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. Ese 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 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 tecleándolos 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
```

Los primeros cuatro dígitos son la orden de escribir el dato, la segunda serie de cuatro indica el registro de memoria en el que se quiere guardar, y el último grupo de cuatro 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 ristras de números.

Por ejemplo: en lugar de usar "1011" como instrucción para guardar un dato en la memoria, se podía usar la palabra MOV. Cada registro de memoria del procesador tenía un nombre (AX, en el ejemplo que hemos visto) y los números se podían escribir en hexadecimal, que es más corto y fácil de recordar que el binario.

Así, nuestra instrucción quedaría más o menos así (96h es 150 escrito en notación hexadecimal):

```
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 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, y 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 entiende 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 están el que suelen 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 como se opera con funciones matemáticas.

1.4 ¿QUÉ HEMOS VISTO EN ESTE TEMA?

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

2

PYTHON

Python es un lenguaje interpretado, de alto nivel y enfocado principalmente a la legibilidad y facilidad de aprendizaje y uso.

Como veremos a lo largo de este libro, Python es un lenguaje orientado a objetos, aunque soporta otros paradigmas como la programación funcional y, por supuesto, la programación imperativa.

Python es un lenguaje multiplataforma, lo que significa que puede usarse en multitud de sistemas distintos. Funciona en ordenadores con sistemas operativos Linux, BSD, Apple, Windows y muchos otros, pero también hay versiones para otros dispositivos, como terminales telefónicos inteligentes, etc.

Naturalmente, Python dispone, por medio del uso de *bibliotecas*, de herramientas para aprovechar las posibilidades concretas que le brinda cada plataforma, pero también es posible escribir programas evitando el uso de esas bibliotecas específicas, de modo que esos programas funcionen indistintamente en cualquier ordenador.

Python es software libre, y se distribuye bajo la licencia "Python Software Foundation License". Esto, entre otras cosas, significa que se distribuye gratuitamente y no necesita del pago de licencias o *royalties* para su uso, ya sea privado o comercial.

Guido van Rossum es el creador y BDLF (*Benevolent Dictator for Life*, benevolente dictador de por vida) de Python; y la filosofía que quiso darle es que el código debe ser limpio y legible, evitando "atajos" o construcciones que dificulten la comprensión del programa. Python es simple sin ser limitado.

Estas características hacen que Python sea un lenguaje ideal para aprender e iniciarse en la programación.

A pesar de que Python es una muy buena elección para aprender a programar, no es un lenguaje diseñado para aprender a programar. Python es un lenguaje completo perfectamente funcional, muy potente, y viene acompañado por una serie de paquetes que facilitan funciones para el trabajo con casi cualquier cosa. A este respecto, se dice que Python viene con "pilas incluidas".

2.1 SOFTWARE LIBRE

Ya hemos mencionado un par de veces que Python es software libre.

Aunque a menudo se interpreta equivocadamente como "software gratuito", el software libre es una filosofía que afirma que el software debe ser accesible a los usuarios.

En particular, los defensores del software libre afirman que hay una serie de derechos o "libertades del software" que todo usuario de un programa debería tener.

Según las propias palabras de la Free Software Fundation (*http://www.gnu.org/philosophy/free-sw.es.html*), las cuatro libertades del software libre son:

- La libertad de ejecutar el programa como se desea, con cualquier propósito (libertad 0).
- La libertad de estudiar cómo funciona el programa, y cambiarlo para que haga lo que usted quiera (libertad 1). El acceso al código fuente es una condición necesaria para ello.
- La libertad de redistribuir copias para ayudar a su prójimo (libertad 2).
- La libertad de distribuir copias de sus versiones modificadas a terceros (libertad 3). Esto le permite ofrecer a toda la comunidad la oportunidad de beneficiarse de las modificaciones. El acceso al código fuente es una condición necesaria para ello.

A lo largo de los últimos tiempos, la filosofía del software libre se ha extendido por todo el mundo, hasta tal punto que el software libre está detrás de la mayor parte de los grandes logros de la informática, desde la propia Internet hasta el sistema operativo Linux pasando por el propio Python.

2.2 EL ZEN DE PYTHON

El zen de Python (*http://www.python.org/dev/peps/pep-0020/*) viene a ser la carta de principios del lenguaje, su filosofía. Describe una serie de reglas que deberían seguirse tanto en el desarrollo del propio lenguaje, como en los programas que se hagan con él.

- Bello es mejor que feo.
- Explícito es mejor que implícito.
- Simple es mejor que complejo.
- Complejo es mejor que complicado.
- Plano es mejor que anidado.
- Disperso es mejor que denso.
- La legibilidad importa.
- Los casos especiales no son lo suficientemente especiales como para romper las reglas.
- Aunque la practicidad gana a la pureza.
- Los errores nunca deberían pasar en silencio.
- A menos que se silencien explícitamente.
- Frente a la ambigüedad, evita la tentación de adivinar.
- *Debería haber una (y preferiblemente solo una) manera obvia de hacerlo.*
- *Aunque puede no ser obvia al principio, salvo que seas holandés.*
- *Ahora es mejor que nunca.*
- *Aunque nunca es mejor que el ahora* correcto.
- *Si la implementación es difícil de explicar, es una mala idea.*
- *Si la implementación es fácil de explicar, podría ser una buena idea.*
- *Los espacios de nombres son una idea genial. ¡Hagamos más de eso!*

2.3 PYTHON 2 VERSUS PYTHON 3

Actualmente, existen dos versiones de Python conviviendo, ambas en desarrollo y con actualizaciones.

Por un lado, está la serie de "Python 2", que se encuentra en su versión 2.7 (en el momento de escribir este libro, la última versión estable es la 2.7.10, concretamente).

Simultáneamente, existe "Python 3", que se encuentra en su versión 3.4 (concretamente, en la 3.4.3).

Ambas versiones son muy similares, pero lo suficientemente diferentes para que un *script* escrito en Python 2 no sea compatible con Python 3, y viceversa.

Aunque, en el futuro, se espera que Python 3 sea la única versión que se mantenga, actualmente conviven las dos. De hecho, la actual versión de Python 2 es *posterior* a la actual versión de Python 3.

Desafortunadamente para las aspiraciones de Python 3, este no ha tenido una aceptación tan rápida como se esperaba. Una gran parte de las aplicaciones y librerías que se usan hoy en día existen para Python 2 pero aún no están portadas a Python 3 o solo lo están parcialmente.

Probablemente por esta razón, Python 2 sigue siendo hoy en día mucho más popular que Python 3. Y, por esta razón, en este libro vamos a usar Python 2 en lugar de Python 3.

¿Significa eso que vamos a aprender una versión de Python destinada a quedarse anticuada? Nada de eso. Por fortuna, las diferencias en la sintaxis de ambos lenguajes son mínimas, y aprender Python 2 es también aprender Python 3, si se tienen en cuenta unos aspectos mínimos.

Al final de este libro hay un apéndice en el que se da una pequeña lista de diferencias y algunos consejos para convertir nuestros *scripts* entre una versión y otra.

2.4 ¿QUÉ HEMOS VISTO EN ESTE TEMA?

Las características de Python y sus variantes. La filosofía que tiene detrás y el modo de enfocar la programación implícito en este lenguaje.

2.4.1 Tareas sugeridas

Quizás es pronto para comenzar con tareas, pero no sería mala idea consultar la página de Wikipedia dedicada a Python, para empezar a conocer algo de este lenguaje de programación.