Dr. D. Y. PATIL INSTITUTE OF TECHNOLOGY, PIMPRI, PUNE-18

Department of First Year Engineering

Programming and Problem Solving

Unit- 3 Notes

Unit III: Functions and Modules

3.1 Need for Functions

Q. Define Function and give its advantages.

Ans:

Function: A *function* is a block of organized and reusable program code that performs a single, specific and well-defined task.

Advantages

- Reducing duplication of code
- Decomposing complex problems into simpler pieces
- Improving clarity of the code
- Reuse of code
- Information hiding

Q. Explain the use / need for functions?

Ans:

- Dividing the program into separate well defined functions facilitates each function to be written and tested separately. This simplifies the process of program development. Following fig. shows that *function A* calls other functions for dividing the entire code into smaller functions.

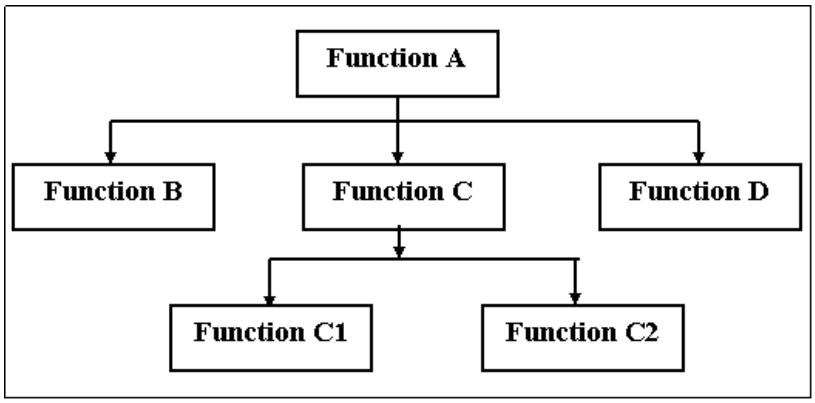

Fig: Top-down approach of solving a problem

- Understanding, coding and testing multiple separate functions are far easier than doing the same for one huge function.
- If big program has to be developed without the use of any function, then there will be large number of lines in the code and maintaining that program will be a big mess.
- All the libraries in Python contains pre-defined and pre-tested functions which the programmers are free to use directly in their programs without worrying about their code in detail.
- When a big program is broken into comparatively smaller functions, then different programmers working on that project can divide the workload by writing different functions.
- Like Python libraries, programmers can also make their own functions and use them from different points in the main program or any other program that needs its functionalities.
- **Code Reuse:** Code reuse is one of the prominent reason to use functions. Large programs follow DRY principle i.e. Don't Repeat Yourself principle. Once a function is written, it can be called multiple times within the same or by different program wherever its functionality is needed. Correspondingly, a bad repetitive code abides by WET principle i.e. Write Everything Twice or We Enjoy Typing. If a set of instructions have to be executed abruptly from anywhere within the program code, then instead of writing these instructions everywhere they are required, a better way is to place these instructions in a function and call that function wherever required.

3.2 Function Definition

Q. What is user defined function? With the help of example illustrate how you can have such functions in your program.

OR

Q. Explain with example how to define and call a function?

Ans:

How to define function

Function definition consists of a function header that identifies the function, followed by the body of the function containing the executable code for that function.

User Defined Function

The user defined functions, are functions created by users in their programs using **def** keyword.

To define user defined function following points must be considered:

- Function blocks start with the keyword **def**
- The keyword is followed by the **function name and parenthesis (( )).** The function name is uniquely identifying the function.
- After the parenthesis a **colon (:)** is placed
- **Parameters or arguments** that function accepts are placed **within parenthesis**. They are optional.
- The first statement of a function can be n optional statement-the **documentation string** of the function or *docstring* describe what the function does.
- The **code block** within the function is properly **indented** to form the block code.
- A function may have a **return[expression]** statement. It is optional.

