

## Unit-II

**Objective:** To use functions and modules to develop python programs.

### **Syllabus:**

**Functions:** Function definition, call, return statement, local and global variables, Types of arguments, Types of Functions: Anonymous, Fruitful, Recursive function and Passing functions as arguments.

**Modules:** The from...import statement, making your own modules, dir() function, modules and namespaces, types of namespaces: global, local and built-in, packages and Modules, introduction to PIP, installing packages via PIP.

### **Learning Outcomes:**

At the end of the unit student will be able to

- understand need for functions, variable scope, and lifetime.
- learn different types of arguments.
- learn different types of functions.
- identify use of modules.
- creating modules and package.
- Install packages via pip command.

### Learning Material

#### Functions

- A function is a block of organized and reusable program code that performs a single, specific, and well-defined task.
- Python enables its programmers to break up a program into functions, each of which can be written more or less independently of the others. Therefore, the code of one function is completely insulated from the codes of the other functions.

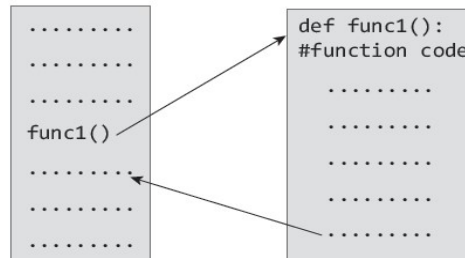


Figure 1: Calling a function

- In figure 1 which explains how a function `func1()` is called to perform a well-defined task. As soon as `func1()` is called, the program control is passed to the first statement in the function. All the statements in the function are executed and then the program control is passed to the statement following the one that called the function.

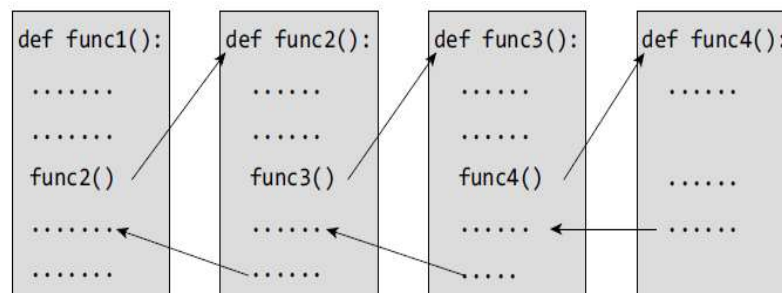


Figure 2: Function calling another function

- In figure 2 `func1()` calls function named `func2()`. Therefore, `func1()` is known as the *calling function* and `func2()` is known as the *called function*. The moment the compiler encounters a function call, instead of executing the next statement in the calling function, the control jumps to the statements that are a part of the called function. After called function is executed, the control is returned back to the calling program.
- It is not necessary that the `func1()` can call only one function, it can call as many functions as it wants and as many times as it wants. For example, a function call placed within for loop or while loop may call the same function multiple times until the condition holds true.

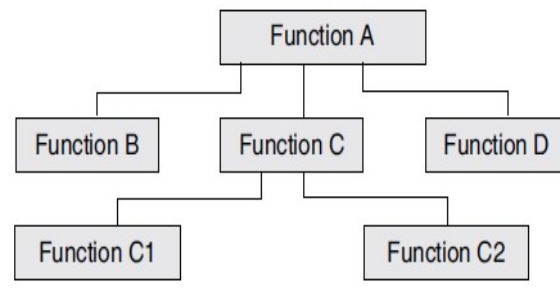
**Need for Functions:**

Figure 3: Top-down approach of solving a problem

- Each function to be written and tested separately.
- Understanding, coding and testing multiple separate functions are far easier than doing the same for one huge function.
- When a big program is broken into comparatively smaller functions, then different programmers working on that project can divide the workload by writing different functions.
- All the libraries in Python contain pre-defined and pre-tested functions which the programmers are free to use directly in their programs, without worrying about their code details. This speeds up program development.
- Like Python libraries, programmers can also make their own functions and use them from different points in the main program or any other program that needs its functionalities. So *code reuse* is one of the most prominent reasons to use functions.

**Function Declaration and Definition:**

- A function, *f* that uses another function *g*, is known as the *calling function* and *g* is known as the *called function*.
- The inputs that the function takes are known as *arguments/parameters*.
- When a called function returns some result back to the calling function, it is said to return that result.
- The calling function may or may not pass parameters to the called function. If the called function accepts arguments, the calling function will pass parameters, else not.
- *Function declaration* is a declaration statement that identifies a function with its name, a list of arguments that it accepts and the type of data it returns.
- *Function definition* consists of a function header that identifies the function, followed by the body of the function containing the executable code for that function.

## Function Definition

There are two basic types of functions

1. built-in functions eg: `dir()`, `len()`, `abs()` etc.,
2. user defined functions.
  - Function blocks starts with the keyword `def`.
  - The keyword is followed by the function name and parentheses `(( ))`.
  - After the parentheses a colon `(:)` is placed.
  - Parameters or arguments that the function accept are placed within parentheses.
  - The first statement of a function can be an optional statement - the *docstring* describe what the function does.
  - The code block within the function is properly indented to form the block code.
  - A function may have a `return[expression]` statement. That is, the `return` statement is optional.
  - You can assign the function name to a variable. Doing this will allow you to call same function using the name of that variable.

```
def diff(x,y):          # function to subtract two numbers
    return x-y
a = 20
b = 10
operation = diff        # function name assigned to a variable
print(operation(a,b))  # function called using variable name
```

### OUTPUT

10

Figure 4: Program that subtracts two numbers using a function.

```
def function_name(variable1, variable2,..)
    documentation string
    statement block
    return [expression]
```

Function Header

Function  
Body

Figure 5: The syntax of a function definition.

