

# PYTHON

## GUÍA PARA SER UN PYTHONISTA





# ■ 2

## Características básicas del lenguaje



# Índice de contenidos

<b><i>Introducción.....</i></b>	<b><i>5</i></b>
<b><i>Mi primer programa en Python.....</i></b>	<b><i>6</i></b>
Ejecutar código Python en una sesión interactiva .....	6
Ejecutar código Python que está contenido en un fichero .....	9
<b><i>Operadores, expresiones y sentencias .....</i></b>	<b><i>12</i></b>
Operadores .....	12
Expresiones .....	12
Instrucciones .....	13
<b><i>Bloques de código y sangrado .....</i></b>	<b><i>15</i></b>
<b><i>Comentarios .....</i></b>	<b><i>17</i></b>
Docstrings .....	20
<b><i>Convenciones de nombres en Python .....</i></b>	<b><i>21</i></b>
<b><i>Palabras reservadas de Python .....</i></b>	<b><i>23</i></b>
<b><i>Constantes .....</i></b>	<b><i>24</i></b>



## Introducción

Después del tema de introducción, este tema trata de mostrarte una primera aproximación al lenguaje, cuáles son sus características básicas y qué lo diferencia del resto de lenguajes de programación.

Aquí realizarás tu primer programa en Python y descubrirás las diferentes maneras de ejecutar código utilizando el intérprete.

Sigue siendo un tema introductorio, pero es muy importante que lo completes. Probablemente vuelvas a él en el futuro.



# Mi primer programa en Python

¡Vamos a escribir el primer programa en Python! Pero antes, un inciso.

Tal y como te adelanté en el tema anterior, existen dos formas de ejecutar código Python (utilizando el intérprete):

- A través de una sesión interactiva.
- Especificando un fichero que contiene las instrucciones y operaciones a ejecutar.

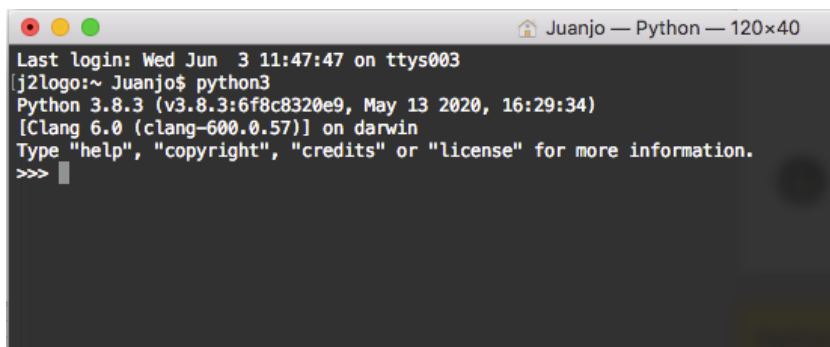
## Ejecutar código Python en una sesión interactiva

Una manera muy común de ejecutar código Python es a través de una sesión interactiva del intérprete. Para iniciar una sesión interactiva, simplemente abre un terminal y ejecuta el comando `python3` (o `python`, según se haya realizado la instalación).

**NOTA:** A partir de aquí no volveré a diferenciar entre el comando `python3` o `python`. Yo utilizaré `python3` porque es el habitual con esta versión del lenguaje. Tú utiliza el comando que tengas disponible.



Tras ello, te debe aparecer algo similar a la siguiente imagen:

A terminal window titled 'Juanjo — Python — 120x40'. The text inside shows the system login 'Last login: Wed Jun 3 11:47:47 on ttys003', the command '[j2logo:~ Juanjo\$ python3]', and the Python 3.8.3 startup banner: 'Python 3.8.3 (v3.8.3:6f8c8320e9, May 13 2020, 16:29:34) [Clang 6.0 (clang-600.0.57)] on darwin'. It then prompts 'Type "help", "copyright", "credits" or "license" for more information.' and shows the prompt '>>>' with a cursor.

¿Qué significa esto? Básicamente que has ejecutado el intérprete y está a la espera de que introduzcas instrucciones para poder ejecutarlas.

Cualquier instrucción o expresión que introduzcas tras los caracteres `>>>`, el intérprete los interpretará y ejecutará inmediatamente.

