Introduction

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF BIO AND CHEMICAL ENGINEERING

Common to : Biotech, BioMed, Chemical, EEE

UNIT – I - SCSA1102 - FUNDAMENTALS OF PYTHON PROGRAMMMING

Why python?

UNIT I

INTRODUCTION

1. Program

A program performs a task in the computer. But, in order to be executed, a program must be written in the machine language of the processor of a computer. Unfortunately, it is extremely difficult for humans to read or write a machine language program. This is because a machine language is entirely made up of sequences of bits. However, high level languages are close to natural languages like English and only use familiar mathematical characters, operators and expressions. Hence, people prefer to write programs in high level languages like C, C++, Java, or Python. A high level program is translated into machine language by translators like compiler or interpreter.

1.1 ABOUT PYTHON

Guido-Van-Rossum

Introduction

Python is a high level programming language that is translated by the python **interpreter**. As is known, an interpreter works by translating line-by-line and executing. It was developed by Guido-Van-Rossum in 1990, at the National Research Institute for Mathematics and Computer Science in Netherlands. Python doesn't refer to the snake but was named after the famous British comedy troupe, Monty Python's Flying Circus.

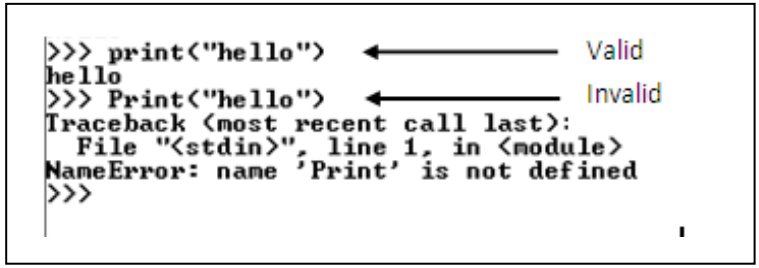
The following are some of the features of Python:

- Python is an Open Source: It is freely downloadable, from the link "http:// python.org/"
- Python is portable: It runs on different operating systems / platforms
- Python has automatic memory management
- Python is flexible with both procedural oriented and object oriented programming
- Python is easy to learn, read and maintain

It is very flexible with the console program, Graphical User Interface (GUI) applications, Web related programs etc.

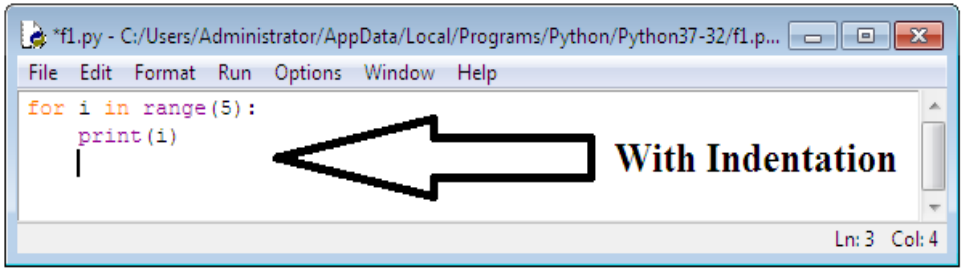
Points to remember while writing a python program :

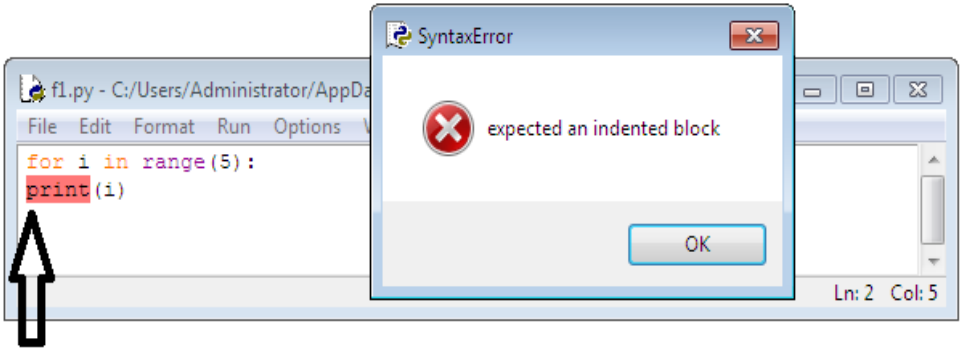
- Case sensitive : Example - In case of print statement use only lower case and not upper case, (See the snippet below)



Introduction

- Punctuation is not required at end of the statement
- In case of string use single or double quotes i.e. ‘ ’ or “ ”
- Must use proper indentation: The screen shots given below show, how the value of “i” behaves with indentation and without indentation.





Without Indentation

- Special characters like (,), # etc. are used
- ( ) -> Used in opening and closing parameters of functions
- # -> The Pound sign is used to comment a line

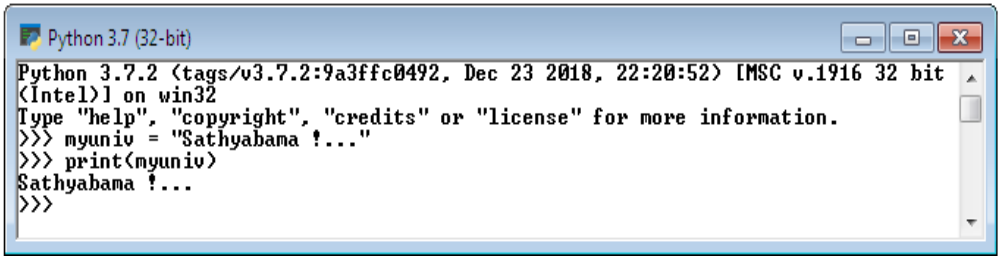
1.2 TWO MODES OF PYTHON PROGRAM

Python Program can be executed in two different modes:

- Interactive mode programming
- Script mode programming

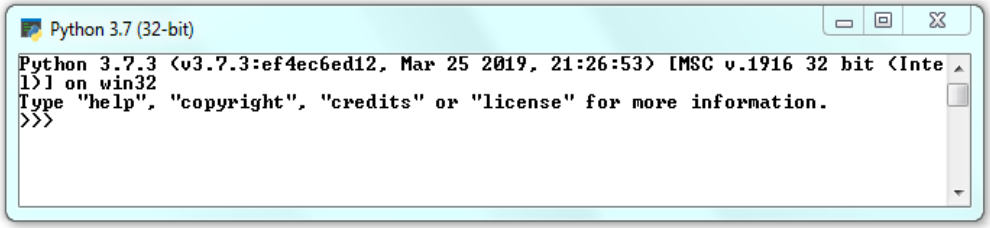
1.2.1 Interactive Mode Programming

It is a command line shell which gives immediate output for each statement, while keeping previously fed statements in active memory. This mode is used when a user wishes to run one single line or small block of code. It runs very quickly and gives instant output. A sample code is executed using interactive mode as below.



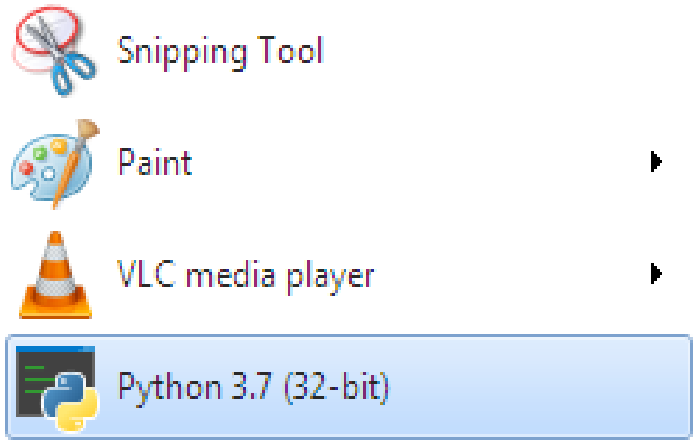
Interactive mode can also be opened using the following ways:

i) From command prompt c :> users\\...>python



The symbol ">>>" in the above screen indicates that the Python environment is in interactive mode.

ii) From the start menu select Python (As shown below)



1.2.2 Script Mode Programming

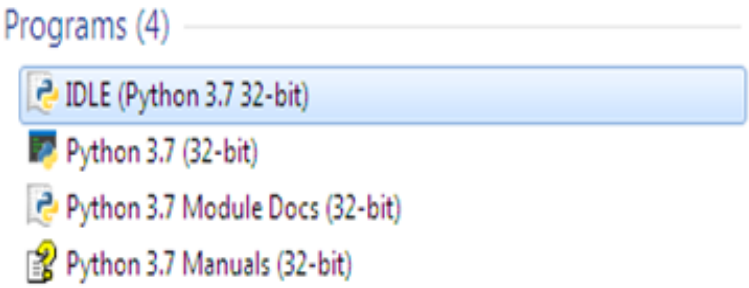
When the programmer wishes to use more than one line of code or a block of code, script mode is preferred. The Script mode works the following way:

i) Open the Script mode

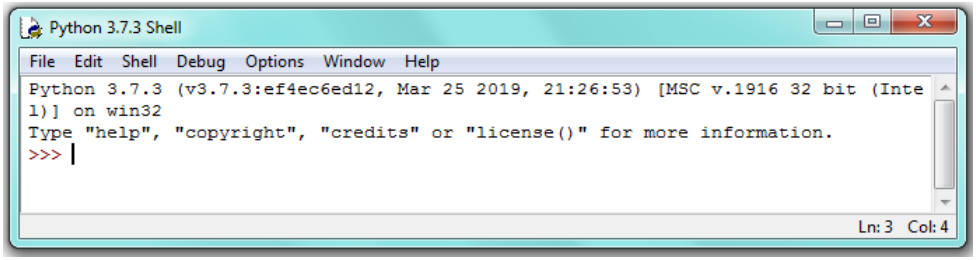
ii) Type the complete program. Comment, edit if required.
iii) Save the program with a valid name.
iv) Run
v) Correct errors, if any, Save and Run until proper output

The above steps are described in detail below:

i) To open script mode, select the menu "***IDLE (Python 3.7 32-bit)"*** from start menu



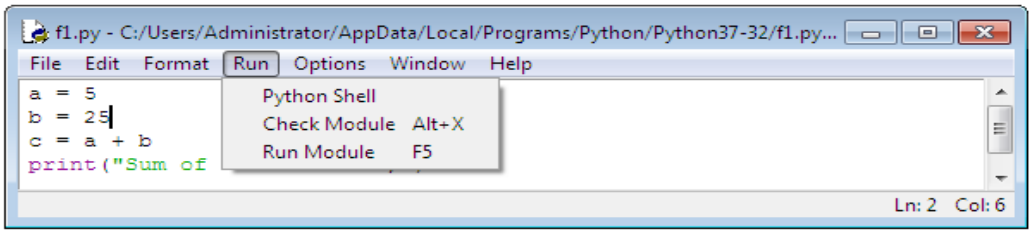
ii) After clicking on the menu "***IDLE (Python 3.7 32-bit)"*** , a new window with the text Python 3.7.3 shell will be opened as shown below:



iii) Select File → New, to open editor. Type the complete program.
iv) Select File again; Choose Save.

