



Aprenda a programar con Python 3

Funciones

Carolina Mañoso, Ángel P. de Madrid y Miguel Romero

Índice

◆ Funciones

- Definición
- Parámetros

◆ Referencias



Funciones: definición (1/2)

◆ Una función es un fragmento de código con un nombre asociado que realiza una tarea concreta (y devuelve un valor).

- Permite reutilizar código

◆ Para definir una función:

- Se utiliza la palabra clave `def` seguida del nombre de la función más paréntesis de apertura y cierre. Como toda estructura de control, la definición finaliza con dos puntos `:`.
- Las sentencias que forman el cuerpo de la función (el algoritmo que desarrolla) empiezan en la línea siguiente con sangría.
- La primera sentencia del cuerpo de la función puede ser una cadena de texto de documentación de la función (`docstring`)


```
def mi_funcion():  
    """ esta función escribe hola mundo """  
    print("Hola mundo desde la función")
```

Funciones: definición (2/2)

- ◆ Una función no ejecuta nada hasta que no es llamada

```
mi_funcion() #llamada
```

- ◆ Una función puede retornar valores, (en vez de imprimirlos) usando `return`, entonces, la función puede ser asignada a una variable, que contendrá esos valores

```
def mi_funcion_con_return():  
    """ esta función retorna hola mundo """  
    return "Hola mundo" 
```

```
#Llamada a la función  
saludo = mi_funcion_con_return()  
print("escribo desde el programa:",saludo)
```

Funciones y parámetros (1/7)

- ◆ Un parámetro es un valor que la función espera recibir cuando es llamada a fin de realizar acciones con el mismo. En la definición estos parámetros van entre los paréntesis separados por comas.

```
def mi_funcion(param1, param2):
```

- ◆ Ejemplo:

```
def mi_funcion_con_param(nombre):  
    print(nombre)
```

```
mi_funcion_con_param('Maria') # Llamada
```

- ◆ Los parámetros son variables de ámbito local.
 - Si uso `nombre` fuera de la definición, nos dará error
`NameError: name 'nombre' is not defined`



Práctica: Funciones y parámetros (2/7)

◆ Paso de un parámetro

- ◆ Calcula el cuadrado de un número

◆ Paso de dos parámetros

- ◆ Calcula la suma de dos números

◆ Sucesión de Fibonacci

$$f_0 = 0$$

$$f_1 = 1$$

$$f_2 = f_1 + f_0 = 1 + 0 = 1$$

$$f_3 = f_2 + f_1 = 1 + 1 = 2$$

...

$$f_n = f_{n-1} + f_{n-2}$$

Soluciones al final de la presentación.



Funciones y parámetros (3/7)

◆ Los parámetros son variables de ámbito local:

```
def incremento_uno(num):  
    num = num + 1  
    return num
```

```
num1 = 1  
num2 = incremento_uno(num1)  
print("primer_num", num1)  
print("segundo_num", num2)
```

◆ Sin embargo, los tipos de datos mutables como listas conjuntos y objetos pueden cambiar


```
def incremento_uno_lista(lista):  
    lista.append(1)  
    return lista
```

```
lista1 = [4,3,2]  
lista2 = incremento_uno_lista(lista1)  
print("primera lista", lista1)  
print("segunda lista", lista2)
```



Funciones y parámetros (4/7)

- ◆ Es posible asignar valores por defecto a los parámetros (se le asigna el valor en la definición).
 - La función podrá ser llamada con menos argumentos de los que espera

```
def mi_funcion(nombre, mensaje='hola') :   
    print(mensaje, nombre)
```

```
mi_funcion('Maria') # Llamada
```



Funciones y parámetros (5/7)

- ◆ Se pueden definir funciones con un número arbitrario de parámetros. Estos argumentos llegan a la función como una tupla. En la definición, se coloca el último parámetro precedido por un asterisco (*)

```
def mi_funcion(param_fijo, *param_arbitrarios):
```

- Ejemplo:

```
def varios(param1, param2, *otros):  
    for val in otros:  
        print(val)
```

```
varios(1,2) # Llamada1  
varios(1,2,3) # Llamada2  
varios(1,2,3,4) # Llamada3
```



