**PYTHON: Intro to Functions**

Una función es un bloque de **código** organizado y reutilizable que se utiliza para **realizar una única acción relacionada**.

**\*args** como argumento hace que valga meter cuantos argumentos queramos dentro de la función. ¡OJO, todos sus elementos los devuelve en una Tuple, o List (según esté definido el argumento)!

**\*\*kwarg** como argumento hace que valga meter cuantos argumentos CON KEY queramos dentro de la función. ¡Te los devuelve en un dic!

→ he hecho una foto con el movil, añadirla

Reglas para determinar una función:

- El inicio de la función comienza con **def** y le sigue el **“nombre de la función”** y **“()”**

**→ def *functionname* ():**

- Cuando dentro de “()” tiene valores se llaman “arguments” que son valores que se podrán utilizar dentro de la propia función.

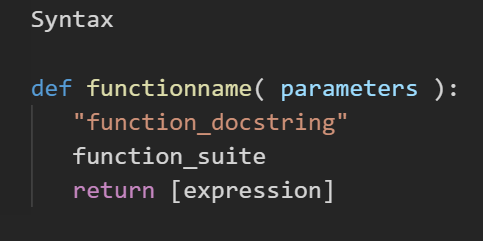
**→** def my\_function(**fname**):

print(fname + " Refsnes")

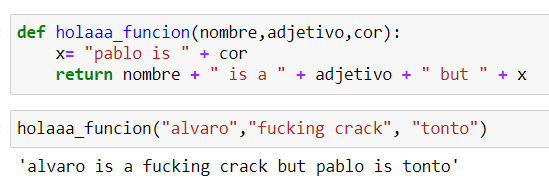
- Los dos puntos (colon) **“:”** son obligatorios.

- El primer statement de una función es opcional

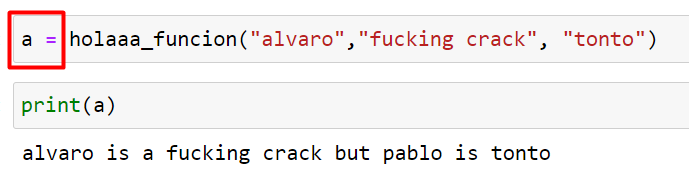
- El statement `**return [expression]**` devuelve el valor al que lo llama. Si no hay argumentos, el return es como devolver nada.



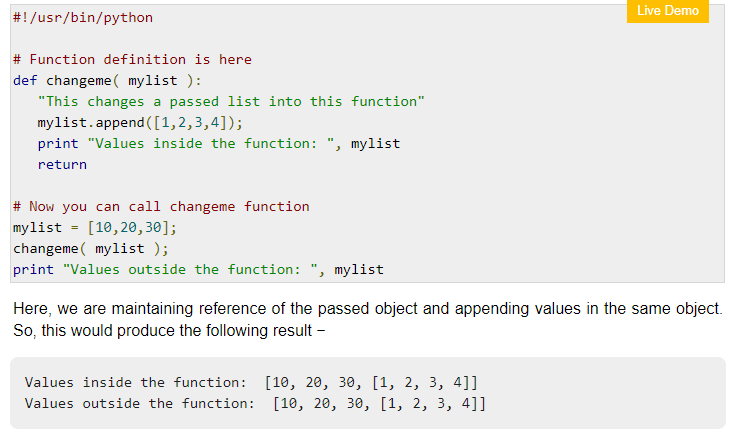
→ para que una función trabaje hay que llamarla, y si tiene argumentos indicar el valor de éstos.



