PY0101ES-1-1-Types

May 18, 2022

Python - ¡Escribiendo Tu Primer Código en Python!

¡Bienvenido! Este cuaderno te va a mostrar lo esencial del lenguaje de programación Python. Aunque la información que presentamos aquí es bastante elemental, es una base importante que te ayudará a leer y escribir código Python. Al finalizar este cuaderno habrás aprendido los fundamentos de Python, incluyendo cómo escribir comandos sencillos, comprendiendo algunos tipos básicos y sabiendo cómo realizar operaciones simples sobre ellos.

Índice de Contenidos

```
<ul>
    <li>
        <a href="https://#hello">Dile "Hola" al mundo en Python</a>
        <ul>
            <li><a href="https://version/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_
            <li><a href="https://comments/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm
            <li><a href="https://errors/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_co
            <li><a href="https://python_error/?utm_medium=Exinfluencer&utm_source=Exinfluencer&
            <li><a href="https://exercise/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm
        </ul>
    </li>
    <li>
        <a href="https://#types_objects">Tipos de objetos en Python</a>
        <ul>
            <li><a href="https://int/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_conte
            <li><a href="https://float/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_con
            <li><a href="https://convert/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_
            <li><a href="https://bool/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_cont
            <li><a href="https://exer_type/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm
        </ul>
    </li>
    <li>
        <a href="https://#https://exp/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_cont
        <ul>
            <li><a href="exp">Expresiones</a></li>
            <li><a href="https://exer_exp/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_
            <li><a href="https://var/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_conte
            <li><a href="https://exer_exp_var/?utm_medium=Exinfluencer&utm_source=Exinfluencer&
        </ul>
    </li>
```

```
</ul>
<p>
    Tiempo estimado requerido: <strong>25 min</strong>
</p>
```

Dile “Hola” al mundo en Python

Cuando se aprende un nuevo lenguaje de programación, se acostumbra a empezar con un ejemplo de “hola mundo”. A pesar de lo simple que es, esta línea de código asegurará que sepamos cómo escribir una cadena de texto en la salida y cómo ejecutar el código de las celdas de un cuaderno.

[1]:
```
# Intenta tu primer mensaje en Python

print('¡Hola Python!')
```

```
¡Hola Python!
```

Tras ejecutar la celda de arriba, deberías ver que Python imprime ¡Hola Python!. ¡Enhorabuena por ejecutar tu primer código en Python!

```
[Idea:] <code>print()</code> es una función. Pasaste la cadena <code>'¡Hola Python!'</code> co
```

¿Qué versión de Python estamos utilizando?

Hoy en día existen dos versiones comunes en uso del lenguaje de programación Python: 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 ofrecerán soporte para Python 2.

Como Python 3 es el futuro, en este curso lo vamos a utilizar de forma exclusiva. ¿Cómo sabemos que nuestro cuaderno se ejecuta bajo un tiempo de ejecución (runtime) 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. Intenta ejecutar el siguiente código:

[2]:
```
# Comprobar la Versión de Python

import sys
print(sys.version)
```

```
3.7.12 | packaged by conda-forge | (default, Oct 26 2021, 06:08:53)
[GCC 9.4.0]
```

```
[Idea:] <code>sys</code> es un módulo integrado que contiene muchos parámetros y funciones esp
```

Escribir comentarios en Python

Además de escribir código, ten en cuenta que siempre es una buena idea incluir comentarios en el. Ayudará a que los demás entiendan lo que intentabas lograr (la razón por la que escribiste un determinado fragmento de código). Esto no sólo sirve para que otras personas entiendan tu código, sino que también puede servir de recordatorio para ti cuando vuelvas a él semanas o meses después.

Para escribir comentarios en Python, usa el símbolo de almohadilla # antes de escribir el comentario. Cuando ejecutes el código, Python ignorará todo lo que vaya detrás de # en una determinada

línea.

