Functions - 2

User Defined Functions

Function Basics

- There are two aspects to every Python function
  - **Function definition.** The definition of a function contains the code that determines the function's behavior.
  - **Function invocation (Calling).** A function is used within a program via a function invocation. Every function has exactly one definition but may have many invocations.
- Function definition consists of three parts
  - Function Name
    - Most Python functions have a name
  - Parameters
    - Every function definition specifies the parameters that it accepts from callers
  - Body
    - Every function definition has a block of indented statements

Definition of the function

- A function is a block of statements that performs a specific task.

  Syntax :- def functionName(Parameters):

  """ docString"""

  statement(s)

  return expression

  - Keyword def marks the start of function header.
  - A function name to uniquely identify it. Function naming follows the same **rules of writing identifiers** in Python.
  - Parameters (arguments) through which we pass values to a function. They are optional.
  - A colon **(:)** to mark the end of function header.
  - Optional documentation string (**docstring**) to describe what the function does.
  - One or more valid python statements that make up the function body. Statements must have same indentation level.
  - An optional **return** statement to return a value from the function.

Example for Defining Function

Function definition begins with "def."

Function name and its parameter.

```
def get_final_answer(filename):
    """Documentation String"""
    line1
    line2
    return total_counter
```

Colon.

The indentation matters…
First line with less
indentation is considered to be
outside of the function definition.

The keyword 'return' indicates the
value to be sent back to the caller.

Advantages of functions

- Using functions one can avoid rewriting the same code again and again
- It is easy to write a function that does a particular job
- The complexity of the entire program can be divided into simple subtask and function, subprograms can be written for each subtask.
- It is very efficient, clear and compact source code.
- The length of the program can be reduced.
- It is easier to write , testing and debugging individual functions
- It increases program readability and helps documentation.
- A function can call itself again ,it is called recursiveness.
- Many calculations can be done easily using recursive process such as calculation of sum of natural numbers, factorial of a number etc.

Sample Program using function

- Here def is keyword.
- **display** is function name and **name** is function argument.
- Inside we have created **docstring** it tells about functions what it does.
- In statements we have used **print()** to display to given string.
- We have just created the function and then called with name of function i.e. **display(x)**

```
#Sample Function Creation
def display(name):
    """To Display the string"""
    print("Hello "+name)
#Function Calling
x=input("Enter your Name: ")
display(x)
```

Arguments or Parameters

- The argument list contains valid variable names separated by commas.
- The argument variable receive values from a calling function .Thus they provide the link between the main program and the function.
- Arguments also known as **parameters** ,the parameters provided the means for communicating between functions.
- We have two types of arguments :- **formal(dummy)** arguments and **actual** arguments.
- The **formal arguments** are defined in the function declaration in the calling function .
- The data, which are passed from the **calling** function to **called** function are called the **actual** arguments .
- The **actual** arguments are passed to **called** function through a function call.
- A function may be without arguments ,in which case the argument list is empty.
- However, even if there are no arguments, the parentheses must be present.

