UNIT 2: Functions and Modules in Python

What is function?

Definition – A Function is a block of program statement that perform ***single***, ***specific*** and well defined ***task***. Python enables it‟s programmers to break the program into functions, each of which has some specific task.

When a function is called, the program control is passed to the first statement in the function. Allthe statements in the function are executed in sequence and the control is transferred back to function call. The function that calls another function is known as "***Calling Function***", and the function that is being called by another function is known as "***Called Function***".

Why we need Functions?

1. Simpler Code

A program‟s code seems to be simpler and easier to understand when it is broken down into functions. Several small functions are much easier to read than one long sequence of statements.

2. Code Reuse

Functions also reduce the duplication of code within a program. If a specific operation needs to be performed in several places in a program, then a function can be written once to perform that operation, and then be executed any number of times. This benefit of using functions is known as***code reuse*** because you are writing the code to perform a task once and then reusing it each time you need to perform that task.

3. Better Testing

When each task of the program is coded in terms of functions, testing and debugging will be simpler. Programmers can test each function in a program individually, to determine whether it iscorrectly performing its operation.

4. Faster Development

The entire program is divided into tasks. Each task is assigned to individual programmer or team. So that it will be easy to accomplish the program in a faster manner.

Types of Functions

There are two different types of functions, **built-in functions** and **user defined functions**. The functions such as input(), print(), min(), max() are example for the built-in functions. The user defined functions are created by user. The user selects his own name for the function name. The naming rules for the function name are same as identifier rule.

Defining Functions

To create a function we write its ***definition***. Here is the general format of a function definition in Python:

```
def function_name(): #function header
        statement1
        statement2          Function Body
        ……..
        Statement N
```

The first line is known as the ***function header***. It marks the beginning of the function definition. The function header begins with the key word **def**, followed by the name of the function, followed by a set of parentheses, followed by a colon (:).

The function body contains one or more statement. These statements are executed in sequence toperform the task for which it is intended to define.

Example function definition for even or odd:

```
#function definition
def eventest(x):
   if x%2==0:
      print("even")
   else:
      print("odd")
```

In the above function definition, “eventest” is the name of the function, “x” is the parameter or argument. The body contains some lines of code for finding whether a given number is even orodd.

Calling Functions

A function definition specifies what a function does, but it does not cause the function to execute. To execute a function, you must ***call*** it. This is how we would call the "eventest" function:

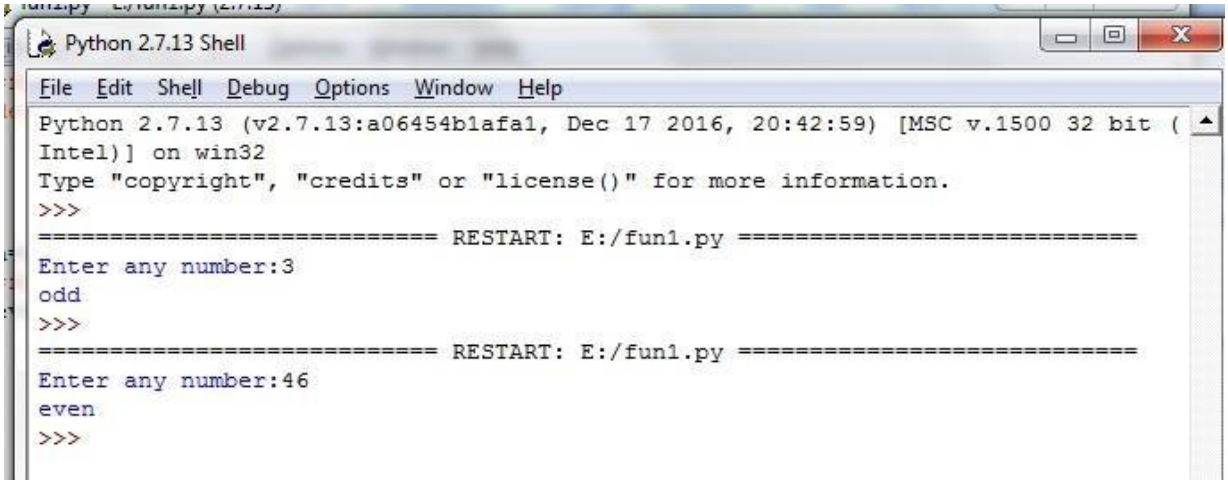
eventest(n)

