El futuro digital es de todos | MinTIC

UNIVERSIDAD EL BOSQUE | MisiónTIC 2022

Ciclo 1

Semana 5

Subprogramas y Funciones

Lectura 2 – *Declaración de Funciones*

Convenio de asociación no. 777 de 2021 suscrito entre el Fondo Único de Tecnologías de la Información y las Comunicaciones y la Universidad el Bosque.

Declaración de Funciones

En **Python** el programador puede definir sus propias funciones, las cuales pueden recibir uno o varios parámetros.

Funciones con un solo parámetro: por ejemplo, una función que recibe un número y retorna el cuadrado de dicho número. El nombre de la función es cuadrado y se define de la siguiente manera (tomado de Marzal):

cuadrado.py

```
1 def cuadrado(x):
2     return x ** 2
```

La función cuadrado se aplica sobre un valor al que se llama x y retorna un número, que es el resultado de elevar x al cuadrado.

```
def cuadrado(x):
    return x**2
print(cuadrado(2))
```

```
4

Process finished with exit code 0
```

A la izquierda se puede ver la definición de la función y luego el llamado que se hace de ella, con el valor de 2 y a la derecha el resultado obtenido. En el ejemplo se puede observar que el valor dado entre paréntesis en el llamado, es utilizado como valor de x durante la ejecución de la función. La línea que empieza con def es la cabecera de la función y la secuencia de instrucciones que le siguen indentadas a la derecha, es el cuerpo de la función. Cuando definimos la función el parámetro se llama parámetro formal, en el ejemplo es x, y el valor que pasamos a la función cuando la invocamos se llama parámetro real o actual. En el momento que se define una función no se está diciendo que se ejecute, es en el llamado o invocación de la función cuando se ejecuta.

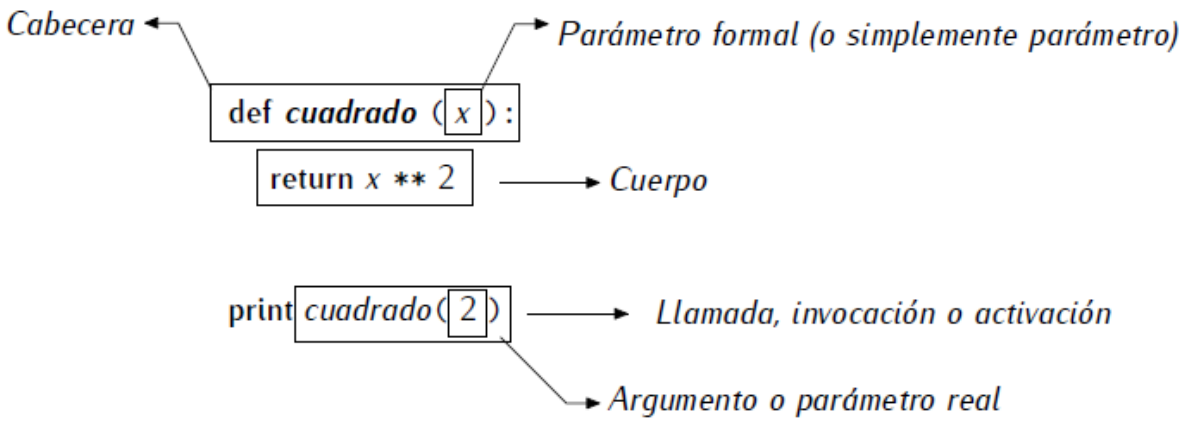


Figura 1: tomada de Marzal

Las reglas para dar nombre a las funciones y a sus parámetros son las mismas que se deben seguir para dar nombre a las variables. Sólo se pueden utilizar letras (del alfabeto inglés), dígitos y el guion bajo o *underscore (_)*. El primer carácter del nombre no puede ser un dígito y no se pueden utilizar palabras reservadas, precisamente porque son reservadas por el lenguaje para su propio uso. Cada nombre debe identificar claramente un único elemento: una variable o una función, por lo cual una variable y una función no deben tener el mismo nombre.

Cuando se define una función cuadrado se ha creado una máquina de hacer cuadrados, se puede representar la función como una caja negra que transforma un dato de entrada en un dato de salida:

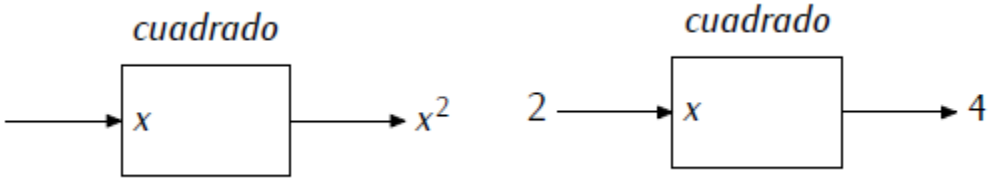


Figura 2: tomada de Marzal

La máquina se pone en funcionamiento cuando se invoca la función y se le da el valor de entrada, y ella produce el resultado esperado. Ahora suponga que se desea definir una función que dado el valor x, le calcule el seno de x, lo multiplique por x y retorne el resultado.

```
import math
def xsen(x):
    return x*math.sin(x)
print(xsen(math.pi/2))
```

```
1.5707963267948966

Process finished with exit code 0
```

Aquí se hizo lo siguiente: primero se importó el módulo math para poder utilizar la función sin y el número pi, luego se definió la función xsen, para recibir una valor x (parámetro formal) y con ese valor calcular x*sin(x), por ultimo por fuera de la función se muestra en consola el resultado invocando a la función xsen con el parámetro real o actual pi/2.

En el cuerpo de una función se pueden utilizar todas las estructuras vistas, es decir estructuras de control de selección y de repetición, bucles, expresiones aritméticas y lógicas, etc., por ejemplo se puede definir la siguiente función:

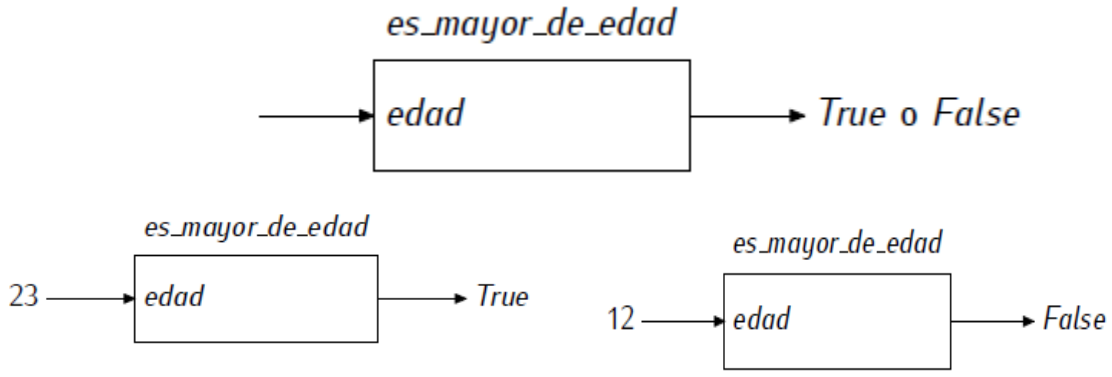


Figura 3: tomada de Marzal

```
def esmayor(edad):
    if edad < 18:
        resultado = False
    else:
        resultado = True
    return resultado

print(esmayor(18))
```

```
True

Process finished with exit code 0
```

Funciones sin parámetros vs funciones con varios parámetros

Lo único que hay que decir en las funciones sin parámetros es que los paréntesis son obligatorios tanto al definir como al invocar la función. Para el caso de funciones con varios parámetros tenemos como ejemplo, una función que recibe el alto y el ancho de un rectángulo y retorna el valor de su área.

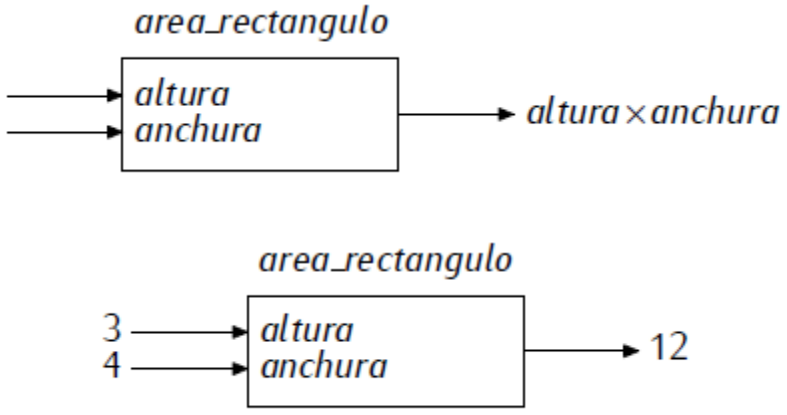


Figura 4: tomada de Marzal

Cualquier programa que se diseñe tendrá tres tipos de líneas o instrucciones: importación de módulos (o funciones y variables de módulos), definición de funciones y sentencias del

programa principal. Para una mayor legibilidad y claridad de un programa, se sugiere la siguiente organización: primero las importaciones, luego las definiciones de funciones y al final las sentencias que conforman la lógica del programa principal.

Procedimientos: Funciones sin devolución de valor

Una Función que no retorna un valor se denomina procedimiento. Debe entenderse que mostrar algo por pantalla no es devolver un valor. Con frecuencia se requieren subprogramas que calculen varios resultados en vez de uno solo, o que realicen el ordenamiento de una serie de números. En estas situaciones las funciones no son apropiadas y se necesitan los procedimientos.

Un procedimiento o subrutina es un subprograma que ejecuta un proceso específico. Ningún valor está asociado con el nombre del procedimiento, por consiguiente, no puede ocurrir en una expresión, como ocurre con las funciones. Un procedimiento se invoca por su nombre, por ejemplo, sort (ordenar). Cuando se invoca el procedimiento, los pasos que lo definen se ejecutan y a continuación se devuelve el control al programa que lo llamó.

Ámbito: variables locales y globales

Las variables según el lugar donde se declaren y definan pueden ser locales o globales. Una variable local es aquella que está declarada dentro de un subprograma y es distinta a variables con el mismo nombre, declaradas en cualquier parte del programa principal. El significado de una variable está confinado al procedimiento donde se declara. Cuando otro subprograma utiliza el mismo nombre se refiere a una posición diferente en memoria. Una variable global es aquella que está declarada para el programa o algoritmo principal, del que dependen todos los subprogramas. La parte del programa o algoritmo en donde una variable se define se conoce como ámbito o alcance (scope en inglés). El uso de variables locales tiene muchas ventajas porque hace a los subprogramas independientes, con la comunicación entre el programa principal y los subprogramas manipulados estructuralmente a través de la lista de parámetros. Para utilizar un procedimiento necesitamos sólo conocer lo que hace y no su diseño, es decir como hacen las cosas. Esto permite que diferentes programadores puedan trabajar los subprogramas de forma independiente. Si hay procedimientos que engloban otros procedimientos, anidados, entonces la noción de alcance debe mirarse con más cuidado. Por ejemplo, ver figura 5.

El ámbito de un identificador (variables, constantes, procedimientos) es la parte del programa donde se conoce el identificador. Si un procedimiento está definido localmente a otro procedimiento, tendrá significado sólo dentro del ámbito de ese procedimiento. A las variables les sucede lo mismo, si están definidas localmente dentro de un procedimiento, su significado o uso se confina a cualquier función o procedimiento que pertenezca a esa definición.