This will automatically save the file with an extension ".py".

v) Select Run → Run Module or Short Cut Key **F5** ***(As shown in the screen below)***



The output of the program will be displayed as below:

>> Sum of a and b is: 30

1.3 VARIABLES

Variable is the name given to a reserved memory locations to store values. It is also known as Identifier in python.

***Need for variable*:**

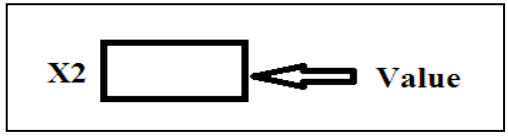
Sometimes certain parameters will take different values at different time. Hence, in order to know the current value of such parameter we need to have a temporary memory which is identified by a name that name is called as variable. For example, our surrounding temperature changes frequently. In order to know the temperature at a particular time, we need to have a variable.

Naming and Initialization of a variable

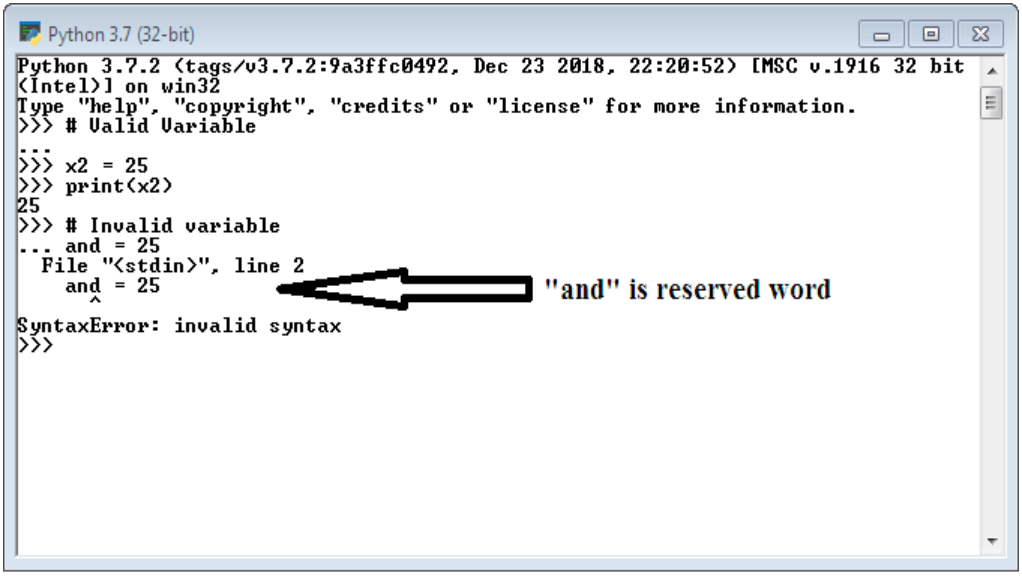
1. A variable name is made up of alphabets (Both upper and lower cases) and digits
2. No reserved words
3. Initialize before calling
4. Multiple variables initialized
5. Dynamic variable initialization

i. Consist of upper and lower case alphabets, Numbers (0-9). E.g. X2

In the above example, a memory space is assigned to variable X2. The value of X2 is stored in this space.

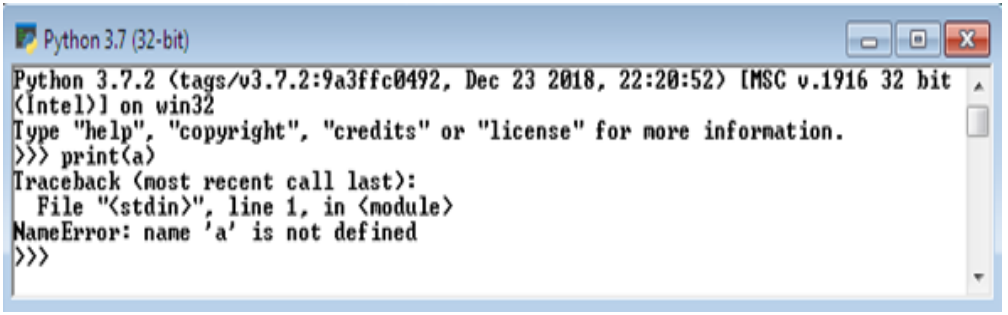


ii. Reserved words should not be used as variables names.



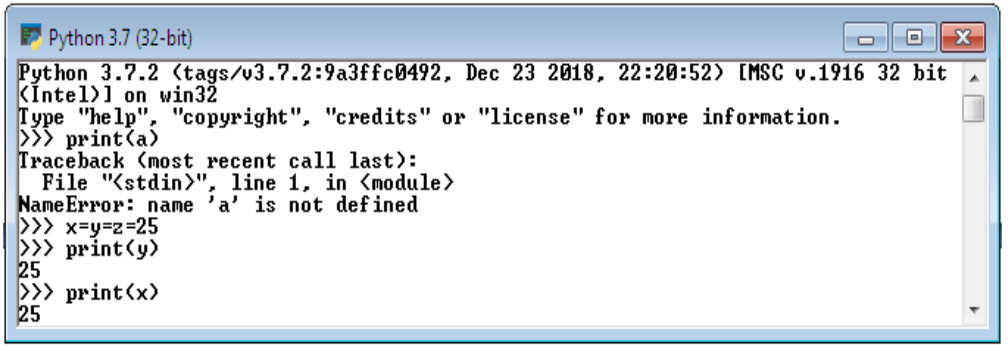
In the above example “and” is a reserved word, which leads to Syntax error

iii. Variables must be initialized before it called , else it reports “is not defined ” error message as below E.g.: a = 5 print(a)



In the above example “a” is called before it initialized. Hence, the python interpreter generates the error message: NameError: ‘a’ is not defined.