Funciones y parámetros (6/7)

- ◆ Si se precede por (**) en vez de una tupla se utilizará un diccionario

```
def porc_aprobados(aprobados, **aulas):  
    '''Calcula el % de aprobados '''  
    total = 0  
    for alumnos in aulas.values():  
        total += alumnos  
    return aprobados * 100 / total
```

```
porcentaje_aprobados = porc_aprobados(48, A=22, B=25, C=21)  
print(porcentaje_aprobados)
```



Funciones y parámetros (7/7)

- ◆ Llamamos *desempaquetado de parámetros* a la situación inversa, la función espera una lista fija de parámetros, pero estos están disponibles en una lista o tupla. En este caso, en la llamada, se debe colocar un asterisco (*) antes del nombre de la lista o tupla que es pasada

```
def calcular(importe, descuento):  
    return importe - (importe*descuento/100)
```

```
datos = [1500, 10]  
print(calcular(*datos)) #llamada 
```

- ◆ Si lo que se pasa es un diccionario ponemos (**) 

```
datos = {"importe":1500,"descuento":10}  
print(calcular(**datos)) #llamada 
```



Soluciones (1/4)

◆ Paso de un parámetro

- ◆ Calcula el cuadrado de un número

#Ejemplo de paso de un parámetro

```
def cal_cuadrado(n):
```

```
    """Calcula el cuadrado de un número"""
```

```
    return n**2
```

#Llamada a la función

```
cal_cuadrado(4)
```

Soluciones (2/4)

◆ Paso de dos parámetros

- ◆ Calcula la suma de dos números

#Ejemplo de paso de dos parámetros

```
def cal_suma(a,b):  
    """Calcula la suma de dos números"""  
    return a+b
```

#Llamada a la función

```
cal_suma(2,3)
```

Soluciones (3/4)

◆ Sucesión de Fibonacci (solución 1)

```
def fib1(n):  
    """Escribe la sucesión de Fibonacci"""  
    a, b = 0, 1  
    while a < n:  
        print(a)  
        a, b = b, a + b  
# Primera llamada  
fib1(30)
```

Soluciones (4/4)

◆ Sucesión de Fibonacci (solución 2)

```
def fib2(n):  
    """Escribe la sucesión de Fibonacci con return"""  
    result = []  
    a, b = 0, 1  
    while a < n:  
        result = result + [a]  
        a, b = b, a + b  
    return result  
  
# Segunda llamada  
f30 = fib2(30)  
print(f30)
```

Referencias

- ◆ La documentación oficial, *Python Tutorial Release 3.5.2*. Guido van Rossum and the Python development team. Python Software Foundation. Junio 25, 2016.
<https://docs.python.org/3/download.html>
- ◆ *Learning Python 5th edition*. Mark Lutz. O'Reilly 2013
- ◆ *Practical Programming: An Introduction to computer Science using Python 3*. P. Gries, J. Campbell & J. Montoyo. Edited by Lynn Beighley. 2nd Edition. The Pragmatic Bookshelf. 2013
- ◆ *Learning Python with Raspberry Pi*. Alex Bradbury, Ben Everard. Wiley. 2014
- ◆ Tutorial: *Python 3 para impacientes*. Antonio Suárez Jiménez. 2014
<http://python-para-impacientes.blogspot.com.es/p/indice.html>
- ◆ *Introducción a la programación con Python*. Andrés Marzal, Isabel Gracia. Publicacions de la Universitat Jaume I. 2009
- ◆ *Raspberry Pi, User Guide*. Eben Upton, Gareth Halfacree. Wiley. 2012

Aviso



Aprenda a programar con Python 3 by C. Mañoso, A. P. de Madrid, M. Romero is licensed under a [Creative Commons Reconocimiento-NoComercial-CompartirIgual 4.0 Internacional License](https://creativecommons.org/licenses/by-nc-sa/4.0/).

Esta colección de transparencias se distribuye con fines meramente docentes.

Todas las marcas comerciales y nombres propios de sistemas operativos, programas, hardware, etc. que aparecen en el texto son marcas registradas propiedad de sus respectivas compañías u organizaciones.