Ficha de ayuda: Funciones

Funciones: Parámetros y argumentos

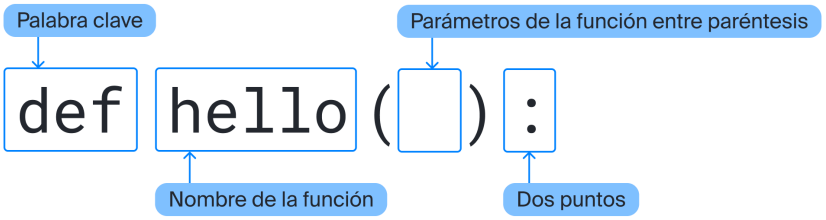
Las funciones son fragmentos de código con un nombre específico que realizan una tarea determinada. Se utilizan para no escribir el mismo código una y otra vez.

Por ejemplo, la función `print()` muestra todo lo que se le pasa. Está integrada en Python. Sin embargo, puedes crear tu propia función en Python si lo necesitas.

Cómo definir una función

Si quieres crear tu propia función, primero debes **definirla**. Para ello, utiliza la palabra clave `def`. A continuación te mostramos un ejemplo:

```
def hello():
```



El equipo de desarrollo define el nombre de la función. Por razones de conveniencia, solo se utilizan letras minúsculas. Además, las palabras se separan con un guión bajo. Por ejemplo, puedes escribir `say_hello` en lugar de solo `hello`.

El código que la función debe ejecutar comienza en la siguiente línea. Este es el **cuerpo** de una función en Python.

Cuerpo de la función

El cuerpo se escribe como una nueva línea, con una sangría de cuatro espacios delante. Así es como se indica dónde comienza y termina la función:

```
def hello():
    print("¡Hola!")# es el cuerpo de la función
```

Llamada de función

Para aplicar una función al código, debes llamarla.

Para ello, escribe el nombre de la función y luego agrega paréntesis.

```
hello()
```

Argumentos y parámetros de la función

Para ayudar a la función a reconocer la variable, debes especificarla. Al hacerlo, la variable desconocida `name` se convierte en un **parámetro de la función**.

Los parámetros se especifican al declarar una función. Al principio de la lección, los paréntesis estaban vacíos:

```
def hello(name):
```

Ahora contienen el cuerpo de la función, pero esta vez con el parámetro `name`:

```
def hello(name):
    print(name + ", ¡Hola")
```

El último paso es sencillo: agrega el primer valor a la variable (tu nombre). El valor de un parámetro se llama argumento de la función. Se especifica entre paréntesis cuando se llama a la función.

```
def hello(name):
    print(name + ", ¡Hola")

hello("Andrea") # la función mostrará: ¡Hola, Andrea!
```

