

Funciones en Python - continuación

OPERADO POR:





RUTA DE APRENDIZAJE 1



Argumentos y parámetros

Al definir una función los valores que está recibe se denominan parámetros, pero cuando llamamos la función los valores que se envían se denominan argumentos.

Por posición:



Cuando enviamos los argumentos a una función, estos se reciben según el orden de los parámetros que definimos. Podemos decir entonces que son argumentos por posición. Posición 1 y 2 para el ejemplo:

```
def suma(a, b):
    return a + b
suma(30, 10)
```

En este ejemplo la función suma necesita de los parámetros a y b para realizar la operación, cuando llamamos la función, los argumentos 30 y 10 deben ir en la posición 1 y 2 respectivamente: a = 30 y b = 10.





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

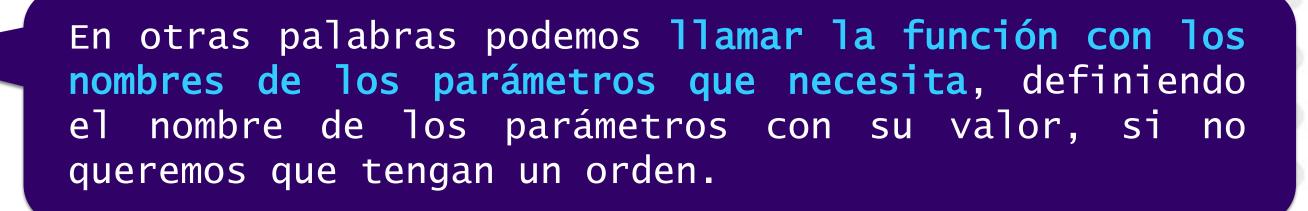
 def suma(a, b): return a + b

> definimos los b=30 valores a=10

suma(a, b)

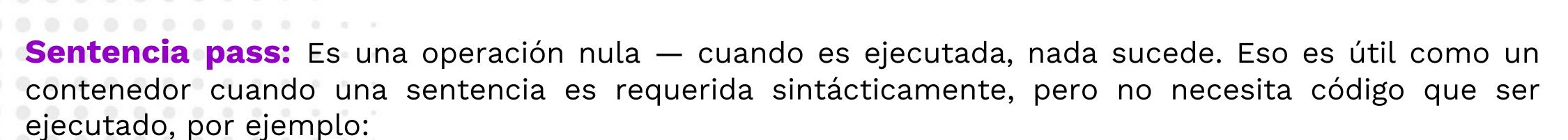
En este caso tenemos la función suma(a, b), pero función con sus respectivos nombres.













La sentencia pass es útil si queremos que en un caso especial el código no entre a la función o la función esta pendiente de ser construida en un futuro.

```
# una función que no hace nada (aun)
def consultar_nombre_genero(letra_genero): pass

type(consultar_nombre_genero)

consultar_nombre_genero("M")
```

Sentencia return: Las funciones pueden comunicarse con el exterior o con el proceso principal del programa usando la sentencia return.

Es la forma que tiene la función para comunicarse con el exterior y regresar valores.







A continuación, un ejemplo de función usando return:



```
M def otra_suma(numero1, numero2):
    print(numero1 + numero2)
    print("\n")

resultado = otra_suma(5,6)
print(resultado)
```

```
M def otra_suma(numero1, numero2):
    print(numero1 + numero2)
    print("\n")
    return numero1 + numero2

resultado = otra_suma(5,6)
print(resultado)
```

En el ejemplo, la función utiliza los dos argumentos, numerol y numero2 para hacer la suma de dos números, sin embargo necesitamos indicarle a la función que debe darnos una respuesta. Vamos a ver los dos casos, donde le pedimos la respuesta a la solución y el segundo caso donde no lo pidamos.

Usar el return es la única manera que tenemos de acceder a esa información, si no se adiciona dentro de la función, no tendremos acceso a ella.







```
def otra_suma(numero1, numero2):
    print(numero1 + numero2)
    print("\n")
resultado = otra_suma(5,6)
print(resultado)
11
None
```

```
M def otra_suma(numero1, numero2):
      print(numero1 + numero2)
      print("\n")
      return numero1 + numero2
  resultado = otra_suma(5,6)
  print(resultado)
  11
```

11

Al ejecutar el código el primer ejemplo no usa return al final de la función que creamos, por lo tanto, nos imprime None como solución, por que no puede comunicarse con el exterior. Cuando usamos return y le decimos que debe regresar una solución, el programa logra imprimir el resultado.