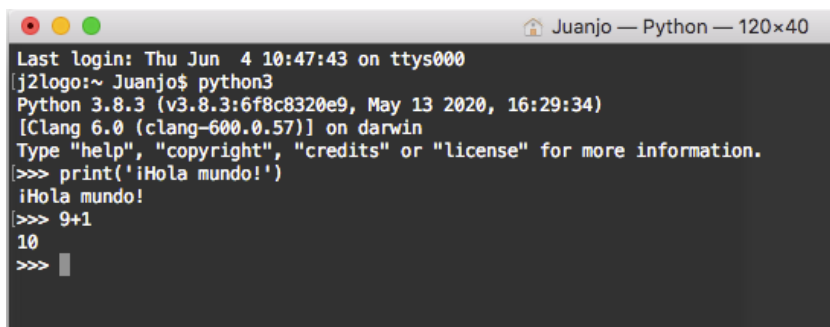
¿Quieres crear tu primer programa en Python? Escribe en el intérprete `print('¡Hola mundo!')` y pulsa la tecla `Enter`, a ver qué pasa.

Efectivamente, el intérprete te muestra el texto `¡Hola mundo!`.

**NOTA:** Acabas de aprender tu primera función del lenguaje, `print()`. Por ahora, simplemente, quédate con que esta función muestra un texto por pantalla. Iremos descubriendo más sobre ella en varios temas posteriores.

Escribe ahora la expresión `9 + 1` y, de nuevo, pulsa `Enter`.

Ahora, el intérprete te muestra el resultado de la suma: `10`.



```
Last login: Thu Jun  4 10:47:43 on ttys000
[j2logo:~ Juanjo$ python3
Python 3.8.3 (v3.8.3:6f8c8320e9, May 13 2020, 16:29:34)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print('¡Hola mundo!')
¡Hola mundo!
>>> 9+1
10
>>> 
```

Utilizando el intérprete puedes escribir y ejecutar el código Python que desees. El único problema es que, cuando cierres la sesión, el código desaparecerá. No obstante, es una buena forma de experimentar con el lenguaje y probar código al instante.





Para terminar la sesión y salir del intérprete, ejecuta las instrucciones `quit()` o `exit()`.

## Ejecutar código Python que está contenido en un fichero

Para hacer pruebas, una sesión interactiva puede estar bien, pero no es así como se programa realmente.

En Python, como en cualquier lenguaje de programación, el código y las instrucciones que componen un programa se escriben en ficheros de texto. Estos ficheros son procesados por el intérprete (como vimos en el tema anterior), que lee y ejecuta cada una de las instrucciones escritas (según el *Modelo de ejecución de Python*).

Por norma, la extensión de un fichero que contiene código Python es `.py`. Recuerda que los ficheros con extensión `.pyc` son generados por el intérprete y contienen *bytecode*.

Veamos a continuación cómo ejecutar un programa escrito en Python que está en un fichero.

Sitúate en un directorio de trabajo (el que prefieras) y crea en él un archivo llamado `main.py` (el nombre



puede ser cualquiera, pero debe acabar con extensión `.py`)

Abre el fichero con un editor de textos (*PyCharm*, *VS Code*, *Sublime Text*, *Atom*, *Bloc de notas*, ...) y escribe lo siguiente:

```
print('¡Hola mundo!')
```

Guarda el fichero.

A continuación, abre el terminal, sitúate en el directorio de trabajo anterior y ejecuta esto:

```
$> python3 main.py
```

¿Ves por pantalla el mensaje `¡Hola mundo!`?

**Esta es la manera habitual de programar y ejecutar un programa en Python.**





Consulta los videotutoriales de este tema en los que te explico cómo crear y ejecutar un programa en Python utilizando *PyCharm* y *Visual Studio Code*.

Una vez que hemos creado nuestro primer programa en Python y nos hemos familiarizado con el intérprete, voy a continuar mostrándote las características básicas del lenguaje.



# Operadores, expresiones y sentencias

Como en otros lenguajes de programación, un programa Python está compuesto por instrucciones, expresiones y operadores.

## Operadores

Un operador es un carácter o conjunto de caracteres que actúa sobre una, dos o más variables y/o literales para llevar a cabo una operación con un resultado determinado.

Ejemplos de operadores comunes son los operadores aritméticos `+` (suma), `-` (resta) o `*` (producto), aunque en Python existen otros operadores. Los veremos en el tema correspondiente.

## Expresiones

Una expresión es una unidad de código que devuelve un valor y está formada por una combinación de operandos (variables y literales) y operadores.

Los siguientes son ejemplos de expresiones:



```
a + 3
5 < edad
user is None
```

## Instrucciones

Una instrucción es una sentencia o declaración que puede ser ejecutada por el intérprete. Algunas instrucciones están formadas por expresiones, otras no.

En definitiva, las instrucciones o sentencias son las acciones que indican qué debe hacer un programa.

