Resumen Python 3

Víctor Mardones Bravo

Febrero de 2021

Índice general

Capítulo 1

Introducción a Python

1.1. ¿Qué es Python?

Python es un lenguaje de programación de alto nivel, con aplicaciones en numerosas áreas, incluyendo programación web, scripting, computación científica e inteligencia artificial.

Es muy popular y usado por organizaciones como Google, la NASA, la CIA y Disney.

No hay limitaciones en lo que se puede construir usando Python. Esto incluye aplicaciones autónomas, aplicaciones web, juegos, ciencia de datos, modelos de machine learning y mucho más.

Dato curioso: Según el creador Guido van Rossum, el nombre de Python viene de la serie de comedia británica "El Circo Volador de Monty Python".

1.2. El Zen de Python

El Zen de Python es una colección de 19 "principios" para escribir programas de computadores que influenciaron el diseño y representan la filosofía de este lenguaje de programación.

El Zen de Python se muestra en pantalla la primera vez que se ejecute la siguiente linea.

```
1 import this
```

Después se mostrará el siguiente texto.

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
```

```
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

1.3. Hola mundo

Para mostrar el texto "Hola mundo" en pantalla se puede usar la función print().

```
1 print('Hola mundo')
```

Cada declaración de impresión print() genera texto en una nueva línea.

```
1 print('Hola')
2 print('Mundo')
```

Capítulo 2

Conceptos básicos

2.1. Comentarios

Los comentarios son anotaciones en el código utilizadas para hacerlo más fácil de entender. No afectan la ejecución del código.

En Python, los comentarios comienzan con el símbolo #. Todo el texto luego de este # (dentro de la misma línea) es ignorado.

```
# Este es un comentario
```

Python no soporta comentarios multilínea, al contrario de otros lenguajes de programación.

A lo largo de este resumen, se usarán comentarios para mostrar la salida de algunas funciones, cuando sea posible.

2.2. Operaciones aritméticas

Python tiene la capacidad de realizar cálculos. Los operadores +, -, * y / representan suma, resta, multiplicación y división, respectivamente.

```
print(1 + 1)  # 2
print(5 - 2)  # 3
print(2 * 2)  # 4
print(4 / 4)  # 1.0
```

Los espacios entre los signos y los números son opcionales, pero hacen que el código sea más fácil de leer.

2.3. La regla PEMDAS

Las operaciones en Python siguen el orden dado por la regla PEMDAS:

1. Paréntesis

2. Exponentes
3. Multiplicación y división
4. Adición y Sustracción

2.4. Paréntesis

Se pueden usar paréntesis () para agrupar operaciones y hacer que estas se realicen primero, siguiendo la regla PEMDAS.

```
print((2 + 3) * 4)
# (2 + 3) * 4
# 5 * 4
# 20
```

2.5. Floats

Para representar números racionales o que no son enteros, se usa el tipo de dato float o punto flotante. Se pueden crear directamente ingresando un número con un punto decimal, o como resultado de una división.

```
print(0.25)  # 0.25
print(3 / 4)  # 0.75
```

Se debe tener en cuenta que los computadores no pueden almacenar perfectamente el valor de los floats, lo cual a menudo conduce a errores.

```
print(0.1 + 0.2)  # 30000000000000004
```

El error mostrado arriba es un error clásico de la aritmética de punto flotante. Incluso tiene su propio sitio web.

Al trabajar con floats, no es necesario escribir un 0 a la izquierda del punto decimal.

```
print(.42)  # 0.42
```

Esta notación se asemeja a decir "punto cinco" en vez de "cero punto cinco".

El resultado de cualquier operación entre floats o entre un float y un entero siempre dará como resultado un float. La división entre enteros también da como resultado un float.

2.6. Exponenciación

Otra operación soportada es la exponenciación, que es la elevación de un número a la potencia de otro. Esto se realiza usando el operador **.

Ejemplo equivalente a $2^5 = 32$.

```
print(2**5)  # 32
```

2.7. Cociente y resto

La división entera se realiza usando el operador //, donde el resultado es la parte entera que queda al realizar la división, también conocida como cociente.

La división entera retorna un entero en vez de un float.

```
print(7 // 2)  # 3
```

Para obtener el resto al realizar una división entera, se debe usar el operador módulo %.

```
print(7 % 2)  # 1
```

Esta operación es equivalente a $7 \mod 2$ en aritmética modular.

Este operador viene de la aritmética modular, y uno de sus usos más comunes es para saber si un número es múltiplo de otro. Esto se hace revisando si el módulo al dividirlo por ese otro número es 0.

```
print(10 % 2)  # 0, 10 es múltiplo de 2
print(15 % 3)  # 0, 15 es múltiplo de 3
print(16 % 7)  # 2, 16 no es múltiplo de 7
```

El caso particular módulo 2 también puede usarse para saber si un número es par o no.

