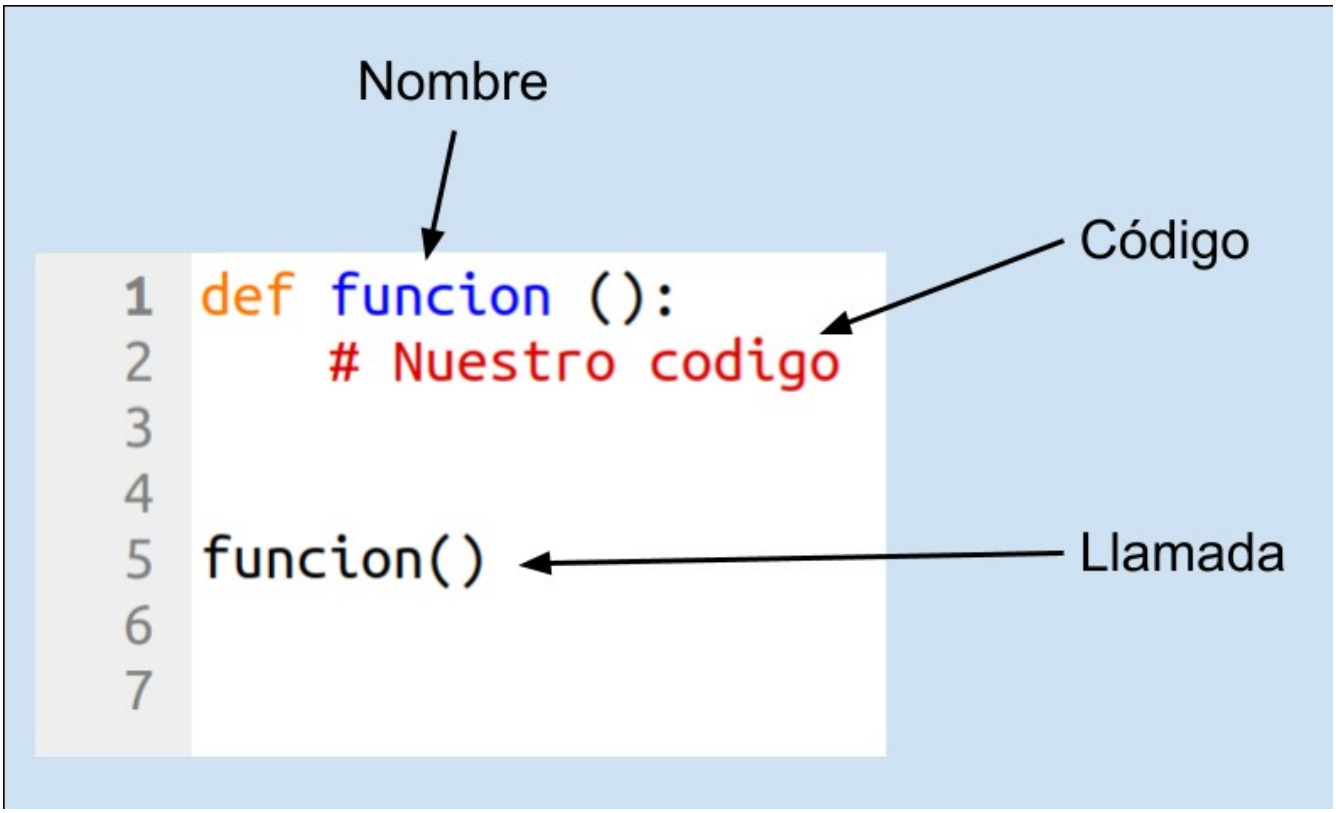
FUNCIONES EN PYTHON



Una función es una estructura que nos va a permitir usar varias veces un bloque de código sin tener que repetirlo.

La función tendrá un nombre y antes de usarla tenemos que definirla, lo que se hace precediendo el nombre de la función de la palabra reservada **def** y tras unos paréntesis pondremos "**:**" que nos indican que a continuación sigue el bloque de código de la función. Pondremos el bloque de código indentado para dejar claro las líneas que lo forman.

Para usar la función, sólo tenemos que llamarla, lo que se hace simplemente poniendo su nombre y los paréntesis.

