Resumen Python 3

Víctor Mardones Bravo

Febrero de 2021

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.

```
1 import this
```

Figura 1: El Zen de Python se muestra en pantalla la primera vez que se ejecute esta línea.

```
1  The Zen of Python, by Tim Peters
2
3  Beautiful is better than ugly.
4  Explicit is better than implicit.
5  Simple is better than complex.
6  Complex is better than complicated.
7  Flat is better than nested.
8  Sparse is better than dense.
9  Readability counts.
10 Special cases aren't special enough to break the rules.
11 Although practicality beats purity.
12 Errors should never pass silently.
13 Unless explicitly silenced.
14 In the face of ambiguity, refuse the temptation to guess.
15 There should be one-- and preferably only one --obvious way to do it.
16 Although that way may not be obvious at first unless you're Dutch.
17 Now is better than never.
18 Although never is often better than *right* now.
19 If the implementation is hard to explain, it's a bad idea.
20 If the implementation is easy to explain, it may be a good idea.
21 Namespaces are one honking great idea -- let's do more of those!
```

Figura 2: El Zen de Python, escrito por Tim Peters.

1.3. Hola mundo

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

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

Figura 3: Hola mundo, el clásico primer programa en cualquier lenguaje de programación.

```
print('Hola')
print('Mundo')
```

Figura 4: Cada declaración de impresión print() genera texto en una nueva línea.

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
```

Figura 5: Ejemplo de un comentario en Python.

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
```

Figura 6: Adición, sustracción, multiplicación y división en Python.

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)  # 20
```

Figura 7: Uso de paréntesis para cambiar el orden de las operaciones.

2.5. Floats

Para representar números 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
```

Figura 8: Los floats representan números racionales.

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
```

Figura 9: Un error clásico de la aritmética de punto flotante.

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

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

Figura 10: 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 **.

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

Figura 11: Ejemplo equivalente a $2^5 = 32$.

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.

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

Figura 12: La división entera retorna un entero en vez de un float.

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

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

Figura 13: Esta operación es equivalente a 7 mód 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
```

Figura 14: El caso particular módulo 2 también puede usarse para saber si un número es par o no.

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")
```

Figura 15: Una cadena de caracteres válida.

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

Figura 16: Un string debe empezar y terminar con comillas del mismo tipo, no se permiten comillas mixtas.

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 = ""
```

Figura 17: 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('\\')  # \
```

Figura 18: Los caracteres ', ɛ
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"')
```

Figura 19: Los caracteres ', ɛ
son especiales, porque normalmente cumplen funciones especiales dentro de strings.