Passing Arguments(values) to a Function

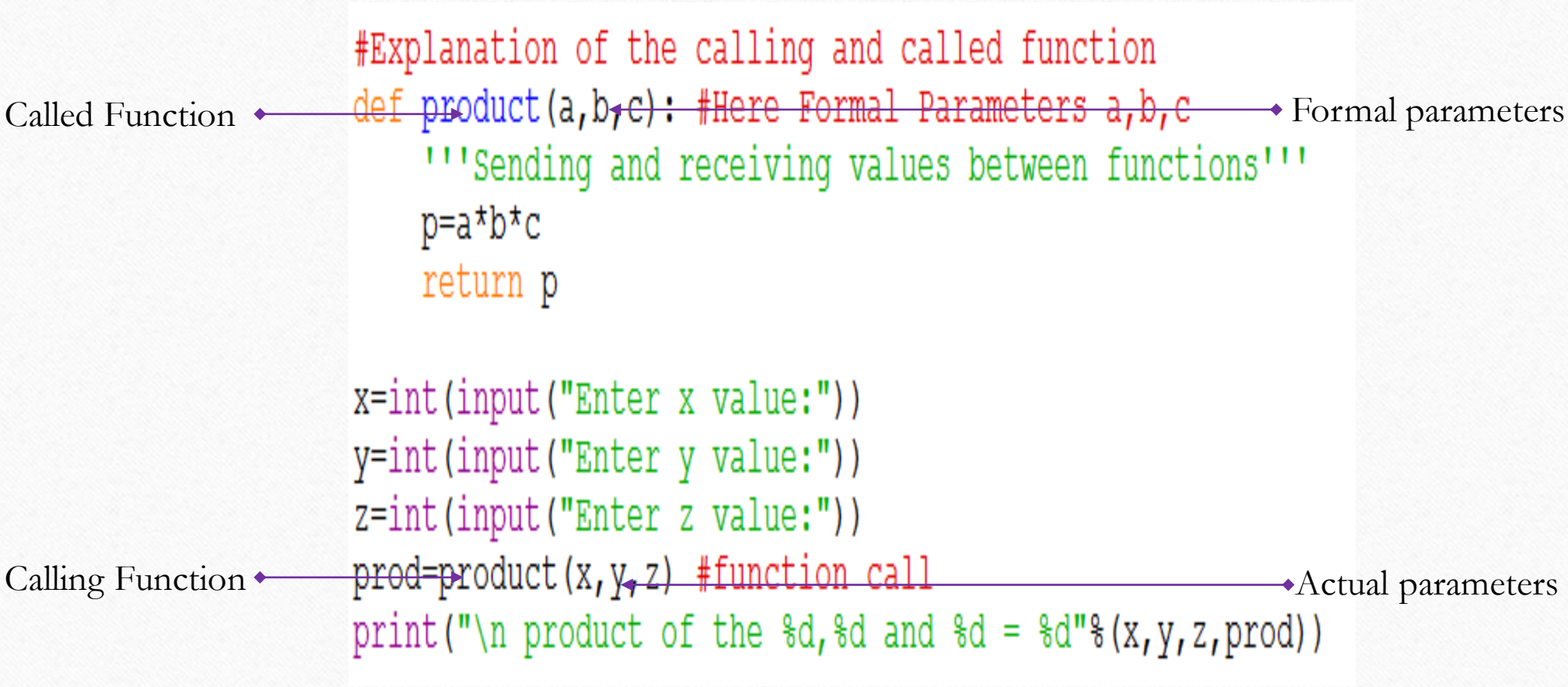
- Arguments to a function are usually passed into two ways.
- The first one is known as (sending the values of the arguments)**passing by value** or **call by value**.
- When a single value is passed to a function via an **actual** argument, the value of the **actual** argument is copied into the function.
- Therefore, the value of corresponding **formal** argument can be changed within the function, but the value of the actual argument within the calling function will not change.
- This procedure for passing the value of an argument to a function is known as **call by value**.
- The Second method is known as **call by reference** (i.e., sending the address of the argument),in which the address of the each argument is passed to the function .
- By this method ,the changes made to the parameters of the function will effect the variables which called the function

Calling a Function

- A function can be called by specifying function name, followed by a list of arguments enclosed in parentheses and separated by commas.
- If a function call does not require any arguments ,an empty pair of parentheses must follow the function's name.
- A function can be called any number of times .
- A function which returns a value can be used in expressions like any other variable.
- A function does not returns any value cannot be used in an expression.
- Any function can be called from any other function
- The following conditions must be satisfied for a function call .
  - There must a one –to –one correspondence between the actual and formal parameters.
  - Corresponding parameters(arguments) must be same data type.

Explanation of the calling and called function

- Sending and receiving values between functions



return statement

- A function may return any type data except sequences .
- **return** statement is used inside a function to exit it and return a value.
- The **return** statement also causes control to be returned to the point from which the function was called.
- A function may or may not send back any value to the calling function. It is an optional statement .