When a function is called, the interpreter jumps to that function ***definition*** and executes the statements in its body. Then, when the end of the body is reached, the interpreter jumps back to the part of the program that called the function, and the program resumes execution at that point. When this happens, we say that the function ***returns***

Complete example for eventest.py

```
#function definition
def eventest(x):    #function header
   if x%2==0:
      print("even")
   else:
      print("odd")
n=int(input("Enter any number:"))
#function calling
eventest(n).
```

Output:



Passing Arguments to Function

An argument is any piece of data that is passed into a function when the function is called. This argument is copied to the argument in the function definition. The arguments that are in the

function call are known as "***Actual Arguments or Parameters***". The arguments that are in function definition are called "***Formal arguments or Parameters***". We can pass one or more number of actual arguments in the function call. The formal argument list and their type must match with actual arguments.

```
#function definition
def eventest(x):    #function header, here x is called Formal Argument or Parameter
    if x%2==0:
        print("even")
    else:
        print("odd")
n=int(input("Enter any number:"))
#function calling
eventest(n). # here n is called Actual Parameter
```

Example Program to calculate Simple Interest

```
#finding the simple interest

def si(p,t,r):    #here p,t,r are formal arguments

    s=(p*t*r)/100;

    print("The simple interest is:",s)

    T=p+s;

    print("The Total amount with interest is:",T)

n1=float(input("Enter principal amount:"))

n2=float(input("Enter number of months:"))

n3=float(input("Enter rate of Interest"))

#function call

si(n1,n2,n3) #here n1,n2,n3 are actual arguments
```

Note: Write an Example Program to calculate Compound Interest (A=p*(1+(r/n))^tn)

Keyword Arguments

When we call a function with some values, these values are passed to the formal arguments in the function definition based on their position. Python also allows functions to be called using the keyword arguments in which the order (position) of the arguments can be changed. The values are not copied not according to their position, but based on their names.

The actual arguments in the function call can be written as follow:

Function_name (Argument_name1=value1, argument_name2=value2)

An argument that is written according to this syntax is known as "***Keyword Argument***".

Example program: (1) Calculating Simple interest using keyword arguments.

```
#keyword arguments
def simpleinter(principal,rate,time): #function Header
    i=(principal*rate*time)/100
    print("The interest is:",i)
    print("Total amount is:",principal+i)
#function call
simpleinter(rate=7.25,time=3,principal=5000)
```

Output:

```
('The interest is:', 1087.5)
('Total amount is:', 6087.5)
```

The order of the actual arguments and formal arguments changed. Here, based on the name of the actual arguments the values are copied to the formal arguments. The position of the arguments does not matter.

Example program: (2) using the keyword arguments

```
#function definition
def disp(name,phone,email): #function header
    print "Your name:",name
    print " Your Phone Number:",phone
    print "Your email id:",email
disp(phone=9704,email="me@gmail.com",name="Rosum") #order of arguments is changed
```

Output:

```
Your name: Rosum
```

```
 Your Phone Number: 9704
Your email id: me@gmail.com
```

Note: keyword arguments make program code easier to read and understand.

Default Arguments

Python allows functions to be called without or with less number of arguments than that are defined in the function definition. If we define a function with three arguments and call that function with two arguments, then the third argument is considered from the default argument.

The default value to an argument is provided using the assignment (=) operator. If the same number of arguments are passed then the default arguments are not considered. The values of actual arguments are assigned or copied to formal arguments if passed, default arguments are considered otherwise. Hence, the formal arguments are overwritten by the actual arguments if passed.

General format of default arguments:

```
#function definition
def function_name(arg1=val1,arg2=val2,arg3=val3)
        Statement 1
        Statement 2
        Statement 3
#function call
function_name() #without arguments
function_name(val1,val2) #with two arguments, third argument is taken from default argument
```

Example Program:

```
#default arguments
def add(x=12,y=13,z=14):
   t=x+y+z
   print("The sum is:",t)
#function call without arguments
add()
#function call with one argument
add(1)
#function call with two arguments
add(10,20)
#function call with three arguments
add(10,20,30)
```

Output:

```
('The sum is:', 39)
('The sum is:', 28)
('The sum is:', 44)
('The sum is:', 60)
```

Variable-length arguments

In some situations, it is not known in advance how many number of arguments have to be passed to the function. In such cases, Python allows programmers to make function calls with arbitrary (or any) number of arguments.

When we use arbitrary arguments or variable-length arguments, then the function definition uses an asterisk (*) before the formal parameter name.