Sobre los parámetros: Un parámetro es un valor que la función espera recibir cuando sea llamada (invocada), a fin de ejecutar acciones en base al mismo.

def mi_funcion(nombre, apellido): # algoritmo

Una función puede esperar uno o más parámetros (que irán separados por una coma) o ninguno.

Los parámetros, se indican entre los paréntesis, como variables, y de esta forma usarlos con esa estructura dentro de la misma función.

Los parámetros que una función espera, serán usados como variables locales, a las cuáles solo la función podrá acceder:

```
    def mi funcion(nombre, apellido):

       nombre_completo = nombre, apellido
       print(nombre completo)
  mi_funcion('David','Alvarez')
```





▶ def mi funcion(nombre, apellido): nombre completo = nombre, apellido print(nombre completo) mi_funcion('David','Alvarez')



Si quisiéramos acceder a esas variables locales, fuera de la función, obtendríamos un error al ejecutar el código:

```
▶ def mi funcion(nombre, apellido):
      nombre completo = nombre, apellido
      print(nombre completo)
  # Retornará el error: NameError: name 'nombre' is not defined
  print(nombre)
```

```
Traceback (most recent call last)
<ipython-input-11-fd3d5d1686b5> in <module>
     5 # Retornará el error: NameError: name 'nombre' is not defined
---> 6 print(nombre)
```

NameError: name 'nombre' is not defined

Al llamar a una función, siempre se le deben pasar sus argumentos en el mismo orden en el que los espera. Pero esto puede evitarse, haciendo uso del paso de argumentos como keywords.











En Python, también es posible, asignar valores por defecto a los parámetros de las funciones. Esto significa, que la función podrá ser llamada con menos argumentos de los que espera:

```
M def saludar(nombre, mensaje='Hola'):
      print(mensaje, nombre)
  saludar('Gamer')
```

Hola Gamer

Keywords como parámetros

En Python, también es posible llamar a una función, pasándole los argumentos esperados, como pares de claves = valor:

```
def saludar(nombre, mensaje='Hola'):
    print(mensaje, nombre)
saludar(mensaje="Buen día", nombre="Juan")
Buen día Juan
```







Ejemplo 1:



Para este ejemplo, solo le estamos pidiendo un parámetro a la función para que haga su trabajo correctamente.

La función: my_function(fname), solo necesita del parámetro fname para hace r sus tareas, en este caso tomar el valor de fname y sumarle una cadena de s ímbolos "Gonzales".

```
M def my_function(fname):
    print(fname + Gonzales")
  my function("Emil")
  my function("Tobias")
  my_function("Linus")
```

Emil Gonzales Tobias Gonzales Linus Gonzales





Siempre debemos recordar, que dentro de la función, las instrucciones deben tener 4 espacios o la sangría para que Python pueda leerla correctamente.

Después de que terminamos de construir nuestra función solo debemos llamarla con los valores que necesitamos entre comillas si son símbolos o palabras, para el parámetro "fname" colocamos un nombre entre comillas.



```
def my_function(fname):
          print(fname + " Gonzales")
Sangrie
        my function("Emil")
        my function("Tobias")
        my_function("Linus")
        Tobias Gonzales Linus Gonzales Linus Gonzales
```





Ejemplo 2:





Es importante recordar que cada cadena de símbolos, o lo que también llama mos cadena de string, tiene un espacio para cada cosa.

En este ejemplo, vamos a decirle a la función que una o sume los dos parámetros que debemos entregarle, en este caso llamamos los parámetros fname y lname.

```
def my_function(fname, lname):
    print(fname + " " + lname)

my_function("Emil", "Henao")
```

Emil Henao

Si solo le damos como instrucción: print(fname + lanme) va a sumar los dos valores sin un espacio entre ellos. Para que exista un espacio debemos decirle que lo sume usando las dos comillas con un espacio entre ellas.: " ", como aparece en el ejemplo.







Ejemplo 3:

Si nosotros le decimos a nuestra nueva función que necesita de dos parámetros para hacer su tarea correctamente, no podemos solo entregarle un valor cuando llamemos la función.

```
def my_function(fname, lname):
    print(fname + " " + lname)

my_function("Emil")
```

Python nos dice que a la función le falta un argumento.

TypeError: my_function() missing 1 required positional argument: 'lname'

En el ejemplo solo le entregamos un nombre a la función: "Emil" y al ejecutar el código nos entrega un error.









OPERADO POR:



