Python

Guía para ser un Pythonista

Juan José Lozano Gómez – j2logo.com

5 Operadores

Índice de contenidos

Introducción 5
Operadores de comparación 6
Consideraciones sobre los operadores de comparación 7
Operadores lógicos o booleanos 8
Operadores aritméticos 10
Operadores a nivel de bits 11
Operadores de asignación 13
Operadores de pertenencia 14
Operadores de identidad 15
Prioridad de los operadores en Python 16
Operador de concatenación de cadenas de caracteres 17

Introducción

En el Tema 2, Características básicas del lenguaje, ya hablé sobre los operadores en Python. Como te indiqué, los operadores son símbolos reservados por el propio lenguaje que se utilizan para llevar a cabo operaciones sobre uno, dos o más elementos llamados operandos.

Los operandos pueden ser variables, literales, el valor devuelto por una expresión o el valor devuelto por una función.

El ejemplo más típico de operador que siempre viene a la mente es el operador suma, `+`, que se utiliza para obtener la suma aritmética de dos valores.

En este tema entraremos más en detalle sobre los distintos tipos de operadores disponibles en Python: operadores de comparación, operadores lógicos o booleanos, operadores aritméticos, operadores a nivel de bits, etc.

¡Comenzamos!

Operadores de comparación

Los operadores de comparación se utilizan, como su nombre indica, para comparar dos o más valores. El resultado de una expresión con estos operadores es siempre `True` o `False`.

Operador	Significado y valor devuelto
>	Mayor que. `True` si el operando de la izquierda es estrictamente mayor que el de la derecha; `False` en caso contrario.
>=	Mayor o igual que. `True` si el operando de la izquierda es mayor o igual que el de la derecha; `False` en caso contrario.
<	Menor que. `True` si el operando de la izquierda es estrictamente menor que el de la derecha; `False` en caso contrario.
<=	Menor o igual que. `True` si el operando de la izquierda es menor o igual que el de la derecha; `False` en caso contrario.
==	Igual. `True` si el operando de la izquierda es igual que el de la derecha; `False` en caso contrario.
!=	Distinto. True si los operandos son distintos; `False` en caso contrario.

Ejemplos:

```
>>> x = 9
>>> y = 1
>>> x < y
False
>>> x > y
True
>>> x == y
False
```

Consideraciones sobre los operadores de comparación

Los objetos de diferentes tipos, excepto los tipos numéricos, nunca se comparan igual. El operador `==` siempre está definido, pero para algunos tipos de objetos (por ejemplo, objetos de clase) es equivalente a `is`.

Las instancias no idénticas de una clase normalmente se comparan como no iguales a menos que la clase defina el método `__eq__()`.

Las instancias de una clase no se pueden ordenar con respecto a otras instancias de la misma clase u otros tipos de objeto, a menos que la clase defina los métodos `__lt__()` y `__eq__()`.

Además, los operadores de comparación se pueden concatenar:

```
>>> x = 9
>>> 1 < x < 20
True
```

Operadores lógicos o booleanos

A la hora de operar con valores booleanos, tenemos a nuestra disposición los operadores `and`, `or` y `not`.

IMPORTANTE: Las expresiones con and, or y not realmente no devuelven True o False, sino que devuelven uno de los operandos como veremos en el cuadro de abajo.

A continuación, te muestro cómo funcionan los operadores booleanos (en orden de preferencia ascendente):

Operación	Resultado	Descripción
a or b	Si `a` se evalúa a falso, entonces devuelve `b`, si no devuelve `a`	Solo se evalúa el segundo operando si el primero es falso
a and b	Si `a` se evalúa a falso, entonces devuelve `a`, si no devuelve `b`	Solo se evalúa el segundo operando si el primero es verdadero
not a	Si `a` se evalúa a falso, entonces devuelve `True`, si no devuelve `False`	Tiene menos prioridad que otros operadores no booleanos

Ejemplos:

```
>>> x = True
>>> y = False
>>> x or y
True
>>> x and y
False
>>> not x
False
>>> x = 0
>>> y = 10
>>> x or y
10
>>> x and y
0
>>> not x
True
```

Operadores aritméticos

En cuanto a los operadores aritméticos, estos permiten realizar las diferentes operaciones aritméticas del álgebra: *suma*, *resta*, *producto*, *división*, ...

El listado completo es el siguiente:

Operador	Descripción
+	Suma dos operandos.
–	Resta al operando de la izquierda el valor del operando de la derecha. Utilizado sobre un único operando, le cambia el signo.
*	Producto/Multiplicación de dos operandos.
/	Divide el operando de la izquierda por el de la derecha (el resultado siempre es un `float`).
%	Operador módulo. Obtiene el resto de dividir el operando de la izquierda por el de la derecha.
//	Obtiene el cociente entero de dividir el operando de la izquierda por el de la derecha.
**	Potencia. El resultado es el operando de la izquierda elevado a la potencia del operando de la derecha.

