Escribiendo tu Primer Código en Python

Objetivos

Tras completar esta práctica serás capaz de:

- Escribir código básico en Python
- Trabajar con varios tipos de datos en Python
- Convertir los datos de un tipo a otro
- Usar expresiones y variables para realizar operaciones

Di Hola mundo en Python

Al aprender un nuevo lenguaje de programación, se acostumbra comenzar con un ejemplo de "hola mundo". Tan simple como es, esta línea de código asegurará que sepamos cómo imprimir una cadena en la salida y cómo ejecutar el código dentro de las celdas de un Google Collab.

[Tip]: Para ejecutar el código de Python en la celda de código a continuación, haz clic en la celda para seleccionarla y presiona `Shift` + `Enter`.

```
1 # Prueba tu primer output de Python
2
3 print('Hello, Python!')
```

```
Hello, Python!
```

Después de ejecutar la celda anterior, debería ver que Python imprime `Hello, Python!`.

[Tip:] `print()` es una función. `'Hello, Python!'` es un String que se le pasa a la función como argumento.

¿Qué versión de Python estamos usando?

Hay dos versiones populares del lenguaje de programación Python en uso hoy en día: Python 2 y Python 3. La comunidad de Python ha decidido pasar de Python 2 a Python 3, y muchas bibliotecas populares han anunciado que ya no serán compatibles con Python 2.

Dado que Python 3 es el futuro, en este curso lo usaremos exclusivamente. ¿Cómo sabemos que nuestro cuaderno es ejecutado por un tiempo de ejecución de Python 3? Podemos mirar en la esquina superior derecha de este cuaderno y ver "Python 3".

También podemos preguntar directamente a Python y obtener una respuesta detallada. Prueba a ejecutar el siguiente código:

```
1 # Comprobar la versión de Python
2
3 import sys
4 print(sys.version)
```

```
3.7.15 (default, Oct 12 2022, 19:14:55)
[GCC 7.5.0]
```

[Tip:] `sys` es una libería nativa que contiene muchos parámetros y funciones específicos del sistema, incluida la versión de Python en uso. Antes de usarla, debemos importarla.

Comentarios

Además de escribir código, tenga en cuenta que siempre es una buena idea agregar comentarios a su código. Ayudará a otros a comprender lo que intentaba lograr (la razón por la que escribió un fragmento de código determinado). Esto no solo ayuda a **otras personas** a comprender su código, sino que también puede servirle como un recordatorio **para usted** cuando vuelva a consultarlo semanas o meses después.

Para escribir comentarios en Python, use el símbolo `#` antes de escribir su comentario. Cuando ejecuta su código, Python ignorará todo lo que esté más allá del `#` en una línea determinada.

```
1 print('Hello, Python!') # Esta línea imprime un String
2 # print('Hola')
```

Después de ejecutar la celda anterior, debe notar que `Esta línea imprime una String` no apareció en el output, porque era un comentario (y por lo tanto ignorado por Python).

¡La segunda línea tampoco se ejecutó porque `print('Hola')` también estaba precedido por el signo de número (`#`)! Dado que este no es un comentario explicativo del programador, sino una línea de código real, podríamos decir que el programador *comentó* esa segunda línea de código.

Errores

Todo el mundo comete errores. En la mayoría de casos, Python le avisará y le dirá que ha cometido un error al darle un mensaje de error. Es importante leer los mensajes de error detenidamente para comprender realmente dónde cometió un error y cómo puede corregirlo.

Por ejemplo, si escribe `print` como `frint`, Python mostrará un mensaje de error. Pruébalo:

```
1 # La siguiente línea da un error
2
3 frint("Hello, Python!")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-4-564039a8822b> in <module>
      1 # La siguiente línea da un error
      2
----> 3 frint("Hello, Python!")

NameError: name 'frint' is not defined
```

SEARCH STACK OVERFLOW

El mensaje de error te dice:

1. dónde ocurrió el error (más útil en celdas de cuaderno grandes), y
2. qué tipo de error fue (NameError)

Aquí, Python intentó ejecutar la función `frint`, pero no pudo determinar qué es `frint` ya que no es una función integrada y no ha sido definida previamente por nosotros tampoco.