A function definition comprises two parts:

- Function header
- Function body

Syntax:

def function_name(variable1, variable2,..): ← **Function Header**

Documentation string
Statement block
return [expression] ← **Function Boy**

How to call a function

- The function call statements invoke the function. When a function is invoked, the program control jumps to the called function to execute the statements that are part of that function.
- Once the called function is executed, the program control passes back to calling function.

Syntax of calling function:
function_name()

- Function call statement has the following syntax when it accepts parameters

function_name(variable1, variable2,..)

 # Program that subtract two numbers using function

```
def diff(x, y):                 #function to subtract two numbers
        return x-y
a=20
b=10
print(diff(a,b))                #function call
```

OUTPUT
10

3.3 Variable Scope and Lifetime

Q. Explain variable scope and lifetime.

Ans:

- **Scope** of the variable is the part of the program in which the variable is accessible.
- **Lifetime** of a variable is the period throughout which the variable exists in the memory.

3.3.1 Local and Global variables:

- **Local variables:**
  - Parameters and variables defined inside a function are not accessible from outside. Hence, they have a local scope.
  - The lifetime of variables inside a function is as long as the function executes.
  - They are destroyed once we return from the function. Hence a function does not remember the value of a variable from its previous calls.
- **Global Variables**:
  - The variables which are defined in the main body of the program file. Outside any function. Are called as global variables.
  - They are visible throughout the program.
- Here is an example to illustrate the scope of a variable inside a function.

```
def var_scope():
        x=5                                  # Here x is local variable
        print("Value inside function:",x)
x=10                                         #Here x is Global variable
var_scope()
print("value outside function:", x)
```

Output:

```
Value inside function: 5
Value outside function: 10
```

- In this example, we can observe that the value of x is 10 initially.
- Then the var_scope() function is called, it has the value of x as 5, but it is only inside the function.
- It did not affect the value outside the function.
- This happens since the variable x inside the function var_scope() is considered as different (local to the function) from the one outside.
- Although they have same names, they are two different variables with different scope.

Q. Differentiate between global and local variables

Ans:

Global variables	Local variables
1. They are defined in the main body of the program file	1. They are defined within a function and is local to that function.
2. They can be accessed throughout the program file	2. They can be accessed from the point of its definition until the end of the block in which it is defined
3.Scope of **global variables** is Throughout the program.	3.Scope of **local variables** is Within a function, inside which they are declared.
4.Life of **global variables:** Remain in existence for the entire time your program is executing.	4.Life of **local variables:** Created when the function block is entered and destroyed upon exit.

3.3.2 Using the global statement:

- To define a variable defined inside a function as a global, you must use the global statement.
- This declares the local or the inner variable of the function to have module scope.

- For ex.

```
x=”Good”
def show()
    global y
    y = “Morning”
    print(“In function x is – “, x)

show()
print(“Outside function y is – “, y)        # accessible as it is global variable
print(“ x is – “, x)
```

Output:

```
In function x is- Good
Outside function y is – Morning
x is - Good
```

3.3.3 Resolution of Names

Q. When we can have variable with same name as that of global variable in program, how is name resolved in python.

Ans:

- If we have variable with same name as Global and Local variable. Then local variable is accessible within the function/block in which it is defined.
- Global variable is defined outside of any function and accessible throughout the program.
- In the code given below, str is a global string because it has been defined before calling the function.
- Program that demonstrates using a variable defined in global namespace.

```
def fun():
        print(str)
str = "Hello World!!!"
fun()
```

```
Output:
    Hello World
```

- You cannot define a local variable with the same name as that of global variable. If you want to do that you just use the global statement.
- Program describes using a local variable with same name as that of global

```
def f():
    global str
    print(str)
    str= "hello world"
    print(str)

str= " welcome to python programming"
f()
```

```
Output:
welcome to python programming
hello world
```

3.4 The return statement

Q. Return statement is optional. Justify this statement with the help of example.

Ans:

- In python, every function is expected to have return statement.
- When we do not specify return statement at the end of our function, Implicit return statement is used as last statement for such function.
- This implicit return statement returns nothing to its caller, so it is said to return none.