[5]:
```
# Práctica de escritura de comentarios

print('¡Hola Python!') # Esta línea imprime una cadena de texto
#print('Hola')
```

```
¡Hola Python!
```

Tras ejecutar la celda de arriba, deberías observar que Esta línea imprime una cadena de texto no apareció en el resultado, porque era un comentario (y por lo tanto, ignorado por Python).

La segunda línea tampoco se ejecutó porque print('Hola') ¡también estaba precedida por el símbolo de almohadilla (#)! Como esto no es un comentario aclaratorio del programador, sino una línea de código real, podríamos decir que el programador comentó esa segunda línea de código.

Errores en Python

Todo el mundo comete errores. Para muchos tipos de errores, Python te dirá que has cometido una equivocación dándote un mensaje de error. Es importante leer los mensajes de error con cuidado para entender realmente dónde has cometido el error y cómo puedes corregirlo.

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

[6]:
```
# Imprimir cadena como mensaje de error

frint("¡Hola Python!")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/tmp/ipykernel_71/4103292370.py in <module>
      1 # Imprimir cadena como mensaje de error
      2
----> 3 frint("¡Hola Python!")

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

El mensaje de error te dice:

dónde se produjo el error (más útil en celdas de cuadernos o scripts de gran tamaño), y

qué tipo de error era (NameError)

Aquí, Python intentó ejecutar la función frint, pero no pudo determinar qué es frint ya que no es una función integrada y tampoco la hemos definido previamente nosotros.

Observarás que si cometemos un tipo de error diferente, olvidándonos de cerrar la cadena, obtendremos un error diferente (es decir, un SyntaxError (Error de Sintaxis)). Pruébalo aquí debajo:

[7]:
```
# Prueba a ver el mensaje de error

print("¡Hola Python!)
```

```
  File "/tmp/ipykernel_71/1786335092.py", line 3
    print("¡Hola Python!)
                         ^
SyntaxError: EOL while scanning string literal
```

¿Ve Python tu error antes de ejecutar el código?

Python es lo que se llama un lenguaje interpretado. Los lenguajes compilados examinan todo el programa en tiempo de compilación, y son capaces de advertirte sobre todo un conjunto de errores antes de la ejecución. Por contra, Python interpreta el script línea por línea mientras lo ejecuta. Python dejará de ejecutar el programa entero en cuanto encuentre un error (a menos que el error sea esperado y lo gestione el programador, un tema más avanzado que cubriremos más adelante en este curso).

Prueba a ejecutar el código de la celda de debajo y mira lo que pasa:

[8]:
```
# Impresión de cadena y error para ver el orden de ejecución

print("Esto se imprimirá")
frint("Esto provocará un error")
print("Esto NO se imprimirá")
```

```
Esto se imprimirá
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/tmp/ipykernel_71/238453031.py in <module>
      2
      3 print("Esto se imprimirá")
----> 4 frint("Esto provocará un error")
      5 print("Esto NO se imprimirá")

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

Ejercicio: Tu Primer Programa

Generaciones de programadores han comenzado sus carreras en la programación simplemente imprimiendo "¡Hola mundo!". Tú también seguirás sus pasos.

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

[9]:
```
# Escribe tu código a continuación y presiona Mayúsculas+Intro para ejecutar
print("¡Hola Mundo!")
```

```
¡Hola Mundo!
```

Haz doble clic en **aquí** para ver la solución.

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

[10]:
```
# Escribe tu código a continuación y presiona Mayúsculas+Intro para ejecutar
print("Hola Mundo") #escribe la frase, pero mantiene el comentario
```

```
Hola Mundo
```

Haz doble clic en **aquí** para ver la solución.

Tipos de objetos en Python

Python es un lenguaje orientado a objetos. Hay muchos tipos diferentes de objetos en Python. Comencemos con los tipos de objetos más comunes: cadenas, enteros y punto flotante. Cada vez que escribes palabras (texto) en Python, estás usando cadenas de caracteres (cadenas para abreviar). Los números más comunes, por otra parte, son los enteros (por ejemplo -1, 0, 100) y los de punto flotante, que representan números reales (por ejemplo 3.14, -42.0).

