Universitat de Lleida
Escola Politècnica Superior

Jordi Virgili
jvirgili@diei.udl.cat

PROCESS MODELS (DEGREE IN COMPUTER SCIENCE)

BUENAS PRÁCTICAS EN PYTHON PT3.

ARGS Y KWARGS:

Gracias a los args y kwargs en Python podemos definir una función con un número variable de argumentos.

Vamos a suponer que queremos una función que sume un conjunto de números, pero no sabemos la cantidad. Podemos definir funciones que aceptan un número indeterminado de parámetros, adaptándose al número de argumentos con los que son llamados.

De hecho, el args viene de arguments en Inglés, o argumentos. No es necesario utilizar este nombre, solamente el puntoero a variable. Haciendo uso de *args en la declaración de la función podemos hacer que el número de parámetros que acepte sea variable.

```
def suma(*args):
    s = 0
    for arg in args:
        s += arg
    return s

suma(1, 3, 4, 2)
#Salida 10
```

En el ejemplo anterior hemos visto como *args puede ser iterado, ya que en realidad es una tupla. Iterando la tupla podemos acceder a todos los argumentos de entrada.

Al igual que en *args, en **kwargs el nombre es una mera convención. Puedes usar cualquier otro nombre siempre y cuando respetes el **. Con ** tenemos un diccionario en lugar de una tupla. Puedes verificarlo de la siguiente forma con type().

```
def suma(**kwargs):
    print(type(kwargs))
    suma(x=3)
#<class 'dict'>
```

A diferencia de *args, los **kwargs nos permiten dar un nombre a cada argumento de entrada, pudiendo acceder a ellos dentro de la función a través de un diccionario.

```
def suma(**kwargs):
  s = 0
  for key, value in kwargs.items():
    print(key, "=", value)
    s += value
  return s

suma(a=3, b=10, c=3)
#Salida
#a = 3
```

```
#b = 10
#c = 3
#16
```

Es posible iterar los argumentos de entrada con items(), y podemos acceder a la clave key (o nombre) y el valor o value de cada argumento.

Por último, se pueden utilizar argumentos para los campos necesarios y *args y **kwargs para los optativos. El término Tuple Unpacking se refiere a reasignar los argumentos individuales de las funciones que la contienen.

```
def funcionHija(*args, **kwargs):
    for arg in args:
        print("args =", arg)
    for key, value in kwargs.items():
        print(key, "=", value)

def funcionPadre(*args, **kwargs):
   funcionHija(*args, **kwargs)

args = [1, 2, 3, 4]
kwargs = {'x':"Hola", 'y':"Que", 'z':"Ase"}

funcionPadre(*args, **kwargs)

#Salida
#a = 10
#b = 20
#args = 1
#args = 2
#args = 3
#args = 4
#x = Hola
#y = Que
#z = Ase
```

Como curiosidad, una tupla se puede desempacar de manera parcial, como en el ejemplo:

```
my_list = [1, 2, 3, 4, 5, 6]
a, *b, c = my_list
print(a)
print(b)
print(c)
#Salida
#1
#[2,3,4,5]
#6
```