- In the example below we can see there is no return statement used.
- So here default return none is used as last statement, and program will execute successfully.

```
def func1():
        x=5
        print("Value of x: ",x)
func1()
print("Hello World!)
```

```
Output:
Value of x: 5
Hello World
```

- So from this we can say that return statement is optional in function definition.

Q. Write a note on return statement with suitable example.

Ans:

- In python, every function is expected to have return statement.
- When we do not specify return statement at the end of our function, Implicit return statement is used as last statement for such function.
- This implicit return statement returns nothing to its caller, so it is said to return none.
- But you can change this default behavior by explicitly using the return statement to return some value to the caller.

- The syntax of return statement is:

return [expression]

- a return statement with no argument is same as **return none**
- A function may or may not return a value.
- The return statement is used for two things:
    1. Return a value to the caller
    2. To end and exit a function and go back to its caller
- Example 1: program to write a function without a return statement and try to print its return value. Such function should return none.

```
def display(str):
        print(str)
x= display("Hello World")
print(x)
print(display("Hello again"))
```

Output:

```
Hello World
None
Hello again
None
```

Note that in output None is return by the function

Example 2: Program for function which returns integer value to the caller.

```
def cube(x):
        return (x*x*x)
num = 10
result= cube(num)
print("Cube of ", num, "=", result)
```

Output:

```
Cube of 10 = 100
```

Note: return statement cannot be used outside the function.

3.5 Types of Parameters

Q. Explain types of function arguments. OR

Q. Write a note on:

1. **Required arguments**
2. **Keyword arguments**
3. **Default arguments**
4. **Variable-length arguments**

Ans: There are four types of function arguments available in Python as follow:

1. Required arguments
2. Keyword arguments
3. Default arguments
4. Variable-length arguments

1. Required Arguments:

- In the required arguments, the arguments are passed to the function in correct positional order.
- The number of arguments in the function call should exactly match with the number of arguments specified in the function definition
- For ex.

```
def person(name, age):
    print(name)
    print( age)
person(“Amar”, 20)
```

```
Output:
 Amar
 20
```

2. Keyword Arguments:

- In keyword arguments the order(or position) of the arguments can be changed.
- The values are assigned to the arguments by using their names
- The python interpreter uses keywords provided in the function call to match the values with parameters.
- Program to demonstrate keyword arguments

```
def person(name, age, salary):
    print(“Name: ”, name)
    print(“Age: ”, age)
    print(“Salary: ”, salary)

person(salary=50000, name=”Ajay”, age=30)
```

```
Output:
  Name: Ajay
  Age: 30
  Salary: 50000
```

- Note:
  - All the keyword arguments passed should match one of the arguments accepted by the function
  - The order of keyword argument is not important
  - In no case argument should receive a value more than once

3. Default Arguments:

- Python allows user to specify function arguments that can have default values
- Function call can have less number of arguments than provided in its definition.
- User can specify default value for one or more arguments which is used if no value provided in call for that argument.

```
def display(name, age=18):
    print("Name: "+ name)
    print("Age: ", age)

display(age=25, name="Ajay")
display("Amar")
```

```
Output:
  Name: Ajay
  Course: 25
Name: Amar
  Course: 18
```

- In the above example, default value is not specified for name, so it is mandatory.

- But age has provided default value, so if value not provided in calling for age, it will take 18.

Note: Non default arguments should be provided before default arguments.

4. Variable Length arguments:

- In some situations, it is not known in advance how many arguments will be passed to a function.
- I such cases variable length arguments can be used, in this function definition uses an asterisk(*) before the parameter name.

Syntax:

```
def function_name([arg1, arg2, …] *var_args_tuple )
    function statements
    return [expression]
```

Program to demonstrate the use of variable length arguments

```
def sumFun(a, *tup):
        sum1=a
        for i in tup:
            sum1= sum1 + i
        print(sum1)
sumFun(10, 20, 30, 40)
```

Output:

```
100
```

- In the above program, in the function definition we have two parameters- one is a and other is variable length parameter tup.
- The first value is assigned to a and other values are assigned to parameter tup.
- In tup parameter, 0 to n values are acceptable.
- The variable length argument if present in the function definition should be the last in the list of formal parameters.
- To see or use all the values in the variable length parameter, we have to use for loop as given in above example.

