**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 en una sola, siendo el resultado un nuevo *string*.

La forma más simple de concatenar dos cadenas en Python es utilizando el operador de concatenación +:

>>> hola = 'Hola'

>>> python = 'Pythonista'

>>> hola\_python = hola + ' ' + python # concatenamos 3 strings

>>> print(hola\_python)

Hola Pythonista

**Operadores lógicos o booleanos**

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

**IMPORTANTE:**Las operaciones 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 de comparación**

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

| Operador | Descripción |
| --- | --- |
| > | 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**

**Operadores aritméticos en Python**

En cuanto a los operadores aritméticos, estos permiten realizar las diferentes operaciones aritméticas del álgebra: suma, resta, producto, división, … Estos operadores Python son de los más utilizados. 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. |

>>> 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 de asignación**

El operador de asignación se utiliza para asignar un valor a una variable. Como te he mencionado en otras secciones, 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, aparece 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 |

**Anteriormente hemos visto los operadores aritméticos, que usaban dos números para calcular una operación aritmética (como suma o resta) y devolver su resultado. En este caso, los operadores de asignación o *assignment operators* nos permiten realizar una operación y almacenar su resultado en la variable inicial. Podemos ver como realmente el único operador nuevo es el =. El resto son abreviaciones de otros operadores que habíamos visto con anterioridad. Ponemos un ejemplo con x=7**

| **Operador** | **Ejemplo** | **Equivalente** |
| --- | --- | --- |
| **=** | **x=7** | **x=7** |
| **+=** | **x+=2** | **x=x+2 = 7** |
| **-=** | **x-=2** | **x=x-2 = 5** |
| **\*=** | **x\*=2** | **x=x\*2 = 14** |
| **/=** | **x/=2** | **x=x/2 = 3.5** |
| **%=** | **x%=2** | **x=x%2 = 1** |
| **//=** | **x//=2** | **x=x//2 = 3** |
| **\*\*=** | **x\*\*=2** | **x=x\*\*2 = 49** |
| **&=** | **x&=2** | **x=x&2 = 2** |
| **|=** | **x|=2** | **x=x|2 = 7** |
| **^=** | **x^=2** | **x=x^2 = 5** |
| **>>=** | **x>>=2** | **x=x>>2 = 1** |
| **&lt&lt=** | **x&lt&lt=2** | **x=x&lt&lt2 = 28** |

**a=7; b=2**

**print("Operadores de asignación")**

**x=a; x+=b; print("x+=", x) # 9**

**x=a; x-=b; print("x-=", x) # 5**

**x=a; x\*=b; print("x\*=", x) # 14**

**x=a; x/=b; print("x/=", x) # 3.5**

**x=a; x%=b; print("x%=", x) # 1**

**x=a; x//=b; print("x//=", x) # 3**

**x=a; x\*\*=b; print("x\*\*=", x) # 49**

**x=a; x&=b; print("x&=", x) # 2**

**x=a; x|=b; print("x|=", x) # 7**

**x=a; x^=b; print("x^=", x) # 5**

**x=a; x>>=b; print("x>>=", x) # 1**

**x=a; x<<=b; print("x<<=", x) # 28**

**Operador =**

**El operador = prácticamente no necesita explicación, simplemente asigna a la variable de la izquierda el contenido que le ponemos a la derecha. Ponemos en negrita variable porque si hacemos algo del tipo 3=5 tendremos un error. Como siempre, nunca te fíes de nada y experimenta con ello.**

**x=2 # Uso correcto del operador =**

**print(x) # 2**

**#3=5 # Daría error, 3 no es una variable**

**Tal vez pienses que el operador = es trivial y apenas merezca explicación, pero es importante explorar los límites del lenguaje. Si sabes lo que es un puntero, o una referencia tal vez el ejemplo siguiente tenga sentido para tí. Vamos a ver, si todo lo que hemos visto anteriormente es cierto, a=[1, 2, 3] asigna [1, 2, 3] a a, por lo que si no tocamos a, el valor de a deberá ser siempre [1, 2, 3]. Bueno, pues en el siguiente ejemplo vemos como eso no es así, el valor de a ha cambiado. Se podría decir que no es lo mismo x=3 con un número que x=[1, 2, 3] con una lista. No te preocupes si no lo has seguido, en otros capítulos lo explicaremos mejor.**

