Operaciones con datos

March 29, 2022

1 Operaciones con datos en Python

1.1 ¿Qué es un operador?

Los operadores son símbolos especiales en Python que realizan cálculos aritméticos o lógicos. Al valor sobre el que actúa el operador se le llama operando. Por ejemplo:

```
[ ]: print(2 + 5)
```

```
7
```

Aquí, `+` es el operador que realiza la suma. `2` y `5` son los operandos y `7` es el resultado de la operación. Ahora veamos con qué operadores contamos en cada tipo de datos:

1.2 Operadores aritméticos

Los operadores aritméticos se utilizan para realizar operaciones matemáticas como sumas, restas, multiplicaciones, etc. Estos son compatibles con cualquier tipo de dato numérico, enteros, flotantes o complejos. En la siguiente tabla veremos los principales operadores aritméticos que tiene python:

Operador	Descripción	Ejemplo
+	Suma	`r = 3 + 2`
-	Resta	`r = 4 - 7`
-	Negación	`r = -7`
	Multiplicación	`r = 2 * 6`
	Exponente	`r = 2 ** 6`
/	División	`r = 3.5 / 2`
//	División entera	`r = 3.5 // 2`
%	Módulo	`r = 7 % 2`

Si bien la suma, resta, multiplicación y las potencias tienen el comportamiento que hemos aprendido desde las matemáticas, como vemos, python tiene 3 operaciones relacionadas con la división, ahora, vamos a ver un ejemplo de uso de todas las operaciones y entrarémos en detalle del comportamiento de estas últimas tres operaciones.

1.2.1 Suma

Ejecuta el siguiente bloque de código insertando diferentes tipos de número:

[ ]:
```
a = eval(input("digita un número"))
b = eval(input("digita otro número"))
print("la suma es:", a + b)
```

```
digita un número412
digita otro número35
la suma es: 447
```

1.2.2 Resta

Ejecuta el siguiente bloque de código insertando diferentes tipos de número:

[ ]:
```
a = eval(input("digita un número"))
b = eval(input("digita otro número"))
print("la resta es:", a - b)
```

```
digita un número 845
digita otro número 165

la resta es: 680
```

1.2.3 Multiplicación

Ejecuta el siguiente bloque de código insertando diferentes tipos de número:

[ ]:
```
a = eval(input("digita un número"))
b = eval(input("digita otro número"))
print("la multiplicación es:", a * b)
```

```
digita un número 658
digita otro número 3.14

la multiplicación es: 2066.12
```

1.2.4 Potenciación

Ejecuta el siguiente bloque de código insertando diferentes tipos de número:

[ ]:
```
a = eval(input("digita un número"))
b = eval(input("digita otro número"))
print("la potencia a^b es:", a ** b)
```

```
digita un número 2
digita otro número 6

la potencia a^b es: 12
```

1.2.5 División

Como se mencionó anteriormente, python cuenta con 3 operaciones relacionadas con la división: la división entera, la división completa y el módulo.

La **división entera** arroja el cociente de la operación, independientemente de que los números sean enteros o flotantes y este resultado ignora la parte decimal que pudiera tener el cociente. Su operador es `//`

[ ]:
```
print('5//2: ', 5 // 2)
print('4525.214//2: ', 4525.214 // 2)
print('54874136.85//65874.6: ', 54874136.85 // 65874.6)
```

```
5//2:  2
4525.214//2:  2262.0
54874136.85//65874.6:  833.0
```

La **división completa** es la operación tradicional de la mayoría de lenguajes de programación, su cociente tiene en cuenta la fracción decimal que sea necesaria para hacer cero el residuo. Su operador es `/`

[ ]:
```
print('5/2: ', 5 / 2)
print('4525.214/2: ', 4525.214 / 2)
print('54874136.85/65874.6: ', 54874136.85 / 65874.6)
```

```
5/2:  2.5
4525.214/2:  2262.607
54874136.85/65874.6:  833.0090330719275
```

La operación de **módulo** retorna el residuo de la división entre dos números y su operador es `%`. Si bien esta operación puede realizarse con flotantes, no es una práctica recomendada, ya que si usamos operandos flotantes obtendremos un valor de punto flotante como cociente y el resto será cero, lo que no generará una respuesta correcta.

