Funciones en Python

Las funciones en Python, y en cualquier lenguaje de programación, son estructuras esenciales de código. Una función es un grupo de instrucciones que constituyen una unidad lógica del programa y resuelven un problema muy concreto.

Este tutorial es una guía que te muestra qué es una función, cuál es su estructura y cómo usarlas en tus aplicaciones.

Qué son las funciones en Python

Como te decía en la introducción, las funciones en Python constituyen unidades lógicas de un programa y tienen un doble objetivo:

- Dividir y organizar el código en partes más sencillas.
- Encapsular el código que se repite a lo largo de un programa para ser reutilizado.

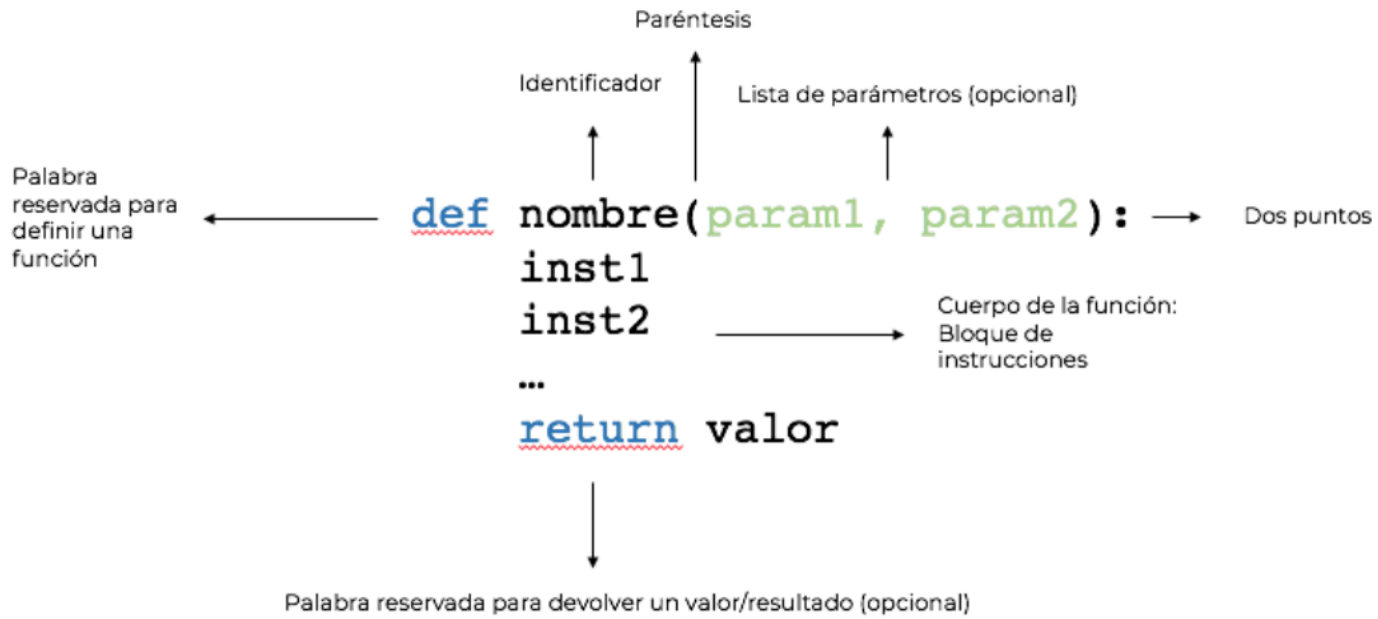
Python ya define de serie un conjunto de funciones que podemos utilizar directamente en nuestras aplicaciones. Algunas de ellas las has visto en tutoriales anteriores. Por ejemplo, la función `len()`, que obtiene el número de elementos de un objeto contenedor como una lista, una tupla, un diccionario o un conjunto. También hemos visto la función `print()`, que muestra por consola un texto.

Sin embargo, tú como programador, puedes definir tus propias funciones para estructurar el código de manera que sea más legible y para reutilizar aquellas partes que se repiten a lo largo de una aplicación. Esto es una tarea fundamental a medida que va creciendo el número de líneas de un programa.

En el siguiente apartado te muestro cómo definir una función en Python.

Cómo definir una función en Python

La siguiente imagen muestra el esquema de una función en Python:



Para definir una función en Python se utiliza la palabra reservada `def`. A continuación viene el nombre o identificador de la función que es el que se utiliza para invocarla. Después del nombre hay que incluir los paréntesis y una lista opcional de parámetros. Por último, la cabecera o definición de la función termina con dos puntos.

Tras los dos puntos se incluye el cuerpo de la función (con un sangrado mayor, generalmente cuatro espacios) que no es más que el conjunto de instrucciones que se encapsulan en dicha función y que le dan significado.

En último lugar y de manera opcional, se añade la instrucción con la palabra reservada `return` para devolver un resultado.

NOTA: Cuando la primera instrucción de una función es un `string` encerrado entre tres comillas simples `'''` o dobles `"""`, a dicha instrucción se le conoce como `docstring`. El *docstring* es una cadena que se utiliza para documentar la función, es decir, indicar qué hace dicha función.

Cómo usar o llamar a una función

Para usar o invocar a una función, simplemente hay que escribir su nombre como si de una instrucción más se tratara. Eso sí, pasando los argumentos necesarios según los parámetros que defina la función.