## Function Call

- Defining a function means specifying its name, parameters that are expected, and the set of instructions.
- The function call statement invokes the function. When a function is invoked the program control jumps to the called function to execute the statements that are a

part of that function. Once the called function is executed, the program control passes back to the calling function.

### Function Parameters

- A function can take parameters which are nothing but some values that are passed to it so that the function can manipulate them to produce the desired result. These parameters are normal variables with a small difference that the values of these variables are defined (initialized) when we call the function and are then passed to the function.
- Function name and the number and type of arguments in the function call must be same as that given in the function definition.
- If the data type of the argument passed does not matches with that expected in function then an error is generated.

```
def func():  
    for i in range(4):  
        print("Hello World")  
func()      #function call
```

**OUTPUT**

Hello World  
Hello World  
Hello World  
Hello World

Figure 6: a function that displays string repeatedly.

```
def func(i):          # function definition header accepts a variable with name i  
    print("Hello World", i)  
    j = 10  
    func(j)           # Function is called using variable j
```

**OUTPUT**

Hello World 10

Figure 7: Program to demonstrate mismatch of name of function parameters and arguments.

Note: Names of variables in function call and header of function definition may vary.

```
def func(i):  
    print("Hello World", i)  
func(5+2*3)
```

**OUTPUT**

Hello World 11

Figure 8: Arguments may be passed in the form of expressions to the called function.

```
def total(a,b):    # function accepting parameters
    result = a+b
    print("Sum of ", a, " and ", b, " = ", result)

a = int(input("Enter the first number : "))
b = int(input("Enter the second number : "))
total(a,b) #function call with two arguments
```

#### OUTPUT

```
Enter the first number : 10
Enter the second number : 20
Sum of 10 and 20 = 30
```

Figure 9: Program to add two integers using functions

### **Variable scope and lifetime:**

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

1. Scope of the variable: Part of the program in which a variable is accessible is called its *scope*.
2. Lifetime of the variable: Duration for which the variable exists it's called its *lifetime*.

### **Local and Global variables:**

A variable which is defined within a function is *local* to that function. A local variable can be accessed from the point of its definition until the end of the function in which it is defined. It exists as long as the function is executing. Function parameters behave like local variables in the function. Moreover, whenever we use the assignment operator (=) inside a function, a new local variable is created.

Global variables are those variables which are defined in the main body of the program file. They are visible throughout the program file. As a good programming habit, you must try to avoid the use of global variables because they may get altered by mistake and then result in erroneous output.

```

num1 = 10    # global variable
print("Global variable num1 = ", num1)
def func(num2):
    # num2 is function parameter
    print("In Function - Local Variable num2 = ",num2)
    num3 = 30    #num3 is a local variable
    print("In Function - Local Variable num3 = ",num3)
func(20)        #20 is passed as an argument to the function
print("num1 again = ", num1)    #global variable is being accessed
#Error- local variable can't be used outside the function in which it is defined
print("num3 outside function = ", num3)

```

#### OUTPUT

```

Global variable num1 = 10
In Function - Local Variable num2 = 20
In Function - Local Variable num3 = 30
num1 again = 10
num3 outside function =
Traceback (most recent call last):
  File "C:\Python34\Try.py", line 12, in <module>
    print("num3 outside function = ", num3)
NameError: name 'num3' is not defined

```

**Programming Tip:** Variables can only be used after the point of their declaration

Figure 10: lists the differences between global and local variables.

### Comparison between global and local variables

Global variables	Local variables
They are defined in the main body of the program file.	They are defined within a function and is <i>local</i> to that function.
They can be accessed throughout the program life.	They can be accessed from the point of its definition until the end of the block in which it is defined.
Global variables are accessible to all functions in the program.	They are not related in any way to other variables with the same names used outside the function.

### Using the Global Statement

To define a variable defined inside a function as global, you must use the global statement. This declares the local or the inner variable of the function to have module scope.

Key points to remember:

You can have a variable with the same name as that of a global variable in the program. In such a case a new local variable of that name is created which is different from the global variable.

```

var = "Good"
def show():
    global var1
    var1 = "Morning"
    print("In Function var is - ", var)
show()
print("Outside function, var1 is - ", var1)      #accessible as it is global
variable
print("var is - ", var)

```

**OUTPUT**

```

In Function var is - Good
Outside function, var1 is - Morning
var is - Good

```

**Programming Tip:** All variables have the scope of the block.

Figure 11: Program to demonstrate the use of global statement.

### Resolution of names

*Scope* defines the visibility of a name within a block. If a local variable is defined in a block, its scope is that particular block. If it is defined in a function, then its scope is all blocks within that function.

When a variable name is used in a code block, it is resolved using the nearest enclosing scope. If no variable of that name is found, then a `NameError` is raised. In the code given below, `str` is a global string because it has been defined before calling the function.

```

def func():
    print(str)
str = "Hello World !!!"
func()

```

**OUTPUT**

```

Hello World !!!

```

Figure 12: Program that demonstrates using a variable defined in global namespace.

### The Return Statement

The syntax of return statement is,

```
return [expression]
```

The expression is written in brackets because it is optional. If the expression is present, it is evaluated and the resultant value is returned to the calling function. However, if no expression is specified then the function will return `none`.

The return statement is used for two things.

- Return a value to the caller
- To end and exit a function and go back to its caller



```
def cube(x):
    return (x*x*x)
num = 10
result = cube(num)
print('Cube of ', num, ' = ', result)
```

### OUTPUT

```
Cube of 10 = 1000
```

Figure 13: Program to write another function which returns an integer to the caller.

### More on defining functions:

In this section we will discuss some more ways of defining a function.