3.6 Lambda or Anonymous functions

Q. What is lambda or anonymous functions in python? Explain with example.

Ans.

- Lambda or anonymous function have **no name**.
- They are created using **lambda** keyword instead of **def**.
- Lambda function contains only single line.
- Lambda function can take any number of arguments.
- It can returns only one value in the form of an expression.
- Lambda function does not have explicit return statement.
- Lambda function cannot contain multiline expressions.
- It cannot access variable other than those in their parameter list
- We can pass Lambda function as arguments in other functions.
- **Syntax:**

  Lambda arguments: expression

  - The **arguments** contains comma separated list of variables
  - The **expression** is an arithmetic expressions that uses these arguments

- **Example:**

```
sum = lambda x, y: x + y
print("sum =", sum(3,5))
```

Output:

Sum=8

3.7 Documentation String

Q. What are docstring?

Ans:

- Docstrings (documentation string) serves the same purpose as that of comments.
- Docstrings are designed to explain code.
- Docstrings are created by putting multiline string to explain the code.
- Docstrings are important as they help tools to automatically generate printed documentation
- Docstrings are very helpful to readers and users of the code to interactively browse the code.
- Docstrings specified in function can be accessed through the **__doc__** attributes of the functions.
- Unlike comments, Docstrings are retained throughout the runtime of the program. (Docstrings ignored by the compilers)
- **Syntax:**

```
def function_name(parameters):
    """ Function written for factorial,
    Calculates factorial of number """
    Function statements
    return [expression]
```

- **Example:**

```
def func():
    """ function just prints message.
     It will display Hello World !!  """
    print("Hello World!!")
func( )
print(func.__doc__)
```

Output:

```
Hello World!!
function just prints message.
It will display Hello World !!
```

3.8 Good Programming Practices

Q. Write a note on good programming practices in python.

Ans:

To develop readable, effective and efficient code programmer has to take care of following practices:

- Instead of tabs, use 4 spaces for indentations.
- Wherever required use comments to explain code.
- Use space around operators and after commas.
- Name of the function should be in lower case with underscore to separate words

  Eg. get_data()
- Use document string that explains purpose of the functions
- Name of the class should be written as first letter capital in each word

  Eg. ClassName()
- Do not use non ASCII character in function names or any other identifier.
- Inserts blank lines to separate functions, Classes, and statement blocks inside functions

3.9 Introduction to Module

Q. What are modules? How do you use them in your program?

Ans:

Module:

- **Definition:** Module is a file with .py extension that has definitions of functions and variables that can be used even in other programs.
- Modules are the pre-written pieces of code.
- They are used to perform common tasks like generate random numbers, performing mathematical operations etc.
- **Using Modules in programs:**
  - Programs in which you want to use functions or variables defined in the module will simply import that particular module (or .py file).

- To use a module in any program, add import module_name as the first line in your program.

  Syntax:

  import module_name

- Then module_name.var is written to access functions and values with the name var in the module.

- **Example:** Program to print sys.path variable

```
import sys
print("PYTHONPATH = \n", sys.path)
```

- Here, sys standard library module is imported.
- When import sys statement is executed, python looks for sys.py module and then executes the statement to print sys.path variable.

- **Module Loading & Execution (optional)**
  - A module imported in a program must be located and loaded into memory before it can be used.
  - Python will search a module:
    - in current working directory
    - if not found there, it searches directories in the environmental variable PYTHONPATH.
    - If not found, then a Python installation specific path (like C:\Python34\Lib) is searched.
    - If the module is not located even there, then an error ImportError exception is generated.
  - In order to make a module available to other programs, it should be either saved in the directory specified in path PYTHONPATH, or stored in Python installation Lib directory.
  - Once a module is located, it is loaded in memory.

