Python Programming

Functions

Outline

- Introduction
- Syntax and Basics of a Function
- Use of a function
- Parameters and Arguments
- Local and Global Scope of a Variable
- Return statement
- Recursive Functions

Introduction

- Functions are common to all programming languages.
- A function is a block of related statements that performs a specific task when called.
- Built-in functions are usually a part of Python packages and libraries, whereas user-defined functions are written by the developers to meet certain requirements.
- In Python, all functions are treated as objects, so it is more flexible compared to other high-level languages.

Syntax and Basics of a Function

- **Syntax of Function**

```
def function_name(parameters):
        statement1
        statement2
        statement3
        .....
        statementN
function_name(arguments)
```

Function Header

Function Body

Function Call

Syntax and Basics of a Function

- 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.
- One or more valid python statements that make up the function body. Statements must have same indentation level (usually 4 spaces).

Syntax and Basics of a Function

```
def user( ):
    print("Python Programming")
    print("This is user defined function")
user( );
```

```
>>>
Python Programming
This is user defined function
>>>
```

Use of a function

- Suppose that you need to find the sum of integers from 1 to 10, 20 to 40 , and 50 to 100. If you create a program to add these three sets of numbers, your code might look like this:

```
sum = 0
for i in range(1, 11):
    sum = sum+i
print("Sum from 1 to 10 is", sum)
sum=0
for i in range(20,41):
    sum = sum+i
print("Sum from 20 to 40  is", sum)
sum = 0
for i in range(50, 101):
    sum = sum+i
print("Sum from 50 to 100 is", sum)
```

Use of a function

```
# Using Function
def sum(x,y):
    s=0
    for i in range(x,y+1):
        s=s+i
    print("Sum of values from",x,'to',y,'is',s)
sum(1,10)
sum(20,40)
sum(50,100)
# End of Program
```

```
>>>
Sum of values from 1 to 10 is 55
Sum of values from 20 to 40 is 630
Sum of values from 50 to 100 is 3825
```

Use of a function

- Functions are reusable code blocks, they only need to be written once, then they can be used multiple times. They can even be used in other applications, too.
- The code is usually well organized, easy to maintain and developer friendly.
- It can support the modular design approach.
- A well-defined and thoughtfully written user-defined function can ease the application development process.

Parameters and Arguments in a Function

- A parameter is a variable defined by a method that receives a value when the method is called.
- An argument is a value that is passed to a method when it is invoked.

```
# Function Program
def  add_numbers(x, y, z):
        a = x + y
        b = x + z
        c = y + z
        print(a, b, c)
add_numbers(1, 2, 3)
```

the number 1 is for the x parameter,
2 is for the y parameter,
3 is for the z parameter.
The number 1,2 and 3 are called arguments.

```
>>>
3  4  5
```

Parameters and Arguments in a Function

- **Function arguments in Python**
- Positional Arguments
- When you call a function, Python must match each argument in the function call with a parameter in the function definition.
- The simplest way to do this is based on the order of the arguments provided. Values matched up this way are called *positional arguments*.
- You can use as many positional arguments as you need in your functions.
- You can get unexpected results if you mix up the order of the arguments in a function call when using positional arguments

Parameters and Arguments in a Function

```
# Positional Arguments
def  display(name,course,per):
        print("Your name is:",name,"\nYou are enrolled for:",course,
          "\nYour percentage is:",per)
display('abc','BE',81.79)
# End of Program
```

```
>>>
Your name is: abc
You are enrolled for: BE
Your percentage is: 81.79
```

Parameters and Arguments in a Function

- **Function arguments in Python**
- Keyword Arguments
- Keyword arguments are relevant for Python function calls.
- The keywords are mentioned during the function call along with their corresponding values. These keywords are mapped with the function arguments so the function can easily identify the corresponding values even if the order is not maintained during the function call.
- All the keyword arguments passed must match one of the arguments accepted by the function. You may change the order of appearance of the keyword.
- Multiple values for same argument are not allowed in function call.

Parameters and Arguments in a Function

```
# Keyword Arguments
def  display(name,course,per):
      print("Your name is:",name,"\nYou are enrolled for:",course,
         "\nYour percentage is:",per)
display(course='BE',per=81.79,name='abc')
```

```
>>>
Your name is: abc
You are enrolled for: BE
Your percentage is: 81.79
```

Parameters and Arguments in a Function

- **Parameter with default values**
- In function's parameters list you can specify a default value(s) for one or more arguments.
- A default value can be written in the format "argument1 = value", therefore you will have the option to declare or not declare a value for those arguments.
- Any number of arguments in a function can have a default value.
- Once you have a default argument, all the arguments to its right must also have default values. i.e. non-default arguments cannot follow default arguments.

