

Python / Guía 6

Funciones

## 

## 

## OBJETIVOS DE LA GUÍA

En esta guía aprenderemos a:

* Crear funciones y parámetros
* Invocar funciones con argumentos
* Entender el ámbito de las variables en una función
* Crear funciones recursivas
* Documentar funciones

# 

# Creación de funciones

La función **es un bloque de código definido por un nombre.** Usamos las funciones cuando necesitamos realizar la misma tarea varias veces sin volver a escribir el mismo código.

Considera un escenario en el que necesitas hacer alguna acción muchas veces. Puedes definir esa acción solo una vez con una función, y llamar a esa función siempre que sea necesario hacer la misma actividad.

La función mejora la eficiencia y reduce los errores, debido a la reutilización del código. Una vez que creamos una función, podemos llamarla en cualquier lugar y en cualquier momento. El beneficio de una función es la reutilización.

Python tiene dos tipos de funciones:

* **Las funciones incorporadas:** son las que vienen junto a Python. Algunas de ellas son print(), input(), range().
* **Las funciones definidas por el usuario.** Son producidas por un programador explícitamente.

Sigue los siguientes pasos para definir una función en Python:

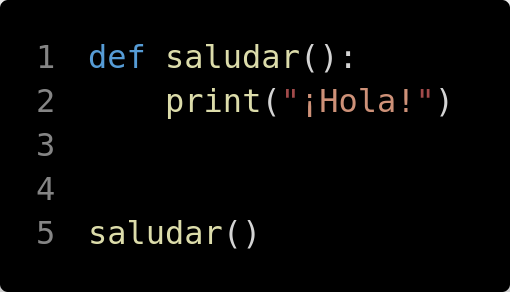
* Utiliza la palabra clave 'def' con el nombre de la función.
* Pasa los parámetros según necesites. Opcional.
* Escribe el cuerpo de la función con un bloque de código, usando la sangría de la indentación.

Su estructura lógica es la siguiente:

def <nombre>(<parámetro opcional>, ...):  
 <cuerpo de la función>  
 return <expresión opcional>

Para invocar (o "llamar") una función se ha de escribir su nombre, seguido de paréntesis. Tienes que invocar la función después de haber sido definida, si no obtendrás un error.

**Ejecuta el siguiente código:**



| Cuando invocas la función, no olvides los paréntesis, o verás la posición en memoria de la función, pero no la ejecutará. |
| --- |

# 

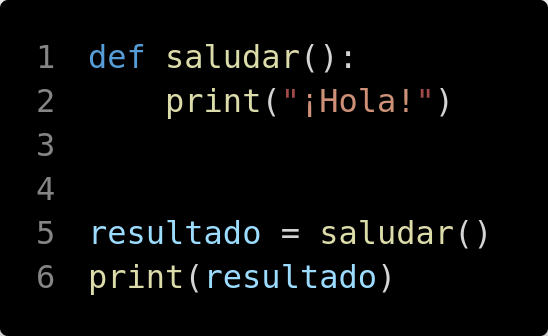
# Retorno de las funciones

## Sin devolución

Una función sin parámetros es el caso del ejemplo de arriba. Observa que no está la instrucción return. Simplemente, imprime algo, pero no devuelve nada.

¿Cómo podemos corroborar que esta función no devuelve nada?

**Ejecuta el siguiente código:**

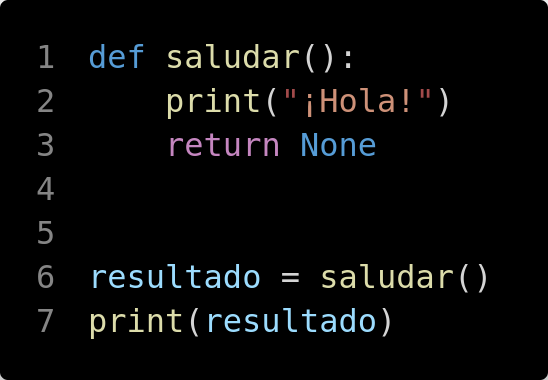


None es un objeto de Python que representa "nada", o "null" o "vacío". Es de mucha utilidad.

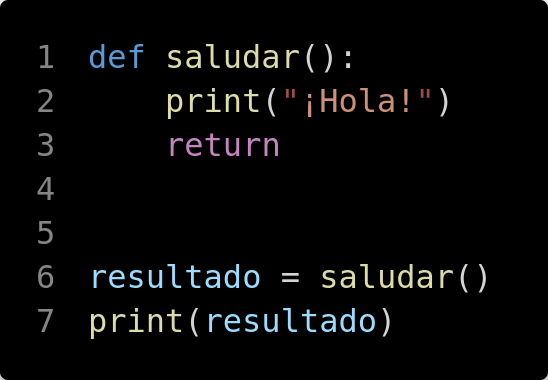
Prueba: print(type(None))

Cuando no escribas de forma explícita "return None", Python asume que la función de forma implícita que tú haces return None, que es lo mismo que un return solo.

