Functions

Function is a block of code that performs specific task.A Function can be called as a section of a program that is written once and can be executed whenever required in the program, this making code reusability.

Types of Functions:

There are two types of Functions.

Built-in Functions: Functions that are predefined. We have used many predefined functions in Python like all,any,sorted functions

User- Defined: Functions that are created according to the requirements.

Advantages of using functions

Simpler Code
A program's code tends to be simpler and easier to understand when it is broken down into functions.
Code Reuse
Functions also reduce the duplication of code within a program. If a specific operation is performed in several places in a program, a function can be written once to perform that operation and then be executed any time it is needed
Better Testing
When each task within a program is contained in its own function, testing and debugging becomes simpler
Faster Development
It doesn't make sense to write the code for these tasks multiple times. Instead, functions can be written for commonly needed tasks & those functions can be incorporated into each program that needs them.
Easier Facilitation of Teamwork
Functions also make it easier for programmers to work in teams. When a program is developed as a set of functions that each performs an individual task, then different programmers can be assigned the job of writing different functions.

Void Functions and Value-Returning Functions
When you call a void function, it simply executes the statements it contains and then terminates. When you call a value-returning function, it executes the statements that it contains, and then it returns a value back to the statement that called it.

Defining a Function

A Function defined in Python should follow the following format:

1. Keyword def is used to start the Function Definition. def specifies the starting of Function block.
2. def is followed by function-name followed by parenthesis.
3. Parameters are passed inside the parenthesis.
4. At the end a colon is marked.
5. Before writing a code, an Indentation (space) is provided before every statement. It should be same for all statements inside the function.

Syntax:

```
def <function_name>([parameters]):
        statement_1
        statement_2
        ......
        .....
```

Example

```
def sum(a,b):
        s=a+b
        print (“Sum of two numbers is”,s)
```

Invoking a Function:

To execute a function it needs to be called. This is called function calling.Function Definition provides the information about function name, parameters and the definition what operation is to be performed. In order to execute the Function Definition it is to be called.

Syntax:<function_name>(parameters)

Example

```
sum(a,b)
```

This program demonstrates a function

Example 1:

```
# First, we define a function named message.
def message( ):
        print("Iam working on it,")
        print("let you know once I finish it off")
# Call the message function.
message( )
```

Example 2:

```
# This program has two functions. First we define the main function.
def main( ):
        print('I have a message for you.')
        message( )
        print('Goodbye!')
# Next we define the message function.
def message( ):
        print("Iam working on it,")
        print("shall let you know once I finish it off")
# Call the main function.
main( )
```

return Statement:

return[expression] is used to send back the control to the caller with the expression. In case no expression is given after return it will return None. In other words return statement is used to exit the Function definition.

Example:

```
def sum(a,b):
        "Adding the two values"
        print("Printing within Function")
        print(a+b)
        return (a+b) # Try with out return
def msg():
        print("Hello")
        return
total=sum(10,20)
print("Printing Outside:",total)
msg( )
print("Rest of code")
```

Argument and Parameter:

There can be two types of data passed in the function.

1. The First type of data is the data passed in the function call. This data is called arguments. Arguments can be variables and expressions.arguments can be called as actual parameters or actual arguments
2. The second type of data is the data received in the function definition. This data is called parameters.Parameters must be variable to hold incoming values.parameters can be called as formal parameters or formal arguments.

Passing Parameters

Python supports following types of formal argument:

1. Positional argument (Required argument).
2. Default argument.
3. Keyword argument (Named argument)
4. Arbitary Arguments

Positional/Required Arguments:
When the function call statement must match the number and order of arguments as defined in the function definition it is Positional Argument matching
Example

```
#Function definition of sum
def sum(a,b):
        "Function having two parameters"
        c=a+b
        print(c)
sum(10,20)
sum(20)
```

Explanation:

1. In the first case, when sum() function is called passing two values i.e., 10 and 20 it matches with function definition parameter and hence 10 and 20 is assigned to a and b respectively. The sum is calculated and printed.
2. In the second case, when sum() function is called passing a single value i.e., 20 , it is passed to function definition. Function definition accepts two parameters whereas only one value is being passed, hence it will show an error.

Default Arguments

Default Argument is the argument which provides the default values to the parameters passed in the function definition, in case value is not provided in the function call.

Example

```
 #Function Definition
def msg(Id,Name,Age=21):
        "Printing the passed value"
        print(Id)
        print(Name)
        print(Age)
#Function call
msg(Id=100,Name='James',Age=20)
msg(Id=101,Name='Ratan')
```

Explanation:

1) In first case, when msg() function is called passing three different values i.e., 100 , Ravi and 20, these values will be assigned to respective parameters and thus respective values will be printed.
2) In second case, when msg() function is called passing two values i.e., 101 and Ratan, these values will be assigned to Id and Name respectively. No value is assigned for third argument via function call and hence it will retain its default value i.e, 21.

