



# PYTHON COMO UNA CALCULADORA

Ahora, se va a mostrar un nuevo lado de la función `print()`. Ya se sabe que la función es capaz de mostrar los valores de los literales que le son pasados por los argumentos.

```
print (2+2)
```

Reescribe el código en el editor y ejecútalo. ¿Puedes adivinar la salida? Deberías de ver el número cuatro. Tómate la libertad de experimentar con otros operadores. Sin tomar esto con mucha seriedad, has descubierto que Python puede ser utilizado como una calculadora. No una muy útil, y definitivamente no una de bolsillo, pero una calculadora sin duda alguna. Tomando esto más seriamente, nos estamos adentrado en el terreno de los **operadores** y **expresiones**.

```
1 print(2+2)
```

Console >\_

4



# LOS OPERADORES BÁSICOS



UN OPERADOR ES UN SÍMBOLO DEL LENGUAJE DE PROGRAMACIÓN, EL CUAL ES CAPAZ DE REALIZAR OPERACIONES CON LOS VALORES.

POR EJEMPLO, COMO EN LA ARITMÉTICA, EL SIGNO DE + (MÁS) ES UN OPERADOR EL CUAL ES CAPAZ DE SUMAR DOS NÚMEROS, DANDO EL RESULTADO DE LA SUMA.

SIN EMBARGO, NO TODOS LOS OPERADORES DE PYTHON SON TAN SIMPLES COMO EL SIGNO DE MÁS, VEAMOS ALGUNOS DE LOS OPERADORES DISPONIBLES EN PYTHON, LAS REGLAS QUE SE DEBEN SEGUIR PARA EMPLEARLOS, Y COMO INTERPRETAR LAS REGLAS QUE REALIZAN.



Se comenzará  
con los  
operadores que  
están asociados  
con las  
operaciones  
aritméticas más  
conocidas:

$+$ ,  $-$ ,  $*$ ,  $/$ ,  $//$ ,  $\%$ ,  $**$



## **OPERADORES ARITMÉTICOS: EXPONENCIACIÓN**



**Un signo de \*\*  
(doble asterisco)  
es un operador de  
exponenciación  
(potencia). El  
argumento a la  
izquierda es la  
base, el de la  
derecha, el  
exponente.**


**Las matemáticas clásicas prefieren una notación con superíndices, como el siguiente:  $2^3$ . Los editores de texto puros no aceptan esa notación, por lo tanto Python utiliza \*\* en lugar de la notación matemática, por ejemplo, `2 ** 3`.**

Observa los ejemplos en la ventana del editor.

Nota: En los ejemplos, los dobles asteriscos están rodeados de espacios, no es obligatorio hacerlo pero hace que el código sea mas **legible**.

Los ejemplos muestran una característica importante de los **operadores numéricos** de Python.

Ejecuta el código y observa cuidadosamente los resultados que arroja. ¿Puedes observar algo?

Operadores >  example

```
1 print(2 ** 3)
2 print(2 ** 3.)
3 print(2. ** 3)
4 print(2. ** 3)
5
```

```
65419
8
8.0
8.0
```

# ¡Recuerda!

**Recuerda:** Es posible formular las siguientes reglas con base en los resultados:

- Cuando **ambos** `**` argumentos son enteros, el resultado es entero también.
- Cuando **al menos un** `**` argumento es flotante, el resultado también es flotante.

Esta es una distinción importante que se debe recordar.



# **OPERADORES ARITMÉTICOS: MULTIPLICACIÓN**

Un símbolo de \* (asterisco) es un operador de **multiplicación**.

Ejecuta el código y revisa si la regla de *entero frente a flotante* aún funciona.

```
operadores > example1.py
1 print(2 * 3)
2 print(2 * 3.)
3 print(2. * 3)
4 print(2. * 3.)
5
```

```
6
6.0
6.0
6.0
```



## **OPERADORES ARITMÉTICOS: DIVISIÓN**

Un símbolo de / (diagonal) es un operador de **división**.

El valor después de la diagonal es el **dividendo**, el valor antes de la diagonal es el **divisor**.

Ejecuta el código y analiza los resultados.

```
operadores > example1.py
1 print(6 / 3)
2 print(6 / 3.)
3 print(6. / 3)
4 print(6. / 3.)
```

```
2.0
2.0
2.0
2.0
```



¿NOTAS ALGO?



Deberías de poder observar que hay una excepción a la regla.

**El resultado producido por el operador de división siempre es flotante,** sin importar si a primera vista el resultado es flotante:  $1 / 2$ , o si parece ser completamente entero:  $2 / 1$ .

¿Esto ocasiona un problema? Sí, en ocasiones se podrá necesitar que el resultado de una división sea entero, no flotante.

Afortunadamente, Python puede ayudar con eso.



## **OPERADORES ARITMÉTICOS: DIVISIÓN ENTERA**

```
operadores > example1.py
```

```
1 print(6 // 3)
```

```
2 print(6 // 3.)
```

```
3 print(6. // 3)
```

```
4 print(6. // 3.)
```

```
5
```

```
2
```

```
2.0
```

```
2.0
```

```
2.0
```

Un símbolo de // (doble diagonal) es un operador de **división entera**. Difiere del operador estándar / en dos detalles:

- El resultado carece de la parte fraccionaria, está ausente (para los enteros), o siempre es igual a cero (para los flotantes); esto significa que **los resultados siempre son redondeados**.
- Se ajusta a la regla *entero frente a flotante*.