Capítulo 3

Cadenas de texto

3.1. Strings o cadenas de caracteres

Las cadenas de caracteres se crean introduciendo el texto entre comillas simples ′′ o dobles ′′′′.

```
print('texto')
print("texto")
```

Un string debe empezar y terminar con comillas del mismo tipo, no se permiten comillas mixtas.

```
# Strings no válidos
print('texto")
print("texto')
```

3.2. Cadena vacía

A veces es necesario inicializa un string, pero sin agregarle información. Una cadena vacía es definida como ′′ o ′′′′.

```
cadena_vacia1 = ''
cadena_vacia2 = ""
```

Estos strings vacíos se inicializan en variables, lo cual se verá en el capítulo siguiente.

3.3. Caracteres especiales

Algunos caracteres no se pueden incluir directamente en una cadena. Para esos casos, se debe incluir la barra diagonal inversa \antes de ellos.

```
print('\'')  # '
print('\"')  # "
print('\\')  # \
```

Los caracteres ′, ″ y \son especiales, porque normalmente cumplen funciones especiales dentro de strings.

Si el string se define entre comillas dobles, no es necesario poner ′ para ingresar comillas simples dentro de él, y viceversa.

```
print("This isn't spanish")
print('Hola "mundo"')
```

3.4. Secuencias de escape

Las secuencias de escape también se pueden incluir usando el símbolo \dentro de cadenas de texto. Su origen viene de las secuencias de escape usadas en las máquinas de escribir.

Algunas de las secuencias de escape más usadas son:

- Nueva línea (new line): Avanza una línea hacia adelante (salto de línea) y deja el cursor al principio de esta línea (retorno de carro). Representado por \n.

```
print('Hola\nMundo')
# Hola
# Mundo
```

  Cualquier caracter después de \n queda en la línea siguiente.

- Tabulador horizontal (horizontal tab): Añade un salto de tabulador horizontal. Representado por \t.

```
print('Hola\tmundo')
# Hola    mundo
```

  El salto de tabulador avanza hasta el siguiente “tab stop” de la misma línea.

- Retorno de carro (carriage return): Mueve el “carro” (cursor) al principio de la línea actual, eliminando todos los caracteres de esa línea. Representado por \r.

```
print('12345\r67')
# 67
```

- Retroceso (backspace): Borra el último carácter y mueve el cursor al carácter anterior. Representado por \b.

```
print('123\b45')
# 1245
```

Otras secuencias de escape que cada vez se usan menos son:

- Tabulador vertical (vertical tab): Añade un salto de tabulador vertical. Representado por \v.

```
print('Hola\vmundo')
```

  La tabulación vertical avanza hasta la siguiente línea que sea una "tab stop".

- Salto de página (form feed): Baja a la próxima "página". Representado por \f.

```
print('Hola\fmundo')
```

  Algunos programadores los usaban para separar distintas secciones de código en "páginas".

3.5. Caracteres Unicode

Las barras diagonales inversas también se pueden usar para escribir caracteres Unicode arbitrarios. Se escriben como \u seguido del código del carácter Unicode (en hexadecimal).

Los códigos Unicode se aceptan sin importar que tengan mayúsculas o minúsculas.

```
print('\u00f1')  # ñ
print('\u00F1')  # ñ
```

El sitio web de Unicode contiene más información sobre estos caracteres y sobre este estándar. Este sitio web tiene una tabla con los códigos.

3.6. Strings multilínea

Este es un tipo especial de string, que se escribe entre comillas triples ''' o """, y que reconoce los saltos de línea sin necesidad de usar la secuencia \n.

```
print("""Esta es
una cadena de
caracteres
multilínea""")
# Esta es
# una cadena de
# caracteres
# multilínea
```

3.7. Comentarios multilínea

Los comentarios multilínea no existen formalmente en Python, pero se puede hacer algo parecido usando

3.8. Concatenación de strings

3.9. Multiplicación de strings

3.10. Opciones del método print()

Capítulo 4

Variables

Capítulo 5

Declaraciones if

Capítulo 6

Listas

Capítulo 7

Bucles

Capítulo 8

Funciones

Capítulo 9

Módulos y la biblioteca estándar

Capítulo 10

El módulo math

Capítulo 11

El módulo random

Capítulo 12

Manejo de excepciones

Capítulo 13

Pruebas unitarias

Capítulo 14

Manejo de archivos

Capítulo 15

Módulos time y datetime

Capítulo 16

Iterables

16.15. Función enumerate()

Capítulo 17

Programación funcional

Capítulo 18

Conjuntos y estructuras de datos

Capítulo 19

El módulo itertools

Capítulo 20

Programación orientada a objetos

20.15. Encapsulamiento

Capítulo 21

Expresiones regulares

Capítulo 22

Empaquetamiento

Capítulo 23

Interfaz gráfica

Capítulo 24

Algoritmos de ordenamiento

Capítulo 25

Algoritmos de búsqueda

Capítulo 26

Algoritmos de matrices

Capítulo 27

Implementación de estructuras de datos

Capítulo 28

La librería NumPy