1. Required arguments
2. Keyword arguments
3. Default arguments
4. Variable-length arguments

### Required Arguments

In the *required arguments*, the arguments are passed to a function in correct positional order. Also, the number of arguments in the function call should exactly match with the number of arguments specified in the function definition

Example:

<pre>def display():     print "Hello" display("Hi")</pre>	<pre>def display(str):     print str display()</pre>	<pre>def display(str):     print str str ="Hello" display(str)</pre>
<b>OUTPUT</b>	<b>OUTPUT</b>	<b>OUTPUT</b>
TypeError: display() takes no arguments (1 given)	TypeError: display() takes exactly 1 argument (0 given)	Hello

### Keyword Arguments

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

Keyword arguments are beneficial in two cases.

- First, if you skip arguments.
- Second, if in the function call you change the order of parameters.

Example:

```
def display(str, int_x, float_y):
    print("The string is : ",str)
    print("The integer value is : ", int_x)
    print("The floating point value is : ", float_y)
display(float_y = 56789.045, str = "Hello", int_x = 1234)
```

**OUTPUT**

```
The string is: Hello
The integer value is: 1234
The floating point value is: 56789.045
```

### Default Arguments

Python allows users to specify function arguments that can have default values. This means that a function can be called with fewer arguments than it is defined to have. That is, if the function accepts three parameters, but function call provides only two arguments, then the third parameter will be assigned the default (already specified) value. The default value to an argument is provided by using the assignment operator (=). Users can specify a default value for one or more arguments.

Example:

```
def display(name, course = "BTech"):
    print("Name : " + name)
    print("Course : ", course)
display(course = "BCA", name = "Arav") # Keyword Arguments
display(name = "Reyansh")             # Default Argument for course
```

**OUTPUT**

```
Name : Arav
Course : BCA
Name : Reyansh
Course : BTech
```

### Variable-length Arguments

In some situations, it is not known in advance how many arguments will be passed to a 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 use an asterisk (\*) before the parameter name. The syntax for a function using variable arguments can be given as,

```
def functionname([arg1, arg2,... ] *var_args_tuple ):
    function statements
    return [expression]
```

Example:

```
def func(name, *fav_subjects):
    print("\n", name, " likes to read ")
    for subject in fav_subjects:
        print(subject)
func("Goransh", "Mathematics", "Android Programming")
func("Richa", "C", "Data Structures", "Design and Analysis of Algorithms")
func("Krish")
```

#### OUTPUT

```
Goransh likes to read Mathematics Android Programming
Richa likes to read C Data Structures Design and Analysis of Algorithms
Krish likes to read
```

### **Lambda Functions or Anonymous Functions**

*Lambda or anonymous* functions are so called because they are not declared as other functions using the def keyword. Rather, they are created using the lambda keyword. Lambda functions are throw-away functions, i.e. they are just needed where they have been created and can be used anywhere a function is required. The lambda feature was added to Python due to the demand from LISP programmers.

Lambda functions contain only a single line. Its syntax can be given as,

```
lambda arguments: expression
```

#### **Example**

```
sum = lambda x, y: x + y
print("Sum = ", sum(3, 5))
```

#### OUTPUT

```
Sum = 8
```

### **Documentation Strings**

Docstrings (documentation strings) serve the same purpose as that of comments, as they are designed to explain code. However, they are more specific and have a proper syntax.

```
def functionname(parameters):
    "function_docstring"
    function statements
    return [expression]
```

#### **Example:**

```
def func():
    """The program just prints a message.
    It will display Hello World !!! """
    print("Hello World !!!")
print(func.__doc__)
```

#### OUTPUT

```
The program just prints a message.
It will display Hello World !!!
```

### Recursive Functions

A recursive function is defined as a function that calls itself to solve a smaller version of its task until a final call is made which does not require a call to itself. Every recursive solution has two major cases, which are as follows:

- *base case*, in which the problem is simple enough to be solved directly without making any further calls to the same function.
- *recursive case*, in which first the problem at hand is divided into simpler sub parts.

Recursion utilized divide and conquer technique of problem solving.

Example:

```
def fact(n):
    if(n==1 or n==0):
        return 1
    else:
        return n*fact(n-1)
n = int(input("Enter the value of n : "))
print("The factorial of",n,"is",fact(n))
```

#### OUTPUT

```
Enter the value of n : 6
The factorial of 6 is 720
```

### Recursion vs Iteration:

Recursion is more of a top-down approach to problem solving in while the original problem is divided into smaller sub-problems.

Iteration follows a bottom-up approach that begins with what is known and then constructing the solution step-by-step.

*Pros*The benefits of using a recursive program are:

- Recursive solutions often tend to be shorter and simpler than non-recursive ones.
- Code is clearer and easier to use.
- Recursion uses the original formula to solve a problem.
- It follows a divide and conquer technique to solve problems.
- In some instances, recursion may be more efficient.

*Cons* The limitations of using a recursive program are:

- For some programmers and readers, recursion is difficult concept.
- Recursion is implemented using system stack. If the stack space on the system is limited, recursion to a deeper level will be difficult to implement.
- Aborting a recursive process in midstream is slow and sometimes nasty.
- Using a recursive function takes more memory and time to execute as compared to its non-recursive counterpart.
- It is difficult to find bugs, particularly when using global variables.

Conclusion: The advantages of recursion pays off for the extra overhead involved in terms of time and space required.

### Modules

- We have seen that functions help us to reuse a particular piece of code. Module goes a step ahead. It allows you to reuse one or more functions in your programs, even in the programs in which those functions have not been defined.
- Putting simply, module is a file with a.py extension that has definitions of all functions and variables that you would like to use even in other programs. The program in which you want to use functions or variables defined in the module will simply import that particular module (or .py file).
- Modules are pre-written pieces of code that are used to perform common tasks like generating random numbers, performing mathematical operations, etc.
- The basic way to use a module is to add `import module_name` as the first line of your program and then writing `module_name.var` to access functions and values with the name var in the module.