Keyword Arguments:

Using the Keyword Argument, the argument passed in function call is matched with function definition on the basis of the name of the parameter.

Example:

```
def msg(id,name):
        "Printing passed value"
        print(id)
        print(name)

msg(id=100,name='Raj')
msg(name='Rahul',id=101)
```

Explanation:

1. In the first case, when msg() function is called passing two values i.e., id and name the position of parameter passed is same as that of function definition and hence values are initialized to respective parameters in function definition. This is done on the basis of the name of the parameter.

2. In second case, when msg() function is called passing two values i.e., name and id, although the position of two parameters is different it initialize the value of id in Function call to id in Function Definition. same with name parameter. Hence, values are initialized on the basis of name of the parameter.

Arbitary Arguments

Sometimes, we do not know in advance the number of arguments that will be passed into a function.Python allows us to handle this kind of situation through function calls with arbitrary number of arguments.

In the function definition we use an asterisk (*) before the parameter name to denote this kind of argument.

Example

```
def greet(*names):
  """This function greets all
  the person in the names tuple."""
  # names is a tuple with arguments
  for name in names:
    print("Hello",name)
greet("Monica","Luke","Steve","John")
```

Scope of Variable:

Scope of a variable can be determined by the part in which variable is defined. Each variable cannot be accessed in each part of a program.
There are two types of variables based on Scope:

1. Local Variable.
2. Global Variable.

1) Local Variables:

Variables declared inside a function body is known as Local Variable. These have a local access thus these variables cannot be accessed outside the function body in which they are declared.
Example:

```
def msg( ):
        a=10
        print("Value of a is",a)
        return
msg( )
print(a )#it will show error since variable is local
```

Output:

```
('Value of a is', 10)
Traceback (most recent call last):
  File "test.py", line 6, in <module>
    print(a )#it will show error since variable is local
NameError: name 'a' is not defined
```

b) Global Variable:

Variable defined outside the function is called Global Variable. Global variable is accessed all over program thus global variable have widest accessibility.

Example:

```
b=20
def msg( ):
        a=10
        print ("Value of a is",a)
        print ("Value of b is",b)
        return

msg( )
print (b)
```

How to access userdefined function in other program

Step 1:Create User defined functions

Step 2: import userdefined function file name as a module name without file extension in another python file.

Step 2: Access those functions with M***odule name. Function name*** Here function name means the name given to the function in the user defined function.

Example

Creating User defined functions

file name : operations.py

```
def add(a,b):

        c=a+b

        return c

def div(a,b):

        c=float(a)/float(b)

        return c
```

file name: sub.py

```
def sub(a,b):
        c=a-b
        return c
```

file name:mul.py

```
def mul(a,b):
        c=a*b
        return c
```

Accesing user defined functions:

file name:calc.py

```
import operations
import mul
import sub
x=input("Enter a number")
y=input("Enter a number")
z=addition.add(x,y)
print z
print addition.div(x,y)
print sub.sub(x,y)
print mul.mul(x,y)
```

Exercise:

1. Write a Python function that checks whether a passed string is palindrome or not?

```
def isPalindrome(string):
        left = 0
        right = len(string) - 1
        while right >= left:
                if not string[left] == string[right]:
                        return False
                left += 1
                right -= 1
        return True
print(isPalindrome('mom'))
```

2. Write a Python function to check whether a number is perfect or not?

```
def perfect_number(n):
```

```
    sum = 0
    for x in range(1, n):
        if n % x == 0:
            sum += x
    return sum == n
print(perfect_number(6))
```

Recursive Function

A function called by itself is called as recursion. Typically when a program employs recursion the function invokes itself with a smaller argument Computing factorial(5) involves computing factorial(4), computing factorial(4) involves computing factorial(3) and so on

Example

```
def factorial(x):
   if x == 1:
      return 1
   else:
      return (x * factorial(x-1))
num = 4
print("The factorial of", num, "is", factorial(num))
```

Advantages of recursion

1. Recursive functions make the code look clean and elegant.
2. A complex task can be broken down into simpler sub-problems using recursion.
3. Sequence generation is easier with recursion than using some nested iteration.

Disadvantages of recursion

1. Sometimes the logic behind recursion is hard to follow through.
2. Recursive calls are expensive (inefficient) as they take up a lot of memory and time.
3. Recursive functions are hard to debug.