- A compiled version of module with file extension .pyc is generated.
- Next time when the module is imported, this .pyc file is loaded instead of .py file.
- A new compiled version of a module is again produced whenever the compiled module is out of date.
- Even the programmer can force the python shell to reload and recompile the .py file to generate a new .pyc file by using reload( ) function.

3.9.1 The from…import statement

- A module may contain definition for many variables or functions.
- When a module is imported, any variable or function defined in that module can be used.
- But if you want to use only selected variables or functions, then use the from…import statement.
- **Syntax**

```
from module_name import member_name
```

- **Example**:

```
from math import pi
print("PI= ",pi)
```

- To import more than one item from a module, use comma separated list. For example, to import value of pi and sqrt( ) from math module write:

  Example:

```
from math import pi, sqrt
print("PI= ", pi)
print("Square root of 4=", sqrt(4))
Output:
PI=  3.141592653589793
Square root of 4= 2.0
```

- To import all identifiers defined in a module, except those beginning with underscore (_), use import * statement.
- **Example:**

```
from math import *
print("PI= ", pi)
print("Square root of 4=", sqrt(4))
Output:
PI=  3.141592653589793
Square root of 4= 2.0
```

3.9.2 Name of Module

- Every module has a name.
- You can find the name of a module by using the __name__ attribute of the module.
- **Example:**

```
print("Hello")
print("Name of module is = ", __name__)
Output:
Hello
Name of module is = __main__
```

- Note that, for every standalone program written by the user the name of module is __main__.

3.9.3 Making your own module

- You can create your own modules.
- Every python program is a module.
- That is, every file that is saved with .py extension is a module.
- For example, let us create our own module say Mymodule.py
- In this file, some functionality is written.

```
File Name: Mymodule.py
def display():
    print("Hello")
```

```
str = "Welcome to World of Python"
```

- Then open another file (main.py) and write code given below.

```
File Name: main.py
import Mymodule
print("Mymodule string=", Mymodule.str)
Mymodule.display()
```

- When you run main.py file, you get following output.

```
Mymodule string= Welcome to World of Python
Hello
```

- Modules should be placed in the same directory as that of the program in which it is imported.
- It can also be stored in one of the directories listed in sys.path.
- The dot operator is used to access members (variables or functions) of module.

3.10 Packages

Q. What are packages in python?

Ans:

- A package is a hierarchical file directory structure.
- A package has modules and other packages within it.
- Every package in python is a directory which must have a special file called __init__.py. This may not even have a single line of code.
- It is simply added to indicate that this is not an ordinary directory and contains a Python package.
- A package can be accessed in other python program using import statement.
- For example, to create a package MyPackage, create directory called MyPackage having module MyModule and __init__.py file.

```
File Name: Mymodule.py
def display():
```

```
print("Hello")
str = "Welcome to World of Python"
```

- Now, to use MyModule in a program, import it in any one of the two ways:

```
import MyPackage.MyModule
or
from MyPackage import MyModule
```

3.11Introduction to Standard Library Modules

Q. Explain any four standard library modules.

Ans:

- Python supports 3 types of modules – those written by the programmer, those that are installed from external sources, and those that are pre-installed with python.
- Modules that are pre-installed in python together are known as the standard library.
- Some useful modules in standard library are string, re, datetime,math, random, os, multiprocessing, socket, email, json, doctest, unittest, pdb, argparse and sys.
- A few standard libraries in python are:

1. Math (import math)

   This is a package for providing various functionalities regarding mathematical operations.

   For ex. ***math.sqrt(9)***, will give you 3 as square root of 9.

2. Random (import random)

   This is a module which supports various functions for generation of random numbers and setting seed of random number generator.

   For ex. ***random.random()***, function will generate random number

3. Sys (import sys)

   It supports all system information related operations.

   For ex***. sys.path*** , will give you information about Current file Path.

4. Os (import os)

   It gives all the information related to Operating System.

   For ex. ***os.name*** , will give you the name of Operating System.