→ Normalmente querremos hacer algo con esa función, en el siguiente ejemplo es popular una variable:



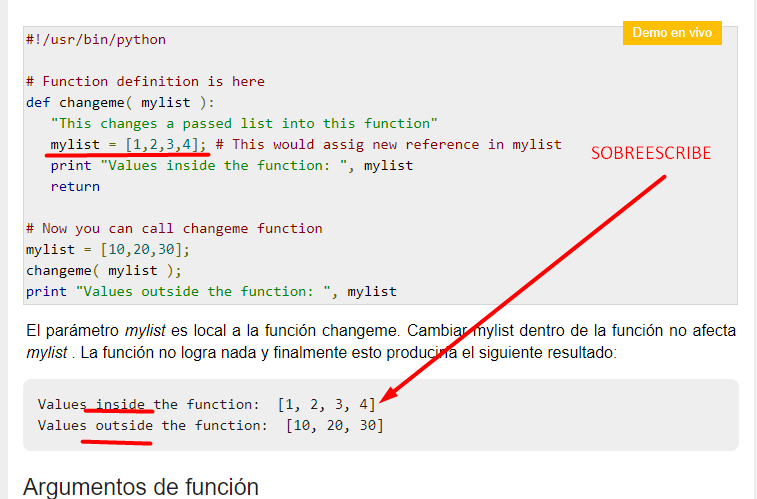
MÁS EJEMPLOS **(añadir valores a un listado)**



**(sobrescribir valores a un listado viendo lo que pasa fuera de la función)**

(sobrescribir valores a un listado, dentro de la función), ya que fuera de la función sigue tomando los valores de antes

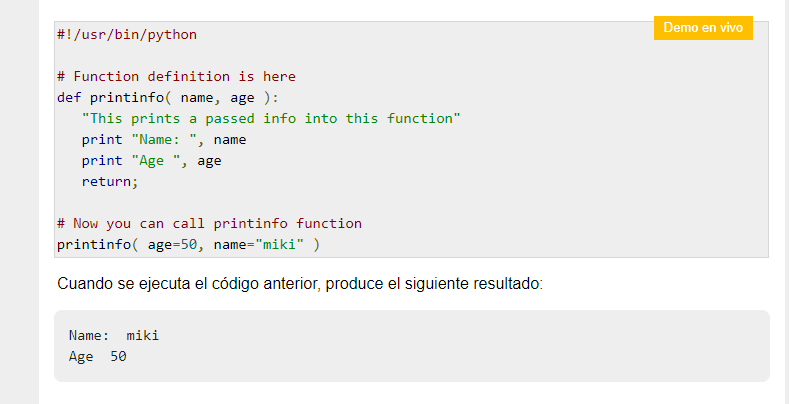
→ IMPORTANTE: La función ejecuta código dentro de ella



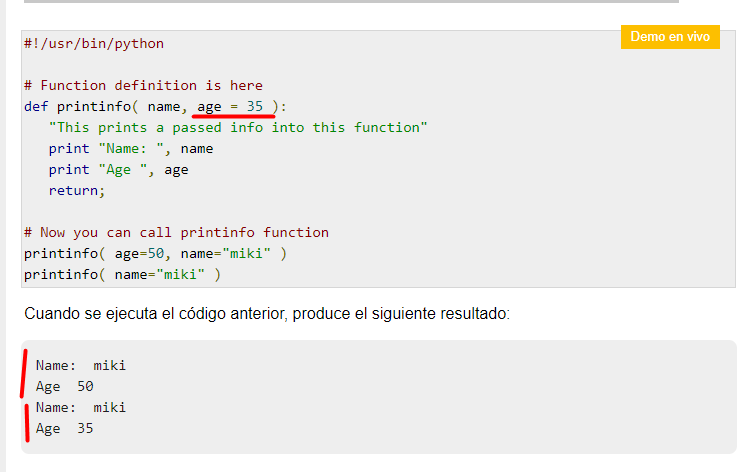
TIPOS DE ARGUMENTOS:

* PALABRAS CLAVE: Los argumentos de palabras clave están relacionados con las llamadas a funciones. Cuando utiliza argumentos de palabras clave en una llamada a función, el llamador identifica los argumentos por el nombre del parámetro.