Purpose of return Statement :-

- Transferring control from the function back to the calling program (None).
- It returns the **value** present in the parentheses after **return** to the calling program.
- General form of the **return statement**
  - return or return (expression)

Characteristics of a Function

- Any function can return only one value.
- Parameter argument list is optional
- **return** statement indicates exit from the function and return to the point from where the function was invoked.
- A function can call any number of times
- Any function cannot be defined in other function
- When a function is not returning any value ,void type can be used as return type.
- Python allows recursion i.e. a function can call itself.

Function Categories

- In python programing language mainly we can categories two types:-
  - **Built – in Functions** are developed along with language we can call it as a pre-defined functions

    (i.e. math.sqrt(9),list.append(5)…etc)
  - **User – defined functions** here user has to develop in existing circumstance.

    def is keyword is used to create user –defined functions in python

  Syntax :- def functionName(Parameters):

  “““““ docString”””””

  statement(s)

  return expression
- A function ,depending on whether arguments are present or not and whether a values is returned or not, may belong to one of the following categories
  - Functions with no arguments and no return values
  - Functions with arguments and no return values
  - Functions with arguments and return values

Functions with no arguments and no return values

- It is simplest way of writing a user defined function in python.
- When function has no arguments , it does not receive any data from the calling function.
- Similarly ,when it does not return a value.
- The calling function does not receive any data from the called function
- In effect ,there is no data communication between a calling and a called function.
- Note that the function does not contain any **return** statement

```
#syntax:-
    def fun(): #called function with no formal arguments
        print()
    fun()#calling function with no actual arguments

#Write a python to find the largest of three given
#numbers using a function ?

def big(): #called function with no parameters
    '''no arguments and no returns values '''
    x=int(input("Enter x value:"))
    y=int(input("Enter y value:"))
    z=int(input("Enter z value:"))
    max=x
    if y>max:max=y
    if z>max:max=z
    print("\nThe Largest Number among three Numbers")
    print("%d,%d and %d = %d"%(x,y,z,max))

#calling Function with no parameters
big()
```

Functions with arguments but no return values

- These functions have some arguments which are passed to a function but the functions does not return back any value to the accessing function.
- It is one way data communication between the calling function and called function.
- The function does not return any value.

```
'''syntax:-
    def fun(x,y): #called function with no formal arguments
        print(x+y)
    fuc(1,2)#calling function with no actual arguments
'''
#Write a python to find the smallest of
#three umbers using a function ?

def small(x,y,z): #called function with Formal parameters
    '''with arguments but no returns values '''
    min=x
    if y<min:min=y
    if z<min:min=z
    print("\nThe smallest of Three Numbers")
    print("%d,%d and %d = %d"%(x,y,z,min))

a=int(input("Enter a value:"))
b=int(input("Enter b value:"))
c=int(input("Enter c value:"))
#calling Function with Actual parameters
small(a,b,c)
```

Functions with arguments and return values

- These functions have some arguments and are passed to a function from the calling program and computed values .
- If any ,are transferred back to the calling function .
- It is two way data communication between the calling and called function.
- These functions receives a predefined form of input and outputs a desired value.

```
'''syntax:-
    def fun(x,y): #called function with no formal arguments
        t=x+y
        retun t
    sum=fuc(1,2)#calling function with no actual arguments
    print(sum)
'''
#Write a python to find the smallest of
#three umbers using function ?

def small(x,y,z): #called function with Formal parameters
    '''with arguments but no returns values '''
    min=x
    if y<min:min=y
    if z<min:min=z
    return min

a=int(input("Enter a value:"))
b=int(input("Enter b value:"))
c=int(input("Enter c value:"))
#calling Function with Actual parameters
minimum=small(a,b,c)

print("\nThe smallest of Three Numbers")
print("%d,%d and %d = %d"%(a,b,c,minimum))
```

Variable Scope and Lifetime

- In python ,you cannot access any variable from any part of the program.
- Some of the variables may not even exist for the entire duration of the program.
- In which part of the program we can access a variable and in which part of the program a variable exists depends on how the variable has been declared.
- Therefore we need to understand two things:

Scope of the variable

Part of the program in which a variable is accessible is called its scope

Lifetime of the variable

Duration for which the variable exists is called its lifetime

Local Variable

- Variable declared within functions are called local variable .
- They are known only in the function in which they are declared and They are not known to other functions.
- Local variable in one function have no relationship to the local variable in another function.
- In other words ,local variables of the same name in different functions are unrelated.
- Local variable are created when a function is called ,and they are destroyed when the function is exited,
- Therefore local variable do not maintain their value between function calls

```
#Exmple for local variable
x=10 # Here x is the local to entire program
def fun():
    x=12
    print("funcation value is",x)
print("Local Value is ",x)
fun()
print("Local Value is ",x)
```

nonlocal Keyword

- The use of **nonlocal** keyword is very much similar to the global keyword.
- nonlocal is used to declare that a variable inside a nested function is not local to it, meaning it lies in the outer inclosing function.
- If we need to modify the value of a non-local variable inside a nested function, then we must declare it with nonlocal.
- Otherwise a local variable with that name is created inside the nested function.
- Here, the innerSum() is nested within the outerSum() function and the variable b is in the outerSum().
- So, if we want to modify it in the innerSum(), we must declare it as nonlocal and notice that b is not a global variable.

```
#Example for nonlocal keyword

a=30 #local or global to the program
def outerSum():
    b = 50
    def innerSum():
        nonlocal b
        b = 70
        sum=a+b
        print("Inner Sum: ",sum)
    innerSum()
    sum=a+b
    print("Outer Sum: ",sum)
outerSum()
#print(b)
```

- The result of not using the nonlocal keyword is as follows:
- Here, we do not declare that the variable a inside the nested function is nonlocal.
- Hence, a new local variable with the same name is created, but the non-local b is not modified

```
#Example for without nonlocal keyword
a=30 #local or global to the program
def outerSum():
    b = 50
    def innerSum():
        b = 70
        sum=a+b
        print("Inner Sum: ",sum)
    innerSum()
    sum=a+b
    print("Outer Sum: ",sum)
outerSum()
#print(b)
```

Global Variable

- Variable declared outside functions are called global variables and they may be accessed by any function in the program.
- It need not be declared in other functions .
- A global variable is also known as external variable.
- The global variables provides two-way communications among the functions.

```
#Exmple for global variable
x=10 # Here x is global any fuction can access
def sumFunc():
    y=12
    sum=x+y
    print("funcation value is",sum)

print("x is global and Local ,Value is ",x)
sumFunc()
print("x is global and Local ,Value is ",x)
```

global Keyword

- **global** is used to declare that a variable inside the function is global.
- If we need to read the value of a global variable, it is not necessary to define it as global and this is understood.
- But if we need to modify the value of a global variable inside a function, then we must declare it with global.
- Otherwise a local variable with that name is created

```
#Exmple for global keyword variable
x=10 # Here x is global any fuction can access
def sumFunc():
    global y
    y=12
    sum=x+y
    print("funcation value is",sum)

print("x is global and Local ,Value is ",x)
sumFunc()
print("y is global and Local to the sumFunc() ,Value is ",y)
```

More on Defining Function

- Required Arguments
- Keyword Arguments
- Default Arguments
- Variable – Length / Arbitrary Arguments

Required arguments

- The number of arguments in the function call should exactly match with the number of arguments specified in the function definition.
- otherwise a typeError or argument error is returned.

```
#Sample Function Creation
def display():
    """To Display the string"""
    print("Hello")
#Function Calling
display('Hi')
#TypeError: display() takes 0 positional arguments but 1 was given

def display(str):
    """To Display the string"""
    print(str)
#Function Calling
display()
#TypeError: display() missing 1 required positional argument: 'str'

def display(str):
    """To Display the string"""
    print(str)
#Function Calling
str1='Hello'
display(str1)
```

Keyword Arguments

- When we call a function with some values , the values are assigned to the arguments based on their position.
- Python allows functions to be called using keyword arguments in which the order of the arguments can be changed.
- The values are not assigned to arguments according to their position but based on their name or keyword.

```
#Sample Function Creation
def display(name,age,salary):
    """To Display the string"""
    print("Name:",name)
    print("Age :",age)
    print("Salary :",salary)
#Function Calling
#display(45,'hi',34)
display(salary=567,name='Jai',age=31)
```

