

Subrutinas (Funciones)

CI-0202 Principios de informática

Sivana Hamer - sivana.hamer@ucr.ac.cr

**Escuela de Ciencias de la Computación e Informática, Universidad de
Costa Rica**

Licencia: CC BY-NC-SA 4.0



UNIVERSIDAD DE COSTA RICA

Función

Utilizar funciones implica un cambio de **paradigma** del imperativo al **funcional**. Una función es una **secuencia de instrucciones nombrada que realiza una operación**, que pueden o no tener **parametros** y pueden o no producir un **resultado**.

Nota

Hemos utilizado funciones anteriormente como por ejemplo **print** e **input**.

Declaración e invocación

Las funciones tienen dos pasos (similares a los de una variable):

- **Declaración:** Paso en que la función es definida (su nombre y cuerpo).
- **Invocación:** Paso en que el nombre es llamado por el programa, ejecutando el código dentro de su cuerpo.

Ejemplo en Python

```
def diga_hello_world():  
    # Bloque dentro de la funcion  
    print("Hello world")
```

} Declaración

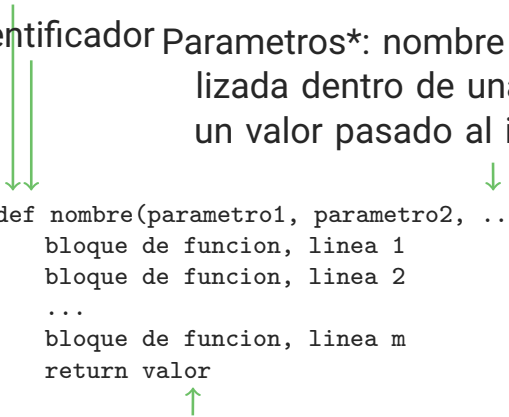
```
diga_hello_world() # Llama la funcion  
diga_hello_world() # Llama de nuevo a la funcion
```

} Invocación

Declaración

Keyword para declarar una función

Identificador Parametros*: nombre de una variable utilizada dentro de una función que fue un valor pasado al invocar la función



```
def nombre(parametro1, parametro2, ..., parametroN):  
    bloque de funcion, linea 1  
    bloque de funcion, linea 2  
    ...  
    bloque de funcion, linea m  
    return valor
```

} Encabezado

} Cuerpo

Return*: detiene la función y retorna el valor

Alcance de las variables

Las variables declaradas son **locales** a la función, lo cual implica que las variables declaradas dentro de una función están **aisladas** de las variables externas de la función. Esto se denomina el **alcance** de la variable. El alcance de cada variable es **dentro del bloque donde se haya declarado**.

Ejemplo en Python

```
def funcion():  
    x=13  
    print(f"Dentro de funcion x = {x}") # 13  
  
x=2  
funcion()  
print(f"Fuera de funcion x = {x}") # 2
```

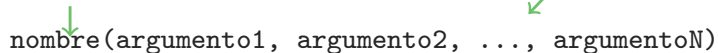
Returns

Fun fact!

En Python las funciones siempre devuelven valores aún cuando no hay un return. En este caso devuelve un **None** (valor especial en Python).

Invocación

Argumentos: valores que se les pasa a una función
Identificador


`nombre(argumento1, argumento2, ..., argumentoN)`

Importante

Las funciones **no** se ejecutan hasta que se invocan.

Nota

Uno puede invocar a una función dentro de otra función.

Tipos de memoria (1)

Los programas manejan 3 tipos de memoria:

- **Estática:** Cuando se pasa por **copia** a una variables. Los tipos de datos se guardan en memoria estática.
- **Dinámica:** Cuando se pasa una **referencia** al objeto dentro de la variable. La referencia indica **donde en memoria se encuentra la variable**. Los objetos y estructuras de datos se guardan en memoria dinámica en Python.
- **Pila:** Python guarda en la pila las variables, objetos y estructuras de datos. Dentro de la pila se guardan temporalmente datos del programa mientras que está en ejecución.

Tipos de memoria (2)

Ejemplo en Python memoria estática

```
def fun(string):  
    string += " Mundo"  
    print(string) # Hola Mundo  
  
string = "Hola"  
fun(string)  
print(string) # Hola
```

Ejemplo en Python memoria dinámica

```
def fun(array):  
    array.append(4)  
    print(array) # [1,2,3,4]  
  
array = [1,2,3]  
fun(array)  
print(array) # [1,2,3,4]
```

Documentación

El código se va volviendo más difícil y complejo. La documentación nos ayuda a acordarnos o avisarle a otros **qué** se hizo en el código. Es considerada buena práctica.

Ejemplo en Python

```
# Funcion esPrimo(num)  
# Retorna True el numero entero num es un numero primo  
# Retorna False de lo contrario  
def esPrimo(num):  
    # Para demostrar que num es primo  
    # se busca si existe un numero i por el que sea divisible  
    for i in range(2, num): # i esta entre 2 y num - 1, ya que un primo solo es divisible  
        entre 1 y si mismo  
        if(num % i == 0): # Si es divisible  
            return False # No es primo  
    return True # Si no es divisible entre ningun numero entre 2 y num - 1, entonces es primo
```

Python permite utilizar **string de documentación** en funciones.

Ejemplo en Python

```
def esPrimo(num):  
    ''' Funcion que calcula si un numero es primo  
  
    Retorna True el numero entero num es un numero primo y False de lo contrario '''  
    # Para demostrar que num es primo  
    # se busca si existe un numero i por el que sea divisible  
    for i in range(2, num): # i esta entre 2 y num - 1, ya que un primo solo es divisible  
        entre 1 y si mismo  
        if(num % i == 0): # Si es divisible  
            return False # No es primo  
    return True # Si no es divisible entre ningun numero entre 2 y num - 1, entonces es primo  
  
print(esPrimo.__doc__) #Imprime la documentacion  
#Al escribir la funcion el IDE tambien muestra la documentacion
```

¿Por qué se usan funciones?

- **Modularidad:** Hacer el sistema por **modulos** implica que se puede **separar el sistema y volverlo a combinar**. Esto implica que se puede **cambiar, adaptar y reutilizar**. Un ejemplo de la vida real son los legos. Cada lego puede ser parte de una figura más compleja.
- **Adaptación:** Implica que el código es **flexible**, permitiendo utilizar el mismo código para problemas similares.
- **Reutilización:** **Volver a utilizar código** hecho anteriormente, lo cual reduce el tiempo y errores al programar.

Referencias I

L. Villalobos, "Subrutinas," Material del curso CI-0202 Universidad de Costa Rica de Leonardo Villalobos, 2019.

C. Swaroop, *A Byte of Python*. Independent, 2020.

Python. (2020) Memory management. [Online]. Available: <https://docs.python.org/3/c-api/memory.html>

J. I. Dobles, "Métodos," Material del curso CI-0202 Universidad de Costa Rica de Jose Ignacio Dobles, 2020.

J. Elkner, A. B. Downey, and C. Meyers, "How to think like a computer scientist: Learning with python," 2012.