Parameters and Arguments in a Function

- **Parameter with default values**

```
# Default Arguments
def   display(name,course='ME',per='81.79'):
        print("Your name is:",name,"\nYou are enrolled for:",course,
          "\nYour percentage is:",per)
display(course='BE',name='abc')
```

```
>>>
Your name is: abc
You are enrolled for: BE
Your percentage is: 81.79
```

Local and Global Scope of a Variable

- There are two types of variables: **global variables** and **local variables**.
- A global variable can be reached anywhere in the code.
- A local variable can be reached only in the area in which they are defined, which is called *scope*.
- Scope of a variable is the portion of a program where the variable is recognized. Parameters and variables defined inside a function is not visible from outside. Hence, they have a local scope.
- Lifetime of a variable is the period throughout which the variable exits in the memory.
- The lifetime of variables inside a function is as long as the function executes. They are destroyed once you return from the function. Hence, a function does not remember the value of a variable from its previous calls.

Local and Global Scope of a Variable

- Accessing a local variable outside the scope will cause an error.
- If a variable with same name is defined inside the scope of function as well then it will print the value given inside the function only and not the global value.
- Any variable which is changed or created inside of a function is local, if it has not been declared as a global variable. To tell Python, that you want to use the global variable, you have to use the keyword "global".
- The keyword global is not needed for printing and accessing. It is only required in a function if you want to do assignments / change them.

Local and Global Scope of a Variable

```
#Local and Global Variable
def  display( ):
        x = 10     #Local Variable
        print("Value inside function:",x)
#Global Variable
x = 20
display( )
print("Value outside function:",x)
```

```
>>>
Value inside function: 10
Value outside function: 20
```

Local and Global Scope of a Variable

```
#Local and Global Variable
x = 20          #Global Variable
def   display( ):
       x = 10      #Local Variable
      print("Value inside function:",x)
display( )
print("Value outside function:",x)
```

```
>>>
Value inside function: 10
Value outside function: 20
```

Local and Global Scope of a Variable

```
#Local and Global Variable
# Global Statement
def  display( ):
      global x
      x = 10
      print("Value inside function:",x)
x=20
display( )
print("Value outside function:",x)
```

```
>>>
Value inside function: 10
Value outside function: 10
```

Return statement

- The return statement is used to return a value from the function.
- It is also used to exit a function and go back to the place from where it was called.
- **Syntax of return**
- return expression_list
- If there is expression in the statement which gets evaluated and the value is returned.
- If there is no expression in the statement or the return statement itself is not present inside a function, then the function will return the None object.

Return statement

```
def square(x):
    y = x ** 2
    return y
result = square(5)
print(result)
```

```
>>>
25
```

```
def square(x):
    y = x ** 2
    # return y
result = square(5)
print(result)
```

```
>>>
None
```

Return statement

- Write a function to compute the discriminator that returns the output depending on the discriminator.
  - If discriminator >0: Two real roots
  - If discriminator=0: One Real root
  - If discriminator<0: Two Complex roots

Return statement

```
import   math
a= eval(input("Enter value of a= "))
b= eval(input("Enter value of b= "))
c= eval(input("Enter value of c= "))
def   roots(a,b,c):
       disc = b**2 - 4*a*c
        if disc >= 0:
           print("Value of a:",a,"Value of b:",b,"Value of c:",c)
           print("Discriminant",disc)
           return ("r1=",(-b + math.sqrt(disc))/(2*a),"r2=",(-b - math.sqrt(disc))/(2*a))
        else:
            print("Value of a:",a,"Value of b:",b,"Value of c:",c)
            print("Discriminant",disc)
           return ('r1=',-b/(2*a),'+i',math.sqrt(disc*(-1))/(2*a),
                    'r2=',-b/(2*a),'-i',math.sqrt(disc*(-1))/(2*a))
print(roots(a,b,c))
```

```
>>>
Enter value of a= 1
Enter value of b= -2
Enter value of c= 1
Value of a: 1.0 Value of b: -2.0 Value of c: 1.0
Discriminant 0.0
('r1=', 1.0, 'r2=', 1.0)
```

```
>>>
Enter value of a= 1
Enter value of b= 7
Enter value of c= 12
Value of a: 1.0 Value of b: 7.0 Value of c: 12.0
Discriminant 1.0
('r1=', -3.0, 'r2=', -4.0)
```

```
>>>
Enter value of a= 1
Enter value of b= 2
Enter value of c= 3
Value of a: 1.0 Value of b: 2.0 Value of c: 3.0
Discriminant -8.0
('r1=', -1.0, '+i', 1.4142135623730951, 'r2=', -1.0, '-i', 1.4142135623730951)
```

Return statement

- It is possible to return multiple values in python.
- It is also possible for a function to perform certain operations, return multiple values and assign the returned multiple values to a multiple variable.

```
# Returning Multiple Values from a Function
def compute(num):
    print("Number=",num)
    return num*num,num*num*num
square,cube=compute(6)
print("Square=",square, ", ", "Cube=",cube)
```

```
>>>
Number= 6
Square= 36 , Cube= 216
```

Return statement

```
# Returning Multiple Values from a Function
num=eval(input("Enter the number:"))
def  compute(n):
        print("Number=",num)
        return num*num,num*num*num
square,cube=compute(num)
print("Square=",square,", ", "Cube=",cube)
```

```
Enter the number:6
Number= 6
Square= 36 , Cube= 216
>>> ================================
>>>
Enter the number:2.5
Number= 2.5
Square= 6.25 , Cube= 15.625
```

Recursive Functions

- Recursion is the process of defining something in terms of itself.
- A physical world example would be to place two parallel mirrors facing each other. Any object in between them would be reflected recursively.
- A function can call other functions. It is even possible for the function to call itself. These type of construct are termed as recursive functions.
- Every recursive function must have a base condition that stops the recursion or else the function calls itself infinitely.