### The from...import Statement

A module may contain definition for many variables and functions. When you import a module, you can use any variable or function defined in that module. But if you want to use only selected variables or functions, then you can use the `from...import` statement. For example, in the aforementioned program you are using only the path variable in the sys module, so you could have better written `from sys import path`.

Example:

```
from math import pi
print("PI = ", + pi)
```

#### **OUTPUT**

```
3.141592653589793
```

To import more than one item from a module, use a comma separated list. For example, to import the value of pi and sqrt() from the math module you can write,

```
from math import pi, sqrt
```

### Making your own Modules

Every Python program is a module, that is, every file that you save as .py extension is a module.

- Modules should be placed in the same directory as that of the program in which it is imported. It can also be stored in one of the directories listed in sys.path.

First write these lines in a file and save the file as MyModule.py

```
def display():          #function definition
    print("Hello")
    print("Name of called module is : ", __name__)

str = "Welcome to the world of Python !!!"    #variable definition
```

Then, open another file (main.py) and write the lines of code given below.

```
import MyModule
print("MyModule str = ", MyModule.str)      #using variable defined in MyModule
MyModule.display()                          #using function defined in MyModule
print("Name of calling module is : ", __name__)
```

When you run this code, you will get the following output.

```
MyModule str = Welcome to the world of Python !!!
Hello
Name of called module is : MyModule
Name of calling module is : __main__
```

### The dir() function

dir() is a built-in function that lists the identifiers defined in a module. These identifiers may include functions, classes and variables. If no name is specified, the dir() will return the list of names defined in the current module.

Example: demonstrate the use of dir() function.

```
def print_var(x):
    print(x)
x = 10
print_var(x)
print(dir())
```

#### **OUTPUT**

```
10
['__builtins__', '__doc__', '__file__', '__name__', '__package__', 'print_var', 'x']
```

### The Python Module:

- We have seen that a *Python module* is a file that contains some definitions and statements. When a Python file is executed directly, it is considered the main module of a program.
- Main modules are given the special name `__main__` and provide the basis for a complete Python program.
- The main module may import any number of other modules which may in turn import other modules. But the main module of a Python program cannot be imported into other modules.

### **Modules and Namespaces**

A namespace is a container that provides a named context for identifiers. Two identifiers with the same name in the same scope will lead to a name clash. In simple terms, Python does not allow programmers to have two different identifiers with the same name. However, in some situations we need to have same name identifiers. To cater to such situations, namespaces is the keyword. Namespaces enable programs to avoid potential name clashes by associating each identifier with the namespace from which it originates.

Example:

```
# module1
def repeat_x(x):
    return x*2

# module2
def repeat_x(x):
    return x**2

import module1
import module2
result = repeat_x(10)    # ambiguous reference for identifier repeat_x
```

### **Local, Global, and Built-in Namespaces**

During a program's execution, there are three main namespaces that are referenced- the built-in namespace, the global namespace, and the local namespace. The built-in namespace, as the name suggests contains names of all the built-in functions, constants, etc that are already defined in Python. The global namespace contains identifiers of the currently executing module and the local namespace has identifiers defined in the currently executing function (if any).

When the Python interpreter sees an identifier, it first searches the local namespace, then the global namespace, and finally the built-in namespace. Therefore, if two identifiers with same name are defined in more than one of these namespaces, it becomes masked.

Example: Program to demonstrate name clashes in different namespaces.



```
def max(numbers):      # global namespace
    print("USER DEFINED FUNCTION MAX....")
    large = -1         # local namespace
    for i in numbers:
        if i>large:
            large = i
    return large

numbers = [9,-1,4,2,7]
print(max(numbers))
print("Sum of these numbers = ", sum(numbers)) #built in namespace
```

**OUTPUT**

```
USER DEFINED FUNCTION MAX....
9
Sum of these numbers = 21
```

**Module Private Variables**

- In Python, all identifiers defined in a module are public by default. This means that all identifiers are accessible by any other module that imports it. But, if you want some variables or functions in a module to be privately used within the module, but not to be accessed from outside it, then you need to declare those identifiers as private.
- In Python identifiers whose name starts with two underscores (\_\_) are known as private identifiers. These identifiers can be used only within the module. In no way, they can be accessed from outside the module.
- Therefore, when the module is imported using the import \* form modulename, all the identifiers of a module's namespace is imported except the private ones (ones beginning with double underscores). Thus, private identifiers become inaccessible from within the importing module.

**Advantages of Modules:**

- Python modules provide all the benefits of modular software design. These modules provide services and functionality that can be reused in other programs.
- Even the standard library of Python contains a set of modules. It allows you to logically organize the code so that it becomes easier to understand and use.