Ejemplos:

```
>>> x = 7
>>> y = 2
>>> x + y  # Suma
9
>>> x - y  # Resta
5
>>> x * y  # Producto
14
>>> x / y  # División
3.5
>>> x % y  # Resto
1
>>> x // y  # Cociente
3
>>> x ** y  # Potencia
49
```

Operadores a nivel de bits

Los operadores a nivel de bits actúan sobre los operandos como si estos fueran una cadena de dígitos binarios. Como su nombre indica, actúan sobre los operandos bit a bit.

Son los siguientes:

Operación	Descripción
x | y	or bit a bit de x e y.
x ^ y	or exclusivo bit a bit de x e y.
x & y	and bit a bit de x e y.
x << n	Desplaza x n bits a la izquierda.
x >> n	Desplaza x n bits a la derecha.
~x	not x. Obtiene los bits de x invertidos.

Supongamos que tenemos el entero 2 (en bits es 00010) y el entero 7 (00111). El resultado de aplicar las operaciones anteriores es:

```
>>> x = 2
>>> y = 7
>>> x | y
7
>>> x ^ y
5
>>> x & y
2
>>> x << 1
4
>>> x >> 1
1
>>> ~x
-3
```

Operadores de asignación

El operador de asignación se utiliza para asignar un valor a una variable (identificar un objeto con un nombre). Como ya sabrás, este operador es el signo `=`.

Además del operador de asignación, existen otros operadores de asignación compuestos que realizan una operación básica sobre la variable a la que se le asigna el valor.

Por ejemplo, `x += 1` es lo mismo que `x = x + 1`. Los operadores compuestos realizan la operación que hay antes del signo igual, tomando como operandos la propia variable y el valor a la derecha del signo igual.

A continuación, te muestro la lista de todos los operadores de asignación compuestos:

Operador	Ejemplo	Equivalencia
+=	x += 2	x = x + 2
-=	x -= 2	x = x – 2
*=	x *= 2	x = x * 2
/=	x /= 2	x = x / 2
%=	x %= 2	x = x % 2
//=	x //= 2	x = x // 2
**=	x **= 2	x = x ** 2
&=	x &= 2	x = x & 2
|=	x |= 2	x = x | 2

^=	x ^= 2	x = x ^ 2
>>=	x >>= 2	x = x >> 2
<<=	x <<= 2	x = x << 2

Operadores de pertenencia

Los operadores de pertenencia se utilizan para comprobar si un valor u objeto se encuentra en un objeto contenedor (`list`, `tuple`, `dict`, `set` o `str`).

Todavía no hemos visto estos tipos, pero son operadores muy utilizados.

Operador	Descripción
in	Devuelve `True` si se encuentra el valor u objeto; `False` en caso contrario.
not in	Devuelve `True` si no se encuentra el valor u objeto; `False` en caso contrario.

Ejemplos:

```
>>> lista = [1, 3, 2, 7, 9, 8, 6]
>>> 3 in lista
True
>>> 4 not in lista
True
```

Operadores de identidad

Por último, los operadores de identidad se utilizan para comprobar si dos variables son, o no, el mismo objeto.

Operador	Descripción
is	Devuelve `True` si ambos operandos hacen referencia al mismo objeto; `False` en caso contrario.
is not	Devuelve `True` si ambos operandos no hacen referencia al mismo objeto; `False` en caso contrario.

Recuerda: Para conocer la identidad de un objeto se usa la función `id()`. Esta función devuelve el identificador único del objeto que se pasa como argumento, el cuál es único e inmutable durante el ciclo de vida del objeto.

```
>>> x = 4
>>> y = 2
>>> lista = [1, 5]
>>> x is lista
False
>>> x is 4
True
```

Prioridad de los operadores en Python

Al igual que ocurre en las matemáticas, los operadores en Python tienen un orden de precedencia. Este orden es el siguiente, de menos prioritario a más prioritario: *asignación*; *operadores booleanos*; *operadores de comparación*, *identidad y pertenencia*; *a nivel de bits* y finalmente los *aritméticos* (estos últimos con el mismo orden de prioridad que en las matemáticas).

Además, el orden de prioridad se puede alterar con el uso de los paréntesis ():

```
>>> x = 5
>>> y = 2
>>> z = x + 3 * y
>>> z
11
>>> z = (x + 3) * y
>>> z
16
```

Operador de concatenación de cadenas de caracteres

Una de las operaciones más básicas cuando se trabaja con cadenas de caracteres es la concatenación. Esto consiste en unir dos cadenas.

Python define el operador de concatenación `+` para concatenar dos cadenas de caracteres. El resultado siempre es un nuevo string.

```
>>> hola = 'Hola'
>>> python = 'Pythonista'
>>> hola_python = hola + ' ' + python
>>> print(hola_python)
Hola Pythonista
```