
 No description has been provided for this image No description has been provided for this image

Python Basics I

Contenidos

- [Operaciones aritméticas](#)
- [Operaciones comparativas](#)

Operaciones aritméticas

[al índice](#)

Hace un par de sesiones ya vimos por encima las principales operaciones aritméticas en Python. Las recordamos:

- Sumar: `+`
- Restar: `-`
- Multiplicar: `*`
- Dividir: `/`
- Elevar: `**`
- Cociente division: `//`
- Resto de la división: `%`

```
In [5]: # Declarar y asignar una variable entera
variable_1 = 100
# Declarar y asignar una variable real o float
variable_2 = 10.5
# Sumar las variables y mostrar el resultado
suma = variable_1 + variable_2
print(suma)
# Mostrar de qué tipo es el resultado
print(type(suma))
# Multiplicar ambas variables y mostrar el resultado
multi = variable_1 * variable_2
print(multi)
# Elevar una a la otra
potencia = pow(variable_1, variable_2)
print(potencia)
```

```
# Dividir una por otra y mostrar además el resto y el cociente
dividir = variable_1 / variable_2
print(dividir, (variable_1//variable_2), (variable_1%variable_2) )
```

```
110.5
<class 'float'>
1050.0
1e+21
9.523809523809524 9.0 5.5
```

In []:

Propiedad conmutativa, asociativa, distributiva y el paréntesis

Si queremos concatenar varias operaciones, ten siempre en cuenta las propiedades matemáticas de la multiplicación

```
In [6]: print("Conmutativa")
print(2 * 3)
print(3 * 2)

print("\nAsociativa") # Recuerda que "\n" se usa para que haya un salto
print((3 * 2) * 5)
print(3 * (2 * 5))

print("\nDistributiva")
print(2 * (3 + 5))
print(2 * 3 + 2 * 5)

print("\nEl Orden de operaciones se mantiene. Siempre podemos usar paré")
print(2 * (2 + 3) * 5)
print((2 * 2 + 3 * 5)/(4 + 7))
```

Conmutativa

```
6
6
```

Asociativa

```
30
30
```

Distributiva

```
16
16
```

El Orden de operaciones se mantiene. Siempre podemos usar paréntesis

```
50
1.7272727272727273
```

Operaciones más complejas

Si salimos de las operaciones básicas de Python, tendremos que importar módulos con más funcionalidades en nuestro código. Esto lo haremos mediante la sentencia `import math`. `math` es un módulo con funciones ya predefinidas, que no vienen por defecto en el núcleo de Python. De esta forma será posible hacer cálculos más complejos como:

- Raíz cuadrada
- Seno/Coseno
- Valor absoluto *...

El módulo es completísimo y si estás buscando alguna operación matemática, lo más seguro es que ya esté implementada. Te dejo por aquí el [link a la documentación del módulo](#)..

```
In [7]: import math
print(math.sqrt(25))
print(math.cos(0))
print(math.fabs(-124.45))
```

5.0

1.0

124.45

Como en todos los lenguajes de programación, suele haber una serie de componentes básicos (variables, operaciones aritméticas, tipos de datos...) con los que podemos hacer muchas cosas. Ahora bien, si queremos ampliar esas funcionalidades, se suelen importar nuevos módulos, con funciones ya hechas de otros usuarios, como en el caso del módulo `math`. Veremos esto de los módulos más adelante.



No
description
has been
provided for
this image

ERRORES Dividir por cero

Cuidado cuando operamos con 0s. Las indeterminaciones y valores infinitos suponen errores en el código. Por suerte, la descripción de estos errores es bastante explícita, obteniendo un error de tipo `ZeroDivisionError`

```
In [8]: 4/0
```

```
-----  
---  
ZeroDivisionError                                Traceback (most recent call la  
st)  
Cell In[8], line 1  
----> 1 4/0  
  
ZeroDivisionError: division by zero
```

```
In [ ]: variable_a/variable_b
```

```
In [ ]: # Hay valores que se salen del dominio de algunas funciones matemáticas  
# como es el caso de las raíces de números negativos
```

```
In [10]: math.sqrt(-12)
```

```
-----  
---  
ValueError                                Traceback (most recent call la  
st)  
Cell In[10], line 1  
----> 1 math.sqrt(-12)  
  
ValueError: math domain error
```

Operaciones comparativas

[al índice](#)

Es bastante intuitivo comparar valores en Python. La sintaxis es la siguiente:

- `==` : Igualdad. No es un `=`. Hay que diferenciar entre una comparativa, y una asignación de valores
- `!=` : Desigualdad
- `>` : Mayor que
- `<` : Menor que
- `>=` : Mayor o igual que
- `<=` : Menor o igual que

En la asignación estamos diciendole a Python que la variable `assign` vale 1, mientras que en la comparación, estamos preguntando a Python si `a` equivale a 5. Como vale 1, nos devuelve un `False`

```
In [11]: # asignación  
assign = 1  
print(assign)  
  
print(assign == 5)  
print(assign)
```

```
1
False
1
```

Veamos otros ejemplos de comparaciones

```
In [12]: print("AAA" == "BBB")
print("AAA" == "AAA")
print(93>12)
print(1==0)
```

```
False
True
True
False
```



No
description
has been
provided for
this image

ERRORES en comparativas

Este tipo de errores son muy comunes, pues es muy habitual comparar peras con manzanas. Cuando se trata de una igualdad (`==`), no suele haber problemas, ya que si las variables son de distinto tipo, simplemente es `False` . Lo ideal sería que Python nos avisase de estas cosas porque realmente lo estamos haciendo mal, no estamos comparando cosas del mismo tipo

```
In [14]: print(1=="1")
print("Hola" == True)
```

```
False
False
```

```
In [ ]: # Obtenemos un TypeError cuando la comparativa es de > o <
```

```
In [15]: print("17" >= 23)
```

```
-----
---
TypeError                                Traceback (most recent call last)
Cell In[15], line 1
----> 1 print("17" >= 23)

TypeError: '>=' not supported between instances of 'str' and 'int'
```

```
In [ ]:
```