Las siguientes celdas de código presentan algunos ejemplos.

[ ]:
```
# Entero

11
```

[ ]:
```
# Punto flotante

2.14
```

[ ]:
```
# Cadena

"¡Hola, Python 101!"
```

Puedes hacer que Python te diga el tipo de una expresión mediante la función integrada type(). Observarás que Python se refiere a los enteros como int, a los números de punto flotante como float, y a las cadenas de caracteres como str.

[11]:
```
# Tipo de 12

type(12)
```

[11]:
```
int
```

[12]:
```
# Tipo de 2.14

type(2.14)
```

[12]:
```
float
```

[13]:
```
# Tipo de "¡Hola, Python 101!"

type("¡Hola, Python 101!")
```

[13]:
```
str
```

En la celda de código de debajo, utiliza la función type() para comprobar el tipo de objeto de 12.0.

[14]:
```
# Introduce el código a continuación. No te olvides de pulsar Mayúsculas+Intro␣
↪para ejecutar la celda
type(12.0)
```

[14]:
```
float
```

Haz doble clic en **aquí** para ver la solución.

Enteros

Aquí hay algunos ejemplos de números enteros. Los números enteros pueden ser números negativos o positivos:

Podemos verificar que este es el caso mediante el uso, lo has adivinado, de la función type():

[15]:
```
# Imprime el tipo de -1

type(-1)
```

[15]:
```
int
```

[16]:
```
# Imprime el tipo de 4

type(4)
```

[16]:
```
int
```

[17]:
```
# Imprime el tipo de 0

type(0)
```

[17]:
```
int
```

Punto Flotante

Los números de punto flotante representan números reales; son un superconjunto de números enteros pero también incluyen "números con decimales". Existen algunas limitaciones cuando se trata de que las máquinas representen números reales, pero los números de punto flotante son una buena representación para la mayoría de los casos. Puedes obtener más información sobre las características específicas de los números de punto flotante en tu entorno de ejecución, comprobando el valor de sys.float_info. Esto también te dirá cuál es el número más grande y más pequeño que se puede representar con ellos.

Una vez más, puedes probar algunos ejemplos con la función type():

[18]:
```
# Imprime el tipo de 1.0

type(1.0) # Observa que 1 es un int, y 1.0 es un float
```

[18]:
```
float
```

[19]:
```
# Imprime el tipo de 0.5

type(0.5)
```

[19]:
```
float
```

[20]:
```
# Imprime el tipo de 0.56

type(0.56)
```

[20]:
```
float
```

[21]:
```
# Ajustes del sistema para el tipo float

sys.float_info
```

[21]:
```
sys.float_info(max=1.7976931348623157e+308, max_exp=1024, max_10_exp=308,
min=2.2250738585072014e-308, min_exp=-1021, min_10_exp=-307, dig=15,
mant_dig=53, epsilon=2.220446049250313e-16, radix=2, rounds=1)
```

Convertir de un tipo de objeto a otro tipo de objeto diferente

Puedes cambiar el tipo del objeto en Python; esto se llama conversión de tipos, o typecasting. Por ejemplo, puedes convertir un entero en un punto flotante (por ejemplo 2 a 2.0).

Vamos a probarlo:

[22]:
```
# Verifica que esto es un entero

type(2)
```

[22]:
```
int
```

Convertir enteros en punto flotante

Vamos a convertir el entero 2 en punto flotante:

[23]:
```
# Convierte 2 a punto flotante

float(2)
```

[23]:
```
2.0
```

[24]:
```
# Convierte el entero 2 a punto flotante y comprobar su tipo

type(float(2))
```

[24]:
```
float
```

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

[25]:
```
# Convertir 1.1 a entero resultará en una pérdida de información

int(1.1)
```

[25]:
```
1
```

Convertir cadenas a enteros o punto flotante

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 mediante int():