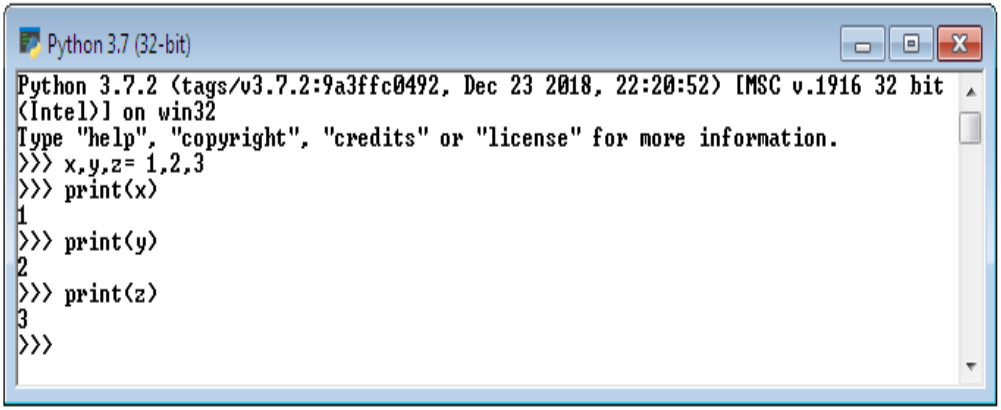
iv. Multiple variables can be initialized with a common value. E.g. : x = y = z = 25



In the above three variables x, y, z is assigned with same value 25.

v. Python also supports dynamic variable initialization. E.g.: x, y, z = 1, 2, 3

Introduction



Proper spacing should be given

- print (10+20+30) → bad style
- print (20 + 30 + 10) → good style

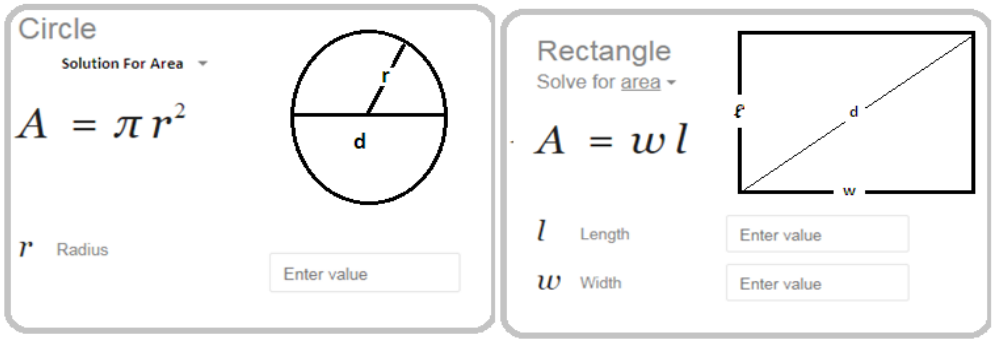
1.3.1 Expression:

An expression is a combination of variables, operators, values and calls to functions. Expressions need to be evaluated.

Need for Expression:

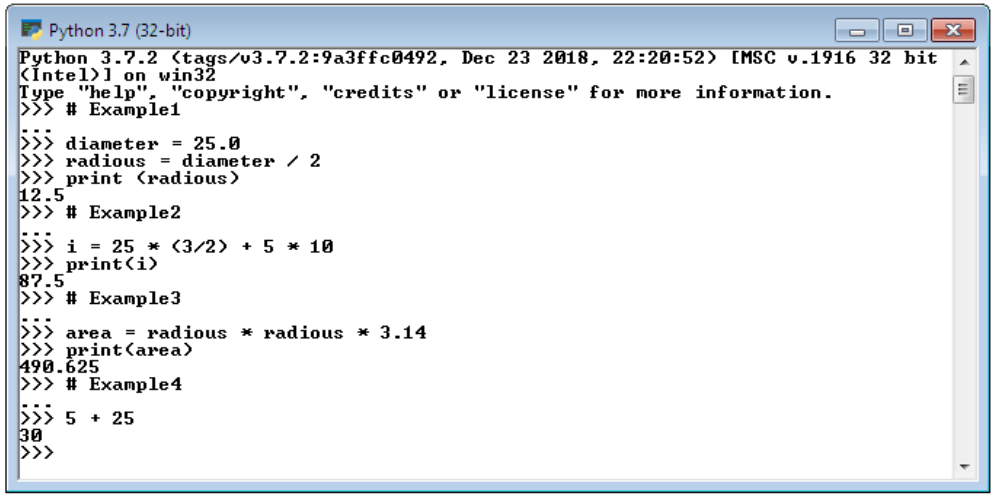
Suppose if you wish to calculate area. Area depends on various parameters in different situations. E.g. Circle, Rectangle and so on…

