Python Basics II

Ya hemos visto cómo declarar variables, qué tipos hay, y otras funcionalidades importantes de Python como sus flujos de ejecución o las formas que tenemos de comentar el código. En este Notebook aprenderás a realizar operaciones con tus variables y descubrirás las colecciones mediante uno de los objetos más usados en Python: las listas.

1. Operaciones aritméticas

En el Notebook *Python Basics I* ya vimos por encima las principales operaciones aritméticas en Python. Las recordamos:

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

Ejercicio de operaciones aritméticas

1. **Declara una variable int**
2. **Declara otra variable float.**
3. **Suma ambas variables. ¿Qué tipo de dato es el resultado?**
4. **Multiplica ambas variables**
5. **Eleva una variable a la potencia de la otra**
6. **Calcula el resto de dividir 12/5**

In [3]:

```
#Declaracion de una variable int
int=12
#Declaracion de una variable float
float=12.3
#Suma de ambas variables
print(int+float)
#El resultado de la operacion es un float

#Multiplicacin de ambas varibales
print(int*float)
#Variable elevada a otra variable
print(int**float)
#Calcular el resto de dividir 12 entre 5
print(12%5)
#Sacar el tipo de dato que es la variable que se le pasa por parámetro
type(int)
```

```
#Uso dek resto
#cuanto tiene que poner cada uno de los 5 amigos
print('Cada amigo tiene que poner',128//5,"euros")
print("Alguien se tiene que sacrificar y poner",128%5,"euros mas")
```

```
24.3
147.60000000000002
18790110166816.06
2

Out[3]:

int
```

Propiedad conmutativa, asociativa, distributiva y el paréntesis

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

In [7]:

```
print("Conmutativa")
print(2 * 3)
print(3 * 2)

print("\nAsociativa") # Recuerda que "\n" se usa para que haya un salto de linea en el ou
tput.
print(2 * (3 + 5))
print(2 * 3 + 2 * 5)

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

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

```
Conmutativa
6
6

Asociativa
16
16

Distributiva
30
30

El Orden de operaciones se mantiene. Siempre podemos usar paréntesis
50
19
```

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 [1]:

```
import math
```

In [2]:

```
#Raiz cuadrada
print(math.sqrt(25))
#Valor absoluto
print(math.fabs(-4))
print(abs(-4.0))
#?
print(math.acos(0))
```

```
5.0
4.0
4.0
1.5707963267948966
```

In [5]:

```
#Importar librerias
import math
```

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.

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 [11]:

```
4/0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
C:\Users\MIGUEL~1\AppData\Local\Temp/ipykernel_9316/1580332654.py in <module>
----> 1 4/0

ZeroDivisionError: division by zero
```

In [12]:

```
math.sqrt(-10)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
C:\Users\MIGUEL~1\AppData\Local\Temp/ipykernel_9316/4187518437.py in <module>
----> 1 math.sqrt(-10)

ValueError: math domain error
```

Ejercicio de operaciones con math

Consulta la documentación de math para resolver este ejercicio

1. **Calcula el valor absoluto de -25. Usa fabs**
2. **Redondea 4.7 a su entero más bajo. Usa floor**
3. **Redondea 4.3 a su entero más alto. Usa ceil**
4. **El número pi**
5. **¿Cuál es el área de un círculo de radio 3?**

In [10]:

```
#Sacar valor absoluto
math.fabs(-25)
#Redondear hacia abajo
math.floor(4.7)
#Redondear hacia arriba
math.ceil(4.3)
#Sacar valor de pi
math.pi
#Redondearvalor de pi a 5 decimales
round(math.pi,5)
#Calcular el area de un círculo de radio 3
area=math.pi*(3**2)
#Sacar por consola el valor del area
print(area)
```

```
28.274333882308138
```

2. Operaciones comparativas

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**

In [21]:

```
x = 1

print(x == 1)
print(x == 5)
```

```
True
False
```

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

In [25]:

```
print("AAA" == "BBB")
print("AAA" == "AAA")

print(1 == 1.0)
print(1 == 1.1)

print(67 != 67)
```

```
False
True
True
False
False
```

In [27]:

```
print(True == 1)
print(False == 0)
print(True == 5)
```

```
True
True
```

False

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 [4]:

```
# comparar elementos de distinto tipo
'1'== 1
```

Out[4]:

```
False
```

In [3]:

```
int('1')==1
```

Out[3]:

```
True
```

3. Operaciones con booleanos

Todas las operaciones que realizabamos en el apartado anterior devolvían un tipo de dato concreto: un booleano. `True` o `False`. Pero ¿cómo harías si se tienen que cumplir 3 condiciones, o solo una de esas tres, o que no se cumplan 5 condiciones? Para este tipo de operaciones recurrimos al *Álgebra de Boole*. Se trata de una rama del álgebra que se utiliza en electrónica, pero que tiene un sin fin de aplicaciones, no solo téncicas, sino aplicables a la vida cotidiana. Estas matemáticas pueden llegar a ser muy complejas aún utilizando únicamente dos valores: `True` y `False`. Las operaciones más comunes son AND, OR, NOT. En las siguientes tablas tienes todos los posibles resultados de las puertas AND, OR, NOT, dependiendo de sus inputs.

Puede parecer complejo pero a efectos prácticos, y sin meternos con otro tipo de puertas lógicas, te recomiendo seguir estas reglas:

- **AND: Se tienen que cumplir ambas condiciones para que sea un `True`**
- **OR: Basta que se cumpla al menos una condicion para que sea `True`**
- **NOT: Lo contrario de lo que haya**

Veamos un ejemplo práctico para aclarar estos conceptos. Imaginemos que queremos comprar un ordenador, pero nos cuesta decidirnos.

In [2]:

```
# Primer ordenador
ram1 = 32
process1 = "i5"
disco1 = 500
precio1 = 850

# Segundo ordenador
ram2 = 8
process2 = "i5"
disco2 = 500
precio2 = 600

# Tercer ordenador
ram3 = 32
process3 = "i3"
disco3 = 500
precio3 = 780
```

Eso sí, tenemos claras las siguentes condiciones a la hora de elegir

- **La RAM me vale que tenga 16, 32 o 64 GB**
- **En cuanto al procesador y disco duro, la combinación que mejor me viene es un i3 con 500GB de disco.**
- **Precio: que no pase de los 800 €**

Veamos cómo implemento esto mediante operaciones booleanas

In [12]:

```
#1
cond1 = (ram1 == 16) or (ram1 == 32) or (ram1 == 64)
print("EL primer ordenador cumple la condicion de la RAM:",cond1)
#2
cond2 = (process1 == "i3") and (disco1 == 500)
print("El primer ordendador cumple con la condicion del procesador y del disco:", cond2)
#3
cond3 = precio1 <= 800
print("El primer ordenador cumple con la condicion del precio:",cond3)
#Sacar las condiciones en una única línea
condition_ord_1= (cond1 == True) and (cond2==True) and (cond3==True)
condition_ord_2= (cond1 and cond2 and cond3)
condition_ord_3=()
print("El primer ordenador cumple con las condiciones:", condition_ord_1)
```

```
EL primer ordenador cumple la condicion de la RAM: True
El primer ordendador cumple con la condicion del procesador y del disco: False
El primer ordenador cumple con la condicion del precio: False
El primer ordenador cumple con las condiciones: False
```

Out[12]:

```
()
```

El primer ordenador cumple el requisito de ram, pero no los de precio y procesador/disco. Veamos los otros dos si los cumplen

In [ ]:

```
#Ordenador 2
cond_ord_2=(ram2==16) or (ram2==32) or (ram2==64)\
and (process2=="i3") and (disco2==500)\
and precio2 <= 800
condition_ord_2

#Ordenador 3
cond_ord_3 = (ram3==16) or (ram3==32) or (ram3==64)\
and (process3=="i3") and (disco3==500)\
and precio3 <= 800
condition_ord_3
```

¡Bingo! El tercer ordenador cumple todas las condiciones para ser mi futura compra. Verás en próximos notebooks que esto se puede hacer todavía más sencillo mediante bucles y funciones.

Si quieres aprender más sobre el Álgebra de Boole, te recomiendo esta página

ERRORES varios

¡No me vas a creer cuando te diga que lo mejor que te puede pasar es que te salten errores por pantalla! Si, estos son los errores más fáciles de detectar y puede que también fáciles de corregir ya que tienes la ayuda del descriptivo del error. El problema gordo viene cuando no saltan errores y ves que tu código no lo está haciendo bien. Para ello tendremos que debugear el código y ver paso a paso que está pasando. Lo veremos en notebooks posteriores. De momento corregiremos el código revisandolo a ojo.

Como ves en el siguiente ejemplo, el resultado del ordenador 3 es `False` cuando debería ser `True`. ¿Por qué?

In [35]:

```
cond_final3 = (ram3 == 16 or ram3 == 32 or ram3 == 64) or (process3 == "i3" and disco3 =
= 500) and (precio3 <=800)
cond_final3
```

Out[35]:

```
True
```

Cuidado cuando tenemos sentencias muy largas, ya que nos puede bailar perfectamente un paréntesis, un `>`, un `and` por un `or` ... Hay que andarse con mil ojos.

Y sobretodo, cuidado con el *copy paste*. Muchas veces, por ahorrar tiempo, copiamos código ya escrito para cambiar pequeñas cosas y hay veces que se nos olvida cambiar otras. Pensamos que está bien, ejecutamos, y saltan errores. Copiar código no es una mala práctica, es más, muchas veces evitamos errores con los nombres de las variables, pero hay que hacerlo con cabeza.

> **Ejercicio de operaciones con booleanos**
> **Sin escribir código, ¿Qué valor devuelve cada una de las siguientes operaciones?**
>
> 1. **not (True and False)**
> 2. **False or False or False or False or False or False or True or False or False or False**
> 3. **True or True or True or True or True or False or True or True or True or True**
> 4. **(False and True and True) or (True and True)**

In [2]:

```
#Valores que devuelve cada operacion
#1.false
#2.true
#3.true
#4.false
```

Out[2]:

```
True
```

4. Funciones *Built in*

Hay una serie de funciones internas, que vienen en el intérprete de Python. Algunas de las más comunes son:

- **Tipos:** `bool()`, `str()`, `int()`, `float()`
- **Min/Max:** `min()`, `max()`
- **print()**
- **type()**
- **range()**
- **zip()**
- **len()**
- **...**

La sintaxis de la función es:

```
nombre_funcion(argumentos)
```

Algunas ya las hemos visto. Sin embargo, hay unas cuantas que las iremos descubriendo a lo largo de estos notebooks. Para más detalle, tienes aquí todas las funciones *built-in* de la documentación.

De momento, en lo que se refiere a funciones, vamos a ir trabajando con funciones ya hechas, pero más adelante crearemos nuestras propias funciones.

In [6]:

```
# Longitud de string
len("Esternocleidomastoideo")
```

```
# maximo de varios números
max(12,3,12,434,1344,3,1)

# mínimo de varios números
min(1,2,3,4,4,4,44,43,4)

#redondear
round(3.442452432435243243 , 3)
```

Out[6]:

```
3.442
```

> **Ejercicio de funciones built-in**
> **Busca en la documentación una función que te sirva para ordenar de manera descendente la siguiente lista**

In [12]:

```
lista=[1,2,434,23442,13,4134,1]
sorted(lista)
sorted(lista,reverse=True)
```

Out[12]:

```
[23442, 4134, 434, 13, 2, 1, 1]
```

5. Métodos

Se trata de una propiedad MUY utilizada en programación. Son funciones propias de las variables/objetos, y que nos permiten modificarlos u obtener más información de los mismos. Dependiendo del tipo de objeto, tendremos unos métodos disponibles diferentes.

Para usar un método se usa la sintaxis `objeto.metodo()`. Ponemos un punto entre el nombre del objeto y el del metodo, y unos paréntesis por si el método necesita de algunos argumentos. Aunque no necesite de argumentos, los paréntesis hay que ponerlos igualmente.

Veamos algunos ejemplos

String

Una variable de tipo string, tiene una serie de métodos que permiten sacarle jugo a la cadena de texto. Aquí tienes todos los métodos que podemos usar en cadenas de texto

In [22]:

```
string_ejemplo = "string en mayusculas"

# Pon todo en mayúscula
print(string_ejemplo.upper())
# Pon todo en minúscula
string_ejemplo.lower()
string_ejemplo.islower()
print(string_ejemplo)
# Cambia la "s" por "S"
print(string_ejemplo.replace("s","S"))
# Septara cada palabra en una lista
print(string_ejemplo.split())
# Busca cuantas "r" hay
print(string_ejemplo.count("r"))
#busca en qué posicion está la r
print(string_ejemplo.index("r"))
```

```
STRING EN MAYUSCULAS
string en mayusculas
String en mayuSculaS
['string', 'en', 'mayusculas']
1
```

```
2
```

Como ves, se pueden hacer muchas cosas en los Strings gracias a sus métodos. Ya verás cómo la cosa se pone más interesante cuando los tipos de los datos sean todavía más complejos.

Los métodos son una manera de abstraernos de cierta operativa. Convertir todos los caracteres de una cadena a minuscula, puede ser un poco tedioso si no existiese el método `lower()`. Tendríamos que acudir a bucles o programación funcional.

ERRORES en métodos

In [23]:

```
str_ejemplo = "string"
str_ejemplo.replace()
#replace necesita un argumento
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_14948\1997718453.py in <module>
      1 str_ejemplo = "string"
----> 2 str_ejemplo.replace()

TypeError: replace() takes at least 2 arguments (0 given)
```

6. Listas

Se trata de otro de los tipos de datos de Python más usados. Dentro de las colecciones, que veremos más adelante, la lista es la colección que normalmente se le da más uso. Nos permiten almacenar conjuntos de variables u objetos, y son elementos de lo más versátiles puesto que podemos almacenar objetos de distintos tipos, modificarlos, eliminarlos, meter listas dentro de listas... Sus dos caractrísticas principales son:

- **Mutables: una vez se ha creado la lista, se puede modificar**
- **Ordenada: Los elementos tienen un cierto orden, lo que nos permite acceder al elemento que queramos teniendo en cuenta tal orden**

En cuanto a su sintaxis, cuando declaremos la lista simplemente hay que separar cada elemento con comas, y rodearlo todo con corchetes.

In [6]:

```
# Lista de números
numeros = [1,2,3,4,5,6,6,7,7,78,65,4,3,2,3,4,5,6,543,2,345,4,3,4,5,64,3,4,5]
```

In [8]:

```
# Lista de strings
cadenas=["qwer","afrgg","adfdf"]
# Listas de booleanos
booleanos=[True,False,False,True]
# Listas mixtas (con strings, con int, con floats, con listas...)
mixture=["asddd",1,2.3,[1,2,4,3,4]]
mixture
# Suma de listas
conjunto=[1,23,3,4,3413],[True,False,True]
```

NOTA: ¿Ves por qué los decimales en Python siempre van con puntos y no con comas? Con las colecciones el intérprete de Python se volvería loco.

Podemos ver tambien el tipo de la lista

In [9]:

```
type(mixture)
```

Out[9]:

```
list
```

Calcular la longitud de la misma mediante el método *built-in* ya visto: `len()`

In [10]:

```
len(mixture)
```

Out[10]:

```
4
```

Accedemos a los elemenos de la lista mediante corchetes `[]`

Importante. El primer elemento es el 0

In [12]:

```
mixture[0]
```

Out[12]:

```
'asddd'
```

Metodos en Listas

Para el tipo de objeto lista, también hay una serie de métodos catacterísticos que nos permiten operar con ellas: añadir valores, quitarlos, indexado, filtrado, etc... En este enlace puedes encontrar todos los métodos que podrás usar con listas.

In [30]:

```
# "append" para añadir elemento a la lista
mixture.append(True)

# "index" para buscar en qué posición está un elemento
mixture.index(False)
# "count" para contar cuantas veces sale un elemento
mixture.count(mixture)
# "clear" para limpiar una lista
mixture.clear()
```

Ejercicio de listas

1. Crea una lista con tus 3 películas favoritas.
2. Imprime por pantalla la longitud de la lista
3. Añade a esta lista otra lista con tus 3 series favoritas

In [28]:

```
#Lista de peliculas
peliculas=["Jumanji","El libro de la selva","Aladdin"]
#Imprimir longitud de la lista
print(len(peliculas))
#Añadir lista con tres series
peliculas.append(["Gambito de dama","El juego del calamar","The walking dead"])
```

```
3
```

7. Resumen

In [ ]:

```
# Operaciones matemáticas
```

```python
# Operaciones matemáticas
print("Operaciones matemáticas")
print(4 + 6)
print(9*2)
print(2 * (3 + 5))
print(10/5)
print(10 % 3)
print(2**10)

# Funciones matemáticas más complejas
import math
print(math.sqrt(25))


# Operaciones comparativas
print("\nOperaciones comparativas")
print("AAA" == "BBB")
print("AAA" == "AAA")
print(1 == 1)
print(1 == 1.0)
print(67 != 93)
print(67 > 93)
print(67 >= 93)


# Operaciones con booleanos
print("\nOperaciones con booleanos")
print(True and True and False)
print(True or True or False)
print(not False)


# Funciones builtin
print("\nFunciones built-in")
string_builtin = "Fin del notebook"
print(string_builtin.upper())
print(string_builtin.lower())
print( string_builtin.replace("o", "O"))
print(string_builtin.replace("o", ""))


# Listas
print("\nListas")
musica = ["AC/DC", "Metallica", "Nirvana"]
musica.append("Queen")
print(musica)
```