[26]:
```
# Convierte una cadena en un entero

int('1')
```

[26]:
```
1
```

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

[27]:
```
# Convierte una cadena en un entero con error

int('1 o 2 personas')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/tmp/ipykernel_71/2295571780.py in <module>
      1 # Convierte una cadena en un entero con error
      2
----> 3 int('1 o 2 personas')

ValueError: invalid literal for int() with base 10: '1 o 2 personas'
```

También puedes convertir cadenas que contienen números de punto flotante en objetos de punto flotante:

[28]:
```
# Convierte la cadena "1.2" en un punto flotante

float('1.2')
```

[28]:
```
1.2
```

[Idea:] Observa que las cadenas pueden representarse con comillas simples (<code>'1.2'</code>)

Convertir números en cadenas

Si podemos convertir las cadenas en números, es lógico suponer que podemos convertir los números en cadenas, ¿verdad?

[29]:
```
# Convierte un entero en una cadena

str(1)
```

[29]:
```
'1'
```

Y no hay razón por la que no podamos convertir los números de punto flotante también en cadenas:

[30]:
```
# Convierte un punto flotante en una cadena

str(1.2)
```

[30]:
```
'1.2'
```

Tipo de datos booleano

El Booleano es otro tipo importante en Python. Un objeto de tipo Booleano puede tener uno de entre dos valores: True (verdadero) o False (falso):

[31]:
```
# Valor verdadero

True
```

[31]:
```
True
```

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

[32]:
```
# Valor falso

False
```

[32]:
```
False
```

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

[33]:
```
# Tipo de True

type(True)
```

[33]:
```
bool
```

[34]:
```
# Tipo de False

type(False)
```

[34]:
```
bool
```

Podemos convertir objetos booleanos en otros tipos de datos. Si convertimos un booleano con un valor de True a un entero o punto flotante obtendremos un uno. Si convertimos un booleano con un valor de False en un entero o en un punto flotante, obtendremos un cero. Del mismo modo, si convertimos un 1 en Booleano, obtendremos True. Y si convertimos un 0 a Booleano, obtendremos False. Vamos a probarlo:

[35]:
```
# Convierte True a int

int(True)
```

[35]:
```
1
```

[36]:
```
# Convierte 1 a booleano

bool(1)
```

[36]:
```
True
```

[37]:
```
# Convierte 0 a booleano

bool(0)
```

[37]:
```
False
```

[38]:
```
# Convierte True a punto flotante

float(True)
```

[38]:
```
1.0
```

Ejercicio: Tipos

¿Cuál es el tipo de dato del resultado de: 6 / 2?

[41]:
```
# Introduce el código a continuación. No te olvides de pulsar Mayúsculas+Intro␣
↪para ejecutar la celda
```

```
type(6/2)
```

[41]:
```
float
```

Haz doble clic en **aquí** para ver la solución.

¿Cuál es el tipo del resultado de: 6 // 2? (Observa la doble barra inclinada //.)

[40]:
```
# Introduce el código a continuación. No te olvides de pulsar Mayúsculas+Intro␣
↪para ejecutar la celda
type(6//2)
```

[40]:
```
int
```

Haz doble clic en **aquí** para ver la solución.

Expresiones y Variables

Expresiones

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

[ ]:
```
# Expresión de operación de suma

43 + 60 + 16 + 41
```

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

[ ]:
```
# Expresión de operación de resta

50 - 60
```

Podemos multiplicar utilizando un asterisco:

[ ]:
```
# Expresión de operación de multiplicación

5 * 5
```

También podemos realizar la división con la barra inclinada:

[ ]:
```
# Expresión de operación de división

25 / 5
```

[ ]:
```
# Expresión de operación de división

25 / 6
```

Como se ve en el cuestionario anterior, podemos usar la doble barra inclinada para la división entera, donde el resultado se redondea al entero más cercano:

[ ]:
```
# Expresión de operación de división entera

25 // 5
```

[ ]:
```
# Expresión de operación de división entera

25 // 6
```

Ejercicio: Expresión

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

[45]:
```
# Introduce el código a continuación. No te olvides de pulsar Mayúsculas+Intro␣
↪para ejecutar la
160/60
```

[45]:
```
2.6666666666666665
```

Haz doble clic en **aquí** para ver la solución.