**Ejecuta el siguiente código:**



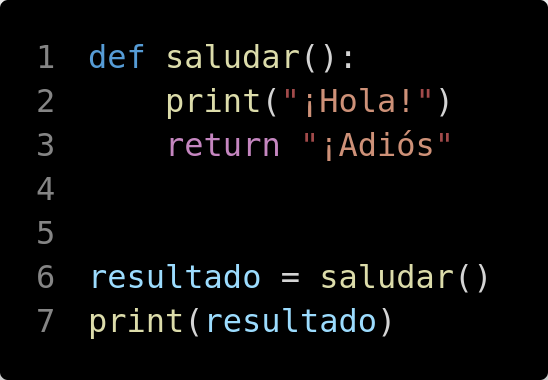
**Ejecuta el siguiente código:**



## Con devolución

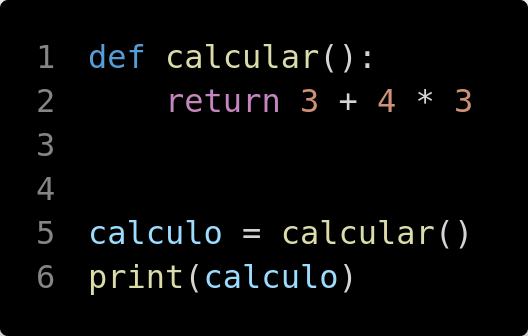
Devolución de tipos de datos primitivos:

**Ejecuta el siguiente código:**



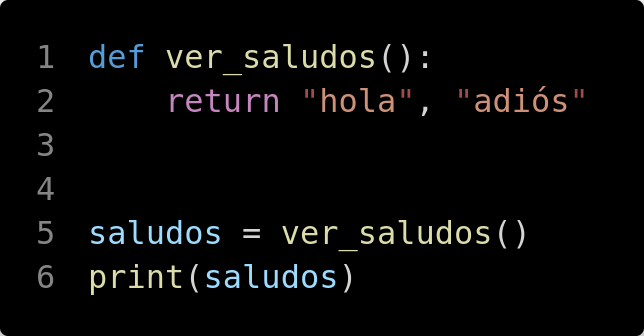
Devolución de alguna expresión o cálculo numérico:

**Ejecuta el siguiente código:**



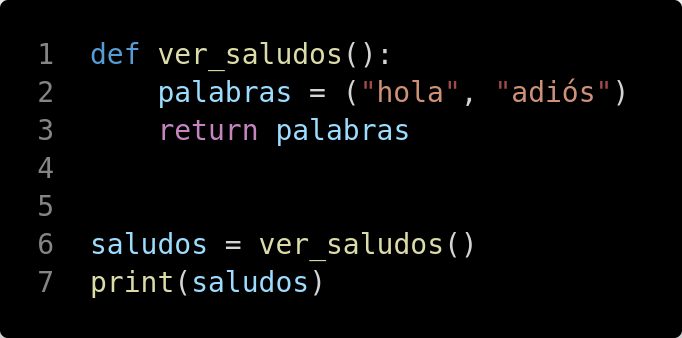
Devolución de varios valores (se hace en forma de tupla):

**Ejecuta el siguiente código:**



Devolución de datos, contenidos en variables:

Ejecuta el siguiente código:



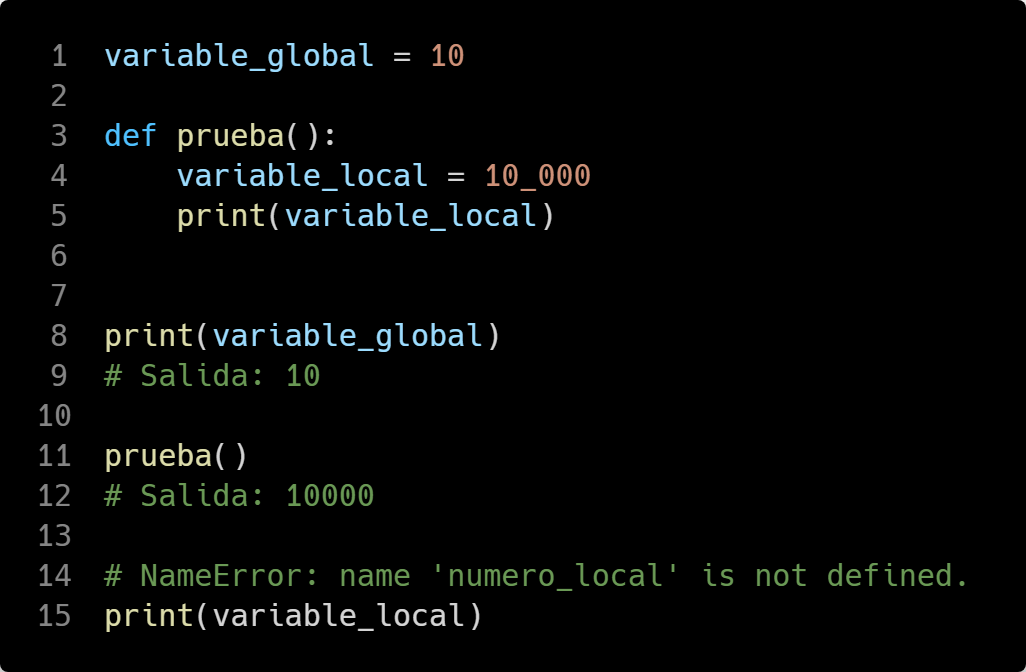
# 

# Ámbito de las variables

‎Cuando defines una función con ‎‎variables‎‎, el alcance de esas variables se limita a esa función. En Python, el ámbito de una variable es el área donde se declara una variable. ‎No podemos acceder a las variables locales desde fuera de la función. Debido a que el ámbito es local, esas variables no son visibles desde el exterior de la función.‎

