2. Explain break statement with the help of an example?

3. Explain continue statement with the help of an example?

4. Explain about pass statement in Python?

5. Write short notes on for-else and while-else?

6. Write a Python program to print prime numbers between 500 to 600.

Unit-4

Functions

Function is a group of related statements that perform a specific task. Functions help break our program into smaller and modular chunks. As our program grows larger and larger, functions make it more organized and manageable. It avoids repetition and makes code reusable.

Syntax of Function

```
def function_name(parameters):          #Keyword def marks the start of function header.
    statement(s)
```

Basically, we can divide functions into the following two types:

1. Built-in functions - Functions that are built into Python.
2. User-defined functions - Functions defined by the users themselves.

Built-in Functions

Functions that come built into the Python language itself are called built-in functions and are readily available to us.

Functions like *print(), input(), len()* etc. that we have been using, are some examples of the built-in function. There are 68 built-in functions defined in Python 3.7.4

Examples:

Name	Description	Example
max(x, y, z, ….)	It returns the largest of its arguments: where x, y and z are numeric variable/expression.	>>>max(80, 100, 1000) 1000 >>>max(-80, -20, -10)

		-1

min(x, y, z, ...)	It returns the smallest of its arguments; where x, y, and z are numeric variable/expression.	>>> min(80, 100, 1000)
		80
		>>> min(-80, -20, -10)
		-80

cmp(x, y)	It returns the sign of the difference of two numbers: -1 if x < y, 0 if x == y, or 1 if x > y, where x and y are numeric variable/expression.	>>>cmp(80, 100) -1 >>>cmp(180, 100) 1

len (s)	Return the length (the number of items) of an object. The argument may be a sequence (string, tuple or list) or a mapping (dictionary).	>>> a= [1,2,3]
		>>>len (a)
		3
		>>> b= "Hello"
		>>> len (b)
		5

range (start, stop[, step])	This is a versatile function to create lists containing arithmetic progressions.	>>> range(10)
		[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
		>>> range(1, 11)
		[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
		>>> range(0, 30, 5)
		[0, 5, 10, 15, 20, 25]
		>>> range(0, 10, 3)
		[0, 3, 6, 9]

User-defined functions

Functions that we define ourselves to do certain specific task are referred as user-defined functions.

Functions that readily come with Python are called built-in functions. If we use functions written by others in the form of library, it can be termed as library functions.

All the other functions that we write on our own fall under user-defined functions. So, our user-defined function could be a library function to someone else.

Advantages of user-defined functions

1. User-defined functions help to decompose a large program into small segments which makes program easy to understand, maintain and debug.

2. If repeated code occurs in a program. Function can be used to include those codes and execute when needed by calling that function.
3. Programmers working on large project can divide the workload by making different functions.

Example 22:

Write a python script to print a message using user-defined function.

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

Docstring

The first string after the function header is called the docstring and is short for documentation string. It is used to explain in brief, what a function does.

Although optional, documentation is a good programming practice.

Example 23:

```
def greet(name):
    """This function greets to
    the person passed in as
    parameter"""
    print("Hello, " + name + ". Good morning!")
print(greet.__doc__)
```

Output:

```
This function greets to
    the person passed into the
    name parameter
```

Function Call (Calling a Function)

Once we have defined a function, we can call it from another function, program or even the Python prompt. To call a function we simply type the function name with appropriate parameters.

Example 24:

```
def greet(name):
    """This function greets to
    the person passed in as
    parameter"""
    print("Hello, " + name + ". Good morning!")

greet('Paul') #call a function
```

Example 25:

```
# 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( )
```

Example 26:

```
#Providing Function Definition
def sum(x,y):
    """Going to add x and y"""
    s=x+y
```

```
        print("Sum of two numbers is")
        print(s)
#Calling the sum Function
sum(10,20)
sum(20,30)
```

The return statement

The return statement is used returns a value from the function and goes back to the place from where it was called with the expression. A return statement may contain a constant, variable, expression. If return is used without anything, it will return **None**.

Example 27:

```
# Function definition is here
def changeme( x ):
  return x  #return statement with a argument

# Now you can call changeme function
x = 5
print changeme( x )
```

Example 28:

```
# Function definition is here
def changeme( x ):
  return  #return statement without argument

# Now you can call changeme function
x = 5
print changeme( x )
```

Void Functions:

Function performs a task, and does not contain a return statement such functions are called void functions.

Void functions do not return anything. 'None' is a special type in Python which is returned after a void function ends.

Example 29:

```
def check (num):
    if (num%2==0):
  print “True”
   else:
       print “False”
 result = check (29)
print (result)
```

Output:

False

None

Exercise:

1. Write a Python function to find the Min of three numbers.

2. Write a Python function to find the Max of three numbers.

3. Write a Python function to calculate the factorial of a number (a non-negative integer). The function accepts the number as an argument.