Notará que si cometemos un tipo diferente de error, al olvidar cerrar la cadena, obtendremos un error diferente (es decir, un `SyntaxError`). Pruébalo a continuación:

```
1 print("""Hello, Python!""")
```

```
Hello, Python!a
```

```
1
```

¿Python sabe acerca de su error antes de ejecutar su código?

Python es lo que se llama un *lenguaje interpretado*. Los lenguajes compilados examinan todo su programa en el momento de la compilación y pueden advertirle sobre toda una clase de errores antes de la ejecución. Por el contrario, Python interpreta su script línea por línea a medida que lo ejecuta. Python dejará de ejecutar todo el programa cuando encuentre un error (a menos que el programador espere y maneje el error, un tema más avanzado que trataremos más adelante en este curso).

Prueba a ejecutar el código de la celda de abajo y mira qué pasa:

```
1 # Imprime el primer string y la segunda línea da error
2
3 print("This will be printed")
4 frint("This will cause an error")
5 print("This will NOT be printed")
```

```
This will be printed
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-37bbfdb5d12a> in <module>
      2
      3 print("This will be printed")
```

Definición de bloques

A diferencia de otros lenguajes que utilizan llaves para definir los bloques de código, cuando Guido Van Rossum creó el lenguaje quiso evitar estos elementos. Es por ello que en Python los bloques de código se definen a través de **espacios en blanco** (preferiblemente 4, ver PEP8).

> Esto puede resultar extraño e incómodo a personas que vienen de otros lenguajes de programación pero desaparece rápido y se siente natural a medida que se escribe código.

Ejercicio: Tu Primer Programa

Generaciones de programadores han comenzado sus carreras de codificación simplemente escribiendo "¡Hola, mundo!". Vas a seguir sus pasos.

En la celda de código a continuación, usa la función `print()` para imprimir la frase: `¡Hola, mundo!`

```
1 # Escribe tu código debajo. No te olvides de pulsar Shift+Enter para ejecutar la celda
2 print("Hola mundo")
```

```
Hola mundo
```

▼ Pulsa aquí para ver la solución

```
print("¡Hola, mundo!")
```

Ahora, vamos a mejorar tu código con un comentario. En la celda de código a continuación, imprime la frase: `¡Hola, mundo!` y coméntala con la frase `Imprime el tradicional hola mundo` todo en una línea de código.

```
1 # Escribe tu código debajo. No te olvides de pulsar Shift+Enter para ejecutar la celda
2 print ("Hola mundo") # Imprime el tradicional hola mundo
```

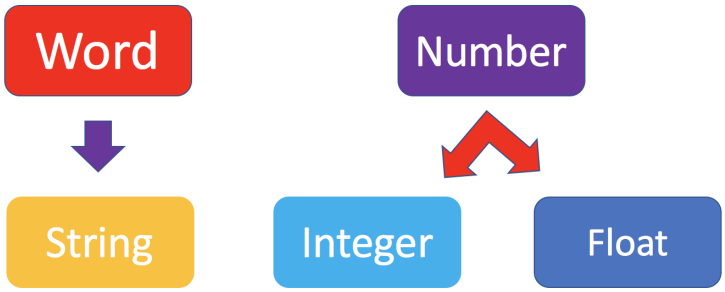
```
Hola mundo
```

▼ Pulsa aquí para ver la solución

```
print("¡Hola, mundo!") # Imprime el tradicional hola mundo
```

Tipos de objetos en Python

Hay muchos tipos diferentes de objetos en Python. Comencemos con los tipos de objetos más comunes: string, ints y float. Cada vez que escribe palabras (texto) en Python, está utilizando cadenas de caracteres (string). Los números más comunes, por otro lado, son los enteros (por ejemplo, -1, 0, 100) y los flotantes, que representan números reales (por ejemplo, 3.14, -42.0).