**Programs:**

1. Write a function cumulative product to compute cumulative product of a list of numbers.

**Program:**

```
def cumulative_product():
    list=[1,2,3,4]
    prod=1
```



```

for i in list:
    prod=prod*i
print prod
cumulative_product()

```

**Output:**

```

C:\Windows\system32\cmd.exe
C:\Python27>cd unit3
C:\Python27\unit3>dir
Volume in drive C has no label.
Volume Serial Number is 961F-44DE

Directory of C:\Python27\unit3
07-12-2017  09:56    <DIR>          .
07-12-2017  09:56    <DIR>          ..
06-12-2017  12:00             130 lengthofstring1_5.py
06-12-2017  11:58             37 lengthofstring_5.py
07-12-2017  12:13             148 productoflist_1.py
07-12-2017  11:27             111 productoflist_iv1.py
06-12-2017  13:57             298 removevowels1_4.py
06-12-2017  14:02             400 removevowels_4.py
06-12-2017  11:53             231 reversestring1_6.py
06-12-2017  11:38             116 reversestring_6.py
               8 File(s)          1,471 bytes
               2 Dir(s)  450,195,415,040 bytes free

C:\Python27\unit3>productoflist_1.py
24
C:\Python27\unit3>_

```

2. Write function to compute gcd, lcm of two numbers. Each function shouldn't exceed one line.

**Program:**

```

from fractions import gcd
print gcd(5,25)
def lcm():
    a=60
    b=40
    print (a * b) // gcd(a, b)

```

lcm()

**output:**

```

C:\Windows\system32\cmd.exe
120
C:\Python27\unit4>dir
Volume in drive C has no label.
Volume Serial Number is 961F-44DE

Directory of C:\Python27\unit4
07-12-2017  14:08    <DIR>          .
07-12-2017  14:08    <DIR>          ..
07-12-2017  12:23             60 first_char_5.py
07-12-2017  14:06             116 gcd2.py
07-12-2017  14:21             122 gcd2v1.py
06-12-2017  10:59             41 sortatuple_2.py
06-12-2017  11:25             227 sumavg1_4.py
06-12-2017  11:21             233 sumavg_4.py
06-12-2017  11:05             187 swap2values_1.py
               7 File(s)          986 bytes
               2 Dir(s)  450,178,555,904 bytes free

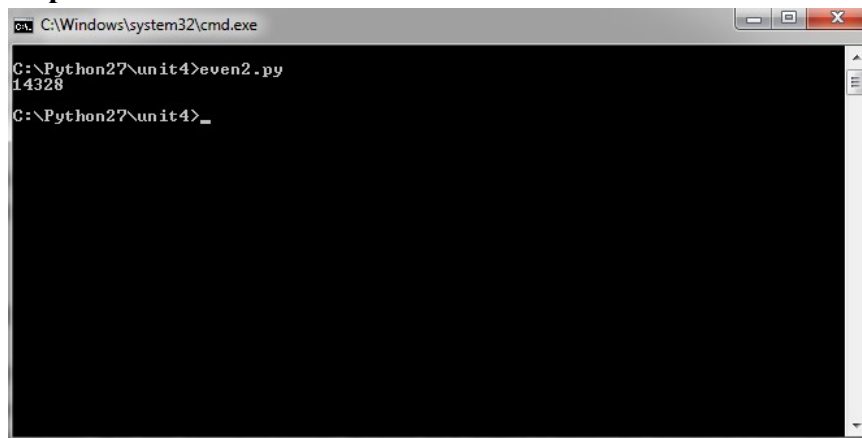
C:\Python27\unit4>gcd2v1.py
5
120
C:\Python27\unit4>

```

3. Find the sum of the even-valued terms in the Fibonacci sequence whose values do not exceed ten thousand.

**program:**

```
i=0
j=1
sum=0
while(i<10000):
    i=i+j
    j=i-j
    if(i%2==0):
        sum+=i
print sum
```

**output:**A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Windows\system32\cmd.exe'. The command prompt shows the execution of a Python script: 'C:\Python27\unit4>even2.py'. The output of the script is '14328'. The prompt then shows 'C:\Python27\unit4>\_'.**Packages and Modules**

- A package is a hierarchical file directory structure that has modules and other packages within it. Like modules, you can very easily create packages in Python.
- Every package in Python is a directory which must have a special file called **`__init__.py`**. This file may not even have a single line of code. It is simply added to indicate that this directory is not an ordinary directory and contains a Python package. In your programs, you can import a package in the same way as you import any module.
- For example, to create a package called MyPackage, create a directory called MyPackage having the module MyModule and the **`__init__.py`** file. Now, to use MyModule in a program, you must first import it. This can be done in two ways.

**import MyPackage.MyModule**  
**or**  
**from MyPackage import MyModule**

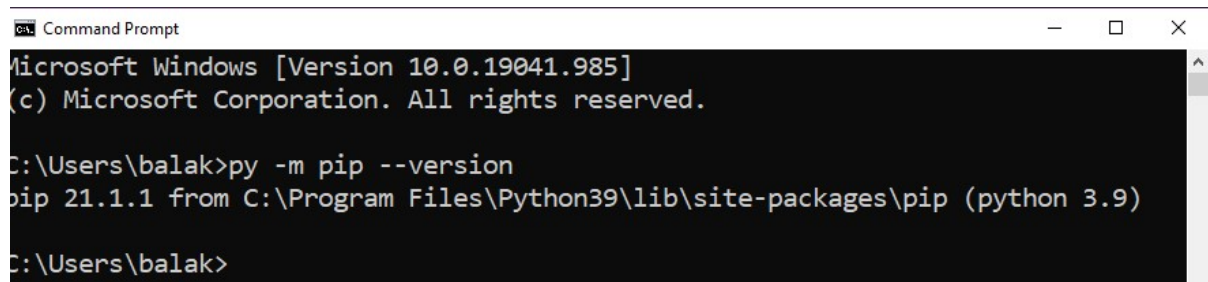
## Introduction to PIP

**PIP** is a package manager for Python packages, or modules if you like.

Note: If you have Python version 3.4 or later, PIP is included by default. Inorder to know the version of pip in local system the command is