Exercise

1. Python program to display the Fibonacci sequence up to n-th term using recursive functions?

```
def recur_fibo(n):
   if n <= 1:
      return n
   else:
      return(recur_fibo(n-1) + recur_fibo(n-2))
nterms = 10
```

```
if nterms <= 0:
   print("Plese enter a positive integer")
else:
   print("Fibonacci sequence:")
   for i in range(nterms):
      print(recur_fibo(i))
```

2. Python program to find the sum of natural numbers up to n using recursive function?

```
def recur_sum(n):
   if n <= 1:
      return n
   else:
      return n + recur_sum(n-1)
num = 16
if num < 0:
   print("Enter a positive number")
else:
   print("The sum is",recur_sum(num))
```

3. Write a Python program to calculate the sum of a list of numbers?

```
def listsum(mylist):
    if len(mylist) == 1:
        return mylist[0]
    else:
        return mylist[0] + listsum(mylist[1:])

print(listsum([2, 4, 5, 6, 7]))
```

4. Write a Python program to get the sum of a non-negative integer?

```
def sumdigits(n):

  if n == 0:
    return 0
  else:
    return n % 10 + sumdigits(int(n / 10))

print(sumdigits(345))
print(sumdigits(45))
```

Lambda or Anonymous Function:

In Python, anonymous function is a function that is defined without a name. While normal functions are defined using the `def` keyword, in Python anonymous functions are defined using the `lambda` keyword.Hence, anonymous functions are also called lambda functions.

Syntax:

lambda [arg1,[args2,[args3.....argsn]:expression

Lambda functions can have any number of arguments but only one expression. The expression is evaluated and returned.

Example 1:

```
double = lambda x: x * 2

print(double(5))
```

Explanation:

In the above program, `lambda x: x * 2` is the lambda function. Here *x* is the argument and `x * 2` is the expression that gets evaluated and returned. This function has no name. It returns a function object which is assigned to the variable `double`

Example 2:

```
#Function Definiton

total= lambda num1,num2:num1+num2

# Function Call

print("Value of total:",total(10,20))

print("Value of total:", total(100,200))
```

Example 3:

```
square=lambda a:a*a

print("Square of number is",square(10))
```

Output : Square of number is 100

Difference between Normal Functions and Anonymous Function:

Have a look over two examples:

Example

Normal function:

```
#Function Definiton

def square(x):
```

return x*x

#Calling square function

print("Square of number is",square(10))

Anonymous function:

#Function Definiton

square=lambda x1: x1*x1

#Calling square as a function

print("Square of number is",square(10))

Explanation:

- ➔ Anonymous is created without using def keyword.
- ➔ lambda keyword is used to create anonymous function.
- ➔ It returns the evaluated expression.

Use of Lambda Function in python

We use lambda functions when we require a nameless function for a short period of time. In Python, we generally use it as an argument to a higher-order function (a function that takes in other functions as arguments). Lambda functions are used along with built-in functions like `filter()`,`map()`.

`map() function`

The `map()` function in Python takes in a function and a list. The function is called with all the items in the list and a new list is returned which contains items returned by that function for each item.

Syntax: map(func, seq)

Example

my_list = [1, 5, 4, 6, 8, 11, 3, 12]

new_list =map(lambda x: x * 2 , my_list)

Output: [2, 10, 8, 12, 16, 22, 6, 24]

print(new_list)

map() can be applied to more than one list. The lists have to have the same length. map() will apply its lambda function to the elements of the argument lists, i.e. it first applies to the elements with the 0th index, then to the elements with the 1st index until the n-th index is reached.

Example

a=[1,2,3,4]

```
b=[5,6,7,8]

list1= list(map(lambda x,y:x+y,a,b))

print list1
```

filter() function

The function filter(function, list) offers an elegant way to filter out all the elements of a list, for which the function *function* returns True. The function filter(f,l) needs a function f as its first argument. f returns a Boolean value, i.e. either True or False. This function will be applied to every element of the list *l*. Only if f returns True will the element of the list be included in the result list.

Syntax: filter(func, seq)

Example

```
fib = [0,1,1,2,3,5,8,13,21,34,55]

result=(filter(lambda x:x%2==0,fib))

print result
```

Exercise

1. Write a program to print lenght of each word of the given string by using lambda function?

```
a="hello how are you"

mylist=a.split()

newlist=(map(lambda x:(x,len(x)),mylist))

print dict(newlist)
```

2. Write a program to find common elements of two lists by using lambda function?

```
list1=[1,2,3,4]

list2=[9,2,7,1]

newlist=filter(lambda x:x in list1,list2)

print (newlist)
```

3.Write a program to append the same string to a list of strings in Python?

```
title = 'Student'

mylist=[1,2,3,4]

name = map(lambda x:title + ' ' + str(x),mylist)

print (list(name))
```