Ejemplos de sentencias son la asignación `=` o las instrucciones `if`, `if ... else ...`, `for` o `while`, entre otras.

```
# Asigna a la variable suma el valor de
# la variable a + 3
suma = a + 3

if edad < 18:
    print('Es menor')
```



A diferencia de otros lenguajes, en los que una instrucción finaliza con el carácter `;`, en Python, una sentencia acaba con el carácter `\n`, es decir, `Enter`.

Como norma general, una instrucción debe ocupar una sola línea. ¿Qué ocurre si es demasiado larga?

Las normas de estilo de Python (recogidas en la *PEP-8*) recomiendan una longitud de línea máxima de 79 caracteres. Si una sentencia ocupa 80 caracteres o más, entonces lo mejor sería dividirla en varias líneas.

La forma preferida de separar varias líneas es usar la continuación de línea implícita dentro de paréntesis `()`, corchetes `[]` o llaves `{}`. Cualquier expresión se puede encerrar entre paréntesis y dividir en varias líneas.

La otra manera, es utilizar el carácter `\` al final de línea. Utiliza esta forma cuando no puedas emplear ninguna de las anteriores. Te muestro todo esto con un ejemplo:

```
suma = (2 + 3
        + 5 + 10)
```

```
suma2 = 1 + 2 \
        + 3 + 4
```





Los editores de texto e IDEs como *PyCharm* y *Visual Studio Code* suelen marcar de algún modo el tamaño máximo de línea. Muchos de ellos, definen un máximo de línea por defecto de 80 caracteres. Este parámetro lo puede modificar el usuario.

## Bloques de código y sangrado

Si ya tienes experiencia con otros lenguajes (y si no, ahora lo descubrirás) sabrás que las instrucciones de un programa se agrupan en bloques de código.

Un bloque de código es un grupo de sentencias relacionadas bien delimitadas. A diferencia de otros lenguajes como *JAVA* o *C*, en los que se usan los caracteres `{ }` para definir un bloque de código, **en Python se usa el sangrado** (o indentación, esto es un anglicismo que no existe realmente en nuestro idioma).

**El sangrado** o indentación **consiste en mover un bloque de texto hacia la derecha insertando espacios o tabuladores al principio de cada línea**, dejando un margen a la izquierda.





Esta es una de las principales características de Python.

**Y te adelanto que debes tener mucho cuidado, ya que un simple espacio donde no debes puede hacer que tu programa no funcione. ¡OJO CON ESTO!**

Por tanto, un bloque de código comienza con un nuevo sangrado y acaba con la primera instrucción cuyo sangrado sea menor. En principio, el bloque de instrucciones principal no tiene ningún sangrado.

De nuevo, las guías de estilo de Python nos hacen una recomendación: Usar los espacios en lugar de las tabulaciones para realizar el sangrado. Yo suelo utilizar 4 espacios para visualizar bien los diferentes bloques (el sangrado también se puede configurar en los editores).

Veámoslo con un ejemplo (que no hace falta que entiendas todavía):





```
numeros = [1, 2, 3, 4]    # Bloque 1
suma = 0                  # Bloque 1

for n in numeros:         # Bloque 1
    if n % 2 == 0:         # Bloque 2
        suma += n         # Bloque 3
        print(n)          # Bloque 3
print(suma)               # Bloque 1
```

## Comentarios

Como cualquier otro lenguaje de programación, Python permite escribir comentarios en el código. Los comentarios son útiles para explicar el **qué** o el **por qué** estamos programando algo o, simplemente, para añadir indicaciones. Te aseguro que son de utilidad cuando se retoma un programa o aplicación en el futuro.



No utilices comentarios para explicar el cómo estás programando algo, eso ya se puede apreciar en el código.

Los comentarios son ignorados por el intérprete de Python. Solo tienen sentido para los programadores.

Para añadir un comentario a tu código, comienza una línea por el carácter `#`.

A continuación, te muestro un ejemplo (por el momento no hace falta que entiendas el código, solo fíjate en el comentario):

```
numeros = [1, 2, 3, 4]
encontrado = False

# Busca un número par
for n in numeros:
    if n % 2 == 0:
        encontrado = True
        break
print(encontrado)
```

Para añadir comentarios de más de una línea, comienza cada línea por el carácter `#`:



```
numeros = [1, 2, 3, 4]
num_par = None

# Busca un número par en la secuencia.
# Si existe, devuelve el primero
# de ellos
for n in numeros:
    if n % 2 == 0:
        num_par = n
        break
print(num_par)
```



Un tipo de comentario muy útil es aquel que comienza por **# TODO**.

Este comentario significa que has dejado algo a medias o que debes revisarlo en un futuro.