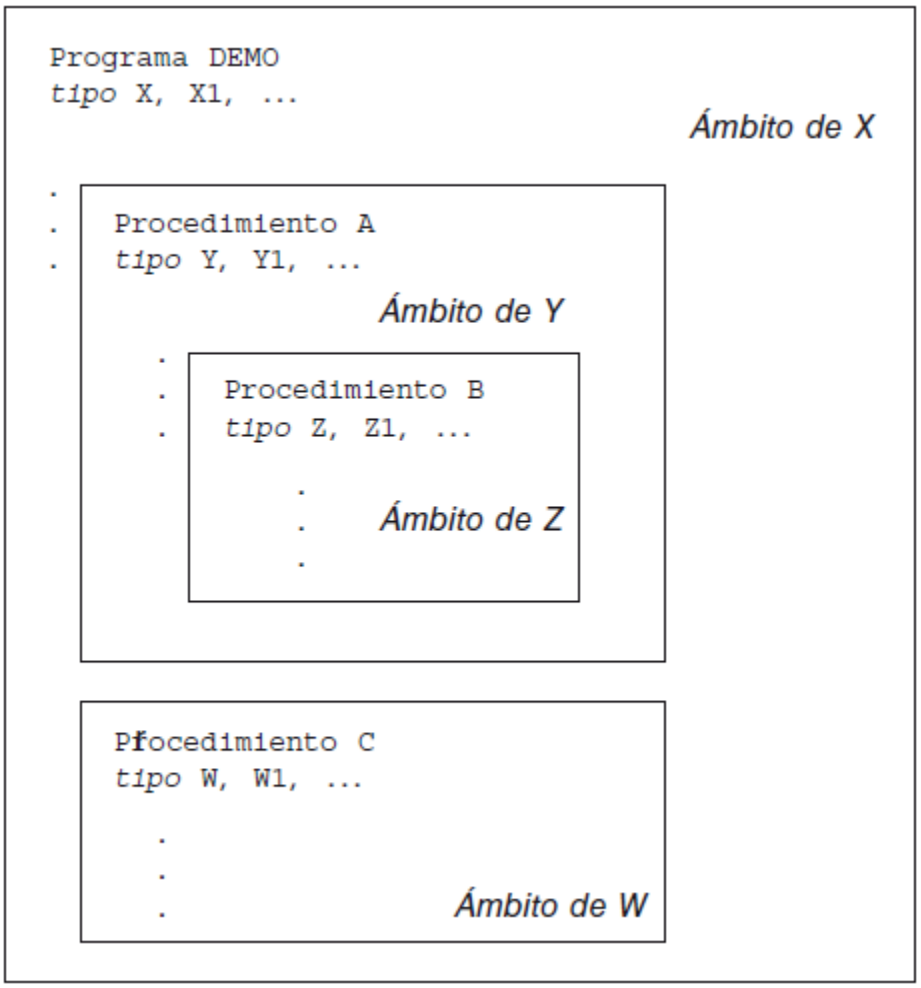
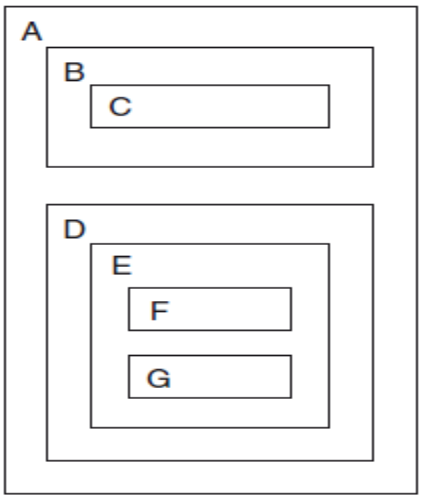


Figura 5: tomado de Joyanes

En la siguiente figura se muestra un esquema de programa con diferentes procedimientos, algunas variables son locales y otras globales.



Variables definidas en	Accesibles desde
A	A, B, C, D, E, F, G
B	B, C
C	C
D	D, E, F, G
E	E, F, G
F	F
G	G

Figura 6: tomado de Joyanes