***py -m pip - -version***

### Example:



```

Microsoft Windows [Version 10.0.19041.985]
(c) Microsoft Corporation. All rights reserved.

C:\Users\balak>py -m pip --version
pip 21.1.1 from C:\Program Files\Python39\lib\site-packages\pip (python 3.9)

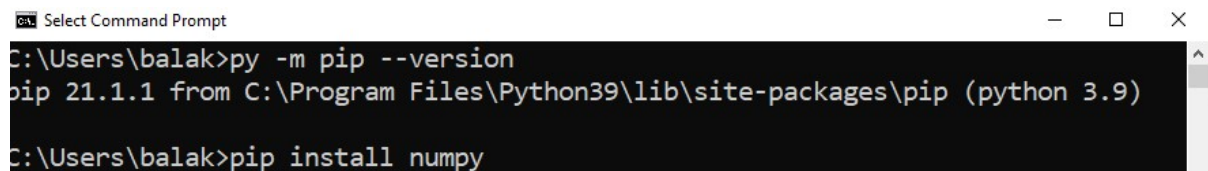
C:\Users\balak>
  
```

## Installing packages via pip

Steps to Install a Package in Python using PIP

1. First, type **Command Prompt** in the Windows search box
2. Now, type the pip install command to install your Python package. The pip install command has the following structure:

**pip install package\_name**

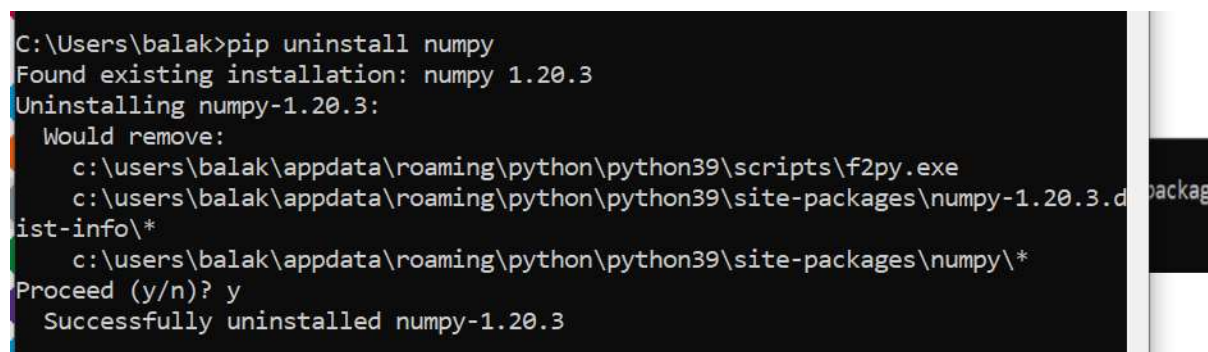


```

C:\Users\balak>py -m pip --version
pip 21.1.1 from C:\Program Files\Python39\lib\site-packages\pip (python 3.9)

C:\Users\balak>pip install numpy
  
```

To *uninstall particular package* in python simply type the command



```

C:\Users\balak>pip uninstall numpy
Found existing installation: numpy 1.20.3
Uninstalling numpy-1.20.3:
  Would remove:
    c:\users\balak\appdata\roaming\python\python39\scripts\fp2py.exe
    c:\users\balak\appdata\roaming\python\python39\site-packages\numpy-1.20.3.dist-info\*
    c:\users\balak\appdata\roaming\python\python39\site-packages\numpy\*
Proceed (y/n)? y
Successfully uninstalled numpy-1.20.3
  
```

## Machine Learning Libraries

### 1. NumPy

NumPy basically provides n-dimensional array object. NumPy also provides mathematical functions which can be used in many calculations.

**Command to install:** *pip install numpy*

### 2. SciPy

SciPy is collection of scientific computing functions. It provides advanced linear algebra routines, mathematical function optimization, signal processing, special mathematical functions, and statistical distributions.

**Command to install:** *pip install scipy*

### 3. matplotlib

matplotlib is scientific plotting library usually required to visualize data. Importantly visualization is required to analyze the data. You can plot histograms, scatter graphs, lines etc.

**Command to install:** *pip install matplotlib*

### 4. scikit-learn

scikit-learn is built on NumPy, SciPy and matplotlib provides tools for data analysis and data mining. It provides classification and clustering algorithms built in and some datasets for practice like iris dataset, Boston house prices dataset, diabetes dataset etc.

**Command to install:** *pip install scikit-learn*

### 5. pandas

pandas is used for data analysis it can take multi-dimensional arrays as input and produce charts/graphs. pandas may take a table with columns of different datatypes. It may ingest data from various data files and database like SQL, Excel, CSV etc.

**Command to install:** *pip install pandas*

## Example Programs

### Example1: #python program to display the main module

```
a=10
b=20
print("The sum is ",a+b)
print(__name__)
```

"""output

The sum is 30  
\_\_main\_\_"""

### Example2: #python program to define function add

```
def add(a,b):#called function
    """This function performs addition of two numbers"""
    c=a+b
    return c
a=int(input("Enter the a value"))#10
b=int(input("Enter the b value"))#20
c=add(a,b)#calling fuction
print("The sum of two numbers is ",c)
```

```
print(__name__)
print(add.__doc__)
```

"""output:

```
Enter the a value10
Enter the b value20
The sum of two numbers is 30
__main__
This function performs addition of two numbers"""
```

**Example3:#python program to create function fact that display factorial of a given number**

```
def fact(n):
    i=1
    fact=1
    while i<=n:
        fact=fact*i
        i=i+1
    print("The factorial of given number is ",fact)
n=int(input("Enter the n value"))
fact(n)
```

"""output:

```
Enter the n value10
The factorial of given number is 3628800"""
```

**Example4:#program to demonstrate the required arguments**

```
def fun(str1):#called function
    print("The value of str1 is",str1)
fun()#calling function
```

"""output:

```
Traceback (most recent call last):
  File "D:/Pythonprogramscseeb/requiredargument.py", line 4, in <module>
    fun()#calling function
TypeError: fun() missing 1 required positional argument: 'str1'"""
```