Así que si usamos operandos flotantes en el operador de módulo, siempre obtendremos la respuesta 0, pero como veremos en el ejemplo, esto no siempre ocurre en Python. Si queremos obtener la respuesta correcta de módulo debemos usar operandos enteros

[ ]:
```
print('5%2: ', 5 % 2)
print('4525.214%2.5: ', 4525.214 % 2.5)
```

```
5%2:  1
4525.214%2.5:  0.2139999999999418
```

1.3 Operadores de comparación

Los operadores de comparación se utilizan para comparar valores. Devuelven `True` o `False` según la condición.

operador	Descripción	Ejemplo
==	a = b	`5 == 3`
!=	a diferente de b	`5 != 3`
<	a menor que b	`5 < 3`
>	a mayor que b	`5 > 3`
<=	a menor o igual que b	`5 <= 3`

operador	Descripción	Ejemplo
>=	a mayor o igual que b	`5 >= 3`

[ ]:
```
x = 69
y = 420

print('x > y es', x > y)
print('x < y es', x < y)
print('x == y es', x == y)
print('x != y es', x != y)
print('x >= y es', x >= y)
print('x <= y es', x <= y)
```

```
x > y es False
x < y es True
x == y es False
x != y es True
x >= y es False
x <= y es True
```

1.4 Operadores lógicos

Permiten realizar operaciones de álgebra booleana, opera directamente sobre los valores de `True` y `False` y retorna uno de estos mismos valores.

operador	Descripción	Ejemplo
and	a y b	`a and b`
or	a o b	`a or b`
not	negación de a	`not a`
is	Verdadero si los operandos son idénticos	`x is True`
is not	Verdadero si los operandos no son idénticos	`x is not True`

[ ]:
```
x = True
y = False
x1 = 5
y1 = 5
x2 = 'Hello'
y2 = 'Hello'

print('x and y es', x and y)
print('x or y es', x or y)
print('not x es', not x)
print(x1 is not y1)
print(x2 is y2)
```

```
x and y es False
x or y es True
not x es False
False
True
```

1.5 Operadores de asignación

Los operadores de asignación se utilizan en Python para asignar valores a las variables.

`a = 5` es un operador de asignación simple que asigna el valor 5 de la derecha a la variable `a` de la izquierda.

Hay varios operadores compuestos en Python como `a += 5` que suma a la variable y luego asigna lo mismo. Es equivalente a `a = a + 5`. Es importante aclarar que debe existir de antemano la variable y tener un valor internamente para que estos operadores funcionen, de lo contrario se obtendrá el siguiente error:

[ ]:
```
l += 5
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-10-0bf8e349eb49> in <module>
----> 1 l+=5

NameError: name 'l' is not defined
```

A continuación se presenta una tabla con los principales operadores de asignación que podríamos utilizar.

Operador	Ejemplo	Equivalente a
=	`x = 5`	`x = 5`
+=	`x += 5`	`x = x + 5`
-=	`x -= 5`	`x = x - 5`
=	`x *= 5`	`x = x * 5`
/=	`x /= 5`	`x = x / 5`
%=	`x %= 5`	`x = x % 5`
//=	`x //= 5`	`x = x // 5`

[ ]:
```
x = 123456
print('1: ', x)
x += 333
print('2: ', x)
x *= 2
print('3: ', x)
x -= 555
print('4: ', x)
```

```
x /= 5
print('5: ', x)
```

```
1:  123456
2:  123789
3:  247578
4:  247023
5:  49404.6
```