Esto le permite omitir argumentos o colocarlos fuera de orden porque el intérprete de Python puede usar las palabras clave proporcionadas para hacer coincidir los valores con los parámetros. También puede realizar llamadas de palabras clave a la función printme () de las siguientes formas:

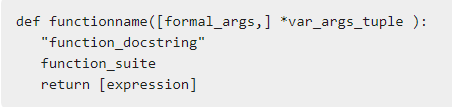


* ARGUMENTOS PREDETERMINADOS: Un argumento predeterminado es un argumento que asume un valor predeterminado si no se proporciona un valor en la llamada a la función para ese argumento. El siguiente ejemplo da una idea sobre los argumentos predeterminados, imprime la edad predeterminada si no se pasa:

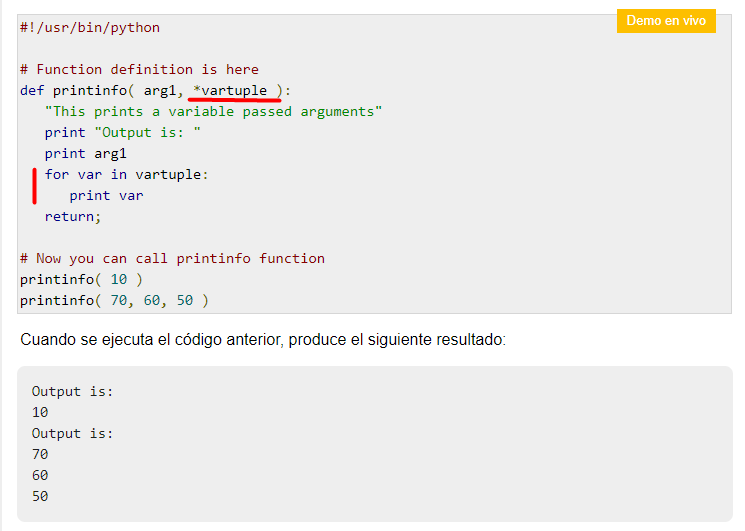


* ARGUMENTOS DE LONGITUD DE VARIABLE: Es posible que deba procesar una función para más argumentos de los que especificó al definir la función. Estos argumentos se denominan argumentos de longitud variable y no se nombran en la definición de la función, a diferencia de los argumentos obligatorios y predeterminados.

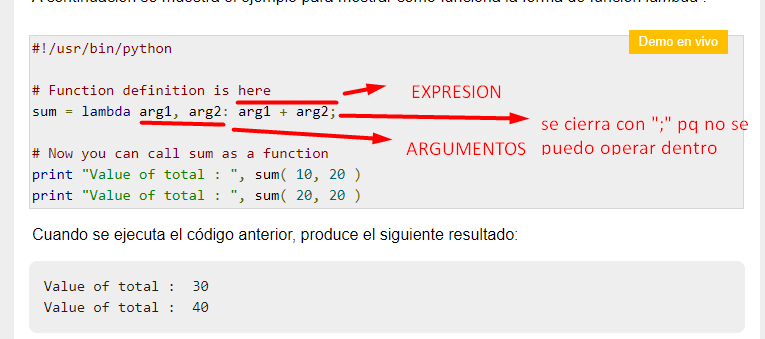
La sintaxis para una función con argumentos variables que no son palabras clave es la siguiente:



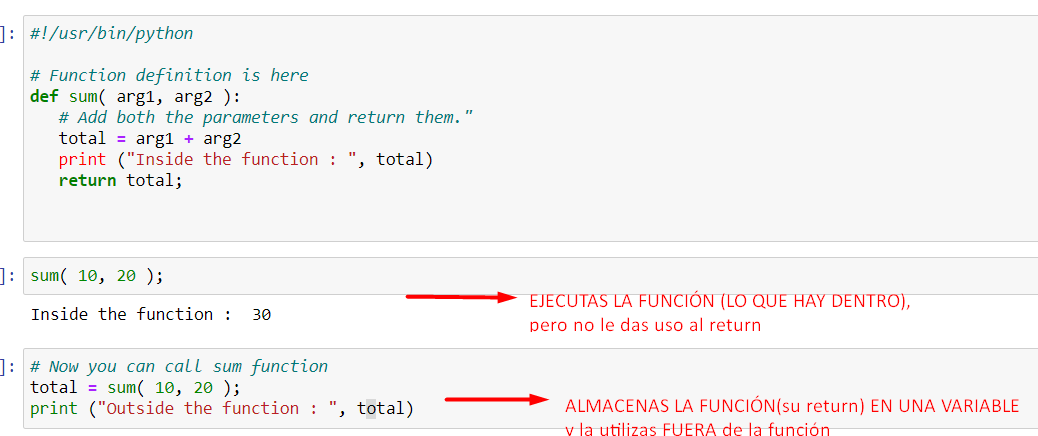
Se coloca un **asterisco (\*)** antes del nombre de la variable que contiene los valores de todos los argumentos de variables que no son palabras clave. Esta tupla permanece vacía si no se especifican argumentos adicionales durante la llamada a la función. A continuación se muestra un ejemplo simple:



* FUNCIONES ANÓNIMAS: Estas funciones se denominan anónimas porque no se declaran de la manera estándar mediante el uso de la palabra clave def. Puede utilizar la palabra clave **lambda** para crear pequeñas funciones anónimas.
* Una función anónima no puede ser una llamada directa a imprimir porque lambda **requiere una expresión**
* Las funciones de Lambda tienen su propio espacio de nombres local y **no pueden acceder a otras variables** **que no sean las de su lista** **de parámetros** y las del espacio de nombres global.



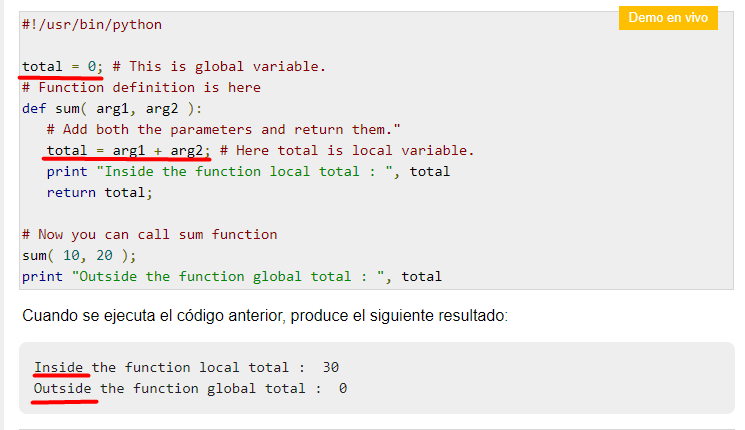
RETURN: En una función tu ejecutas cosas dentro de ella, pero podemos almacenar su return para utilizarlo FUERA de ella.



ALCANCE: variables GLOBALES vs LOCALES:

Las variables que se definen dentro de un cuerpo de función tienen un alcance local y las definidas fuera tienen un alcance global.

Esto significa que solo se puede acceder a las variables locales dentro de la función en la que están declaradas, mientras que todas las funciones pueden acceder a las variables globales en todo el cuerpo del programa. Cuando llamas a una función, las variables declaradas dentro de ella entran en el ámbito. A continuación se muestra un ejemplo simple:



¡OJO, si la vamos a utilizar la variable global como una operación dentro de la función, está no funciona! En el ejemplo anterior sí que funciona porque estamos volviendo a definir (referenciar) “total” como una suma de parámetros locales que sí que existen)!

a = 1

def nombre\_funcion(b)

a = a+b

return a

LANZAMOS nombre función(8) → no funciona pq estamos intentando REFERENCIAR “a” CON “a” (cuando localmente a no existe) - Si fuese solo usarla sí que se podría, pero referenciarla consigo mismo no

a = 1

def nombre\_funcion(b)

global a

a = a+b

return a

LANZAMOS nombre función(8) → funciona pq le hemos puesto un global a (si no daría error de referencia, pq localmente no existiría)

def nombre\_funcion(b)

c = a+b

return c

LANZAMOS nombre función(8) → así si funcionaría pq estamos referenciando otra variable.