INFORMATORIO

Taller de Programación Web

Funciones

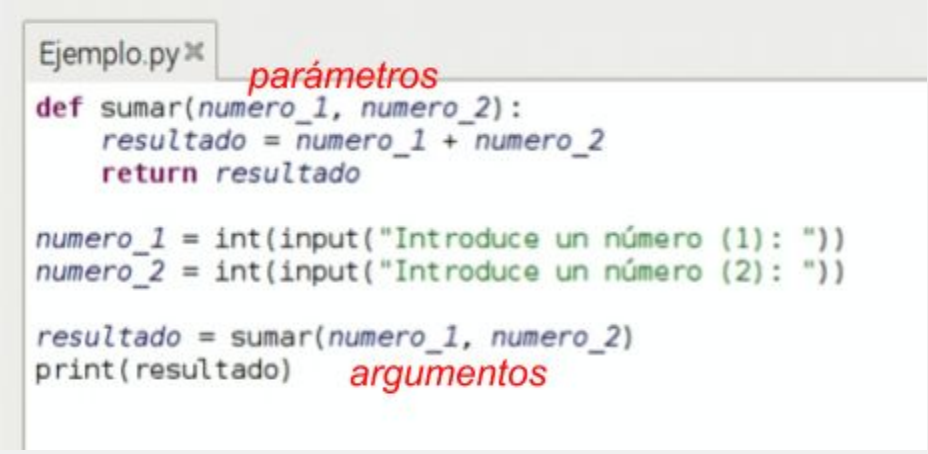
Subsecretaría de Empleo
Chaco Gobierno de todos

Ministerio de Producción, Industria y Empleo
Chaco Gobierno de todos

CHACO
Gobierno de todos

Argumentos y parámetros

Al definir una función los valores que se reciben se denominan **parámetros**, pero durante la llamada los valores que se envían se denominan **argumentos**.



- Por posición

Cuando se envían argumentos a una función, estos se reciben por orden en los parámetros definidos.
La asociación de los parámetros y los valores pasados a la función se hace normalmente de izquierda a derecha: como, en el ejemplo anterior, la función *sumar* recibe el valor de la variable *numero_1* y la suma con el valor que recibe en segundo lugar, *numero_2*.
En este caso se dice que son **argumentos por posición**.

- Por nombre

Sin embargo también es posible modificar el orden de los parámetros si indicamos el nombre del parámetro al que asociar el valor a la hora de llamar a la función:

```
sumar(numero_2 = 2, numero_1=3)
```

- Llamada sin argumentos

El número de valores que se pasan como parámetro al llamar a la función tiene que coincidir con el número de parámetros que la función acepta según la declaración de la función.
Si al momento de llamar una función la cual tiene definidos unos parámetros no pasa los argumentos correctamente provocará una excepción *TypeError*. Por ejemplo:

Código

```
Imprimir los valores pasados como parámetro.

def mi_funcion(param1, param2):
        """Esta función imprime los dos valores pasados como parámetros"""
    print(param1)
    print(param2)

mi_funcion("Hola")
```

Resultado

```
Traceback (most recent call last):
  File "<input>", line 5, in <module>
TypeError: mi_funcion() missing 1 required positional argument: 'param2'
```

- Parámetros por defecto

Para solucionar la excepción `TypeError` ejecutada al momento de la llamada a una función sin argumentos, entonces se puede asignar unos **valores por defecto nulos** a los parámetros, de esa forma puede hacer una comprobación antes de ejecutar el código de la función:

Código

```
Dado dos números calcular la resta de ambos números. Si al llamar a la función se pasa solo un parámetro imprimir un mensaje informando que se requieren dos números para realizar la operación.

def resta(num1=None, num2=None):
      """Esta función imprime la resta de los dos valores pasados como parámetros"""
    if a == None or b == None:
        print("Error, debes enviar dos números a la función")
        return
    return a - b

resta(30, 10)

resta()
```

Resultado

```
20
Error, debes enviar dos números a la función
```

Como se puede ver en el código anterior, se indica el final de la función luego de la sentencia print, usando la sentencia return aunque no devuelva nada, ya que en este caso se utiliza para finalizar la ejecución de la llamada a la función y "devolver" el resultado que en este caso no está definido por lo que será el valor por defecto (None).
Tener en cuenta que las declaraciones posteriores a las declaraciones de devolución no se ejecutan.