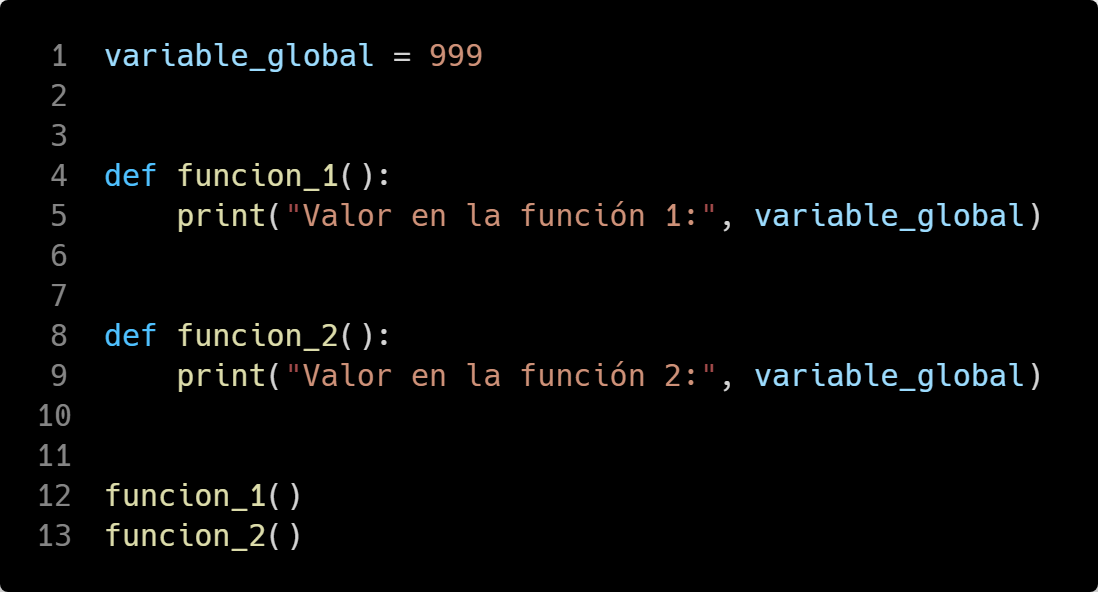
Cuando estamos ejecutando una función, las variables viven durante el tiempo de ejecución. Una vez que regresamos de la función, esas variables se destruyen. Por lo tanto, la función no recuerda el valor de una variable de su llamada anterior.‎

**Ejecuta el siguiente código:**



‎Una variable global es una variable que se declara fuera de una función. El alcance de una variable global es amplio: es accesible desde todas las funciones del mismo módulo.‎

**Ejecuta el siguiente código:**

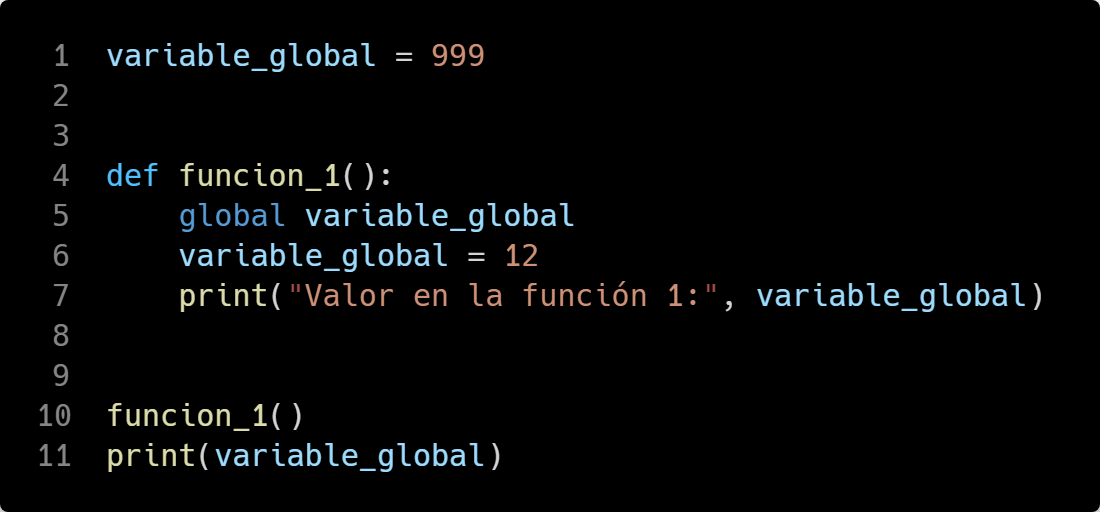


## 

## Global

global es la palabra clave utilizada para poder acceder o modificar una variable global.

**Ejecuta el siguiente código:**



**¡MANOS A LA OBRA!**

**Ejercicio 1**

1º Crea dos variables globales que tengan como valor algún número.  
2º Crea una función que se llame "media" y que imprima el promedio de las dos variables  
3º Invocar la función.

| Puedes crear variables globales arriba o abajo de las funciones, pues lo primero que se ejecuta es el ámbito global. |
| --- |

## Revisemos lo aprendido hasta aquí

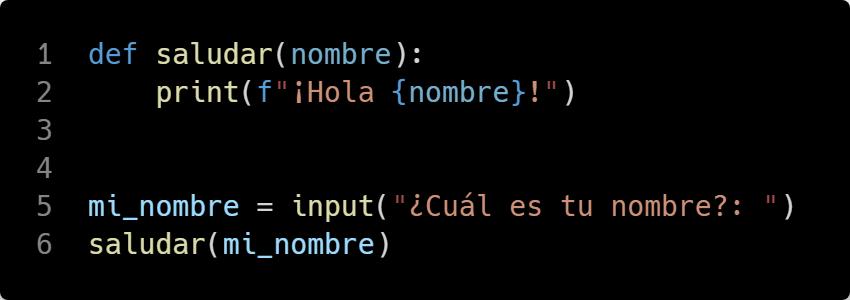
* Crear funciones sin devolución
* Crear funciones con devolución
* Entender el ámbito de las variables, ámbito global y local

# 

# Funciones con parámetros

Cuando una función tiene un parámetro o más, la función debe ser invocada con la misma cantidad de argumentos. Es decir, los parámetros son requeridos.