¿Por qué utilizarlo? Porque los editores de código como *PyCharm* o *Visual Studio Code* identifican este tipo de comentarios y te crean accesos directos hacia ellos.

Cuando tu programa está formado por miles de líneas de código separadas en múltiples ficheros, es una buena forma de recordar rápidamente cosas que has dejado pendientes.

## Docstrings

Los *docstrings* son un tipo de comentarios especiales que se usan para documentar un módulo, función, clase o método (ya veremos qué es cada uno de estos conceptos a su debido tiempo). En realidad, son la primera sentencia de cada uno de ellos.

A diferencia de los comentarios comunes, se encierran entre tres comillas simples `'''` o dobles `"""`.

Los *docstrings* son utilizados para generar la documentación de un programa. Además, suelen utilizarlos los entornos de desarrollo para mostrar la documentación al programador de forma fácil e intuitiva.

Veamos un ejemplo de *docstring* de una función `suma` (de nuevo, no hace falta que entiendas el código, solo fíjate en el *docstring*):



```
def suma(a, b):  
    """  
    Esta función devuelve la suma  
    de dos números.  
    :param a: Primer sumando  
    :param b: Segundo sumando  
    :return: Devuelve la suma de los  
             sumandos a y b  
    """  
    return a + b
```

Hablaremos más en profundidad de los *docstrings* en una clase dedicada a ellos.

## Convenciones de nombres en Python

Si estás haciendo este curso, imagino que ya tienes nociones de programación y, más o menos, sabes lo que es una variable, una función, una clase, etc. Si no es así y estás aprendiendo a programar, vuelve a esta sección cuando sepas qué es cada uno de estos conceptos. Hay un tema dedicado a cada uno de ellos, de manera que, si los desconoces o solo tienes una ligera idea, muy pronto serás un experto/a en la materia.



Te decía todo esto porque me voy a poner un poco técnico. A continuación, voy a darte una serie de indicaciones sobre convenciones de nombres en Python que son muy importantes que conozcas.

A la hora de dar un nombre a una variable, una función, un módulo, una clase, etc. en Python, siempre se siguen las siguientes reglas y recomendaciones:

- Un identificador puede ser cualquier combinación de letras (mayúsculas y minúsculas), números y el carácter guion bajo `_`.
- Un identificador no puede comenzar por un número.
- A excepción de los nombres de clases, es una convención que todos los identificadores se escriban en minúsculas, separando las palabras con el guion bajo. Ejemplos: `contador`, `suma_enteros`.
- Es una convención que los nombres de clases sigan la notación *Camel Case*, es decir, todas las letras en minúscula a excepción del primer carácter de cada palabra, que se escribe en mayúscula. Ejemplos: `Coche`, `VehiculoMotorizado`.



- No se pueden usar como identificadores las palabras reservadas (las veremos en la siguiente sección).
- Como recomendación, usa identificadores que sean expresivos. Por ejemplo, `contador` es mejor que simplemente `c`.
- Python diferencia entre mayúsculas y minúsculas, de manera que `variable_1` y `Variable_1` son dos identificadores totalmente diferentes.

## Palabras reservadas de Python

El lenguaje Python tiene una serie de palabras clave reservadas, por tanto, no pueden usarse como nombres de variables, funciones, etc.

Estas palabras clave se utilizan para definir la sintaxis y estructura del propio lenguaje.

La lista completa de palabras reservadas es la siguiente:

```
and, as, assert, break, class, continue, def, del,  
elif, else, except, False, finally, for, from,  
global, if, import, in, is, lambda, None, nonlocal,  
not, or, pass, raise, return, True, try, yield,  
while y with
```



¿Te parecen muchas o pocas? Da igual porque las vas a aprender todas en este curso, jaja.

## Constantes

Terminamos este tema sobre las características básicas de Python señalando que, a diferencia de otros lenguajes, en Python no existen las constantes.

Entendemos como constante una variable que, una vez que se le ha asignado un valor, este no se puede modificar.

Se puede simular este comportamiento, siempre desde el punto de vista del programador y atendiendo a convenciones propias, pero no podemos cambiar la naturaleza mutable de las variables.

No obstante, sí es cierto que el propio Python define una serie de valores constantes en su propio *namespace* (¿Esto qué es? Ya lo verás en su tema). Los más importantes son:

- `False`: El valor *falso* del tipo `bool`.
- `True`: El valor *verdadero* del tipo `bool`.
- `None`: El valor del tipo `NoneType`. Generalmente `None` se utiliza para representar la ausencia de





valor, la nada o el desconocimiento. En otros lenguajes este valor se representa como `NULL`, `null` o `nil`.