Recursive Functions

- **Advantages of Recursion**
  - Recursive functions make the code look clean and elegant.
  - A complex task can be broken down into simpler sub-problems using recursion.
  - Sequence generation is easier with recursion than using some nested iteration.
- **Disadvantages of Recursion**
  - Sometimes the logic behind recursion is hard to follow through.
  - Recursive calls are expensive (inefficient) as they take up a lot of memory and time.
  - Recursive functions are hard to debug.

Recursive Functions

```
# Factorial of a number
num=int(input("Enter the number:"))
def  factorial(n):
      if n==0:
         return 1
      return n*factorial(n-1)
print(factorial(num))
```

```
>>>
Enter the number:10
3628800
>>> ====================
>>>
Enter the number:7
5040
```

Recursive Functions

```
#Fibonacci series Program using Recursion
# Recursive Function Beginning
def  fibo(num):
      if(num==0):
           return 0
      elif(num==1):
           return 1
      else:
            return(fibo(num-2)+fibo(num-1))
# End of the Function
# Fibonacci series will start at 0 and continue up to specified range
num=int(input("Enter the Range for fibonacci series:"))
# Find & Displaying Fibonacci series
for  i  in range(0,num):
      print(fibo(i))
```

Recursive Functions

```
>>>
Enter the Range for fibonacci series:13
0
1
1
2
3
5
8
13
21
34
55
89
144
```

Programs

- The built-in function eval takes a string and evaluates it using the Python interpreter.
- Write a function called eval_loop that iteratively prompts the user, takes the resulting input and evaluates it using eval, and prints the result.
- It should continue until the user enters 'done', and then return the value of the last expression it evaluated.

Programs

```
def eval_loop():
    while True:
        n = input('Input Expression::\n')
        if n == 'done':
            break
        else:
            result = eval(n)
            print(result)
    print(result)
eval_loop()
```

```
Input Expression::
12+45+88+23
168
Input Expression::
67*88*56
330176
Input Expression::
45.89/89
0.5156179775280899
Input Expression::
'1+2+3'
1+2+3
Input Expression::
'Hello,'+'How are you?'
Hello,How are you?
Input Expression::
'Are you understanding Python?'
Are you understanding Python?
Input Expression::
done
Are you understanding Python?
```

Programs

- Write a function called '*do_plus*' that accepts two parameters and adds them together with the " + " operation.

```
a=int(input("Enter first number:"))
b=int(input("Enter second number:"))
def  do_plus(x,y):
      return x+y
print("Sum of the given two numbers is:",do_plus(a,b))
```

```
>>>
Enter first number:23
Enter second number:56
Sum of the given two numbers is: 79
```

Programs

- If you are given three sticks, you may or may not be able to arrange them in a triangle. For any three lengths, there is a simple test to see if it is possible to form a triangle:
- If any of the three lengths is greater than the sum of the other two, then you cannot form a triangle. Otherwise, you can. (If the sum of two lengths equals the third, they form what is called a "degenerate" triangle.)
  - # Write a function name is_triangle that takes three integers as arguments ,and that prints either "Yes" or "No" depending on whether you can or cannot form a triangle from sticks with the given lengths.
  - # Write a function that prompts the user to input three stick lengths, converts them to integers and uses is_triangle to check whether sticks with the given lengths can form a triangle.

Programs

```
# Condition1
def is_triangle(x, y, z):
    if z > (x+y) or y > (x+z) or x > (y+z):
                print('No')
    else:
        print('Yes')
is_triangle(1, 2, 3)          # it's possible to arrange a triangle
is_triangle(1, 2, 9)          # it's not possible to arrange a triangle
print( )
# Condition2
def triangle( ):
    x = int(input('Please enter the length of the 1st stick:\n'))
    y = int(input('Please enter the length of the 2nd stick:\n'))
    z = int(input('Please enter the length of the 3rd stick:\n'))
    is_triangle(x, y, z)
triangle( )
```

Programs

```
>>>
Yes
No

Please enter the length of the 1st stick:
1
Please enter the length of the 2nd stick:
2
Please enter the length of the 3rd stick:
3
Yes
>>> ================================ RESTART
>>>
Yes
No

Please enter the length of the 1st stick:
1
Please enter the length of the 2nd stick:
2
Please enter the length of the 3rd stick:
9
No
```

Programs

- Python Program to Find LCM
- Python Program to Find Factorial of Number Using Recursion
- Python Program to Make a Simple Calculator
- The built-in function eval takes a string and evaluates it using the Python interpreter.
  - Write a function called eval_loop that iteratively prompts the user, takes the resulting input and evaluates it using eval, and prints the result.
  - It should continue until the user enters 'done', and then return the value of the last expression it evaluated.

THANKS.....