Veamos un ejemplos sencillo de una función llamada "saludo" y que nos mostrará nuestro famoso "Hola Python":

```
def saludo():
    '''
    Muestra un saludo en pantalla
    '''
    print('Hola Python')
```

La ejecutaremos con:

```
>>> saludo()
```

```
>>> Hola Python
```

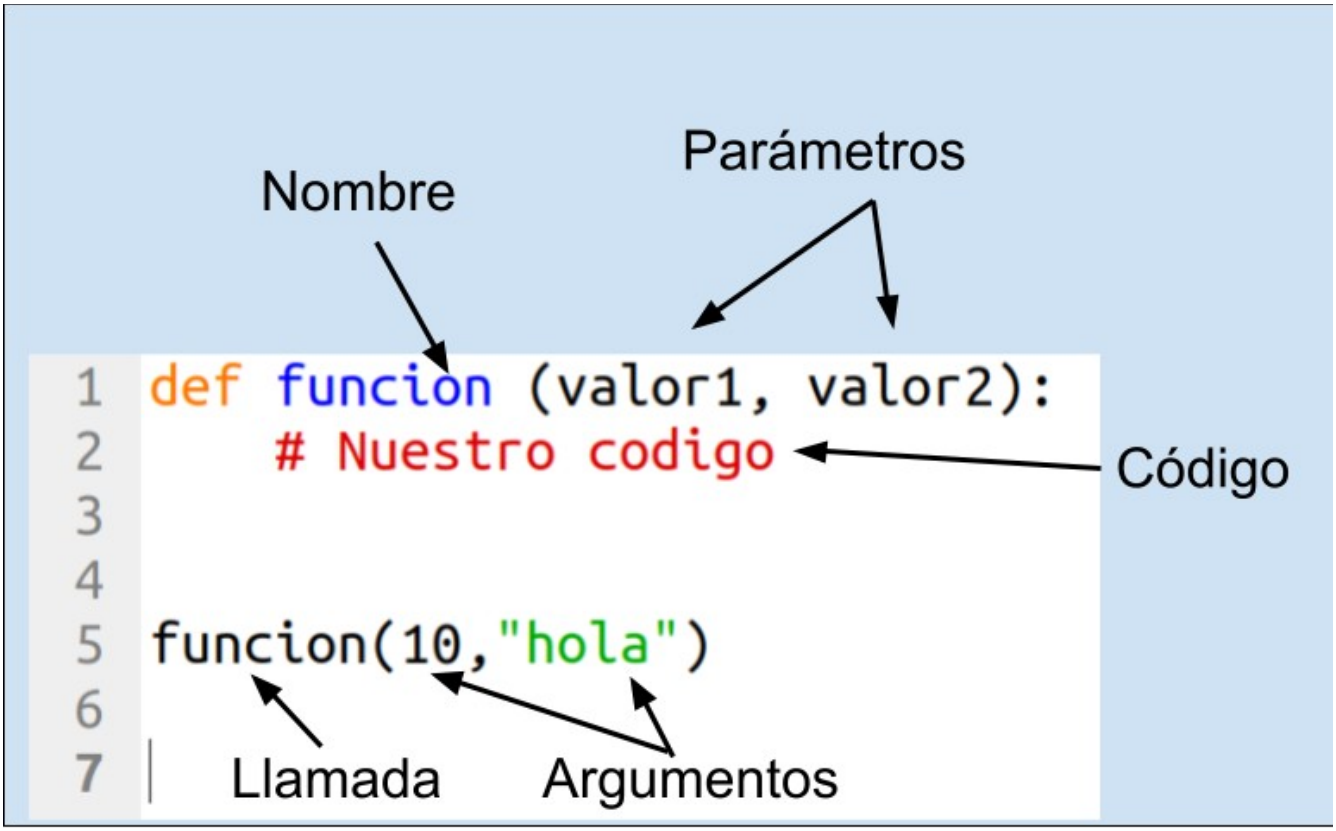
Vamos a documentar la función, para lo que incluiremos un comentario extenso al principio, además de incluir información de su uso, permitimos acceder a esa información desde el comando **help**. Ahora si usamos **help(saludo)**, no dará la información que hemos incluído en el comentario.

Parámetros y argumentos

Ya que tenemos una función que saluda, sería una buena idea que nos permitiera un poco más de flexibilidad al saludar, permitiéndonos hacer un saludo a cualquier persona.

Para ello vamos a añadir lo que se conoce como un **parámetro**, que es una variable que incluímos entre los paréntesis de la definición y que podremos usar dentro de nuestro código.

Llamamos **parámetro** a las variables que usamos en la función y **argumentos** a los valores que damos a esas variables cuando llamamos a la función para que esta se ejecute.



Veamos un ejemplo de una función con un parámetro y como para llamarla, tenemos que pasarle un argumento:

```
def saludo(nombre):
    '''
    Muestra un saludo en pantalla
```

```
    param nombre: a quien saludamos
    '''
    print(f'Hola {nombre}')

>>> saludo('Juan')
```

Argumentos no obligatorios

A veces queremos evitar tener que dar valor a un parámetro necesariamente, pero sí que queremos tener la flexibilidad de poder usar distintos argumentos.