Introduction



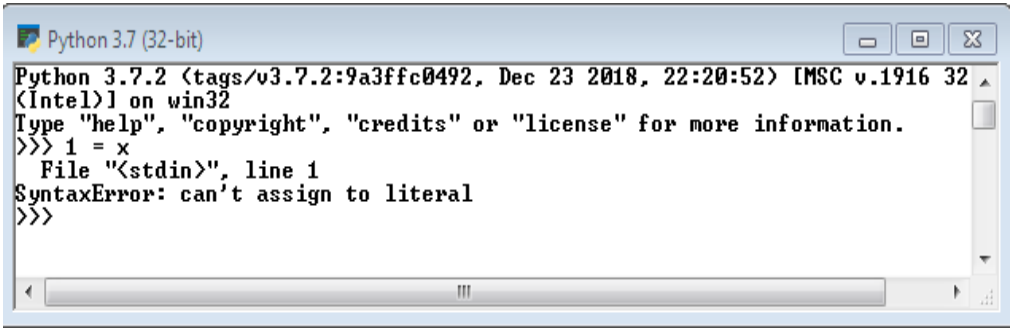
In order to find area of circle, the expression π * r * r must be evaluated and for the rectangle the expression is w * l in case of rectangle. Hence, in this case a variable / value / operator are not enough to handle such situation. So expressions are used. Expression is the combination of variables, values and operations.

A simple example of an expression is 10 + 15. An expression can be broken down into operators and operands. Few valid examples are given below.



Invalid Expression :

Always values should be assigned in the right hand side of the variable, but in the below example, the value is given in the left hand side of the variable, which is an invalid syntax for expression.



1.4 Data Types

A Data type indicates which type of value a variable has in a program. However a python variables can store data of any data type but it is necessary to identify the different types of data they contain to avoid errors during execution of program. The most common data types used in python are str(string), int(integer) and float (floating-point).

Strings: Sequence of characters inside single quotes or double quotes.

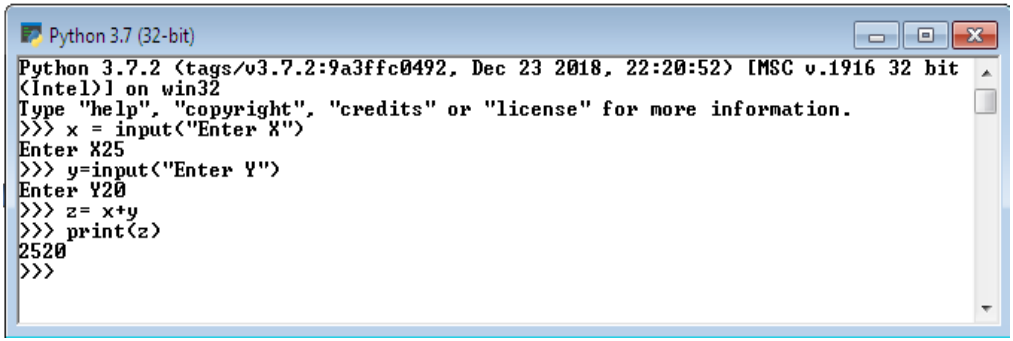
E.g. myuniv = “Sathyabama !..”

Integers: Whole number values such as 50, 100,-3

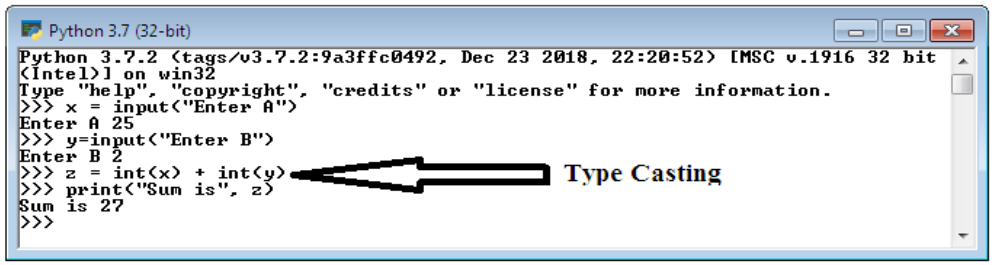
Float: Values that use decimal point and therefore may have fractional point E.g.: 3.415, -5.15

By default when a user gives input it will be stored as string. But strings cannot be used for performing arithmetic operations. For example while attempting to perform arithmetic operation add on string values it just concatenates (joins together) the values

together rather performing addition. For example : ‘25’ + ‘20’ = ‘45’ (As in the below Example)



Fortunately python have an option of converting one date type into another data type (Called as “Casting”) using build in functions in python. The build function int() converts the string into integer before performing operation to give the right answer. (As in the below Program)



1.4.1 Compound Data Types in Python:

i) *List*

The List is an ordered sequence of data items . It is one of the flexible and very frequently used data type in Python. All the items in a list are not necessary to be of the same data type.

Introduction

Declaring a list is straight forward methods. Items in the list are just separated by commas and enclosed within brackets [ ].

```
>>> list1 = [3.141, 100, ‘CSE’, ‘ECE’, ‘IT’, ‘EEE’]
```

Methods used in list

Table 1.1 List Method

list1.append(x)	To add item x to the end of the list “list1”
list1.reverse()	Reverse the order of the element in the list “list1”
list1.sort()	To sort elements in the list
list1.reverse()	To reverse the order of the elements in list1.

ii) Tuple

Tuple is also an ordered sequence of items of different data types like list. But, in a list data can be modified even after creation of the list whereas Tuples are immutable and cannot be modified after creation.

The advantages of tuples is to write-protect data and are usually very fast when compared to lists as a tuple cannot be changed dynamically.