4. Write a Python function that accepts a string and calculate the number of upper case letters and lower case letters.

5. Write a Python function that takes a number as a parameter and check the number is prime or not.

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

Flow of Execution

Execution always begins at the first statement of the program. Statements are executed one at a time, in order from top to bottom. Function definition does not alter the flow of execution of program, as the statement inside the function is not executed until the function is called.

On a function call, instead of going to the next statement of program, the control jumps to the body of the function; executes all statements of the function in the order from top to bottom and then comes back to the point where it left off. This remains simple, till a function does not call another function. Simillarly, in

the middle of a function, program might have to execute statements of the other function and so on.

Parameters and Arguments:

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

The First type of data is the data passed in the function call. This data is called arguments. Arguments can be variables and expressions.
Argument is the actual value of this variable that gets passed to function.

The second type of data is the data received in the function definition(function header). This data is called parameters. Parameters must be variable to holding coming values.
Parameter is variable in the declaration of function.

More on Defining Functions-Passing parameters/Arguments:

Python Supports following types arguments.
1. Required Arguments
2. Keyword Arguments
3. Default Arguments
4. Variable-length or Arbitrary Arguments

1. **Required Arguments:** In the required arguments, the arguments are passed to a function in correct positional order. The number of arguments in the function call should exactly match with the number of parameters specified in the function definition.

Example 30:

def display(): print (“hello”) display(“hi”) Output: TypeError	def display(x): print (x) display() Output: TypeError	def display(x): print (x) display(“hello”) Output: hello

2. **Keyword Arguments:** Python allows functions to be called using keyword arguments. When we call functions in this way, the order (position) of the arguments can be changed.

Example 31:

```
def display(name,x,y):
        print ("The name of person is", name)
        print ("The x value is", x)
        print ("The Y value is", y)
display(y=20,name="RAMESH",x=40)
```

3. **Default Arguments:** A default argument is an argument that assumes a default value which defined in the called function if a value is not provided in the function call for that argument.

Example 32:

```
def display(name,course="BTECH"):
        print ("The name of person is", name)
        print ("Course: ", course)
display("RAJESH","MBA")
display(course="MSC",name="RAJU")
display(name="RAVI")
```

4. **Variable-length or Arbitrary 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. Here is an example.

Example 33:

```
def display(*names):
    for i in names:
        print("Hello",i)
display("Monica","Luke","Steve","John")
```

Scope of Variables:

Scope of variable refers to the part of the program in which a variable is accessible. Some of the variables may not exist for the entire duration of the program. In which part of the program can access a variable and which part of the program a variable exists depends on how the variable has been declared.

There are two types of scope of variables

1) Global variables (Global Scope)

2) Local variables (Local Scope)

1) **Global Variables :** The variables which are defined (declared) in the main body of the program called *global variables*. Those variables can be accessed anywhere in the program.

Example 34:

```
x=50      #global variable
def test ( ):
        print  ("Inside test x is" , x)
print  ("Value of x is" , x)
```

on execution the above code will produce

Inside test x is 50

Value of x is 50

Example 35:

```
x=50      # global variable
def test ( ):
                x = 10
                print  ("Inside test x is", x)
print  ("Value of x is", x)
```

will produce

Inside test x is 10

Value of x is 60

2) **Local variables:** The variables which are defined within a function is called *local variables.* Local variables can be accessed only within the function. It exists as long as the function is executing. Function parameters behave like local variables in the function.

Example 36:

```
X=50  #global variable
def test ( ):
```

```
        y = 20              #local variable
        print ("Value of x is", X,   "y is" , y)
test()
print ("Value of x is ", X, "y is" , y)
```

On executing the code we will get

Value of x is 50; y is 20

The next print statement will produce an error, because the variable y is not accessible outside the function body.

Example 37:

```
x=50          #global variable
def test ( ):
        x=5             #local variable
        y=2             #local variable
        print ("Value of x & y inside the function are" , x , y)
test()
print ("Value of x outside the function is" , x)
```

This code will produce following output:

Value of x & y inside the function are 5 2

Value of x outside the function is 50

Difference between Global and Local variables

Global Variables	Local Variables
1. They are defined in the main body of the program.	1. They are defined within a function and is local to that function.
2. They can be accessed throughout the program.	2. They can be accessed within a function only. We cannot access the variables outside of the function.

3. Global variables are accessible to all functions in the program.	3. They are not related in any way to other variables with the same name used outside the function.

Using the Global Keyword:

Want to modify local variable as global variable inside the function then add 'global' keyword before the variable name. This declares the local or the inner variable of the function to have module scope.

Example 38:

```
var= "Good"
def show():
        global var1
        var1= "Morning"
        print ("In function var is –", var)
show()
print ("Outside function, var1 is – ", var1)
print ("var is ", var)
```

Output:

In function var is – Good

Outside function, var1 is – Morning

var is – Good

Modules

Module is a python program file which contains a python code including python functions, class, statements and variables. Modules in Python provide us the flexibility to organize the code in a logical way. Modules provide reusability of code. Similar type of attributes can be placed in one module.

A file containing Python code, for e.g., example.py, is called a module and its module name would be example.