En este caso podemos hacer que los parámetros tengan un valor por defecto que se usará si no damos ningún argumento. Lo indicaremos dando un valor al parámetro en la definición.

Nuestra función saludo quedaría así, si le ponemos al parámetro nombre, el valor "Python" por defecto:

```
def saludo(nombre = 'Python'):
    '''
    Muestra un saludo en pantalla

    param nombre: a quien saludamos
    '''
    print(f'Hola {nombre}')

>>> saludo('Juan')
>>> Hola Juan
>>> saludo()
>>> Hola Python
```

Funciones en Python

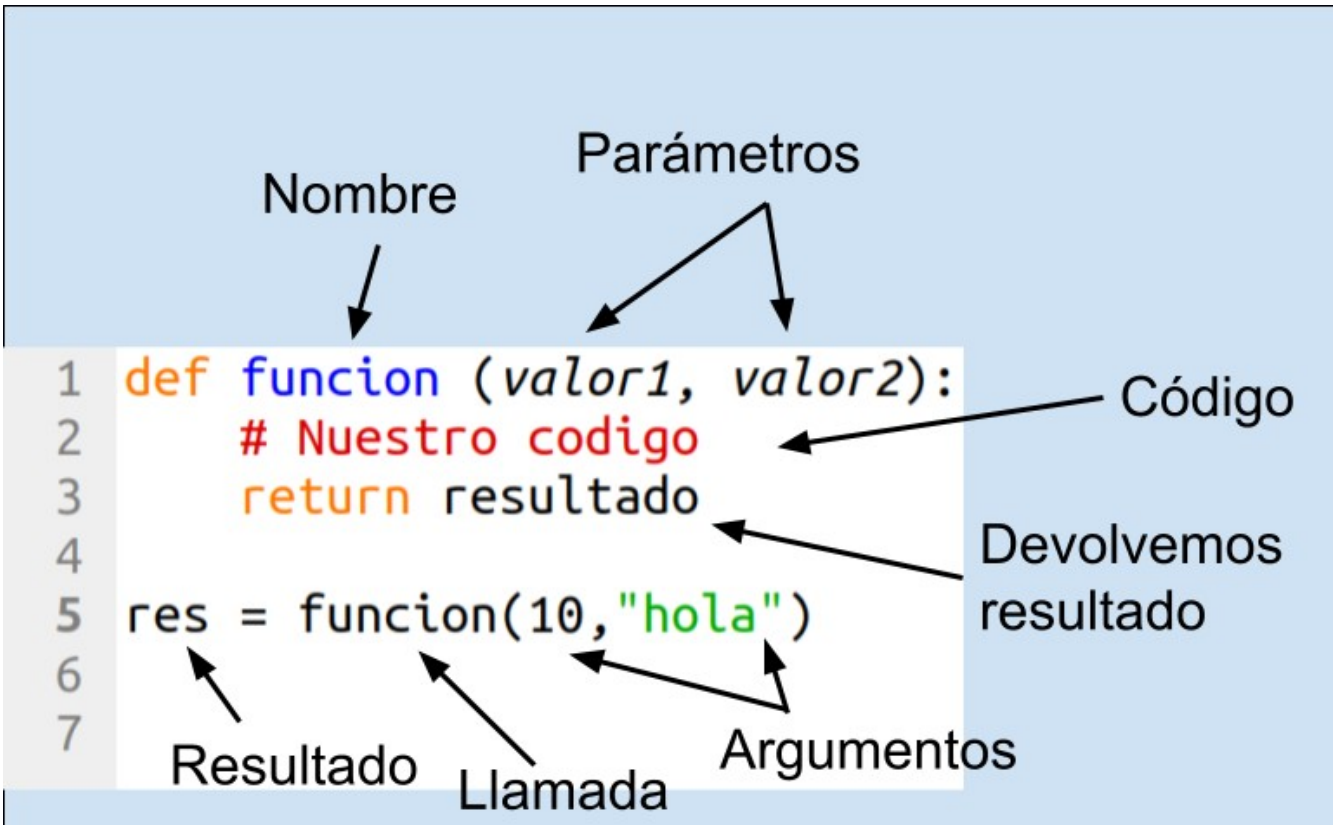
Ahora que sabemos lo que son, nos damos cuenta de que llevamos tiempo usando funciones en Python, como son "print", "input", "len", help, ...

De hecho a lo largo del curso, vamos a seguir viendo multitud de funciones y muchas más, que crearemos nosotros.

Funciones que devuelven valores

Algo muy frecuente es que una función devuelva un valor, un ejemplo puede ser la función **input()** o la función **len()** que ya hemos utilizado.

