17-funciones

May 28, 2020

1 Funciones

Una función es un bloque de código con un nombre asociado, que recibe cero o más argumentos como entrada, sigue una secuencia de sentencias, la cuales ejecuta una operación deseada y devuelve un valor y/o realiza una tarea, este bloque puede ser llamados cuando se necesite.

Para crear un función en python debemos usar la sentencia `def`. La sintaxis para en función en python sería:

```
def nombre_función(parámetros):
    """documentación de la función"""
    sentencia
    return [expresion]
```

nombre_función: es el nombre que debemos usar cada vez que vamos a usar la función. **parámetros:** es la lista de parámetros que puede recibir una función. **sentencia:** bloque de código que realiza cierta operación. **return:** es la sentencia *return* en código python. **expresión:** es la expresión o variable que devuelve la sentencia return.

1.1 Ejemplo

crear un función que reciba dos parámetros y realice la suma de estos dos datos.

[1]:
```
def sumar(a, b):
    """función creada para realizar la suma de dos (2) números"""
    return a + b
```

Ahora vamos a guardar el resultado de esta operación en una variable para posteriormente mostrar la su valor

[2]:
```
# guardar el resultado de la operación en una variable
operacion = sumar(33, 57)
# mostrar el resultado en la consola
print(f"33 + 57 = {operacion}")
```

```
33 + 57 = 90
```

1.1.1 Parámetros por posición

Cuando enviá argumentos a una función, estos se reciben por orden en los parámetros definidos. Se dice por tanto que son argumentos por posición

[3]:
```
def suma(a, b):
    return a + b

suma(45, 60)
```

[3]: 105

En el ejemplo anterior el argumento `45` es la posición `0` por consiguiente es el parámetro de las función `a`, seguidamente el argumento `60` es la posición 1 por consiguiente es el parámetro de la función `b`.

1.1.2 Parámetros por nombre

Es posible evadir el orden de los parámetros si indica durante la llamada que valor tiene cada parámetro a partir de su nombre.

[4]:
```
def suma(a, b):
    return a + b

suma(b = 5, a = 45)
```

[4]: 50

Cuando llamamos una función y no le pasamos todos los argumentos provoca una exepción `TypeError`

[5]:
```
suma(4)
```

```
␣
↪---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call␣
↪last)

<ipython-input-5-09bad68fb336> in <module>
----> 1 suma(4)


TypeError: suma() missing 1 required positional argument: 'b'
```

[6]:
```
suma()
```

```
␣
↪---------------------------------------------------------------------------
```

```
TypeError                                 Traceback (most recent call␣
↪last)

<ipython-input-6-1f6ae598e611> in <module>
----> 1 suma()


TypeError: suma() missing 2 required positional arguments: 'a' and 'b'
```

1.1.3 Parámetros por defecto

Para solucionar la excepción `TypeError` ejecutada al momento de la llamada a una función sin argumentos, podemos asignar unos valores por defecto *nulos* o un *estaticos* a los parámetros.

[7]:
```
def suma(a, b=5):
    return a + b
```

[8]:
```
suma(45, 89)
```

[8]:
```
134
```

Observamos que podemos enviar ambos parámetros y reemplazamos el valor por defecto del parámetro `b`

[9]:
```
suma(45)
```

[9]:
```
50
```

Aquí podemos ver que si no enviamos el parámetro `b` la función no provoca la excepción `TypeError`, ya que por defecto tiene la asignación de `5`

1.1.4 Argumentos indertiminados por posición

En ocasión no sabemos previamente cuantos elementos necesitas enviar a una función. En estos casos puede utilizar los parámetros indeterminados por posición y por nombre.

[10]:
```
def indeterminado_posicion(*args):
    for arg in args:
        print(arg)

indeterminado_posicion("hola", ["manzana", False, "pera", 48.9, "mandarina"],␣
↪(25, 89, 89.0, True, "nombre"), 78.9, False)
```

```
hola
['manzana', False, 'pera', 48.9, 'mandarina']
(25, 89, 89.0, True, 'nombre')
78.9
False
```

1.1.5 Argumentos indeterminados por nombre

Para recibir un número indeterminado de parámetros por nombre (clave-valor o en inglés keyword args), debemos crear un diccionario dinámico de argumentos definiendo el parámetro con dos asteriscos.

[11]:
```
def indeterminado_nombre(**kwargs):
    for kwarg in kwargs:
        print(f"{kwarg} => {kwargs[kwarg]}")

indeterminado_nombre(nombre="Edwin", edad=33, sexo="m", lista=[1, 2, 3],␣
↪tupla=(False, True))
```

```
nombre => Edwin
edad => 33
sexo => m
lista => [1, 2, 3]
tupla => (False, True)
```