The elements of the tuples are separated by commas and are enclosed inside open and closed brackets.

```
>>> t = (50,'python', 2+3j)
```

Table : 1.2 List Vs Tuple

List	Tuple

>>> list1[12,45,27]
>>> list1[1] = 55
>>> print(list1)
>>> [12,55,27]
	>>> t1 = (12,45,27)
	>>> t1[1] = 55
	>>> Generates Error Message # Because Tuples are immutable

iii) Set

The Set is an unordered collection of unique data items. Items in a set are not ordered, separated by comma and enclosed inside { } braces. Sets are helpful in performing operations like union and intersection. However, indexing is not done because sets are unordered.

Table : 1.3 List Vs Set

List	Set

>>> L1 = [1,20,25]
>>> print(L1[1])
>>> 20
	>>> S1= {1,20,25,25}
	>>> print(S1)
	>>> {1,20,25}
	>>> print(S1[1])
	>>> Error , Set object does not support indexing.

iv) Dictionary

The Python Dictionary is an unordered collection of key-value pairs.Dictionaries is optimized for retrieving data when there is huge volume of data. They provide the key to retrieve the value.

In Python, dictionaries are defined within braces {} with each item being a pair in the form key: value. Key and value can be of any type.

```
>>> d1 = {1:'value','key':2}

>>> type(d)
```

1.5 PYTHON'S BUILT-IN DATA TYPE CONVERSION FUNCTIONS

Table 1.4 : Python Built-in Data Type

Function	Description	Out Put
int(x)	Converts x into integer whole number	>>>a = int(input("Enter a")) >>>b = int(input("Enter b")) >>>c = a + b >>>print("Sum is ",c)

float(x)	Converts x into floating-point number	>>> x = 5
		>>> print(float(5))
		>>> 5.0

Function	Description	Out Put

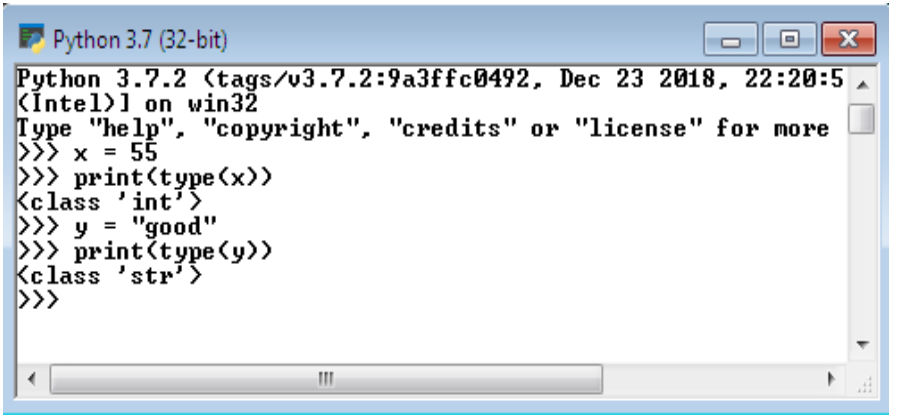
str(x)	Converts x into a string representation	>>> x = 30
		>>> y = 70
		>>> z = str(x) + str(y)
		>>> print(z)
		>>> 3070

chr(x)	Converts integer x into a character	>>> x = 65
		>>> print(chr(x))
		>>> A
		>>>

hex(x)	Converts integer x into a hexadecimal string	>>> x = 14
		>>> print(hex(x))
		>>> 0xe

oct(x)	Converts integer x into an octal string	>>> x = 9
		>>> print(oct(x))
		>>> 0o11

However to identify the data type of a variable, an in-built python function “type ( )” is used. (Example Below)



1.5.1 Python Built-in Functions

Table 1.5 : Built-in Functions

Simple Functions		
Function	**Description**	**Output**

abs( )	Return the absolute value of a number. The argument may be an floating point number or a integer.	>>> a = -10
		>>> print(abs(a))
		>>> 10

max( )	Returns the largest number from the list of numbers	>>> max(12,20,30) >>> 30
min( )	Returns the smallest number from the list of numbers	>>> min(12,20,30) >>> 12
pow( )	Returns the power of the given number	>>> pow(5,2) >>>25
round( )	It **rounds** off the number to the nearest integer.	# E.g. 1: >> round(4.5) >> 5 # Eg 2 >> round(4.567,2) >> 4.57
Mathematical Functions (Using math module)		
ceil(x)	It rounds x up to its nearest integer and returns that integer	>> math.ceil(2.3) >> 3 >> math.ceil(-3.3) >> -3
floor(x)	It rounds x down to its nearest integer and returns that integer	>>math.floor(3.2) >> 3 >> math.floor(-3.4) >> -4

Function	Description	Example
cos(x)	Returns the cosine of x , where x represents angle in radians	>> math.cos(3.14159/2) >> 0 >> math.cos(3.14159) >> -1
sin(x)	Returns the sine of x, where x represents angle in radians	>> math.sin(3.14159/2) >> 1 >> math.sin(3.14159) >> 0
exp(x)	Returns the exponential of x to the base 'e'. i.e. e^x	>> math.exp(1) >> 2.71828
log(x)	Returns the logarithm of x for the base e (2.71828)	>>> math.log(2.71828) >>> 1
log(x,b)	Returns the logarithm of x for the specified base b.	>>> math.log(100,10) >>> 2
sqrt(x)	Returns the square root of x	>>> math.sqrt(16) >>> 4

Note: To include the math module, use the following command:

import math

1.6 CONDITIONAL STATEMENTS

When there is no condition placed before any set of statements, the program will be executed in sequential manure. But when some condition is placed before a block of statements the flow of execution might change depends on the result evaluated by the condition. This type of statement is also called decision making statements or control statements. This type of statement may skip some set of statements based on the condition.