Para devolver un valor, sólo tenemos que añadir como última sentencia de la función la palabra reservada **return** seguida del valor que queremos devolver.



Vamos a hacer como ejemplo una función que devuelve la suma de 2 valores:

```
def suma(sumando1, sumando2):
    '''
    suma: devuelve la suma de 2 valores

    param sumando1: primer sumando
    param sumando2: segundo sumando

    devuelve el resultado de sumar los dos valores
    '''
    resultado = sumando1 + sumando2
    return resultado
```

También podemos usar la sentencia *return* para salir de una función en cualquier momento. Aunque la función no devuelva ningún valor podemos usarla:

```
def funcion1():
    while True:
        # nuestro codigo
        if error:
            print('Se ha producido un error')
            return # saldrá automáticamente de la función
```

Indicación de Tipos en argumentos

Como ya vimos, a partir de la versión 3 de Python, podemos indicar el tipo que va a tener una variable. También podemos indicar el tipo que tendrá un argumento y el tipo de variable que devolverá la función.

```
>>> def saludo(nombre: str) -> str:
>>> # Esperamos un argumento de tipo str y devolvemos un str
>>>     return 'Hola ' + nombre
>>> saludo('pepe')
'Hola pepe'
>>> print(saludo('pepe'))
Hola pepe
>>> print(saludo(7))
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
  File "<pyshell>", line 2, in saludo
TypeError: must be str, not int
```

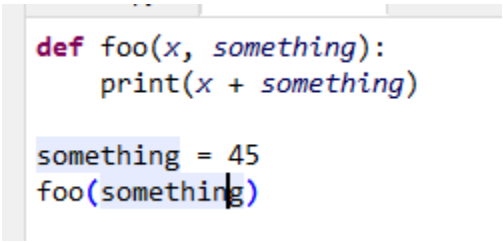
Como vemos en el ejemplo, en el caso de que no sea válido el tipo, se produce una *Exception* de tipo *TypeError*.

VARIABLES LOCALES Y VARIABLES GLOBALES

Las variables que llamamos **globales** son aquellas que están definidas en todo el ámbito del programa y que nos van a permitir utilizarlas en cualquiera de las funciones.

Por defecto todas las variables que declaramos en funciones son **locales**, es decir, solamente están definidas dentro de la función en la que estamos trabajando. Podemos utilizar una de estas variables como global, sin más que anteponer la palabra **global** al nombre de esta variable.

En esta imagen vemos que Thonny nos muestra, resaltando, lo que entiende que son una misma variable:



La zona donde podemos usar una variable se llama **ámbito** (scope en inglés) y para evitar errores, es muy importante tener claro cuál es en cada caso.

Vamos a ver todo esto con un ejemplo donde, aunque tenemos 2 variables llamadas igual, no son la misma, si no que la variable que está en la función, actúa de modo local, diremos que es una **variable local** y su ámbito será la función.

La variable que está definida fuera de la función diremos que es una **variable global** y que se puede usar en cualquier parte del programa.

```
def saludo(nombre = 'Python'):
    '''
    Muestra un saludo en pantalla

    param nombre: a quien saludamos
    '''
    print(f'Hola {nombre}')

nombre = 'Pepe'
saludo('Juan')
print(nombre)
```

Si abrimos la pestaña **Variables**, veremos que Thonny identifica a una de ellas como local y a otra como global.

Para que Python interprete las dos como una misma variable debemos indicarlo incluyendo dentro de la función la referencia **global** delante de la variable idioma:

```
def saludo(nombre = 'Python'):
    '''
    Muestra un saludo en pantalla

    param nombre: a quien saludamos
    '''
    global idioma
    idioma = 'en'
    if  idioma =='es':
        print(f'Hola {nombre}')
    else:
        print(f'Hello {nombre}')

idioma = 'es'
saludo('Juan')
```

En este caso también tenemos la alternativa de hacer que la función **saludo** tenga un parámetro "idioma" que tiene un **valor por defecto**:

```
def saludo(nombre = 'Python', idioma = 'en'):
    '''
    Muestra un saludo en pantalla

    param nombre: a quien saludamos
    '''


    if  idioma =='es':
        print(f'Hola {nombre}')
    else:
        print(f'Hello {nombre}')

idioma = 'es'
saludo('Juan',idioma)
saludo(nombre='Manolo', idioma='en')
```

```
saludo(idioma='es', nombre='Felipe')
```

Como ya hemos visto, dado que los parámetros tienen nombre, podemos usarlos en el orden que queramos, siempre que indiquemos el parámetro al que queremos dar cada argumento.