Default Arguments

- Python allow users to specify function arguments that can have default values .
- Function can be called with fewer arguments than it is defined to have .
- If the function accepts three parameters , but function call provides only two arguments ,then the third parameter will be assigned the default value.
- The default value to an argument is provided by using the assignment operator(=)

```
#Sample Function Creation
def display(name,msg='How are you?'):
    """To Display the string"""
    print("Hello ",name+','+msg)

#Function Calling
display('Kumar')
display('Abhi','Goodmorning')
```

Variable – Length / Arbitrary Arguments

- In some situation ,it is not known in advance how many arguments will be passed to a function.
- In such case python allows programmers to make function calls with arbitrary (or any) number of arguments .
- When use arbitrary argument or variable length arguments ,then the function definition uses an asterisk (*)before parameter name.

  Syntax: def funName(*names):
- Inside the called function ,for loop is used to access the arguments.

```
#Sample Function Creation
def display(*name):
    """To Display the string"""
    for i in name:
        print("Hello ",i)
#Function Calling
display('Hi','How are you')
```

Recursive Functions

- A function which invokes itself repeatedly until some condition is satisfied ,is called a recursive function.
- A recursive function is defined as a function that calls itself to solve a smaller version of its task until final call is made which does not require a call to itself.
- Every recursive function has two major cases : **base case ,recursive case**
- **Base case :-** in which case the problem is simple enough to be solved directly without making any further calls to the same function.
- **Recursive case:-** in which case
  - First the problem at hand is divided into simpler sub-parts,
  - Second ,the function calls itself but with sub-parts of the problem obtained in the first step.
  - Third ,the result is obtained by combining the solutions of simpler sub-parts
- The recursive utilized **divide and conquer technique** of the problem solving.

- Divide and conquer technique is a method of solving a given problem by dividing into two or more smaller instances.
- Each of the smaller instance is recursively solved ,and the solutions are combined to produce a solution for original problem.
- Every recursive function must have at least one base case, otherwise, the recursive function will generate an infinite sequence of calls thereby resulting in an error condition known as infinite stack.
- Let us calculating factorial of a number, **n!**
- What we have to do is multiply the number with factorial of number that is 1 less than that number, in other words n!=n*(n-1)!

Let us take n=5 than we need to find n!=5!

```
def factorial(n):
    if n==1 or n==0:#base case
        return 1
    else:
        return n*factorial(n-1) #recursive case
x=int(input("Enter the value of n:"))
print("The Factorial of ",x,"is",factorial(x))
```

```
#Simple Recursion Program
'''
n!=5!
5!
5 * 4!
5 * 4 * 3!
5 * 4 * 3 * 2!
5 * 4 * 3 * 2 * 1!
'''
```

- Note :- The base case of a recursive function act as the terminating condition.
- Write a program to find the **Greatest Common Divisor**

Explanation: -

a=62,b=8

gcd(62,8)

rem=62 % 8=6

gcd(8,6)

rem=8 % 6=2

gcd(6,2)

rem=6%2=0

return 2

return 2

return 2

```
#Write a program to find the Greatest Common Divisor
def gcd(x,y):
    '''The Greatest Common Divisor
    of 2 numbers'''
    rem=x%y
    if rem==0:#base case
        return y
    else:
        return gcd(y,rem)#recursive case
a=int(input("Enter the Value of a:"))
b=int(input("Enter the value of b:"))
if b>a:
    x=b
    y=a
else:
    x=a
    y=b
print("The GCD of number is ",gcd(x,y))
```

Practice Problems

- Write a program to print the Fibonacci series using recursion?
- Write a program to calculate the exponent(x,y) using recursion?
- Write a program using functions and return statement to check whether a number is even or odd?
- Write a program to convert hours into minutes?
- Write a program to determine whether a given number is prime or not using a function?
- Create a simple calculator using functions ?
- Write a program to find the reverse number using functions?
- Write a program to find the sum of digits of from given number using functions?
- Write a program in python to find out the number of words in a line of text using a function (Avoid Pre-defined Methods) ?
- Write a program to display the number into words (i.e., 12 – OneTwo)?