**Ejecuta el siguiente código:**

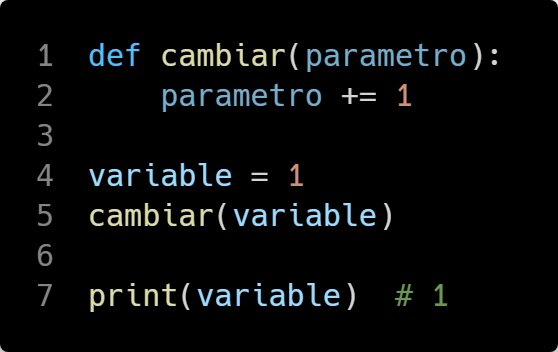


| Se llama **parámetro** a la variable que está dentro de la definición de la función (en este último ejemplo es 'nombre'). Y se llama **argumento** al valor que se envía desde la llamada de la función (es este último ejemplo es 'mi\_nombre') |
| --- |

## Argumentos de tipo inmutable

Cuando pasas un **tipo de dato inmutable:** (int, float, str, bool, tuple) se crea una copia local de la variable dentro de la función.

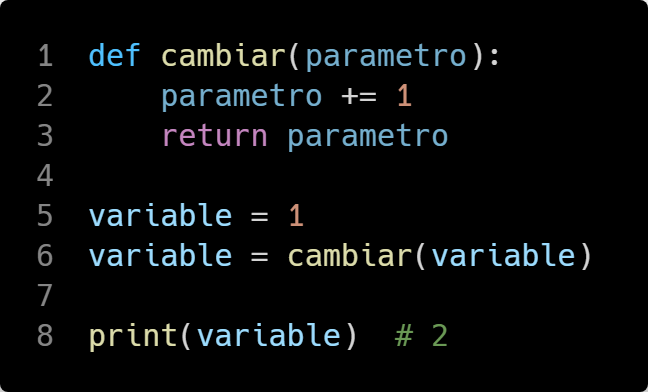
**Ejecuta el siguiente código:**



| Una forma muy útil de saber lo que pasa por debajo de Python, es haciendo uso de la función id(). Esta función nos devuelve un identificador único para cada objeto. Prueba con print(id(variable)) |
| --- |

Para poder modificarlo, debes devolverlo con 'return', y reasignarlo:

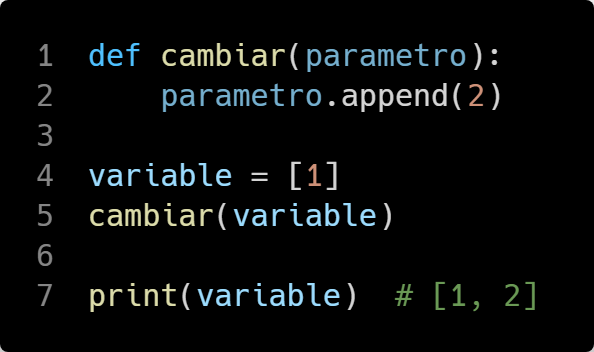
**Ejecuta el siguiente código:**



## Argumentos de tipo mutable

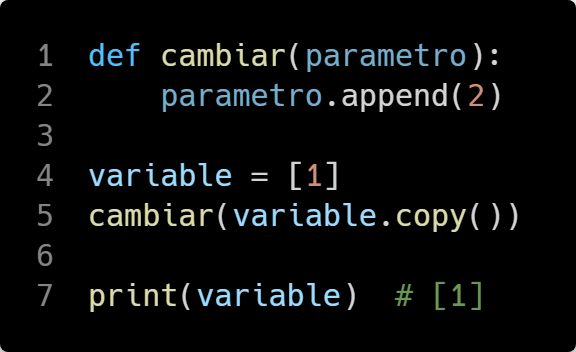
Cuando pasas un **tipo de dato mutable:** (list, set, dict) se maneja directamente la variable, y, en efecto, los cambios que haces dentro de la función la afectarán también afuera.

**Ejecuta el siguiente código:**



Para que no se puedan modificar, debes enviar una copia:

**Ejecuta el siguiente código:**

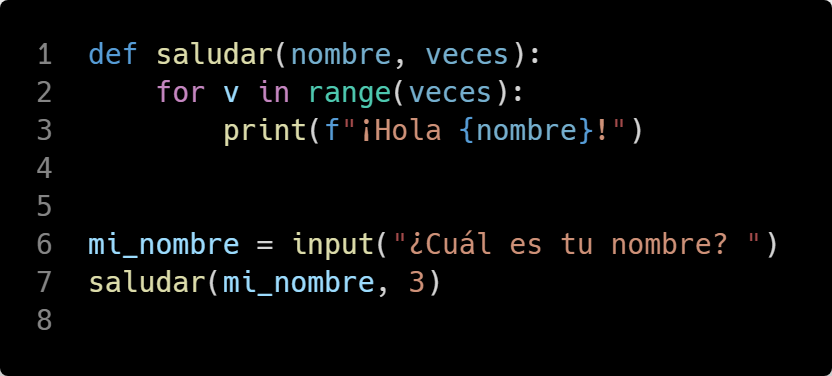


## 

## Argumentos por posición

Los argumentos pueden ser escritos de dos formas (hacen exactamente lo mismo): argumentos por posición y argumentos por nombre. Los argumentos por posición deben ocupar el mismo lugar de orden de los parámetros.