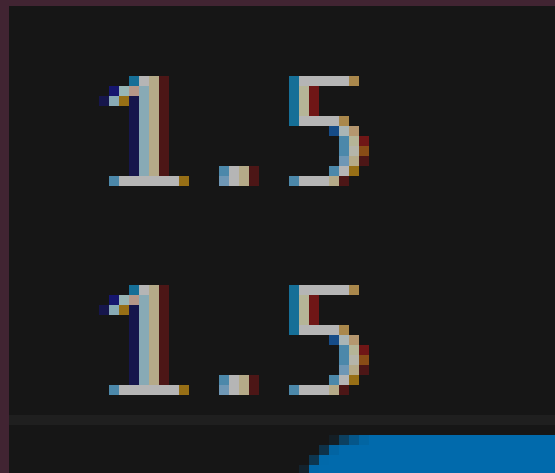
Ejecuta el ejemplo debajo y observa los resultados:

Como se puede observar, *una división de entero entre entero* da un **resultado entero**. Todos los demás casos producen flotantes.

Hagamos algunas pruebas mas avanzadas.  
Observa el siguiente fragmento de código:

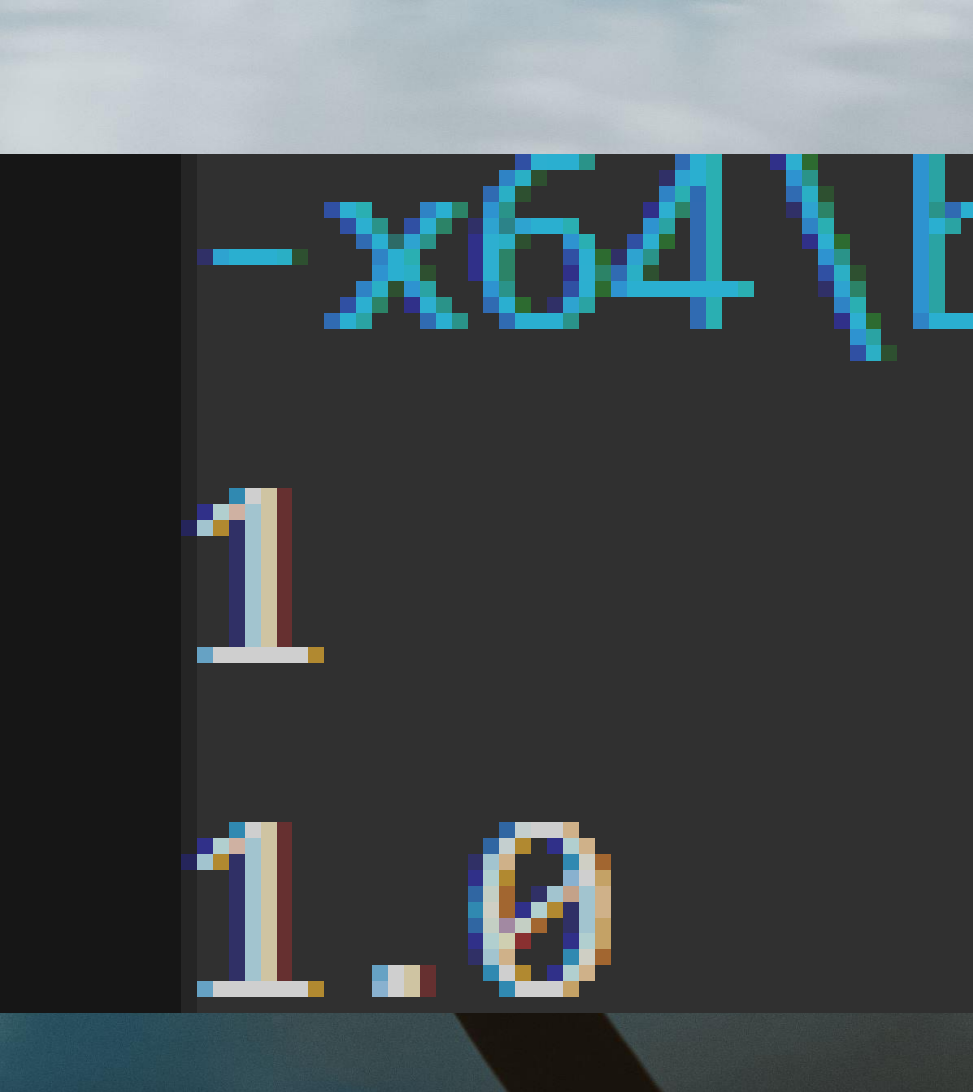
```
operadores > example1.py
1 print(6 // 4)
2 print(6. // 4)
```

```
operadores > example1.py
1 print(6 / 4)
2 print(6. / 4)
```



Imagina que se utilizó / en lugar de // -  
¿Podrías predecir los resultados?

Si, sería 1.5 en ambos casos. Eso está claro.



Lo que se obtiene son dos unos, uno entero y uno flotante.

Esto es muy importante: **el redondeo siempre va hacia abajo.**