Nombre	Tipo	Mutable	Ejemplos
Booleano	`bool`	❌	`True, False`
Entero	`int`	❌	`21, 34500, 34_500`
Flotante	`float`	❌	`3.14, 1.5e3`
Complejo	`complex`	❌	`2j, 3 + 5j`
Cadena	`str`	❌	`'tfn', "tenerife", '''tenerife - islas canarias'''`
Tupla	`tuple`	❌	`(1, 3, 5)`
Lista	`list`	✅	`['Chrome', 'Firefox']`
Conjunto	`set`	✅	`set([2, 4, 6])`
Diccionario	`dict`	✅	`{'Chrome': 'v79', 'Firefox': 'v71'}`

Tipos de datos en Python - Documentación oficial

Las siguientes casillas contienen algunos ejemplos..

```
1 # Enteros
2
3 11
```

```
11
```

```
1 # Flotantes
2
3 2.14
```

```
2.14
```

```
1 # Texto
2
3 "¡Hola, Python 101!"
```

```
'¡Hola, Python 101!'
```

Puede hacer que Python le diga el tipo de una expresión usando la función `type()`. Notarás que Python se refiere a los números enteros como `int`, números con decimales a `float`, y el texto a `str`.

```
1 # Tipo de 12
2
3 type(12)
```

```
int
```

```
1 # Tipo de 2.14
2
3 type(2.14)
```

```
float
```

```
1 # Tipo de "Hola Mundo!"
2
3 type("Hola Mundo!")
```

```
str
```

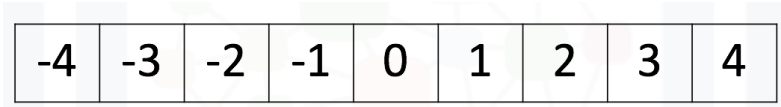
En la siguiente casilla, usa la función `type()` para comprobar el tipo del objeto `12.0`.

```
1 # Escriba tu código a continuación. No olvides presionar Shift+Enter para ejecutar la celda
2 type("Hola")
```

```
    str
```

Enteros

Estos son algunos ejemplos de números enteros. Los números enteros pueden ser números negativos o positivos:



Podemos comprobarlo una vez más con la función `type()`:

```
1 # Imprime el tipo de -1
2
3 type(-1)
```

```
    int
```

```
1 # Imprime el tipo de 4
2
3 type(4)
```

```
1 # Imprime el tipo de 0
2
3 type(0)
```

Números flotantes

Los flotantes representan números reales; son un superconjunto de números enteros pero también incluyen números con decimales.

```
1 # Imprime el tipo de 1.0
2
3 type(1.0) # Observa que 1 es un int y 1.0 es un float
```

```
1 # Imprime el tipo de 0.5
2
3 type(0.5)
```

```
1 # Imprime el tipo de 0.56
2
3 type(0.56)
```

Conversión de un tipo de objeto a otro diferente

Puedes cambiar el tipo del objeto en Python; esto se llama typecasting. Por ejemplo, puedes convertir un *integer* en un *float* (ej. 2 a 2.0).

Vamos a probarlo:

```
1 # Verifica que es un número entero
2
3 type(2)
```

Casting de números enteros a flotantes

Convirtamos el entero 2 en flotante:

```
1 float(2)
```

```
    2.0
```

```
1 # Convierte el entero 2 en un flotante y comprueba su tipo
2
```

```
3 type(float(2))
```

```
float
```

Cuando convertimos un número entero en flotante, en realidad no cambiamos el valor (es decir, el significado) del número. Sin embargo, si convertimos un flotante en un entero, potencialmente podríamos perder algo de información. Por ejemplo, si convertimos el flotante 1.1 en entero, obtendremos 1 y perderemos la información decimal (es decir, 0.1):

```
1 # La conversión de 1.1 a entero resultará en la pérdida de información
2
3 int(1.1)
```