**Ejecuta el siguiente código:**

****

## 

## Argumentos por nombre

En los argumentos por nombre, el orden del argumento no es importante, pero el número de argumentos debe coincidir; de lo contrario, obtendrás un error. Se utiliza una variable y la asignación de un valor. Ejemplo:

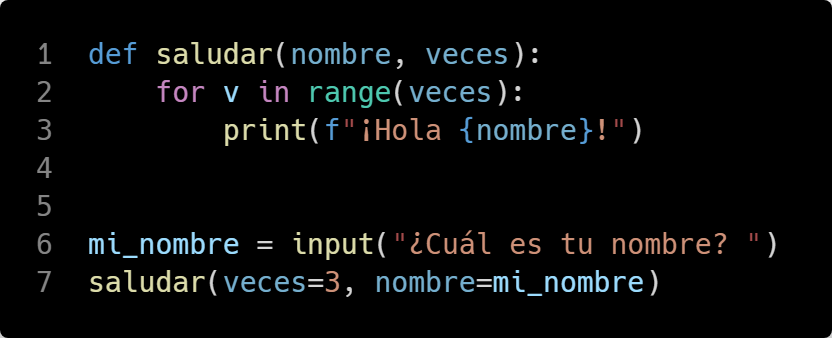
def función(palabra, veces):  
 print(palabra \* veces)  
  
 función(palabra="Hola", veces=2)

‎A primera vista, los argumentos por posición pueden parecer un poco limitantes y contrarios a la legibilidad. En tal caso, los **argumentos por nombre** son una buena opción.

Quizás no haya muchas ocasiones en que los argumentos posicionales mejoren tu código.‎ ‎Sin embargo, pueden darte cierta flexibilidad al diseñar funciones. Por ejemplo, puedes usar argumentos posicionales para refactorizar (rehacer) más fácilmente tus funciones, es decir, no hace falta que cambies el nombre de sus parámetros, lo que significa es que no debes preocuparte de que otra parte del código dependa de esos nombres.‎

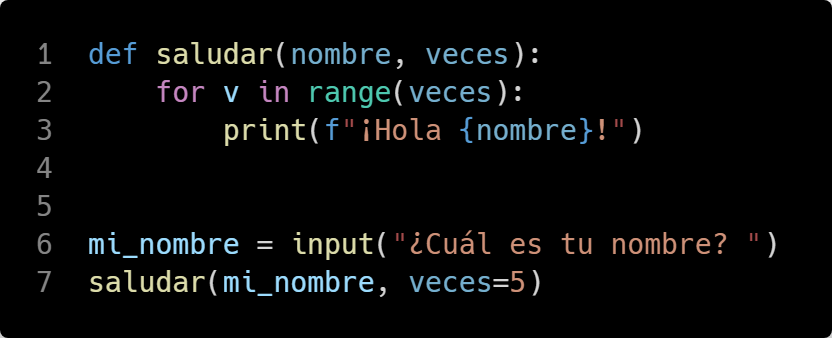
Pero es una buena práctica usar los argumentos por nombre, siempre y cuando tengas en cuenta el contexto que se describe en el párrafo anterior.

**Ejecuta el siguiente código:**

****

Cuando uses argumentos por posición y por nombre simultáneamente, primero debes pasar los argumentos por posición y luego los argumentos por nombre; de lo contrario, obtendrás un error.

**Ejecuta el siguiente código:**

****

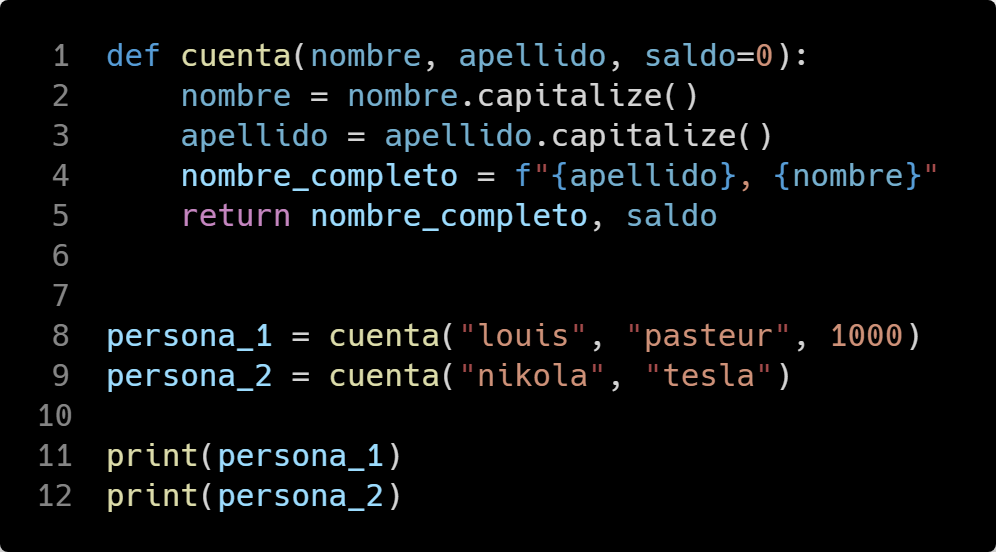
## 

## Parámetros predeterminados

Los argumentos predeterminados toman el valor predeterminado durante la llamada a la función, si no le pasas un valor. Puedes asignar un valor predeterminado a un argumento en la definición de la función utilizando el operador de asignación '='.