**a = [1, 2, 3]**

**b = a**

**b += [4]**

**print(a)**

**# [1, 2, 3, 4]**

**Operador +=**

**Como podemos ver, todos los operadores de asignación no son más que atajos para escribir otros operadores de manera más corta, y asignar su resultado a la variable inicial. El operador += en x+=1 es equivalente a x=x+1. Sabiendo esto, sería justo preguntarse ¿realmente merece la pena crear un operador nuevo que hace algo que ya podemos hacer pero de manera mas corta? Bien, la pregunta no es fácil de responder y en cierto modo viene heredado de lenguajes como C que en los años 1970s introdujeron esto.**

**x=5 # Ejemplo de como incrementar**

**x+=1 # en una unidad x**

**print(x)**

**# 6**

**Para saber más: Aunque se podría decir que el operador x+=1 es igual que x=x+1, no es del todo cierto. De hecho el operador que Python invoca por debajo es "\_\_iadd\_\_" en el primer caso frente a "\_\_add\_\_" para el segundo. A efectos prácticos, se podría considerar lo mismo.**

**Se puede jugar un poco con el operador += y aplicarlo sobre variables que no necesariamente son números. Como vimos en otros capítulos, se podría emplear sobre una lista.**

**x=[1,2,3] # En este caso la x es una lista**

**x+=[4,5] # Se aplica el operador sobre otra lista**

**print(x) # Y el resultado de la unión de ambas**

**# [1, 2, 3, 5, 6]**

**Es muy importante, que si x es una lista, no podemos aplicar el operador += con un elemento que no sea una lista, como por ejemplo, un número. El siguiente código daría error porque el operador no esta definido para un elemento lista y otro entero.**

**x=[1,2,3] #**

**#x+=3 # ERROR! TypeError**

**Operador -=**

**El operador -= es equivalente a restar y asignar el resultado a la variable inicial. Es decir, x-=1 es equivalente a x=x-1. Si vienes de otros lenguajes de programación, tal vez conozcas la forma x--, pero en Python no existe. El operador es muy usado para decrementar el valor de una variable.**

**i = 5**

**i -= 1**

**print(i) # 4**

**Y algo que nunca se haría en la realidad, pero nos permite explorar los límites del lenguaje, sería restar -1, lo que equivale a sumar uno. Pero de verdad, no hagas esto, en serio.**

**i = 0**

**i-=-1 # Aumenta el contador**

**print(i) # 1**

**Operador \*=**

**El operador \*= equivale a multiplicar una variable por otra y almacenar el resultado en la primera, es decir x\*=2 equivale a x=x\*2. Hasta ahora hemos usado todos los operadores de asignación con una variable y un número, pero es totalmente correcto hacerlo con dos variables.**

**a=10; b=2 # Inicializamos a 10 y 20**

**a\*=b # Usando dos variables**

**print(a) # 20**

**Operador /=**

**El operador /= equivale a dividir una variable por otra y almacenar el resultado en la primera, es decir, x/=2 equivale a x=x/2. Acuérdate que en otros capítulos vimos como 5/3 en versiones antiguas de Python, podía causar problemas ya que el resultado no era un numero entero. En el siguiente ejemplo podemos ver como Python hace el trabajo por nosotros, y cambia el tipo de la variable x de lo que inicialmente era int a un float con el objetivo de que el nuevo valor pueda ser almacenado.**

**x = 10**

**print(type(x)) # <class 'int'>**

**x/=3**

**print(type(x)) # <class 'float'>**

**Operador %=**

**El operador %= equivale a hacer el módulo de la división de dos variables y almacenar su resultado en la primera.**

**x = 3**

**x%=2**

**print(x) # 1**

**Una curiosidad a tener en cuenta, es que el operador módulo tiene diferentes comportamientos en Python del que tiene en otros lenguajes como C cuando se usan números negativos tanto de dividendo como de divisor. Así que ten cuidado si haces cosas como las siguientes.**

