FUNCTIONS & SCOPE OF VARIABLES

Functions

Definition: A function is a block of organized, reusable code that is used to perform a single, related action.

Functions can be divided into 2 types:

- Built-in Functions
- User defined functions

Built-in Functions

- The functions whose functionality is pre-defined in Python.
- The python interpreter has several functions that are always present for use.

A few examples of built-in functions are:

`chr()`, `abs()`, `len()`, `tuple()`, `dict()`.....etc.

User defined Functions

- A function that you define yourself in a program is known as user defined function.
- You can give any name to a user defined function, however you cannot use the Python keywords as function name.
- In python, we define the user-defined function using `def` keyword, followed by the function name (follows standard Python variable naming rules)

You can create a function in the following manner:

```
def func_name():
    #code
    #code
```

Built in Functions vs User Defined Functions

Python has a rich collection of built in functions. However, we still need user defined functions for the following reasons:

1. Functions help us divide the program into modules. This makes the code easier to manage and debug
2. It implements code reuse.

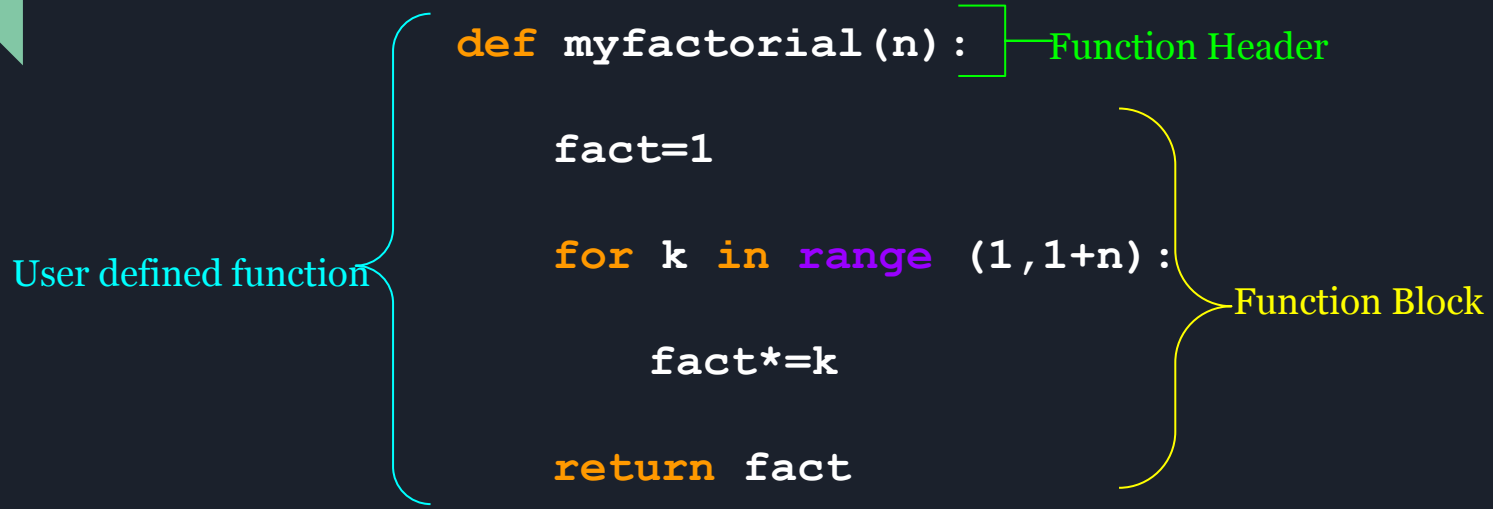
Henceforth, we will be solely dealing with User defined functions

Components of a function

It has 2 components:

- **Function Header**: it starts with keyword `def`, contains the name of the function and an optional list of formal parameters (*all functions need not have parameters)
- **Function Block**: it is the block after the function header. Function block contains equally indented statements that carry out certain actions inside the functions. Function block may contain the optional `return` statement.

Components of a function



Name of the function is `myfactorial()`, function returns a value and it has a formal parameter `n`. The function's `return` statement served 2 purposes:

- It terminates the function
- Returns a value to the caller

Return Statement

A function with a `return` statement will always return a value to the caller.

A function can return more than one values to the caller.

A `return` statement is followed by either a variable/constant.

Just a `return` statement without any variable/constant will return `None` to the caller.

A function without a `return` statement also returns a value `None`.

Parameters

Parameter is used to transfer data between caller and user defined function.

The data transfer can be either one way or two ways.

Parameter used in the function header is called formal parameter and the one used during the function call is called actual parameter. Formal parameter receives value from the actual parameter.

Datatype of the formal parameter depends on the data type of the actual parameter. An actual parameter can be a variable, expression or constant. But formal parameter is a variable.

When invoking a function, list of actual parameters must be equal to list of formal parameters. List of actual parameters must be compatible with the list of formal parameters.

Scope of Variables

The scope of a variable refers to the places that you can see or access a variable.

The variable a is therefore said to be **local** to the function. Put another way, the variable a has local scope.

If you define a variable at the top level of your script or module or notebook, this is a **global** variable

Global Variable vs Local Variable

All variables in a program may not be accessible at all locations in that program. This depends on where a variable has been declared. The scope of a variable determines the portion of the program where a Python script can access a particular variable.

There are 2 basic scopes of variables in Python:

- Global Variable
- Local Variable

Variables that are declared inside the function body (block) have a local scope, and a variable declared outside (either after or before) have a global scope.
This means that local variable can be accessed only inside the function in which they are declared, whereas global variable can be accessed throughout the Python script (anywhere in the program). When a function is called, the variable declared inside the body is brought into scope.

Scope of Variables: Examples

```
x=60                        #x is a Global Variable
def add(m,n):
    z=m+n                   #z is a Local Variable
    print(x,y,z)            #Displays 60 UCL 30

y='UCL'                     #y is a Global Variable
add(10,20)
print (x,y)                 #Displays 60 UCL
```

Scope of Variables: Examples

What happens when a local variable is accessed outside its scope?

```
x=100                    #x is a Global Variable
def foo():
    y='London'           #y is a Local Variable
    print(x,y,z)         #Displays x,y and z

z=7.2                    #z is a Global Variable
foo()
print (x,y,z)            #SYNTAX ERROR
```

Because the local variable `y` is accessed outside its scope

Keyword: `global`

As discussed earlier, a global variable can be accessed throughout the program. But this may not be true always.

```
x=10
def foo():
    x, y=100, 200
    print(x,y)

y=20
print(x,y)
foo()
print(x,y)
```

OUTPUT:

```
10 20
100 200
10 20
```

The output seems a little strange. Function `foo()` did not update global variables x and y. A global variable can be *accessed* anywhere in the program but global variable cannot be updated inside a function.

So, how can we do this?

Keyword: `global`

By using the keyword **`global`**, we can update a global variable.

```
x=10
def foo():
    global x, y
    x, y=100, 200
    print(x,y)

y=20
print(x,y)
foo()
print(x,y)
```

The keyword **`global`** will make variables **`x`** and **`y`** as global variables and hence, variables **`x`** and **`y`** can be updated inside the function `foo()`. The keyword **`global`** is to be used before accessing or updating the global variables x and y. Without using the keyword **`global`**, variable created inside a function with same name as a global variable, becomes the local variable.

OUTPUT:

```
10 20
100 200
100 200
```

THE END