Unit 8. Functions Basics

Héctor D. Menéndez

Endless Science

endsci.net

Index

Index

Functions

A function is a piece of code that performs a specific task.

It is normally formed by three parts:.

- The function name.

- The parameters.

- The output.

Some examples of functions are: `print()`, `sum()`, or `input()`.

Calling a functions

When we call a function we need to use its name with the parameters that we want to pass it.

For example, in the function `print("Hello")`, `print` is the name and the string `"Hello"` is the parameter.

The parameters allow the function to perform different tasks, in this case, to show different messages.

Example of common functions

```
print("Hello") #Says hello
name = input("What's your name?") #gets the user name
len(name) #calculates the number of characters of name
myList  = [1,2,3,4]
sum(myList) #Sum all the elements of myList
range(5) #Outputs the range from 0 to 4
round(3.4) #Rounds the value of the number
min(myList) #Gets the minimum of the list
max(myList) #Gets the maximum of the list
```

Listing 1: Common functions

Index

Function Declaration

We can **define our own functions** although it is important to have clear what's the behaviour that we want to encapsulate.

To declare a function we need to define: the **name**, the **inputs** and the **output**.

A function has the header and the body.

The **header** contains the name and parameters variables (or inputs), and the **body** is the function's code.

The function body will be in a **new block** and will normally end with a **return** statement where the output is defined.

Function Definition

```
1 def functionName(param1,param2,param3): #Header
2   ... #The function's code or body
3   return output #The value that the function returns.
```

Listing 2: Common functions

Function Example

```
1 def maxList(myList): #Header
2   maxElem=myList[0]
3   for elem in myList:
4     if(elem > maxElem):
5       maxElem=elem
6   return maxElem
7
8 print(maxList([1,2,3,4])) #This calls the function
```

Listing 3: Common functions

Inside of a Function

The function's body can have its **own variables**.

The variables only exist inside of the function, if you try to use them outside, the will not exist.

This is called the **function's scope**.

Function Example

```
1 def maxList(myList): #Header
2   maxElem=myList[0]
3   for elem in myList:
4     if(elem > maxElem):
5       maxElem=elem
6   return maxElem
7
8 print(maxList([1,2,3,4])) #This calls the function
9 print(maxElem) #This gives you an error
```

Listing 4: Common functions

Index

Function Call by Value

The function's parameters are not the real variables, when they are atomic types, they are normally copies.

If you modify these parameters inside of the function, the original variables are not modified.

This is called **function called by value**, because the original value is copied.

Called by Value Example

```
1 def sumTwoNumbers(number1,number2): #Header
2   number1=number1+number2
3   return number1
4
5 a=2
6 b=5
7 print(sumTwoNumbers(a,b)) #This calls the function
8 print(a) #This shows that a did not change
```

Listing 5: Common functions

Function Call by Reference

The function's parameters are **references** to the real variables when they are complex types.

If you change the **value of the "reference"** the original variable will not be changed, however, if you change a **value inside** of the variable, it will be changed in the original one.

This is because complex variables use **memory buffer** and Python do not copy these buffers only their references.

Called by Value Example

```
1 def changeElem(myList): #Header
2     myList[3] = 0
3     myList=[5,7,8]
4     return myList
5
6 myList=[1,2,3,4,5,6]
7 print(changeElem(myList)) #This calls the function
8 print(myList) #This shows that the original list
    changed
```

Listing 6: Common functions

Index

Return statement

The function's return statement provides the **function's output** or outputs.

There can be **none, one or multiple** return functions.

Once a return in reached during the code's execution, the rest of the function will **not be executed**.

No Return Example

```
1 def changeElem(myList): #Header
2     myList[3] = 0
3
4 myList=[1,2,3,4,5,6]
5 changeElem(myList)
6 print(myList)
```

Listing 7: Common functions

Multiple Return Example

```
1 def maxOfTwo(num1,num2): #Header
2     if(num1 > num2):
3         return num1
4     return num2
5
6 print(maxOfTwo(1,2))
7 print(maxOfTwo(2,1))
```

Listing 8: Common functions

Index

Exercise 1: List of Even Numbers

Write a function called `evenList` that receives a list of natural numbers as a parameter (you don't need to check that the list is correct). This function will return a new list containing every even number of the list. Call the function to test it (use the module % operator to check whether the number it even).

```
1 $> print(evenList([1,2,3,4,5,6,7,8,9,10]))
2 [2,4,6,8,10]
```

Exercise 2: Swap Function

Create a function called swap that receives three parameters: a list and two numbers. The numbers are indices or the list. The function will swap the elements in the two indexes from the original list. It will not return anything.

```
$> print(swap([1,2,3],0,1))
[2,1,3]
$> print(swap([1,2,3],2,1))
[1,3,2]
```

Exercise Extra: Bubble Sort

Extend the previous exercise to sort a list of numbers using the Bubble Sort algorithm. The algorithm's pseudocode is:

```
Set N = length(list)
for i = 0 to N-1:
  for j=0 to N-i-2:
     if list[j] > list[j+1]:
      swap(list,j,j+1)
```

And the behaviour will be:

```
$>myList=[4,2,1,3]
$>bubbleSort(myList)
$>print(myList)
[1,2,3,4]
```