Resumen Python 3

Víctor Mardones Bravo

Índice general

Capítulo 1

Números

1.1. 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.

Sintaxis:

```
#contenido
```

Ejemplo:

```
# Este es un comentario

print("# Este no es un comentario")
```

Salida:

```
# Este no es un comentario
```

La mayoría de editores de código marcan los comentarios con su propio color, distinto al del resto de código.

Ejemplo:

```
# Este es un comentario
print('Hola mundo') # Este es otro comentario
# Otro comentario más
```

Salida:

```
Hola mundo
```

Python no tiene comentarios multilínea para fines generales como lo tienen otros lenguajes de programación tales como C. Para comentar varias líneas de código se debe antener un comentario a cada línea.

Ejemplo:

```
# Python no soporta
# comentarios multilínea.
# Esta es la
# única opción
# en esos casos.
```

Los comentarios no tienen fines meramente explicativos. También se pueden usar para esconder líneas de código, con el propósito de hacer más fácil revisar que funciona bien.

Ejemplo:

```
#print("Hola mundo")
print("Probando código...")
```

Salida:

```
Probando código...
```

Se debe recordar borrar los comentarios de este tipo después de terminar las pruebas de código, ya que no son de utilidad y hay herramientas mejores, como Git (un sistema de control de versiones), para mantener guardado código antiguo en caso de que algún error ocurra.

Por último, se debe recordar no abusar del uso de comentarios. No es conveniente llenar el código de comentarios o comentar cosas que son demasiado obvias.

Ejemplo:

```
# Ejemplo de comentario innecesario:

# Muestra el texto "hola mundo" en pantalla
print('hola mundo')
```

1.2. Números enteros

Los números enteros (integer) son el tipo de dato más básico en muchos lenguajes de programación, y Python no es la excepción.

Sintaxis:

```
numero
```

Para mostrar un número en pantalla, sólo basta con usar la función `print()` y entregarle el número. El número debe escribirse sin usar comillas y Python inferirá su tipo automáticamente.

Ejemplo:

```
# positivo
print(5)
print(10)

# neutro
print(0)

# negativo
print(-10)
```

Salida:

```
5
10
0
-10
```

A diferencia de otros lenguajes de programación, los números en Python 3 son de largo arbitrario, en otras palabras, soportan cualquier cantidad de dígitos. Esto hace que los cálculos con números grandes sean mucho más convenientes, pero reduce el rendimiento.

Ejemplo:

```
print(99999999999999999999999999999)
```

Salida:

```
99999999999999999999999999999
```

1.3. Operaciones aritméticas

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

Sintaxis:

```
sumando + sumando
```

Sintaxis:

```
minuendo - sustraendo
```

Sintaxis:

```
factor * factor
```

Sintaxis:

```
dividendo / divisor
```

Ejemplo:

```
print(1 + 1)  # suma
print(5 - 2)  # resta
print(2 * 2)  # multiplicación
print(4 / 4)  # división
```

Salida:

```
2
3
4
1.0
```

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

El resultado de una división es un número decimal. Esto lo convierte al tipo de dato float, el cual se verá más tarde en este capítulo.

Ejemplo:

```
print(5 / 2)
```

Salida:

```
2.5
```

Todas las operaciones se pueden combinar entre sí como si el intérprete de Python fuera una calculadora.

Ejemplo:

```
print(1 + 2 - 3 + 10 - 7)
```

Salida:

```
3
```

1.4. La regla PEMDAS

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

1) Paréntesis `()`

2) Exponentes `**`

3) Multiplicación `*` y **División** `/` (de izquierda a derecha)

4) Adición `+` y **Sustracción** `-` (de izquierda a derecha)

A continuación se muestra un ejemplo detallando el orden en el que se procesan las operaciones.

Ejemplo:

```
print(2*7 - 1 + 4/2*3)
# 2*7 - 1 + 4/2*3
# 14 - 1 + 6.0
# 19.0
```

Salida:

```
19.0
```

En realidad, la regla PEMDAS es parte de una jerarquía de operadores mucho más grande, que incluye todos los operadores que se pueden usar en Pyhon. Esta jerarquía se verá más tarde.

1.5. Paréntesis

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

Sintaxis:

```
operaciones (operaciones que se realizan primero) operaciones
```

Ejemplo:

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

Salida:

```
20
```

1.6. 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.

Sintaxis:

```
parte entera.parte decimal
```

Ejemplo:

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

Salida:

```
0.25
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.

Ejemplo:

```
print(0.1 + 0.2)
```

Salida:

```
0.30000000000000004
```

El error mostrado arriba es un error clásico de la aritmética de punto flotante. Es un error tan conocido que incluso tiene su propio sitio web.

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

```
print(.42)
```

Salida:

```
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.
Ejemplo:

```
# entero y float
print(1 + 2.0)
print(15.0 - 7)
print(3 * 4.0)
print(10.0 / 2)

# enteros
print(10 / 2)
```

Salida:

```
3.0
8.0
12.0
5.0
5.0
```

Las operaciones entre floats y enteros son posibles porque Python convierte los enteros en floats silenciosamente al momento de realizarlas.

1.7. 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 `**`.
Sintaxis:

```
base ** exponente
```

Ejemplo equivalente a $2^5 = 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 = 32$.
Ejemplo:

```
print(2**5)
```

Salida:

```
32
```

Las exponenciaciones se pueden encadenar cuantas veces se desee.
Ejemplos equivalentes a $2^{3^2} = 512$ y 10^{100} (un gúgol).
Ejemplo:

```
print(2**3**2)
print(10**100)
```

```
512
10000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪  00000000000000000000
```

Se debe tener cuidado con los paréntesis al encadenar potencias.
Ejemplos equivalentes a $2^{3^2} = 512$ y $(2^3)^2 = 64$.
Ejemplo:

```
print(2**3**2)

print((2**3)**2)

print(2**(3**2))
```

Salida:

```
512
64
512
```

La exponenciación también se puede hacer con floats, donde se puede usar para obtener raíces. El resultado será un float.
Sintaxis:

```
radicando ** (1 / indice)
```

Ejemplos equivalentes a $\sqrt{9} = 3$ y $\sqrt[3]{8} = 2$.
Ejemplo:

```
print(9**(1 / 2))
print(8**(1 / 3))
```

Salida:

```
3.0
2.0
```

Se debe tener en cuenta que el resultado será un float.

1.8. 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.

Sintaxis:

```
dividendo // divisor
```

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

Ejemplo:

```
print(7 // 2)
```

Salida:

```
3
```

También se puede usar la división entera con floats, lo cual dará como resultado otro float.

Ejemplo:

```
print(6.25 // 0.5)
```

Salida:

```
12.0
```

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

Sintaxis:

```
dividendo % divisor
```

Ejemplo:

```
print(7 % 2)
```

Salida:

```
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.

Ejemplo:

```
print(10 % 2)
print(15 % 3)
print(16 % 7)
```

Salida:

```
0
0
2
```

En el caso mostrado anteriormente, se infiere que 10 es múltiplo de 2, que 15 es múltiplo de 3 y que 16 no es múltiplo de 7. El caso particular módulo 2 también puede usarse para saber si un número es par o no.

El operador módulo `%` también puede usarse con floats.

Ejemplo:

```
print(6.25 % 0.5)
print(2.5 % 1.25)
```

Salida:

```
0.25
0.0
```