


Open · Opened 4 days ago by  **Rubén Montero**

Introducción a Python

Resumen

- Hablaremos del lenguaje de programación Python
- Compararemos lenguajes de programación compilados e interpretados
- Nos aseguraremos de que podemos instalar y ejecutar código Python
- Distinguiremos entre el *intérprete* Python y un *fichero de código fuente* Python
- Escribiremos un sencillo fichero de código
- (*Y como siempre, subiremos los cambios al repositorio*)

Descripción

El lenguaje de programación [Python](#) es un lenguaje interpretado y dinámicamente tipado.

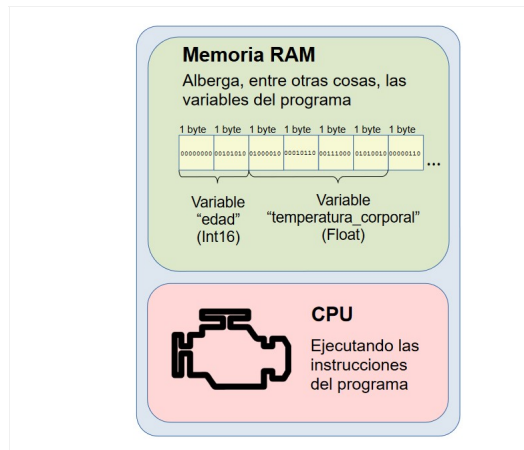
¿Interpretado?

Un lenguaje **interpretado**, en contraposición a uno compilado, es aquel cuyas instrucciones se van ejecutando en la máquina **paso a paso**. Es decir, existe un intérprete que va **leyendo** las instrucciones del programa, traduciéndolas a *código máquina* **bajo demanda** y ejecutándolas **al momento**.

Por otro lado, un lenguaje compilado atraviesa un proceso de *compilación* antes de ser ejecutado. Es decir, el programa **entero** se traduce a *código máquina* antes de ejecutarse.

¿Dinámicamente tipado?

Uno de los conceptos fundamentales en programación es el de **variable**. Una variable es un hueco en la memoria RAM de nuestros ordenadores que puede albergar cualquier valor. Dependiendo del tipo de dato que queremos albergar, hará falta más o menos espacio.



Si este programa fue escrito en un lenguaje de programación fuertemente tipado, como [Java](#), entonces el programador habrá tenido que teclear a mano el tipo de las variables en el momento de declararlas en el código:

```
short edad;  
float temperatura_corporal;
```

En contraposición, si este programa fue escrito en un lenguaje **dinámicamente tipado** como Python, el programador no se preocupó necesariamente de *qué* tipo serían las variables. Trabajó con ellas directamente y el intérprete les asignó el espacio necesario en memoria RAM.

Las variables, por lo tanto, no son *declaradas*. Se **asignan** directamente.

```
edad = 42  
temperatura_corporal = 37.555
```

Además, una variable puede *cambiar de tipo* durante la ejecución:

```
edad = 42
```

```
edad = "42 años"
```

Otro detalle sobre Python

Convenciones de nombrado

En Java estamos acostumbrados a usar `UpperCamelCase` para los nombres de las clases, y `lowerCamelCase` para los nombres de las variables y métodos.

Pues bien, en Python usaremos `snake_case` para las variables. Las clases continuarán siendo `UpperCamelCase`.

Usando Python

Existen **dos** versiones de Python: `Python 2` y `Python 3`.

Aunque lo parece, una **no** *sustituye* a la otra. Debes considerarlas como dos lenguajes separados, con muchas cosas en común que a veces facilitan que el código sea intercambiable, pero **no necesariamente**.

Nosotros instalaremos Python 3 desde la [página de descargas](#). Se aconseja descargar e instalar [Python 3.10.5](#), el más reciente en el momento de definir estas tareas.

Basta con:

- **Ejecutar** el `.exe` descargado
- **Marcar** la opción *Add Python 3.10 to PATH*.
 - Esto facilita que al lanzar un terminal (cmd.exe), el comando `python` funcione directamente. Si te olvidas, puedes [añadirlo al PATH a posteriori](#)
- **Install Now**
- Cuando termine, **Close**

¡Listo!

Debes poder abrir un terminal (Símbolo del sistema) con **Tecla de Windows > cmd** y escribir lo siguiente:

```
python --version
```

Producirá una salida como esta:

```
Python 3.10.5
```

Si **ejecutas el comando Python** directamente:

```
python
```

Debe salir algo como esto:

```
Python 3.10.5 (tags/v3.10.5:f377153, Jun 6 2022, 16:14:13) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

¡Observa que el *prompt* ha cambiado! Ahora es `>>>`.

Estamos ejecutando el [intérprete](#). Ya podemos escribir código Python.

¿Intérprete?

Se trata de un programa (`python.exe`) que, cuando se ejecuta, permite **de forma interactiva** escribir **código Python** y va *imprimiendo* los resultados automáticamente.

Es **útil** para efectuar pruebas de código si dudamos de cómo hacer algo.

Por ejemplo, si escribes `2+2`, verás el resultado automáticamente:

```
>>> 2+2
4
```

También puedes definir una variable:

```
>>> message = 'Hello, world!'
```

...y, al teclearla y darle a ENTER, el **intérprete** muestra su contenido:

```
>>> message  
'Hello, world!'
```

¡No confundir con código fuente!

Este **intérprete** sirve para **probar** código, pero **eso es todo**.

Cuando cerramos el terminal o escribimos `close()`, la sesión se elimina y **lo que hemos hecho se pierde**.

Oh...

Si queremos escribir programas, debemos crear **ficheros de código fuente Python**. Por convención, terminan en `.py`

Por ejemplo, la línea anterior que **define e inicializa** una variable:

```
message = 'Hello, world!'
```

...es **código Python** válido.

En esta ocasión, si queremos que se imprima por pantalla tendremos que hacer un `print`:

```
message = 'Hello, world!'  
print(message)
```

No lleva punto y coma (;)

Correcto. Al contrario que en Java, ahora **no** necesitamos escribir punto y coma (;) al final de cada sentencia.

Arrancando motores

Vamos a ponernos manos a la obra y escribir ese código fuente Python en nuestro repositorio.

En primer lugar, usa un terminal (cmd.exe) en tu ordenador y cambia el directorio activo hasta la ubicación de tu repositorio usando `cd`. Luego, haz:

```
git pull
```

...para traer los cambios de remoto a local. Contienen correcciones del proyecto anterior y **el esqueleto** del **nuevo proyecto**.

Se encontrará en **una carpeta llamada** `python-introduction`.

Usa tu **IDE** favorito

Para trabajar con Python existen muchas alternativas. Puedes usar [IntelliJ IDEA](#) con un *plugin para Python*, [VisualStudio Code](#), [Geany](#), [Sublime Text](#), o, si no tienes miedo a nada, opciones más rudimentarias como [Vi](#) o [Bloc de notas](#).

Con tal de que te resulte cómodo y ágil a la hora de escribir código y lanzar los *tests*, es suficiente.

PyCharm

Nosotros veremos cómo trabajar con [PyCharm Community Edition \(CE\)](#). Puedes descargarlo desde [aquí](#), abrir el **.exe** e instalarlo con las opciones por defecto.

Una vez instalado, puedes abrirlo buscándolo en el Menú Inicio. *Abriremos* nuestro proyecto desde el *popup* inicial, o bien, desde **File > Open**. Buscamos y seleccionamos la carpeta `python-introduction` de nuestro repositorio local.

- Si te dice que el proyecto es de un autor desconocido, puedes darle a **Trust Project** sin problema.
- Si indica que el certificado de `raspi` es inválido, puedes confiar sin problemas *siempre que el SHA-256 sea el siguiente*:

```
93 90 31 B3 49 80 4F 9A BA A9 A2 4B 68 33 26 92  
50 2A 1E 67 66 43 FE 9D 5E E2 33 60 E1 89 89 21
```

- Si indica que existe un `requirements.txt`, puedes crear un entorno virtual con las dependencias.
 - Nosotros iremos viendo cómo configurar e instalar tu entorno en caso de que omitas este paso.

¡Listos para trabajar!

Si haces *click* en el pequeño botón **Projects** de la barra vertical de la izquierda, verás las carpetas del proyecto.

```
- advanced/  
  
- api/  
  
- docs/  
  
- intro/  
  |  
  |- helloworld.py  
  
- tests/
```

- **advanced/** y **api/** son dos carpetas en las que trabajaremos más adelante.
- **docs/** contiene algunas imágenes e información. En general, es una buena práctica tener una carpeta separada para *documentación*.
- **intro/** contiene el módulo **intro**. Trabajaremos con él.
- **tests/** contiene los ficheros con el código de las [pruebas automatizadas en Python](#).

Hola, mundo

Comenzaremos escribiendo este sencillo programa.

Abrimos el fichero **helloworld.py** cuyo contenido es:

```
# TO-DO: Write a hello world!
```

...y escribimos:

```
message = 'Hello, world!'  
print(message)
```

¿Cómo probarlo?

Las cosas ahora son distintas

En Java, solíamos darle al botón **'Play'** y eso ejecutaba el método **main** en la clase **Main.java**. Decíamos que era nuestro *punto de entrada* en la aplicación.

Ahora, cualquier fichero .py puede ser el **punto de entrada** de la aplicación.

¿Un fichero de código Python puede depender de otro?

Sí. Existe la sentencia **import**.

¿Y cómo ejecuto mi **helloworld.py**?

Vamos a ver dos formas:

1) Desde un terminal (cmd.exe), nos posicionaremos en la carpeta **python-introduction/intro**. Desde ahí, podemos usar **python** igual que antes; **esta vez**, pasando **el nombre del fichero** como **argumento** (separado con espacio). Así:

```
python helloworld.py
```

Ejecutará el programa y producirá la salida esperada.

2) Configurando nuestro IDE para que permita ejecutar los ficheros con uno o dos *clicks*. Por ejemplo, en PyCharm CE podemos hacer *click* abajo a la derecha en **Add Interpreter** y eso nos permitirá decidir *qué intérprete* debe usar el IDE. Cualquier opción nos vale, prefiriendo usar System (intérprete del sistema) o un entorno virtual (venv).

También podemos llegar a este menú desde **File > Settings**, y en el panel lateral de la ventana, **Project > Python Interpreter**.

Si lo conseguimos configurar, podremos hacer *click* derecho > **Run 'helloworld.py'** y veremos la salida por pantalla.

¡Listo!

¡Enhorabuena! Has escrito y probado tu primer programa en Python.

Por último

Prueba el *test* asociado a la tarea. Lo ejecutaremos siguiendo los pasos de arriba, y esta vez lanzando `test_001helloworld.py`.

Si ves algo como...

```
Ran 1 test in 0.003s
```

OK

¡Tarea terminada!

Haz `commit` y `push` para subir los cambios al repositorio.

No está de más visitar la página de GitLab y verificar que el *commit* se ha subido.



Rubén Montero @ruben.montero changed milestone to [%Sprint 3](#) 4 days ago