Python Programming

Módulo 1

Argumentos de longitud variable

Argumentos de longitud variable

Imaginemos que queremos una función `sumar()` como la de los ejemplos, pero en este caso necesitamos que sume todos los números de entrada que se le pasen, sin importar si son 3, 100 o 1000. Una primera forma de hacerlo sería con una lista:

```
def sumar(numeros):
    parcial = 0
    for n in numeros:
        parcial += n
    return parcial

sumar([10,0,5,4])

out: 19
```

La propuesta es válida, cumple con nuestro requisito, pero realmente no estamos trabajando con argumentos de longitud variable. En realidad, tenemos un solo argumento, una lista.

Por suerte para nosotros, Python tiene una solución muy potente y sencilla a la vez. Si declaramos un argumento con ***** (asterisco típicamente acompañado con el nombre *args*, pero no es obligatorio usar ese nombre).

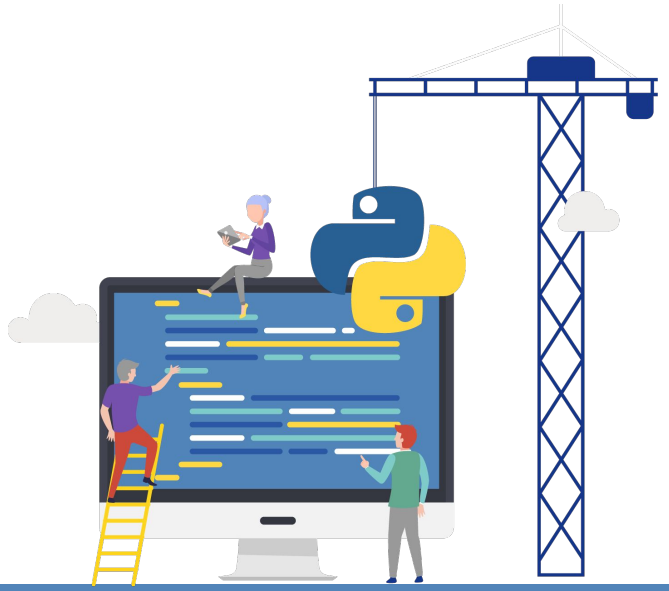
Al declarar la función con un parámetro *, hará que los argumentos, sean la cantidad que fueran, lleguen individualmente y que una vez recibido la función lo empaquete en una tupla de manera automática.

*Atención: No confundir ***** con los punteros en otros lenguajes de programación, no tiene nada que ver ¡En Python existen los punteros a memoria!*

```
def sumar(*args):
    print(type(args))
    print(args)
    total = 0
    for n in args:
        total += n
    return total

sumar(10,0,5,4)
```

```
out: <class 'tuple'>
out: (10,0,5,4)
out: 19
```



Usando doble **** es posible también tener como parámetro de entrada una cantidad arbitraria de elementos almacenados en forma de clave y valor (diccionario).

El doble asterisco ** (usualmente acompañado por el nombre *kwargs*) captura cualquier keyword argument que no haya sido definido junto con la función. Los argumentos capturados por este operador son almacenados, como dijimos, en un diccionario que tiene como claves los strings que representan los nombres de los argumentos, y como valor, el valor del argumento. Este operador debe ir al final de la definición de otros parámetros, si los hubiera.

Ejemplo:

```
def sumar(**kwargs):
      print(type(kwargs))
      print(kwargs)
      total = 0
      for n in kwargs:
            total+= kwargs[n]
      print(total)

sumar(a=5, b=20, c=23)

out: <class 'dict'>
out: {'a': 5, 'b': 20, 'c': 23}
out: 48
```

Orden para usar diferentes argumentos

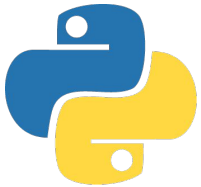
Si tuviéramos una función que hiciera uso de diferentes tipos de argumentos; deberíamos usarlos con un cierto orden.

Primero **argumentos posicionales** (argumentos comunes), después los a**rgumentos arbitrarios posicionales** (*args), a continuación, los **keywords arguments** (argumentos por defecto), y por último **número arbitrario de keywords arguments** (**kwargs).

En caso de usarlos de otra manera el resultado puede ser un error, o no poder hacer uso correcto del argumento por defecto.

Ejemplo:

```
def funcion(a,b,*args,c=100,**kwargs):
    print(a)
    print(b)
    print(args)
    print(c)
    print(kwargs)
```



¡Sigamos trabajando!