Python sigue las convenciones matemáticas ampliamente aceptadas al evaluar las expresiones matemáticas. En el siguiente ejemplo, Python suma 30 al resultado de la multiplicación (es decir, 120).

[ ]:
```
# Expresión matemática

30 + 2 * 60
```

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

[ ]:
```
# Expresión matemática

(30 + 2) * 60
```

Variables

Como con la mayoría de los lenguajes de programación, podemos almacenar valores en variablespara poder usarlos más tarde. Por ejemplo:

[ ]:
```
# Almacena el valor en una variable

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

Para ver el valor de x en un Cuaderno, podemos simplemente colocarlo en la última línea de una celda:

[ ]:
```
# Imprime el valor de la variable

x
```

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

[ ]:
```
# Utiliza otra variable para almacenar el resultado de la operación entre␣
↪variable y valor

y = x / 60
y
```

Si guardamos un valor en una variable ya existente, el nuevo valor sobrescribirá el valor anterior:

[ ]:
```
# Sobrescribe la variable con un nuevo valor

x = x / 60
x
```

Es una buena práctica utilizar nombres de variables con significado, para que tanto tú como los demás puedan leer el código y entenderlo más fácilmente:

[ ]:
```
# Da un nombre a las variables que tenga un significado

total_min = 43 + 42 + 57 # Duración total de los álbumes en minutos
total_min
```

[ ]:
```
# Da un nombre a las variables que tenga un significado

total_horas = total_min / 60 # Duración total de los álbumes en horas
total_horas
```

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

[ ]:
```
# Expresión compleja

total_horas = (43 + 42 + 57) / 60  # Total de horas en una sola expresión
total_horas
```

Si prefieres tener el total de horas como un número entero, puedes, por supuesto, reemplazar la división en punto flotante por la división entera (es decir, //).

Ejercicio: Expresión y Variables en Python

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

[46]:
```
# Introduce el código a continuación. No te olvides de pulsar Mayúsculas+Intro␣
↪para ejecutar la celda
7
```

Haz doble clic en **aquí** para ver la solución.

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

[ ]:
```
# Introduce el código a continuación. No te olvides de pulsar Mayúsculas+Intro␣
↪para ejecutar la celda
10
```

Haz doble clic en **aquí** para ver la solución.

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

[ ]:
```
# Introduce el código a continuación. No te olvides de pulsar Mayúsculas+Intro␣
↪para ejecutar la celda
17
```

Haz doble clic en **aquí** para ver la solución.

¡El último ejercicio!

Enhorabuena, has completado la primera lección y el laboratorio práctico de Python. Sin embargo, hay una cosa más que debes hacer. La comunidad de Ciencias de los Datos anima a compartir los trabajos. La mejor manera de compartir y mostrar tu trabajo es compartirlo en GitHub. Compartiendo tu cuaderno en GitHub no sólo estarás forjando una reputación entre tus colegas que son científicos de datos, sino que también podrás mostrarlo cuando busques empleo. Aunque este haya sido tu primer trabajo, nunca es demasiado pronto para empezar a crear buenos hábitos. Así que, por favor, lee y sigue este artículo para aprender cómo compartir tu trabajo.

¡Consigue IBM Watson Studio gratis!

```
<p><a href="https://cocl.us/PY0101EN_edx_add_bbottom?utm_medium=Exinfluencer&utm_source=Exinflu
```

Sobre los autores:

Joseph Santarcangelo es un Científico de Datos en IBM, y tiene un doctorado en Ingeniería Eléctrica. Su investigación se centró en el uso de Machine Learning, Procesamiento de Señal y Visión Artificial para determinar cómo los videos impactan en la cognición humana. Joseph ha estado trabajando para IBM desde que concluyó su doctorado.

Otros colaboradores: Mavis Zhou