Logical Conditions Supported by Python

- Equal to (==) Eg : a == b
- Not Equal (!=)Eg : a != b
- Greater than (>) Eg : a > b
- Greater than or equal to (>=) Eg : a >= b
- Less than (<) Eg : a < b
- Less than or equal to (<=) Eg : a <= b

Indentation

To represent a block of statements other programming languages like C, C++ uses "{ …}" curly – brackets, instead of this curly braces python uses indentation using white space which defines scope in the code. The example given below shows the difference between usage of Curly bracket and white space to represent a block of statement.

Table 1.6 : ***C- Program Vs Python***

C Program	Python

x = 500
y = 200
if (x > y)
{
  printf("x is greater than y")
}
else if(x == y)
{
  printf("x and y are equal")
}
else
{
 printf("x is less than y")
}
	x = 500
	y = 200
	if x > y:
	  print("x is greater than y")
	elif x == y:
	  print("x and y are equal")
	else:
	  print("x is less than y")
	 
	Indentation (At least one White Space instead of curly bracket)

Without proper Indentation:

```
x = 500
y = 200
if x > y:
print("x is greater than y")
```

In the above example there is no proper indentation after if statement which will lead to Indentation error.

1.6.1 If statement:

The ‘if’ statement is written using “if” keyword, followed by a condition. If the condition is true the block will be executed. Otherwise, the control will be transferred to the first statement after the block.

Syntax :

if <Boolean>:

    <block>

In this statement, the order of execution is purely based on the evaluation of boolean expression.

Example:

```
x = 200
y = 100
if x > y:
  print("X is greater than Y")
print(“End”)
```

Output:

X is greater than Y

End

In the above the value of x is greater than y , hence it executed the print statement whereas in the below example x is not greater than y hence it is not executed the first print statement

```
x = 100
y = 200
if x > y:
  print("X is greater than Y")
print(“End”)
```

Output:

```
End
```

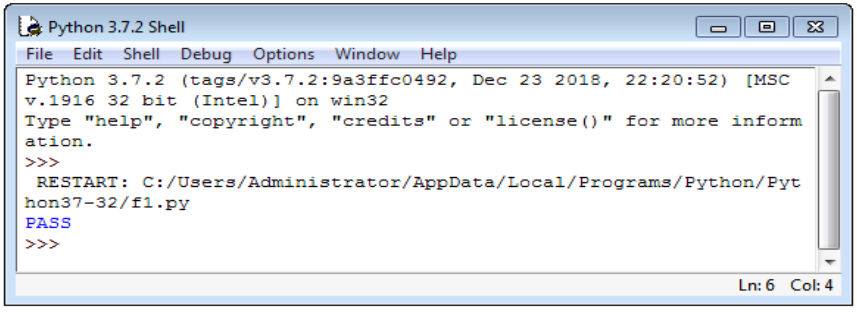
1.6.2 elif

The **elif** keyword is useful for checking another condition when one condition is false.

Example :

```
mark = 55
if (mark >=75):
  print("FIRST CLASS")
elif mark >= 50:
  print("PASS")
```

Output:



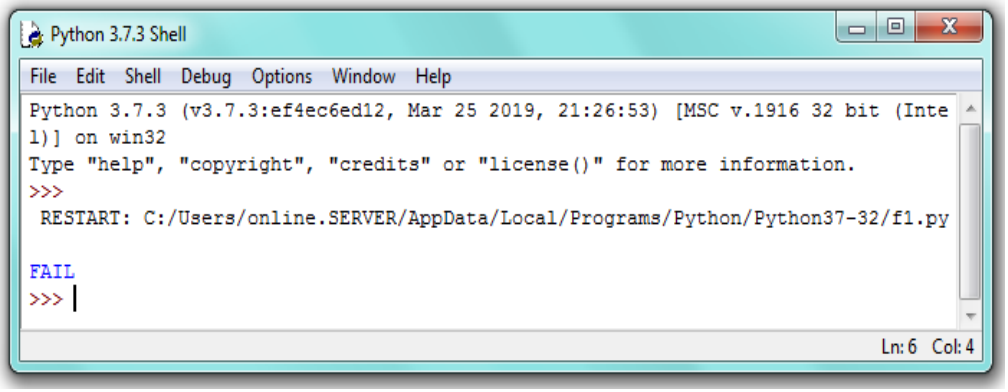
In the above the example, the first condition (mark >=75) is false then the control is transferred to the next condition (mark >=50), Thus, the keyword **elif** will be helpful for having more than one condition.

1.6.3 Else

The **`else`** keyword will be used as a default condition. i.e. When there are many conditions, when the **if-condition** is not true and all **elif-conditions** are also not true, then **else** part will be executed.

`Example:`

```
mark = 10
if mark >= 75:
  print("FIRST CLASS")
elif mark >= 50:
  print("PASS")
else:
  print("FAIL")
```



In the example above, condition 1 and condition 2 fail. None of the preceding condition is true. Hence, the **else** part is executed.

1.7 ITERATIVE STATEMENTS

Sometimes certain section of the code (block) may need to be repeated again and again as long as certain condition remains true. In order to achieve this, the iterative statements are used. The number of times the block needs to be repeated is controlled by the test condition used in that statement. This type of statement is also called as the "Looping Statement". Looping statements add a surprising amount of new power to the program.

