Javascript, PHP, ...
    - Dentro de esta categoría están:
        - **Lenguajes estructurados**: Que sólo permitan tres estructuras: Secuencia, selección e iteración. Esto hace innecesario el uso de saltos (*go to*) y se consideran una mala práctica, ya que empeoran la legibilidad y el mantenimiento del código. Sin embargo, se siguen utilizando saltos como el *break* en algunos casos.
            - Ejemplos: Fortan, Algol, Cobol, Basic, Pascal, C, Ada, ...
        - **Lenguajes orientados a objetos**: Es una estrategia de construcción de programas basada en una abstracción del mundo real. Los objetos son combinación de datos (*atributos*) y métodos que nos permiten interactuar con él. Este paradigma se basa en: abstracción, encapsulación, modularidad, jerarquía y polimorfismo.
            - Ejemplos: Java, C++, Python, C#, ...
- **Paradigma declarativo**: Los lenguajes declarativos fijan un objetivo, pero no el camino para llegar. Se indica qué valor se desea obtener y el dispositivo encuentra la solución siguiendo una lógica interna. Son lenguajes más cercanos a las matemáticas.
    - Ejemplos: SQL, Prolog, LISP, ...

2.4. ¿Cómo elegir lenguaje de programación?

Resolver esta pregunta podría llevarnos días enteros divagando entre todos los posibles paradigmas y tipos de lenguajes y llegaríamos a la conclusión de: **DEPENDE**.

Cada lenguaje de programación tiene unas características determinadas que lo hacen más o menos apropiado para el contexto donde lo utilizas, pudiendo ser determinantes estos factores:

1. El sector productivo al que va dirigida tu aplicación: Medicina, académico, videojuegos, data mining, ...
    - ¿Existen librerías que te facilitan la labor?
2. El tipo de dispositivo donde se ejecutará: móviles, escritorio, IoT, web, ...
3. El sistema operativo sobre la que se ejecutará: Linux, Windows, Android, Ios,...