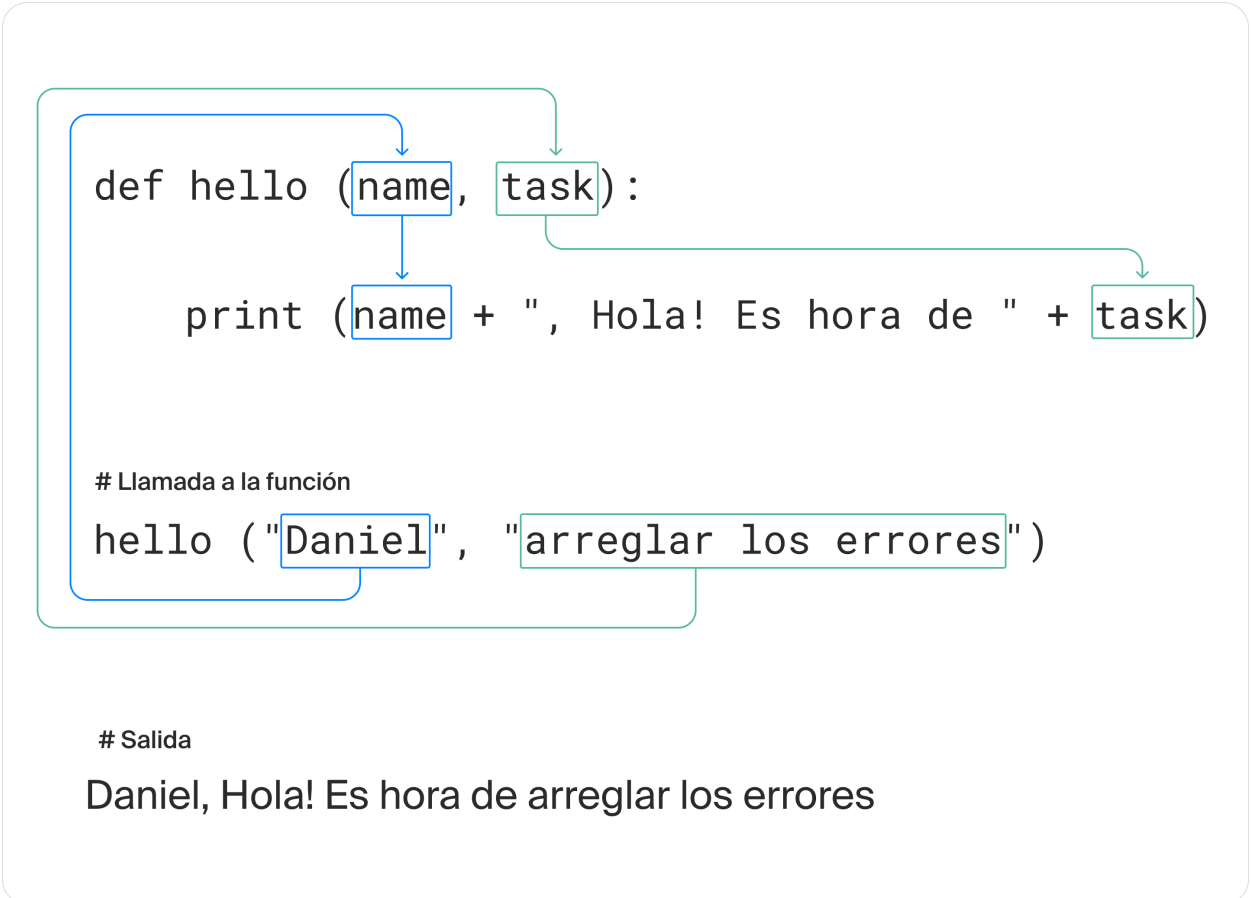
Varios parámetros

Una función puede tener más de un parámetro.

En una llamada a la función, los argumentos se pasan según el orden en el que se enumeran: el primer argumento se pasa como primer parámetro, el segundo argumento como segundo parámetro, y así sucesivamente.

```
def hello(name, task):
    print(name + ", Hola! Es hora de " + task)
hello("Daniel", "arreglar los errores")
hello("Jennifer", "escribir casos de prueba")
hello("Gabriel", "elevar el espíritu del equipo")
```

El siguiente diagrama de flujo muestra dos parámetros (`name` y `task`), así como dos argumentos (`"Daniel"` y `"arreglar los errores"`). La salida entre los argumentos será la misma para todos los nombres y tareas: `¡Hola! Es hora de`.



Funciones: Valores de retorno

Sin embargo, las funciones no solo pueden imprimir los argumentos, sino también **devolverlos** al código para continuar trabajando con los valores de retorno.

La palabra clave `return`

El valor de retorno se coloca después de la palabra clave `return` (devolver) en el cuerpo de la función.

