



El futuro digital  
es de todos

MinTIC



# Funciones en Python - continuación

OPERADO POR:



Misión  
TIC 2022

ruta de aprendizaje 1





# Argumentos y parámetros

Al definir una función los valores que está recibiendo 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:

```
1 2  
▶ 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**.



## Por nombre:

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  
  
b=30  
a=10  
suma(a, b)
```

definimos los valores

En este caso tenemos la función `suma(a, b)`, pero vamos a **definir los valores de a y b fuera de la función** con sus respectivos nombres.

En otras palabras podemos **llamar la función con los nombres de los parámetros que necesita**, definiendo el nombre de los parámetros con su valor, si no queremos que tengan un orden.







**Sentencia pass:** Es una operación nula — cuando es ejecutada, nada sucede. Eso es útil como un contenedor cuando una sentencia es requerida sintácticamente, pero no necesita código que ser ejecutado, por ejemplo:



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**:



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

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

```
▶ 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, **numero1** 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

```
▶ 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)
```

```
-----  
NameError                                Traceback (most recent call last)  
<ipython-input-11-fd3d5d1686b5> in <module>  
      4  
      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:

```
▶ 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 hacer sus tareas, en este caso tomar el valor de `fname` y sumarle una cadena de símbolos "Gonzales".

```
► 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.

*Sangría*

```
▶ def my_function(fname):  
    print(fname + " Gonzales")  
  
my_function("Emil")  
my_function("Tobias")  
my_function("Linus")
```

Emil Gonzales  
Tobias Gonzales  
Linus Gonzales

*Llamar la función.*

## Ejemplo 2:



Es importante recordar que cada cadena de símbolos, o lo que también llamamos 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 + lname)` 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")
```

```
-----  
TypeError                                 Traceback (most recent call last)  
<ipython-input-4-49655f8043be> in <module>  
      2     print(fname + " " + lname)  
      3  
----> 4 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.





El futuro digital  
es de todos

MinTIC

**GRACIAS**

**OPERADO POR:**