Veámoslo con un ejemplo. Vamos a crear una función que muestra por pantalla el resultado de multiplicar un número por cinco:

```
def multiplica_por_5(numero):
print(f'{numero} * 5 = {numero * 5}')

print('Comienzo del programa')
multiplica_por_5(7)
print('Siguiente')
multiplica_por_5(113)
print('Fin')
```

La función `multiplica_por_5()` define un parámetro llamado `numero` que es el que se utiliza para multiplicar por 5. El resultado del programa anterior sería el siguiente:

```
Comienzo del programa
7 * 5 = 35
Siguiente
113 * 5 = 565
Fin
```

Como puedes observar, el programa comienza su ejecución en la *línea 4* y va ejecutando las instrucciones una a una de manera ordenada. Cuando se encuentra el nombre de la función `multiplica_por_5()`, el flujo de ejecución pasa a la primera instrucción de la función. Cuando se llega a la última instrucción de la función, el flujo del programa sigue por la instrucción que hay a continuación de la llamada de la función.

IMPORTANTE: Diferencia entre ***parámetro*** y ***argumento***. La función `multiplica_por_5()` define un *parámetro* llamado `numero`. Sin embargo, cuando desde el código se invoca a la función, por ejemplo, `multiplica_por_5(7)`, se dice que se llama a multiplica por cinco con el *argumento* `7`.

Sentencia return

Anteriormente te indicaba que cuando acaba la última instrucción de una función, el flujo del programa continúa por la instrucción que sigue a la llamada de dicha función. Hay una excepción: usar la

sentencia `return`. `return` hace que termine la ejecución de la función cuando aparece y el programa continúa por su flujo normal.

Además, `return` se puede utilizar para devolver un valor.

La sentencia `return` es opcional, puede devolver, o no, un valor y es posible que aparezca más de una vez dentro de una misma función.

A continuación te muestro varios ejemplos:

return que no devuelve ningún valor

La siguiente función muestra por pantalla el cuadrado de un número solo si este es par:

```
>>> def cuadrado_de_par(numero):
...       if not numero % 2 == 0:
...            return
...       else:
...            print(numero ** 2)
...
>>> cuadrado_de_par(8)
64
>>> cuadrado_de_par(3)
```

Varios return en una misma función

La función `es_par()` devuelve `True` si un número es par y `False` en caso contrario:

```
>>> def es_par(numero):
...       if numero % 2 == 0:
...            return True
...       else:
...            return False
...
>>> es_par(2)
True
>>> es_par(5)
False
```

Devolver más de un valor con return en Python

En Python, es posible devolver más de un valor con una sola sentencia `return`. Por defecto, con `return` se puede devolver una tupla de valores. Un ejemplo sería la siguiente función `cuadrado_y_cubo()` que devuelve el cuadrado y el cubo de un número:

```
>>> def cuadrado_y_cubo(numero):
...       return numero ** 2, numero ** 3
...
>>> cuad, cubo = cuadrado_y_cubo(4)
>>> cuad
16
>>> cubo
64
```

Sin embargo, se puede usar otra técnica devolviendo los diferentes resultados/valores en una lista. Por ejemplo, la función `tabla_del()` que se muestra a continuación hace esto:

```
>>> def tabla_del(numero):
...      resultados = []
...      for i in range(11):
...           resultados.append(numero * i)
...           return resultados
...
>>> res = tabla_del(3)
>>> res
[0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30]
```

Parámetros de las funciones en Python

Tal y como te he indicado, una función puede definir, opcionalmente, una secuencia de parámetros con los que invocarla. ¿Cómo se asignan en Python los valores a los parámetros? ¿Se puede modificar el valor de una variable dentro de una función?

Antes de contestar a estas dos preguntas, tenemos que conocer los conceptos de programación *paso por valor* y *paso por referencia*.

- **Paso por valor:** Un lenguaje de programación que utiliza paso por valor de los argumentos, lo que realmente hace es copiar el valor de las variables en los respectivos parámetros. Cualquier modificación del valor del parámetro, no afecta a la variable externa correspondiente.
- **Paso por referencia:** Un lenguaje de programación que utiliza paso por referencia, lo que realmente hace es copiar en los parámetros la dirección de memoria de las variables que se usan como argumento. Esto implica que realmente hagan referencia al mismo objeto/elemento y cualquier modificación del valor en el parámetro afectará a la variable externa correspondiente.

Muchos lenguajes de programación usan a la vez paso por valor y por referencia en función del tipo de la variable. Por ejemplo, paso por valor para los tipos simples: entero, float, ... y paso por referencia para los objetos.

Sin embargo, en Python todo es un objeto. Ya vimos esto en el tutorial sobre variables. Entonces, ¿cómo se pasan los argumentos en Python, por valor o por referencia? Lo que ocurre en Python realmente es que se pasa por valor la referencia del objeto ¿Qué implicaciones tiene esto? Básicamente que si el tipo que se pasa como argumento es inmutable, cualquier modificación en el valor del parámetro no afectará a la variable externa pero, si es mutable (como una lista o diccionario), sí se verá afectado por las modificaciones.

Bibliografía:

- Extraído de <<Funciones en Python>>. 26 de septiembre de 2022. Consultado el 28 de noviembre del 2022