Por ejemplo, la carta de un restaurante proporciona información sobre la cantidad de `calories` en cada plato:

```
calories = {
   'Hamburguesa': 600,
   'Hamburguesa con queso': 750,
   'Hamburguesa vegetariana': 400,
   'Hamburguesa vegana': 350,
   'Batatas': 230,
   'Ensalada': 15,
   'Té helado': 70,
   'Limonada': 90,
}
```

Puedes automatizar el cálculo de calorías en la comida de tres platos escribiendo una función con la palabra clave `return`.

```
calories = {
   "Hamburguesa": 600,
   "Hamburguesa con queso": 750,
   "Hamburguesa vegetariana": 400,
   "Hamburguesa vegana": 350,
   "Batatas": 230,
   "Ensalada": 15,
   "Té helado": 70,
   "Limonada": 90,
}

def calories_counter(item_a, item_b, item_c): # declarar una nueva función
    return calories[item_a] + calories[item_b] + calories[item_c] # devolver valores
print(calories_counter("Hamburguesa", "Hamburguesa con queso", "Té helado")) # llamar a una nueva función
```

La función funciona así:

La sentencia `def calories_counter(item_a, item_b, item_c):`

Contiene tres parámetros: `item_a, item_b, item_c`. Estos parámetros sirven como plantillas, ya que más adelante serán reemplazados por los platos del restaurante.

La sentencia `return calories[item_a] + calories[item_b] + calories[item_c]`

El siguiente paso es escribir la palabra clave `return`, indicar una nueva variable (`calories`) y su parámetro entre corchetes. En conjunto, forman el operador de retorno. Devuelve los valores de la función cuando es llamada.

Ahora el programa puede calcular la cantidad de calorías en una comida de tres platos.

La sentencia `print(calories_counter("Hamburguesa", "Hamburguesa con queso", "Té helado"))`

Se muestran los resultados del cálculo. Los argumentos (tres platos) se utilizan en lugar de parámetros: una hamburguesa, una hamburguesa con queso y un té helado. El resultado son 1240

calorías.

Probar funciones. La palabra clave `assert`

Para comprobar si la función devuelve un resultado esperado, puedes mostrar el valor y ver si la respuesta es correcta.

Sin embargo, intentemos automatizar esta prueba por conveniencia. Ahí es cuando la palabra clave `assert` (afirmar) resulta útil. Funciona como una expresión lógica porque devuelve `True` o `False`. Por ejemplo, si la suma de los números en realidad es igual a 6, la función devolverá `True` como resultado; de lo contrario, muestra `False`.

Cómo usar `assert`

Imagina que estás probando una calculadora y tu tarea es asegurarte de que suma correctamente números enteros de un solo dígito.

En primer lugar, debes crear un caso de prueba con pasos a seguir.

La calculadora suma correctamente números enteros de un solo dígito

Pasos:

1. Ingresa 5.
2. Presiona `+`.
3. Ingresa 6.
4. Presiona `=`.

Resultado esperado:

El resultado del cálculo es igual a 11.

Para automatizar este caso de prueba, debes seguir los pasos en el código y luego ver si el resultado esperado es realmente 11.

Por eso recomendamos utilizar la palabra clave `assert` en tu automatizada. También se requiere un operador de comparación, como `==`. Así es como se ve en el código:

Es como si dijeras: "Comprueba que el resultado sea 11".

Si la declaración es correcta y se supera la prueba, el programa seguirá funcionando.

El error `AssertionError`

En caso de una prueba no aprobada, el programa devolverá un mensaje de error: `AssertionError` (error de afirmación).

Si el código contiene otras pruebas, el programa no las ejecutará, lo que te dará la oportunidad de notar el error y corregirlo de inmediato.

El error muestra tanto los resultados esperados como los actuales, así:

```
 assert 5 == 4
    +5
    -4

test__calc.py:40: AssertionError
```

`assert` y operadores aritméticos

`assert` te permite usar varios operadores aritméticos, no solo `==`, sino también `!=`, `>`, `<` y algunos otros. Todos ellos ayudan a diseñar diferentes pruebas.

Digamos que necesitas comprobar si el saldo de una cuenta es mayor que cero. En este caso, utilizarás `>`:

```
def test_deposit_more_than_zero():

    deposit = get_user_deposit()  # recibir los datos de la base de datos y guardarlos en la variable
    assert deposit > 0   # comprobar si el saldo de usuario o usuaria es mayor que 0
```