Los valores de un parámetro por defecto pueden ser de cualquier tipo, así que retomando el ejemplo de mi_funcion, podemos asignar diferentes valores por defecto a los parámetros:

Código

```
Imprimir los valores pasados como parámetro.

def mi_funcion(param1="Hola", param2=0):
        """Esta función imprime los dos valores pasados como parámetros"""
    print(param1)
    print(param2)

mi_funcion()
```

Resultado

```
Hola
0
```

- Argumentos indeterminados

En alguna ocasión no podremos determinar previamente cuantos elementos se necesita enviar a una función. En estos casos se puede utilizar los **argumentos indeterminado**s por posición y por nombre.

  - Por posición

Se debe crear una lista dinámica de argumentos, es decir, un tipo tupla, definiendo el parámetro con un asterisco, para recibir los parámetros indeterminados por posición:

Código

```
Imprimir los valores pasados como parámetro.

def indeterminados(*args):
    """Esta función imprime los valores pasados como parámetro"""
    for arg in args:
        print(arg)


indeterminados(5, “Hola”, [1, 2, 3, 4, 5])
```

Resultado

```
5
Hola
[1, 2, 3, 4, 5]
```

Esta sintaxis funciona creando una tupla en la que se almacenan los valores de todos los parámetros extra pasados como argumento. Para el caso de que se llame a la función sin parámetros, la tupla estaría vacía, por lo tanto no se imprimiría nada. En la llamada a la función que se hace en el ejemplo se crea una tupla con los valores (5, "Hola", [1, 2,3, 4, 5]).

- Por nombre

Para recibir un número indeterminado de parámetros por nombre (clave-valor o en inglés keyword args), se debe crear un diccionario dinámico de argumentos definiendo el parámetro con dos asteriscos. Las claves de este diccionario serían los nombres de los parámetros indicados al llamar a la función y los valores del diccionario, los valores asociados a estos parámetros:

Código

```
Imprimir los valores pasados como parámetro.

def indeterminados(**kwargs):
    """Esta función imprime los valores pasados como parámetro"""
    print(kwargs)


indeterminados(n=5, c="Hola", l=[1, 2, 3, 4, 5])
```

Resultado

```
{'n': 5, 's': 'Hola Plone', 'l': [1, 2, 3, 4, 5]}
```

Al recibirse como un diccionario, puede iterarlo y mostrar la clave y valor de cada argumento:

Código

```
Imprimir los valores pasados como parámetro.

def indeterminados(**kwargs):
    """Esta función imprime los valores pasados como parámetro"""
    for kwarg in kwargs:
        print(kwarg, "=>", kwargs[kwarg])
```

```
indeterminados(n=5, c="Hola", l=[1, 2, 3, 4, 5])
```

Resultado

```
n => 5
c => Hola
l => [1, 2, 3, 4, 5]
```

- Por posición y nombre

Si requiere aceptar ambos tipos de parámetros simultáneamente en una función, entonces debe crear ambas colecciones dinámicas. Primero los argumentos indeterminados por valor y luego los cuales son por clave y valor:

Código

```
Imprimir los valores pasados como parámetros por posición y nombre.

def super_funcion(*args, **kwargs):
     """Esta función imprime los valores pasados como parámetro por posición y nombre"""
    total = 0
    for arg in args:
        total += arg
    print("Total", "=>", total)

    for kwarg in kwargs:
        print(kwarg, "=>", kwargs[kwarg])


super_funcion(50, -1, 1.56, 10, 20, 300, c="Hola", n=3)
```

Resultado


```
Total => 380.56
c => Hola
n => 3
```

Los nombres args y kwargs no son obligatorios, pero se suelen utilizar por convención. Muchos frameworks y librerías los utilizan por lo que es una buena práctica llamarlos así.