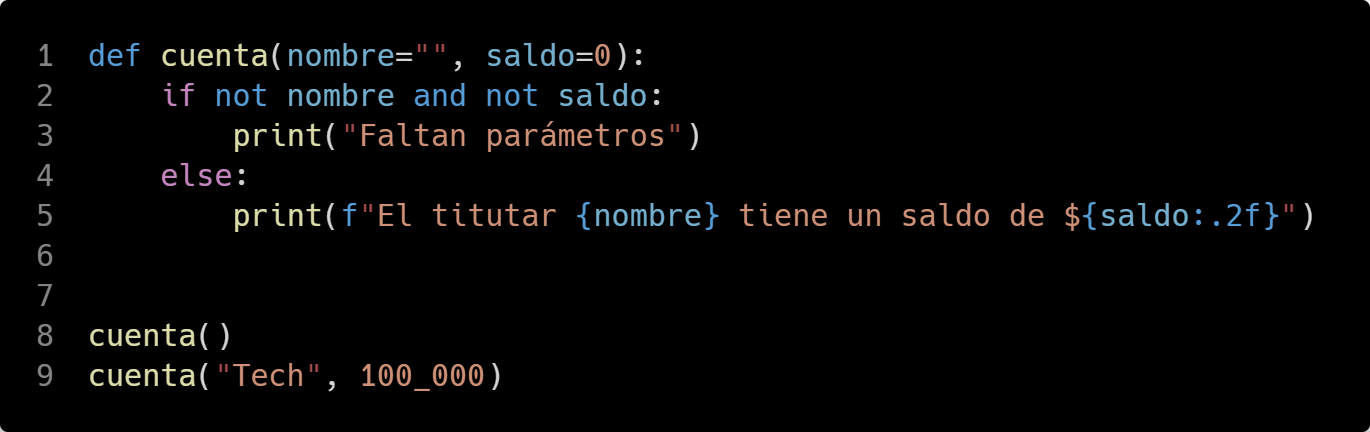
Un parámetro requerido y un parámetro opcional:

**Ejecuta el siguiente código:**



Dos parámetros opcionales:

**Ejecuta el siguiente código:**



**¡MANOS A LA OBRA!**

**Ejercicio 2**

1º Crea una función llamada "media". Tendrá 2 parámetros donde recibirá números. Devolverá un número, que consistirá en el promedio de los dos parámetros. No usar print dentro de la función.  
2º Crear 2 variables con valores numéricos.  
3º Invoca la función y guardar el resultado  
4º Muestra el resultado.

## Revisemos lo aprendido hasta aquí

* Crear funciones con parámetros
* Invocar funciones con argumentos mutables e inmutables
* Invocar funciones con argumentos por posición y por nombre
* Crear parámetros predeterminados

# 

# Desempaque de argumentos

Otra forma de probar la función del ejemplo anterior es usando una colección de datos. Para eso, es necesario desempacar la colección para que los datos sean enviados "desarmados":

**Ejecuta el siguiente código:**



Si tienes más datos anidados, debes iterar sobre ellos:

**Ejecuta el siguiente código:**



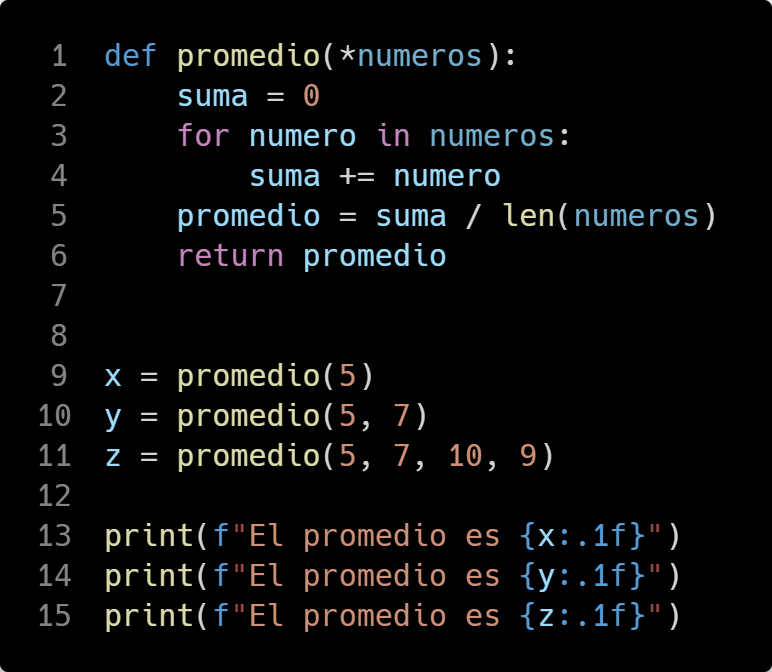
# 

# Argumentos indeterminados

## Argumentos de longitud variable

Cuando necesites pasar una cantidad indeterminada de argumentos, utiliza el asterisco en un único parámetro, de esta forma, llegarán todos los argumentos en forma de tupla. Puedes utilizar el slice para acceder a un elemento de la variable parámetro, o puedes iterar sobre la tupla:

**Ejecuta el siguiente código:**

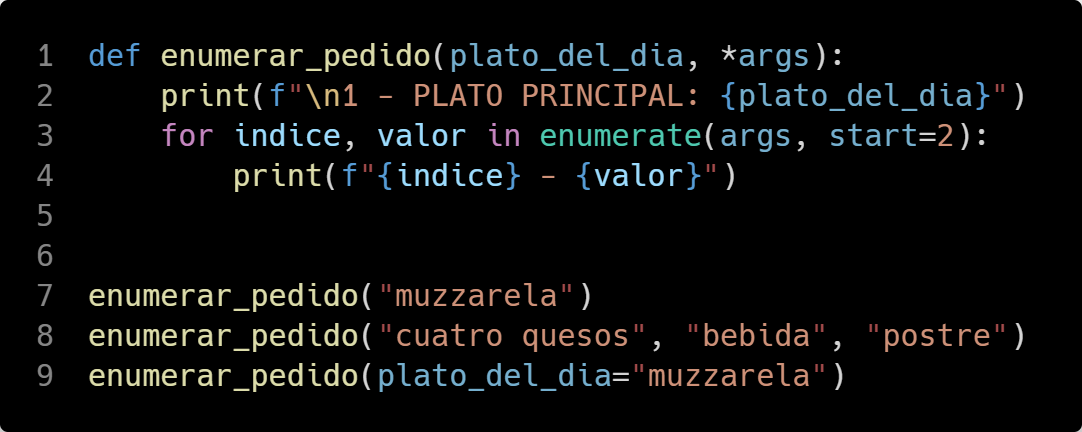


| Por convención, el nombre del parámetro que se usa es 'args' |
| --- |

| Puedes formatear una variable f-string de la siguiente manera:  {variable: <formato>} Observa los dos puntos. Luego de eso puedes alinear texto con los caracteres < > ^ seguido de un número. También puedes acortar la parte decimal usando el punto, un número y la letra f {número:.2f}, como en el ejercicio anterior |
| --- |

Puedes recibir argumentos obligatorios junto a argumentos de longitud variable, para eso, debes definirlo en los parámetros de la función:

**Ejecuta el siguiente código:**



## Argumentos de longitud variable por nombre

Para recibir un número indeterminado de parámetros por nombre (clave-valor), debes crear un diccionario de argumentos. A su vez, debes definir el parámetro con dos asteriscos: \*\*parámetro.

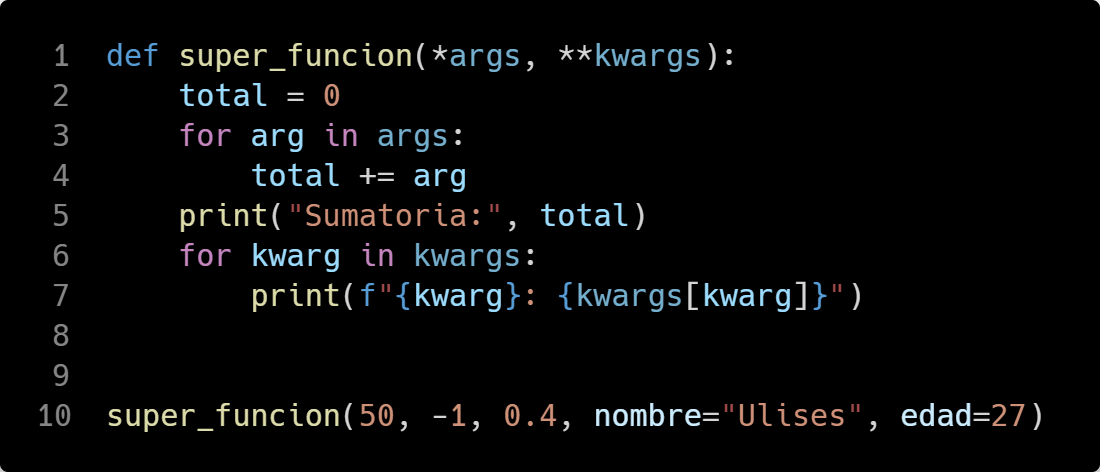
| Por convención, el nombre del parámetro que se usa es 'kwargs', |
| --- |

**Ejecuta el siguiente código:**



Si quieres ambos tipos de parámetros, debes crear primeramente los argumentos indeterminados por valor, y luego los que son por clave y valor:

**Ejecuta el siguiente código:**



## Orden de los argumentos

Si tuviéramos una función que hiciera uso de diferentes tipos de argumentos; deberíamos usarlos con un cierto orden. Primero, argumentos posicionales, después los argumentos posicionales arbitrarios (\*args), a continuación, los argumentos por defecto, y por último los argumentos clave-valor arbitrarios (\*\*kwargs). En caso de usarlos de otra manera, el resultado puede ser un error, o no poder hacer uso correcto del argumento por defecto.

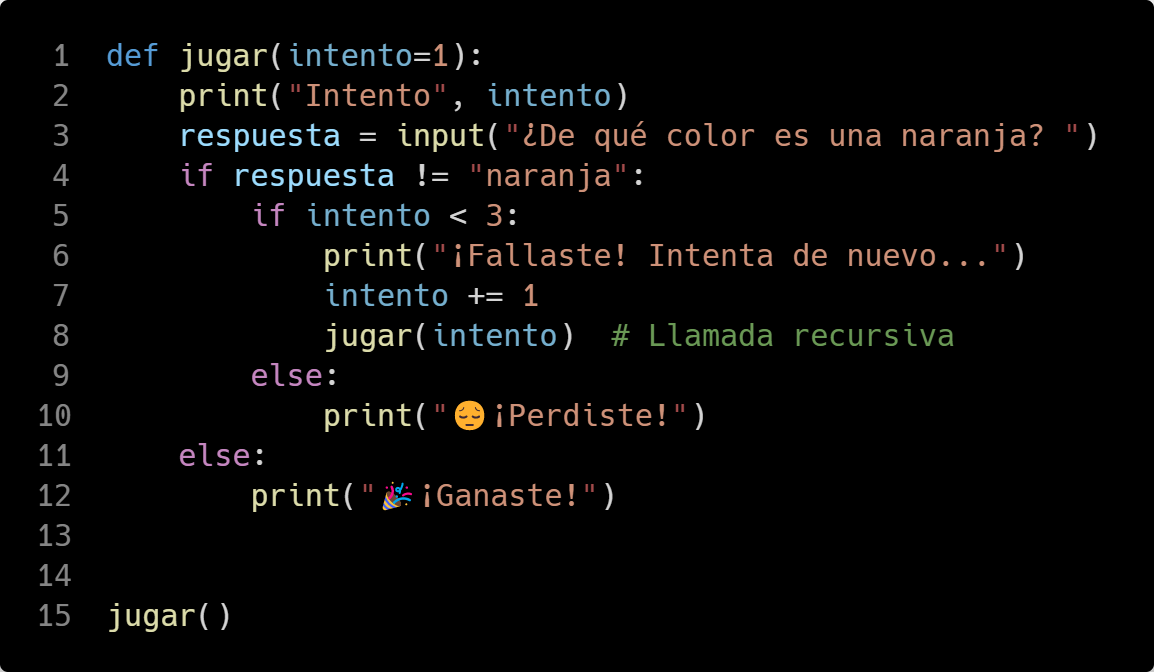
Ejemplo:

def función(a, b, \*args, c=100, \*\*kwargs):  
 print(a)  
 print(b)  
 print(args)  
 print(c)  
 print(kwargs)

# Funciones recursivas

Las funciones recursivas son las que se llaman a sí mismas durante su ejecución. Debemos planificar el momento en que dejan de llamarse a sí mismas, o tendremos una función recursiva infinita.