Conversión de cadenas a números enteros o flotantes

A veces, podemos tener una cadena que contiene un número dentro de ella. Si este es el caso, podemos convertir esa cadena que representa un número en un número entero usando `int()`:

```
1 # Convierte una cadena en un número entero
2
3 int('1')
```

```
1
```

Pero si intentas hacerlo con una cadena que no coincide perfectamente con un número, te aparecerá un error. Prueba lo siguiente:

```
1 # Convierte una cadena en un entero de manera errónea
2
3 int('1 or 2 people')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-21-b96e914cb969> in <module>
      1 # Convierte una cadena en un entero de manera errónea
      2
----> 3 int('1 or 2 people')

ValueError: invalid literal for int() with base 10: '1 or 2 people'
```

SEARCH STACK OVERFLOW

También puedes convertir cadenas que contengan números flotantes en objetos *float*:

```
1 # Convierte la cadena "1.2" en un número flotante
2
3 float('1.2')
```

[Tip:] Ten en cuenta que las cadenas se pueden escribir con comillas simples (`'1.2'`) o dobles (`"1.2"`), pero no puedes mezclarlas (e.g., `"1.2'`).

Conversión de números a cadenas

Si podemos convertir cadenas en números, es natural asumir que podemos convertir números en cadenas, ¿no?

```
1 # Convierte un número entero en una cadena
2
3 str(1)
4
```

```
'1'
```

Y no hay razón por la cual no podamos convertir números flotantes en cadenas también:

```
1 # Convierte un número flotante en una cadena
2
3 str(1.2)
```

Dato de tipo booleano

El *Booleano* es otro tipo importante en Python. Un objeto de tipo *Booleano* puede tomar uno de dos valores: `Verdadero o Falso`:

```
1 # Valor verdadero
2
3 True
```

```
True
```

Observa que el valor `True` tiene una "T" mayúscula. Lo mismo para `False` (es decir, debes usar la "F" mayúscula).

```
1 # Valor falso
2
3 False
```

Cuando le pides a Python que muestre el tipo de un objeto booleano, mostrará `bool` que significa *booleano*:

```
1 # Tipo de True
2
3 type(True)
```

```
bool
```

```
1 # Tipo de False
2
3 type(False)
```

Podemos convertir objetos booleanos a otros tipos de datos. Si convertimos un booleano con el valor `True` en un número entero o flotante, obtendremos un uno. Si convertimos un booleano con el valor `False` en un número entero o flotante, obtendremos un cero. Igualemnte, si convertimos un 1 en un booleano, obtienes un `True`. Y si convertimos un 0 en un booleano obtendremos un `False`. Vamos a intentarlo:

```
1 # Convierte un True en un entero
2
3 int(True)
```

```
1
```

```
1 # Convierte un 1 en un booleano
2
3 bool(1)
```

```
True
```

```
1 # Convierte un 0 en un booleano
2
3 bool(0)
```

```
False
```

```
1 # Convierte un True en un número flotante
2
3 float(True)
```

```
1.0
```

Ejercicio: Tipos

¿Qué tipo de dato es el resultado de: `6 / 2`?

```
1 # Escriba tu código a continuación. No olvides presionar Shift+Enter para ejecutar la celda
2 type (6/2)
```

```
float
```

¿Qué tipo de dato es el resultado de: `6 // 2`? (Nótese la doble barra `//`.)

```
1 # Escriba tu código a continuación. No olvides presionar Shift+Enter para ejecutar la celda
2 type (6//2)
```

```
int
```

Expresiones y Variables

Expresiones

Las expresiones en Python pueden incluir operaciones entre tipos compatibles (enteros y flotantes). Por ejemplo, operaciones aritméticas básicas como sumar varios números:

```
# Expresión de operación de suma

43 + 60 + 16 + 41
```

Podemos realizar operaciones de resta usando el operador menos. En este caso el resultado es un número negativo:

```
# Expresión de operación de resta

50 - 60
```

Podemos hacer una multiplicación usando un asterisco:

```
# Expresión de operación de multiplicación

5 * 5
```

También podemos hacer unas división con una barra:

```
# Expresión de operación de división

25 / 5
```

```
# Expresión de operación de división

25 / 6
```

Como se ve en el cuestionario anterior, podemos usar la barra doble para la división de enteros, donde el resultado se redondea a la baja al entero más cercano:

```
# Expresión de operación de división de enteros

25 // 5
```

```
# Expresión de operación de división de enteros

25 // 6
```

Ejercicio: Expresión

Escribe una expresión que calcule cuántas horas hay en 160 minutos:

```
# Escriba tu código a continuación. No olvides presionar Shift+Enter para ejecutar la celda
160//60
```

```
2
```

Python sigue bien los convenios matemáticos al evaluar expresiones matemáticas. En el siguiente ejemplo, Python suma 30 al resultado de la multiplicación (en total, 120).

```
# Expresión matemática

30 + 2 * 60
```

```
150
```

Y al igual que en matemáticas, las expresiones entre paréntesis tienen prioridad. Entonces lo siguiente multiplica 32 por 60.

```
# Expresión matemática

(30 + 2) * 60
```

Variables

Al igual que con la mayoría de los lenguajes de programación, podemos almacenar valores en *variables*, para que podamos usarlos más adelante. Por ejemplo:

```
# Guarda el valor en una variable

x = 43 + 60 + 16 + 41
```

Para ver el valor de `x` en un Google Collab, simplemente podemos colocarlo en la última línea de la celda:

```
# Imprime el valor de la variable

x
```

```
160
```

También podemos realizar operaciones en `x` y guardar el resultado en una nueva variable:

```
# Usa otra variable para almacenar el resultado de la operación entre la variable y el valor

y = x / 60
y
```

```
2.6666666666666665
```

Si guardamos un valor en una variable existente, el nuevo valor reemplazará al valor anterior:

```
# Sobreescribe una variable con un nuevo valor

x = x / 60
x
```

```
2.6666666666666665
```

Es una buena práctica usar nombres de variables significativos, para que tú y otras personas puedan leer el código y comprenderlo más fácilmente:

```
# Pone un nombre con significado a la variable

total_min = 43 + 42 + 57 # La duración total de varios álbumes en minutos
total_min
```

```
142
```

```
# Pone un nombre con significado a la variable

total_hours = total_min / 60 # La duración total de varios álbumes en horas
total_hours
```

```
2.3666666666666667
```

En las celdas de arriba, añadimos la duración de tres álbumes en minutos y la almacenamos en `total_min`. Luego lo dividimos entre 60 para calcular la duración total, `total_hours`, en horas. También puedes hacerlo todo a la vez en una sola expresión, siempre que uses paréntesis para sumar la duración de los álbumes antes de dividir, como se muestra a continuación.

```
# Expresión complicada

```

```
3 total_hours = (43 + 42 + 57) / 60  # Horas totales en una sola expresión
4 total_hours
```

Si prefieres tener el total de horas como un número entero, puedes reemplazar la división de números flotantes con una división de números enteros. (p.e., `//`).

Ejercicio: Expresiones y Variables en Python

¿Cuál es el valor de `x` donde `x = 3 + 2 * 2`?

```
1 # Escriba tu código a continuación. No olvides presionar Shift+Enter para ejecutar la celda
2 x=3+2*2
```

Cuál es el valor de `y` donde `y = (3 + 2) * 2`?

```
1 # Escriba tu código a continuación. No olvides presionar Shift+Enter para ejecutar la celda
2 y=(3+2)*2
```

```
1 y
```

```
10
```

Cuál es el valor de `z` donde `z = x + y`?

```
1 # Escriba tu código a continuación. No olvides presionar Shift+Enter para ejecutar la celda
2 z=x+y
```

```
1 z
```

```
17
```

```
1
```

✓ 0 s completado a las 11:01