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.

Qué son las funciones en Python

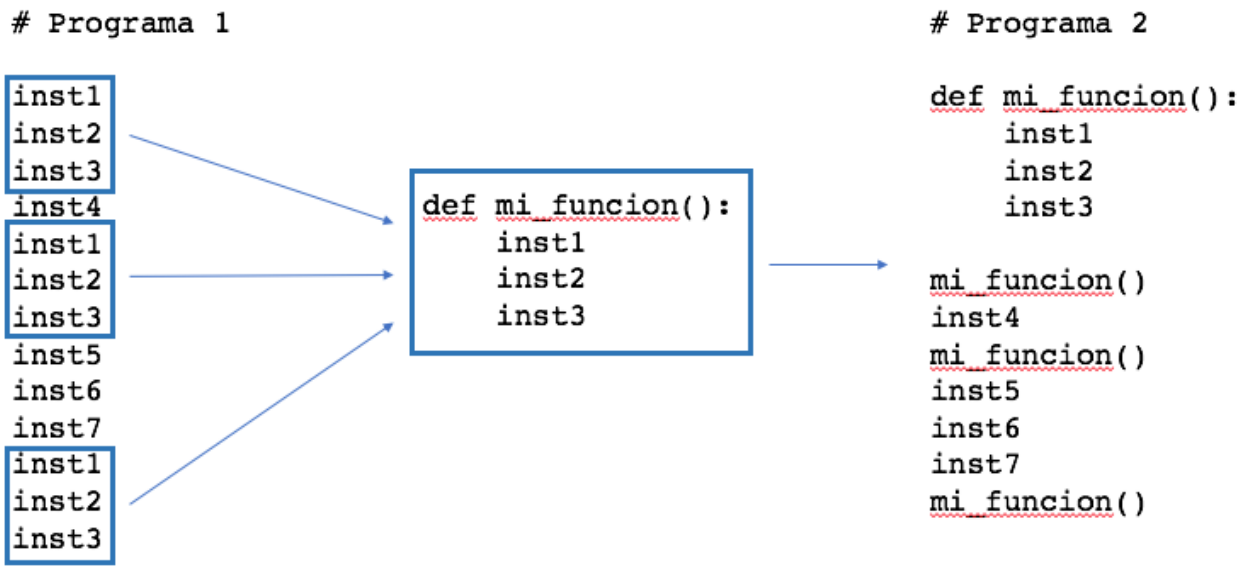
Las funciones en Python constituyen unidades lógicas de un programa y tienen un doble objetivo:

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

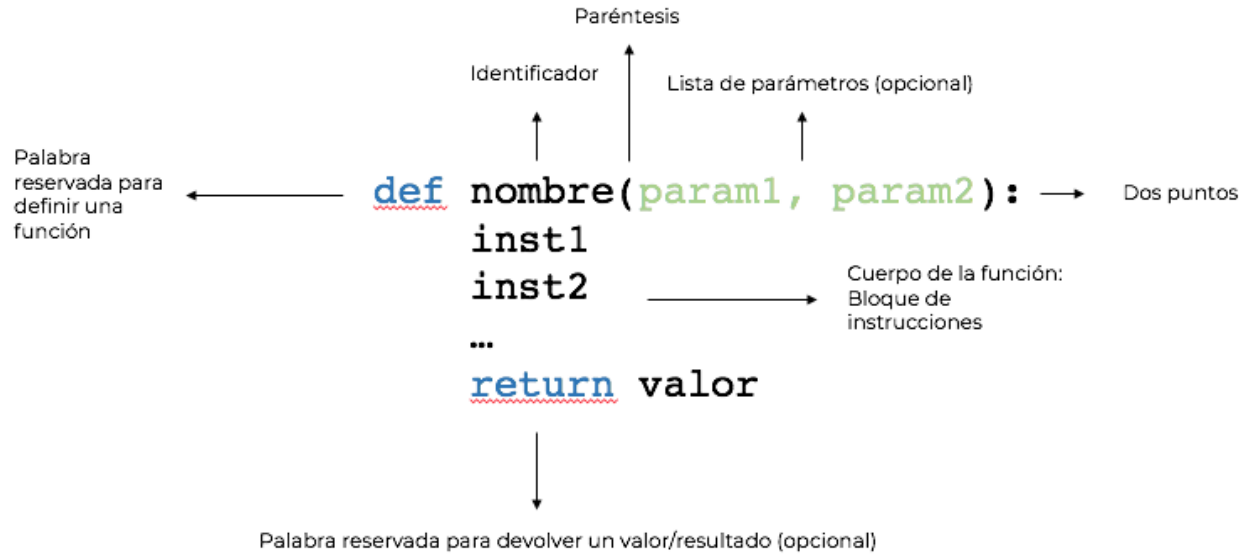
Python ya define de serie un conjunto de funciones que podemos utilizar directamente en nuestras aplicaciones. Algunas de ellas, por ejemplo, la función len(), que obtiene el número de elementos de un objeto contenedor como una lista, una tupla, un diccionario o un conjunto. También hemos visto la función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

Sin embargo, 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.

La idea la puedes observar en la siguiente imagen:



Cómo definir 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.

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.

Veámos 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
```

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 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]
```

En Python una función siempre devuelve un valor

Python, a diferencia de otros lenguajes de programación, no tiene procedimientos. Un procedimiento sería como una función pero que no devuelve ningún valor.

¿Por qué no tiene procedimientos si hemos vistos ejemplos de funciones que no retornan ningún valor? Porque Python, internamente, devuelve por defecto el valor `None` cuando en una función no aparece la sentencia `return` o esta no devuelve nada.

```
>>> def saludo(nombre):
... print(f'Hola {nombre}')
```

```
...
>>> print(saludo('j2logo'))
Hola j2logo
None
```

####[1]

Como puedes ver en el ejemplo anterior, el `print` que envuelve a la función `saludo()` muestra `None`.

1**Cadenas "f"**

A partir de la versión 3.6 de Python, se añadió (PEP 498), una nueva notación para cadenas llamada ***cadenas "f"***, que hace más sencillo introducir variables y expresiones en las cadenas. Una cadena "f" contiene variables y expresiones entre llaves "{}" que se sustituyen directamente por su valor. Las cadenas "f" se reconocen porque comienzan por una letra f antes de las comillas de apertura.

```
nombre = "Alicia"
edad = 35
print(f"Me llamo {nombre} y tengo {edad} años.")
```