**print(-5%-3) # -2**

**print(5%-3) # -1**

**print(-5%3) # 1**

**print(5%3) # 2**

**Operador //=**

**El operador //= realiza la operación cociente entre dos variables y almacena el resultado en la primera. El equivalente de x//=2 sería x=x//2.**

**x=5 # El resultado es el cociente**

**x//=3 # de la división**

**print(x) # 1**

**Operador \*\*=**

**El operador \*\*= realiza la operación exponente del primer número elevado al segundo, y almacena el resultado en la primera variable. El equivalente de x\*\*=2 sería x=x\*\*2.**

**x=5 # Eleva el número al cuadrado**

**x\*\*=2 # y guarda el resultado en la misma**

**print(x) # 25**

**Otro ejemplo similar, sería empleando un exponente negativo, algo que es totalmente válido y equivale matemáticamente al inverso del número elevado al exponente en positivo. Dicho de otra forma, x−2 equivale a 1/x2.**

**x=5 # Elevar 5 a -2 equivale a dividir**

**x\*\*=-2 # uno entre 25.**

**print(x) # 0.04**

**Operador &=**

**El operador &= realiza la comparación & bit a bit entre dos variables y almacena su resultado en la primera. El equivalente de x&=1 sería x=x&1**

**a = 0b101010**

**a&= 0b111111**

**print(bin(a))**

**# 0b101010**

**Operador |=**

**El operador |= realiza el operador | elemento a elemento entre dos variables y almacena su resultado en la primera. El equivalente de x|=2 sería x=x|2**

**a = 0b101010**

**a|= 0b111111**

**print(bin(a))**

**# 0b111111**

**Operador ^=**

**El operador ^= realiza el operador ^ elemento a elemento entre dos variables y almacena su resultado en la primera. El equivalente de x^=2 sería x=x^2**

**a = 0b101010**

**a^= 0b111111**

**print(bin(a))**

**# 0b10101**

**Operador »=**

**El operador >>= es similar al operador >> pero permite almacenar el resultado en la primera variable. Por lo tanto x>>=3 sería equivalente a x=x>>3**

**x = 10**

**x>>=1**

**print(x) # 5**

**Es importante tener cuidado y saber el tipo de la variable x antes de aplicar este operador, ya que se podría dar el caso de que x fuera una variable tipo float. En ese caso, tendríamos un error porque el operador >> no esta definido para float.**

**x=10.0 # Si la x es float**

**print(type(x)) # <class 'float'>**

**#x>>=1 # ERROR! TypeError**

**Operador «=**

**Muy similar al anterior, <<= aplica el operador << y almacena su contenido en la primera variable. El equivalente de x<<=1 sería x=x<<1**

**x=10 # Inicializamos a 10**

**x<<=1 # Desplazamos 1 a la izquierda**

**print(x) # 20**

**Sería justo pensar que si << realiza un desplazamiento de bits a la izquierda y >> lo realiza a la derecha, tal vez un desplazamiento << una unidad, podría equivaler a -1 desplazamiento a la derecha.**

**#x<<=-1 # ERROR! Python no define un desplazamiento negativo a la izquierda**

**#x>>=-1 # ERROR! Python no define un desplazamiento negativo a la derecha**

**Operadores de pertenencia**

Los operadores de pertenencia se utilizan para comprobar si un valor o variable se encuentran en una secuencia (list, tuple, dict, set o str).

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

| Operador | Descripción |
| --- | --- |
| in | Devuelve True si el valor se encuentra en una secuencia; False en caso contrario. |
| not in | Devuelve True si el valor no se encuentra en una secuencia; False en caso contrario. |

A continuación vemos unos ejemplos que son muy intuitivos:

>>> lista = [1, 3, 2, 7, 9, 8, 6]

>>> 4 **in** lista

**False**

>>> 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().

>>> x = 4

>>> y = 2

>>> lista = [1, 5]

>>> x is lista

**False**

>>> x is y

**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 prioridad. 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 (con el mismo orden de prioridad que en las matemáticas).

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

>>> x = 5

>>> y = 2

>>> z = x + 3 \* y # El producto tiene prioridad sobre la suma

>>> z

11

>>> z = (x + 3) \* y # Los paréntesis tienen prioridad

>>> z

16