Syntax:

def fun_name([arg1,arg2,..argn],*var_length_tuple)

Example program:

```
#variable-length argument
def var_len_arg(name,*args):
    print "\n",name,"Hobbies are:"
    for x in args:
        print(x)
#function call
#subash is assigned to name, and rest of arguments are assigned to *args
var_len_arg("Subash","Cricket","Movies","Traveling")
#rajani is assigned to name, and rest of arguments are assigned to *args
var_len_arg("Rajani","Reading Books","Singing","Tennis")
```

Output:

```
Subash Hobbies are:
Cricket
Movies
Traveling
--------------------------

Rajani Hobbies are:
Reading Books
Singing
Tennis
```

Anonymous Functions

Lambda or anonymous functions are so called because they are not declared as other functionsusing the ***def*** keyword. Rather, they are declared using the lambda keyword. Lambda functions are throw-away functions, because they are just used where they have been created. Lambda functions contain only a single line. Its syntax will be as follw:

lambda arguments : expression

The arguments contain a comma separated list of arguments and the expression is an arithmetic expression that uses these arguments. The function can be assigned to a variable to give it a name.

Write a Python program using anonymous function to find the power of a number?

Program	**Output**
#lambda or anonymouse function n=lambda x,y: x**y x=int(input("Enter value of x :")) y=int(input("Enter value of y :")) ***#function call in the print() function*** print(x,"power",y,"is",n(x,y))	Enter value of x :3 Enter value of y :4 (3, 'power', 4, 'is', 81)

Properties of Lambda or Anonymous functions:

- ✓ Lambda functions have no name
- ✓ Lambda functions can take any number of arguments
- ✓ Lambda functions can just return a single value
- ✓ Lambda functions cannot access variables other than in their parameter list.
- ✓ Lambda functions cannot even access global variables.

Calling lambda function from other functions

It is possible to call the lambda function from other function. The function that uses the lambdafunction passes arguments to lambda function. The Lambda function will perform its task, and returns the value to caller.

Write a program to increment the value of x by one using lambda function.

Program	**Output**
#lambda or anonymouse functiondef inc(y): return(lambda x: x+1)(y) x=int(input("Enter value of x :")) #function call in the print() function	Enter value of x :3 ('the increment value of ', 3, 'is', 4)

print("the increment value of ",x,"is",inc(x))	

Fruitful Functions (Function Returning Values)

The functions that return a value are called "Fruitful Functions". Every function after performingits task it return the program control to the caller. This can be done implicitly. This implicit return return nothing to the caller, except the program control. A function can return a value to the caller explicitly using the "***return***" statement.

Syntax:

```
return (expression)
```

The expression is written in parenthesis that computes a single value. This return statement is used for two things: **First**, it returns a value to the caller. **Second**, To end and exit a function and go back to the caller.

Write a program using fruitful function to compute the area of a circle.

Program	Output

#finding the area of a circle
def area(r):
  a=3.14*r*r
  return(a) ***#function returning a value to the caller***
#begining of main function
x=int(input("Enter the radius :"))
#calling the function
r=area(x)
print("The area of the circle is:",r)
	Enter the radius :5
	('The area of the circle is:', 78.5)

Scope of the Variables in a Function

All variables in a program may not be accessible at all locations in that program. This depends on where you have declared a variable. The part of the program in which a variable can be accessed is called its ***scope***. The duration for which a variable exists is called its "***Lifetime***". If a variable is declared and defined inside a function its scope is limited to that function only. Itcannot be accessed to outside that function. If an attempt is made to access it outside that function an error is raised.

The scope of a variable determines the portion of the program where you can access a particular identifier. There are two basic scopes of variables in Python:

- **Global Scope** -variables defined outside the function and part of main program
- **Local Scope** - variables defined inside functions have local scope

Global vs. Local variables

Variables that are defined inside a function body have a local scope, and those defined outside have a global scope.

This means that local variables can be accessed only inside the function in which they are declared, whereas global variables can be accessed throughout the program body by all functions. When you call a function, the variables declared inside it are brought into scope. Following is a simple example –

```
total=0;# This is global variable.
# Function definition is here
def sum( arg1, arg2 ):
# Add both the parameters and return them."
    total= arg1 + arg2;# Here total is local variable.
        print"Inside the function local total : ", total
        return total;
# Now you can call sum function
sum(10,20);
print"Outside the function global total : ", total
```

```
tot=10
def sum(a,b):
    global tot #refering the global variable
    tot=a+b
    print("The sum is:",tot)
sum(12,3)
print("The global value of tot is:",tot)
```

Note: To refer the global variable "global" keyword is used.