**Example5: #program to demonstrate the keyword arguments**

```
def fun(a,b,c):
    print("The value of a is",a)
    print("The value of b is",b)
    print("The value of c is",c)
fun(c=12.34,a=10,b="hello")
```

"""output:-

```
The value of a is 10
The value of b is hello
The value of c is 12.34"""
```

**Example6: #python program to demonstrate the default arguments**

```
def func(name,course="B.Tech"):
    print("student ",name,"is presently studying",course)
func(name="narasimha")
func(name="pradeep",course="B.Sc")
func(name="rakesh",course="B.Tech")
```

**"""output:**

```
student narasimha is presently studying B.Tech
student pradeep is presently studying B.Sc
student rakesh is presently studying B.Tech"""
```

**Example7: #python program to demonstrate the variable length arguments**

```
def func(name,*hobbies):
    print(name,"likes",end=' ')
    for i in hobbies:
        print(i,end=',')
func("c5","WatchingTv","Playing Games","Chating with friends")
print()
func("Chaitanya","Reading books","Watching Movies")
print()
func("Sarh","Watchingtv")
```

**""output:-**

```
c5 likes WatchingTv,Playing Games,Chating with friends,
Chaitanya likes Reading books,Watching Movies,
Sarh likes Watchingtv,"
```

**Example8:Program to demonstrate the usage of return**

```
def fun():
    return 1
    return 2
print("The function return value is",fun())
```

**""output:-**

```
The function return value is 1""
```

**Example9:Program to demonstrate the usage of local and Global Variables**

```
a=10#a,b global variables
b=20
def fun():
    c=30
    print("The value of c is",c)#local
    print("The value of a is",a)#global
    print("The value of b is",b)
d=40
fun()
print("The value of d is",d)
print("The value of a is",a)#global
print("The value of b is",b)
```

"""**output:-**

The value of c is 30  
The value of a is 10  
The value of b is 20  
The value of d is 40  
The value of a is 10  
The value of b is 20"""

**Example10:#python program to demonstrate the global statement**

```
def fun():  
    global var  
    var="hello cse-b"  
    print("The value of var is",var)  
fun()  
print("The value of var is ",var)
```

"""**output:-**

The value of var is hello cseb  
The value of var is hello cseb"""

**Example11:#program to demonstrate the resolution of names**

```
def fun():  
    str1="hello cseb"  
    print("The value of str is",str1)  
str1="Good Morning"  
fun()
```

"""**output:-**

The value of str is hello cseb"""

**Example12:#python program to create the list dynamically based on size**

```
l=[]  
n=int(input("Enter the value of n"))  
print("Enter the list elements")  
for i in range(n):  
    ele=int(input())  
    l.append(ele)  
print("The list is",l)
```

"""**output:-**

Enter the value of n5  
Enter the list elements  
10  
20  
30  
40  
50  
The list is [10, 20, 30, 40, 50]"""

**week4a :-write python program to perform cumulative product of list of items**

```
def cumulative_product(n):
    l=[]
    product=1
    print("read the list elements upto",n)
    for i in range(n):
        ele=int(input())
        l.append(ele)
    for i in l:
        product=product*i
    return product
n=int(input("Enter the size of the list"))
print("The cumulative product of the list is",cumulative_product(n))
```

**"""output:-**

```
Enter the size of the list5
read the list elements upto 5
1
2
3
4
5
The cumulative product of the list is 120
```

**output2:-**

```
Enter the size of the list6
read the list elements upto 6
11
22
33
44
55
66
The cumulative product of the list is 1275523920"""
```

**Week4b: write a python program to perform evenvalued fib sequence upto 10000**

```
def evenfib(n):
    evensum,prev,next1=0,0,1
    fib=[0]
    while(next1<=n):
        fib.append(next1)
        if next1%2==0:
            evensum=evensum+next1
        prev,next1=next1,prev+next1
    print("The sum of even valued fib sequence",fib,"is",evensum)
n=10000
evenfib(n)
```



**"""output:-**

The sum of even valued fib sequence [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765] is 3382"""

**#python program to demonstrate the lamda functions**

```
square= lambda x : x**2
x=int(input("Enter the value of x"))
print("The square of x is",square(x))
```

**"""output:-**

Enter the value of x3  
The square of x is 9"""

**#python program to perform sum of two numbers**

```
add=lambda a,b : a+b
a=int(input("Enter the first number"))
b=int(input("Enter the second number"))
print("The sum of numbers is",add(a,b))
```

**"""output:**

Enter the first number3  
Enter the second number4  
The sum of numbers is 7"""

**#python program to implement factorial using recursion**

```
def fact(n):
    if(n==0 or n==1):
        return 1
    else:
        return n*fact(n-1)
n=int(input("Enter the value of n"))
print("The factorial is",fact(n))
```

**"""output:-**

Enter the value of n5  
The factorial is 120"""

**#python program to implement fib sequence using recursion**

```
def fib(i):
    if i==0:
        return 0
    elif i==1:
        return 1
    else:
        return fib(i-1)+fib(i-2)
n=int(input("Enter the value of n"))
print("The fib sequence is")
for i in range(n+1):
    print(fib(i),end=' ')
```

"""output:-

Enter the value of n5

The fib sequence is

0 1 1 2 3 5 """

**#function as argument**

```
def square(x):
```

```
    return x**2
```

```
def cube(x):
```

```
    return x**3
```

```
print("The cube of x is",cube(square(2)))
```

"""output:-

The cube of x is 64"""

**#function argument**

```
def square(x):
```

```
    return x**2
```

```
def fun(arg1,n):
```

```
    return arg1(n)
```

```
arg1=square
```

```
print(fun(arg1,3))
```