**Ejecuta el siguiente código:**

****

| No es una buena práctica usar funciones recursivas en Python, salvo necesidad, ya que la función que llama a la otra, queda en memoria, en forma de pila, y Python tolera hasta 1000 funciones en pila, número que se puede ampliar, pero no es una buena práctica.  Para evitar desbordamientos de memoria, utiliza return, bucles, o generadores, que se verán más adelante. |
| --- |

# 

# Documentación en funciones

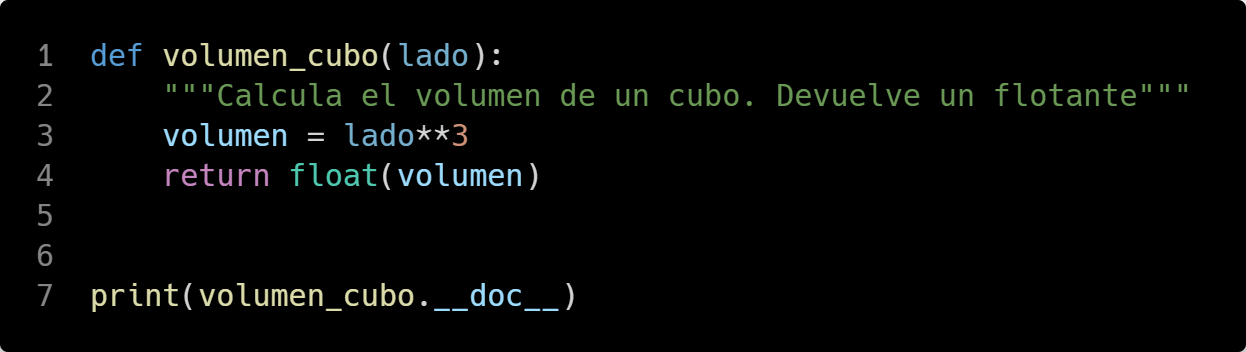
‎En Python, la cadena de documentación también se denomina ‎‎docstring‎‎. Es un texto descriptivo (como un comentario) escrito por un programador para que otros sepan lo que hace el bloque de código.‎

‎Escribimos docstrings después de la definición de la función.

‎Se declara usando comillas simples-triples o comillas dobles-triples. ‎(''' ''')(""" """)

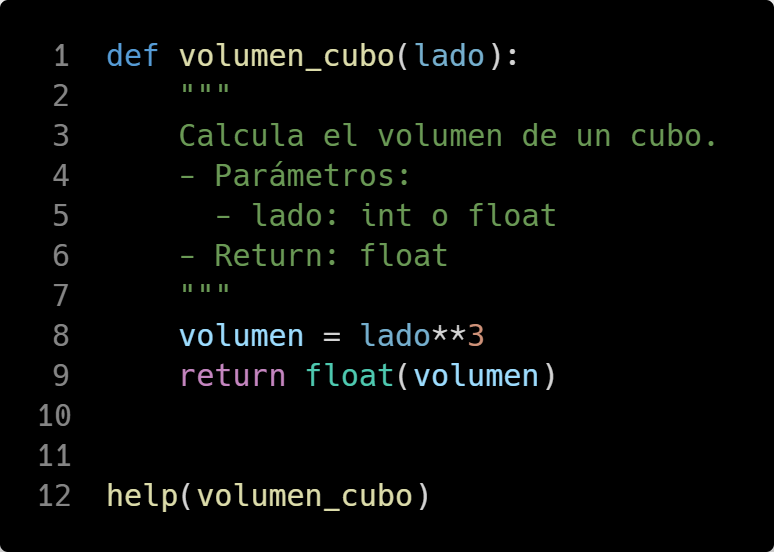
Puedes acceder a las docstrings usando el atributo \_\_doc\_\_ para cualquier objeto, de la siguiente manera:

**Ejecuta el siguiente código:**



Un docstring multilínea, se hace de la siguiente manera, o de forma parecida (observa la función **help,** puedes usarla con funciones incorporadas de Python):

**Ejecuta el siguiente código:**



| En Visual Studio Code, puedes pasar el cursor del mouse por encima del nombre de la función y tendrás la ayuda. |
| --- |

## Revisemos lo aprendido hasta aquí

* Desempacar argumentos
* Usar los argumentos \*ars y \*\*kwargs
* Crear funciones recursivas
* Documentar funciones

**¡MANOS A LA OBRA!**

**Ejercicio 3**

1º Crea una función llamada "media". Tendrá 1 parámetro para argumentos indeterminados. Devolverá un número, que consistirá en el promedio de todos los números pasados a la función. No usar print dentro de la función. Usa 'for' para iterar y hacer el cálculo.  
2º Crea una colección de números.  
3º Invoca la función pasando la colección de números.   
4º Muestra el resultado: "El promedio de los números {colección} es {promedio}". Donde el promedio estará redondeado a 1 decimal.

**Ejercicio 4**

Mejora el ejercicio 3:

1º Documenta la función.  
2º En vez de utilizar 'for' en la función, utiliza la función 'sum()'.  
3º En vez de usar variables, permite que el usuario introduzca cuántos números quiere introducir, y luego que introduzca los números.