1

LENGUAJES DE PROGRAMACIÓN

Durante los últimos años, en apenas el espacio de una generación, la informática ha pasado de ser una simple curiosidad a dominar prácticamente todos los aspectos de nuestra vida.

Pese a que el ordenador personal es el primer dispositivo que nos viene a la mente cuando pensamos en computación, hoy en día es posible escribir programas para casi cualquier aparato imaginable, desde grandes máquinas industriales hasta pequeños dispositivos. En todos nuestros electrodomésticos, automóviles, teléfonos, televisores, redes de comunicación, cajeros... hay un ordenador programable que se ocupa de llevar a cabo las tareas que se le han encomendado.

Hoy en día, todo el mundo tiene una idea, aunque sea más o menos intuitiva, de lo que es un "programa".

A grandes rasgos, **un programa informático no es más que un archivo con un conjunto de instrucciones para que el ordenador haga algo**:

- puede ser una tarea simple como sumar dos dígitos o,
- algo muy complejo, como la gestión que hace un sistema operativo de todas las tareas que el ordenador ejecuta en un momento determinado.

Los programas informáticos actúan sobre datos, sobre otros programas o sobre dispositivos físicos.

1.1. ALTO Y BAJO NIVEL

Los ordenadores (o, más exactamente, sus procesadores) usan internamente un lenguaje propio llamado código máquina. De hecho, hay más de un código máquina, y el que reconoce a un tipo de ordenadores no es reconocido por otros. El código máquina no es más que sucesiones de unos y ceros, y no está pensado para ser escrito ni leído por personas.

Pese a todo, originalmente, la gente literalmente programaba ordenadores escribiendo unos y ceros. Bien te criándolos en una consola, bien activando o desactivando interruptores, o bien perforando agujeros en una tarjeta de cartón.

El nivel de abstracción que esto requiere es desmesurado.

Por ejemplo, para almacenar el número 150 en un determinado registro de memoria del procesador, la instrucción en código máquina podría ser algo así:

1011000010010110

- 1011: los primeros cuatro dígitos son la orden de escribir el dato,
- 0000: la segunda serie de cuatro indica el registro de memoria en el que se quiere guardar,
- 10010110: el último grupo de dígitos se corresponde con el número 150 en representación binaria.

Como escribir instrucciones así es aburrido, difícil, y propenso a errores, se inventó el lenguaje ensamblador.

El lenguaje ensamblador, más que un lenguaje de programación en sí, es un simple sistema nemotécnico. En lugar de escribir largas ristras de números, se escribían algunas instrucciones cortas que, posteriormente, un programa traducía a estas listas de números:

- 1011 → MOV (para guardar un dato en la memoria)
- 0000 → AX (cada registro de memoria del procesador tenía un nombre)
- 10010110 → 96h (150 en notación hexadecimal, más corto y fácil de recordar que el binario)

Así, nuestra expresión quedaría más o menos así: **MOV AX 96h**

Esto era un avance, pero aún se sigue trabajando con instrucciones limitadas a registros de memoria, muy próximas a la forma de trabajar del ordenador, pero muy lejos de la lógica humana.

Naturalmente, conforme se iban escribiendo programas más complejos, esto se volvió impracticable. Hubo que inventar *lenguajes de programación* propiamente dichos, más sofisticados que el ensamblador.

Estos lenguajes permiten usar instrucciones que ya no se traducen a una sola instrucción de código máquina, sino a complejos conjuntos de ellas. Por ejemplo, para sumar varios números ya no hacía falta operar paso a paso a moviendo bits entre distintos registros de memoria, teniendo en cuenta la longitud de los dígitos, controlando el acarreo, moviendo el resultado a cierta zona de memoria, etc., sino que, simplemente, se podía escribir algo como **“5000 + 745 + 46”**.

Hoy en día existen multitud de lenguajes de programación pero, aunque los lenguajes ensambladores se siguen usando para ciertas tareas muy concretas, con el tiempo los lenguajes de programación se han ido aproximando más a nuestro propio lenguaje natural (o, para ser exactos, al inglés) y a la forma de pensar de los seres humanos.

En esto, como en casi todo, hay grados, y llamamos:

- Lenguajes de ***bajo nivel***: a aquellos que se acercan al modo de trabajar del ordenador.
- Lenguajes de ***alto nivel***: a los que se parecen más al modo de trabajar de los seres humanos.

1.2. LENGUAJES COMPILADOS E INTERPRETADOS

Para salvar el escollo entre lenguajes que entienden el procesador y lenguajes que entienden los humanos hay dos perspectivas: los lenguajes compilados y los interpretados.

Un ***lenguaje compilado*** es el que, tras escribir el programa, debe ser pasado por un programa especial (llamado compilador) que lo lee y crea a partir de él una versión en código máquina que es comprensible por el procesador. Al código escrito en el lenguaje de programación se le llama *código fuente* y a la versión compilada se le llama normalmente *binario*. Si hacemos algún cambio en el código fuente es necesario volver a compilarlo para obtener un nuevo programa.

Entre sus ventajas está en el que suele ser más rápidos que los lenguajes interpretados y que, una vez compilados, funcionan autónomamente, sin un programa que los interprete.

Por otro lado, al ser binarios, dependen de la plataforma en la que se ejecutan (procesadores distintos usarán binarios distintos), hay que compilar versiones distintas para cada plataforma y, cada vez que se modifican, necesitan ser compilados de nuevo.

Los lenguajes compilados más conocidos son el lenguaje C y sus variantes (C++, C#).

Un **lenguaje interpretado**, sin embargo, es el que se puede ejecutar sin necesidad de ser compilado. Para ello, en lugar de un compilador, tenemos lo que se llama un *intérprete*, que lee el código y ejecuta las instrucciones en él contenidas. Al no haber compilación no existe un binario, y los programas escritos en lenguajes interpretados se suelen llamar *scripts*.

Como ventajas tienen el que son portables entre plataformas (siempre que dispongan del intérprete adecuado) y que no necesitan ser compilados cada vez que se modifican.

Como desventajas, el que suelen ser más lentos que sus contrapartidas compiladas (aunque no siempre) y que necesitan de un intérprete.

Algunos de los lenguajes interpretados más famosos serían Perl, Ruby y, por supuesto, **Python**.

1.3. PARADIGMAS DE PROGRAMACIÓN

Otra cosa que distingue a unos lenguajes de programación de otros es el paradigma de programación en el que se basan.

Un paradigma de programación es “una forma de hacer las cosas". Existen muchos paradigmas, pero los más populares hoy en día son estos:

1.3.1. Programación imperativa

Es el paradigma más básico, el que subyace en cierto modo a cualquier otro paradigma, y el más usado.

En la programación imperativa el lenguaje se expresa a través de *instrucciones*, que son órdenes que cambian el estado de los datos sobre los que trabaja el programa.

1.3.2. Programación orientada a objetos

La programación orientada a objetos (POO) es probablemente el paradigma más de moda hoy en día, y el que ha supuesto algunos de los mayores avances en los últimos tiempos.

En este paradigma se trabaja con *objetos*, que son estructuras que permiten tanto almacenar como manipular datos de forma coherente e identificada.

1.3.3. Programación funcional

Este paradigma opera sobre los datos por medio de funciones que describen esos datos, de un modo muy parecido a cómo se opera con funciones matemáticas.

1.4. ¿QUÉ HEMOS VISTO EN ESTE TEMA?

Qué son los lenguajes de programación, para qué sirven, como se clasifican y las distintas formas (paradigmas) de usarlos.