"""output:-

9"""

### **Week5a**

#### **ArithmeticPackage.py**

```
def addtwo(a,b):
```

```
    return a+b
```

```
def subtwo(a,b):
```

```
    return a-b;
```

```
def multtwo(a,b):
```

```
    return a*b
```

```
def divtwo(a,b):
```

```
    return a/b
```

### **Week5a.py**

```
from ArithmeticPackage import ArithmeticDemo
```

```
a=int(input("Enter the a value"))
```

```
b=int(input("Enter the b value"))
```

```
print("The addition of two numbers is",ArithmeticDemo.addtwo(a,b))
```

```
print("The sub of two numbers is",ArithmeticDemo.subtwo(a,b))
```

```
print("The multiplication of two numbers is",ArithmeticDemo.multtwo(a,b))
```

```
print("The division of two numbers is",ArithmeticDemo.divtwo(a,b))
```

### **Week5b**

```
from math import gcd
```

```
def gcd1(a,b):
```

```
    return gcd(a,b)
```

```
def lcm1(a,b):  
    return a*b//gcd(a,b)  
a=int(input("Enter the value of a"))  
b=int(input("Enter the value of b"))  
print("The gcd of two numbers is",gcd1(a,b))  
print("The lcm of two numbers is",lcm1(a,b))
```

## Unit-II Assignment cum Tutorial Questions

### A. Objective Questions

- User-defined functions are created by using the \_\_\_\_\_ keyword.
- The \_\_\_\_\_ is used to uniquely identify the function.
- The return statement is optional [Yes/No]
- DRY principle makes the code [ ]  
 a) Reusable    b) Loop forever    c) Bad and repetitive    d) Complex
- \_\_\_\_\_ of a variable determines the part of the program in which it is accessible [ ]  
 a) Scope    b) Lifetime    c) Data Type    d) Value
- Arbitrary arguments have which symbol in the function definition before the parameter name? [ ]  
 a) &    b) #    c) %    d) \*
- \_\_\_\_\_dir()\_\_\_\_\_ is built-in function that lists the identifiers defined in a module.
- Arguments may be passed in the form of expressions to the called function [yes/No]
- In Python a string is appended to another string by using which operator? [ ]  
 a) +    b) \*    c) []    d) +=
- Which error is generated when a character in a string variable is modified? [ ]  
 a) IndexError    b) NameError    c) TypeError    d) BoundError
- The code will print how many numbers? [ ]  

```
def display(x):
    for i in range(x):
        print(i)
    return
display(10)
```

 a) 0    b) 1    c) 9    d) 10
- How many times will the print() execute in the code given below? [ ]  

```
def display():
    print('a')
    print('b')
    return
print('c')
print('d')
```

 a) 1    b) 2    c) 3    d) 4
- What is the output of this code? [ ]  

```
import random as r
print(random.randomint(1,10))
```

 a) An error occurs    b) 1    c) 10    d) any random value.
- Identify the correct way of calling a function named display() that prints Hello on the screen. [ ]  
 a) print(display)    b) displayHello    c) result = display()    d) displayHello()
- Find the error in following Python code. [ ]  

```
def func():
    print("Hello world")
```

 a) Hello world    b) "Hello world"    c) no function call    d) none of the above

$$[ \quad ]$$

a) 20 20      b) 20      c) 9      d) 20'var' is not defined

$$[ \quad ]$$

[ ]

d) Indentation Error:

[ ]

b)str' object does not support item assignment      d) Hello wworld

### B. *Subjective Questions*

1. Define function and give its advantages. [ BL-1]
2. Distinguish between local and global variables. [ BL-4]
3. What are modules? How do you use them in your programs? [BL-1]
4. Demonstrate the Keyword arguments and Default arguments. [BL-2]
5. Outline format operator with an example. [BL-2]
6. With the help of an example, explain how we can create string variables in Python. [BL-2]
7. Explain user-defined functions with the help of an example. [BL-2]
8. Explain Recursion with an example and list out the Advantages and disadvantages of it. [BL-2]
9. Make use of python program to find the factorial of a given number using recursion. [BL-4]
10. Demonstrate any 5 Built-in string methods and functions usage and example. [BL-2]
11. Distinguish between Error and Exception. [BL-4]
12. Explain different keywords used in Exception. [BL-2]
13. How to raise more than one exception in try block with suitable example. [BL-1]
14. Make use of python program to handle multiple errors with one except statement. [BL-4]
15. Make use of python program to create a user-defined exception named “ShortInputException” that rises when the input text length is less than 3. [BL-4]
16. Make use of Python program to find out the power passing through the circuit and voltage difference at the end points of the circuit where current (I) and Resistance (R) are given. ( $P=I^2R$ ,  $V=IR$ ) [BL-4]
17. Make use of python program to find equivalent resistances ( $R_1$ ,  $R_2$ ,  $R_3$ ) while converting from delta to star, where delta resistances are given  $R_a$ ,  $R_b$ ,  $R_c$  respectively
 

$$R_1 = R_b * R_c / (R_a + R_b + R_c)$$

$$R_2 = R_a * R_c / (R_a + R_b + R_c)$$

$$R_3 = R_a * R_b / (R_a + R_b + R_c)$$
[BL-4]
18. Make USE of python program to find young’s modulus, tensile stress and tensile strain of a material where area (A) , length (L), change in length ( $\Delta l$ ) and force (F) applied on the material are given repectively.
 

$$\text{Tensile stress} = F/A$$

$$\text{Tensile Strain} = \Delta l/L$$

$$\text{Young’s modulus} = \text{Tensile Stress} / \text{Tensile Strain}$$
[BL-4]