Need for Looping / Iterative Statement

Suppose the programmer wishes to display the string "Sathyabama !..." 150 times. For this, one can use the print command 150 times.

```
print ("Sathyabama !...")
print ("Sathyabama !...")
        .....
        .....
print ("Sathyabama !...")
```

150 times

The above method is somewhat difficult and laborious. The same result can be achieved by a loop using just two lines of code. (As below)

```
for count in range (1,150):
    print ("Sathyabama !...")
```

Introduction

Types of looping statements

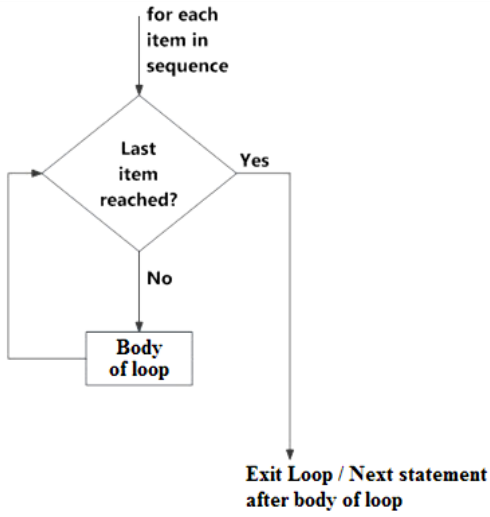
1) **for** loop
2) **while** loop

1.7.1 The 'for' Loop

The **for** loop is one of the powerful and efficient statements in python which is used very often. It specifies how many times the body of the loops needs to be executed. For this reason it uses control variables which keep tracks, the count of execution. The general syntax of a 'for' loop looks as below:

```
for <variable> in range (A,B):

    <body of the loop >
```

Flow Chart:



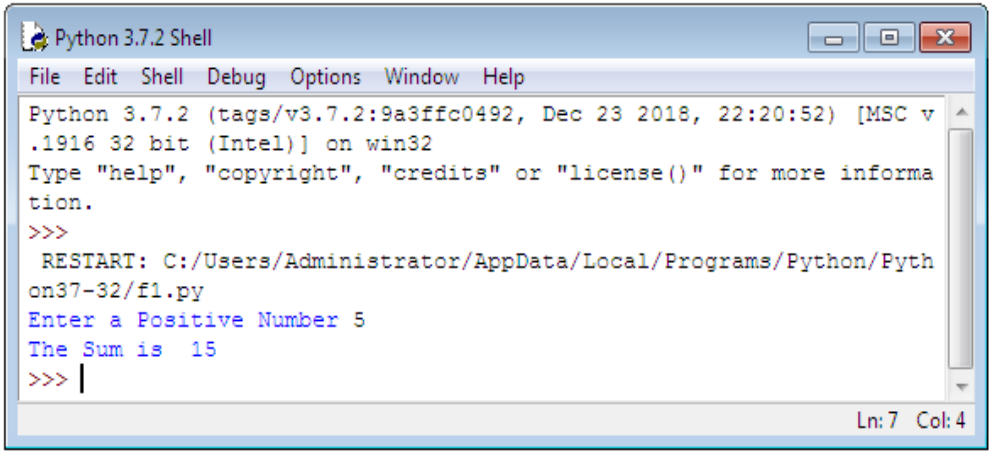
Introduction

Example 1: To compute the sum of first n numbers (i.e. 1 + 2 + 3 + ……. + n)

```
# Sum.py
total = 0
n = int (input ("Enter a Positive Number"))
for i in range(1, n+1):
    total = total + i
print ("The Sum is ", total)
```

Note: Why (n+1)? Check in table given below.

Output:



In the above program, the statement total = total + i is repeated again and again ‘n’ times. The number of execution count is controlled by the variable ‘i’. The range value is specified earlier before it starts executing the body of loop. The initial value for the variable i is 1 and final value depends on ‘n’. You may also specify any constant value.

1.7.2 The range( ) Function:

The **range()** function can be called in three different ways based on the number of parameters. All parameter values must be integers.

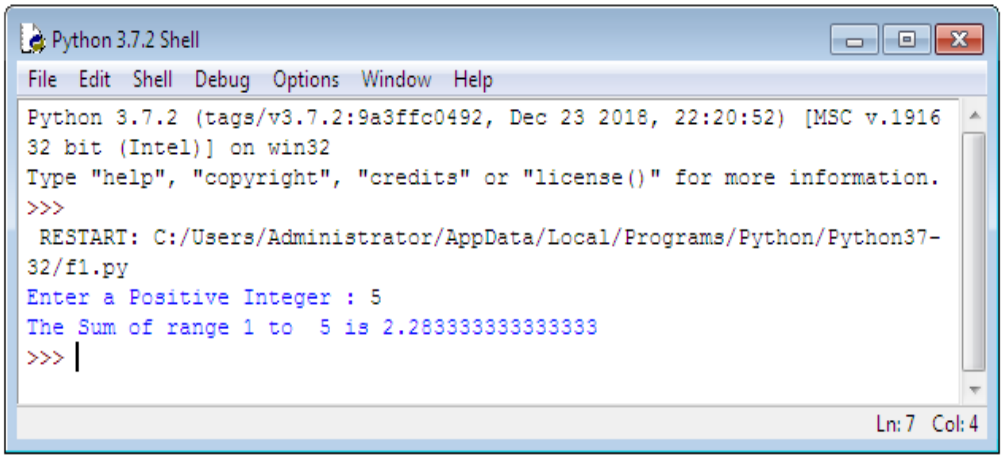
Table 1.7: Categories of range function

Type	Example	Explanation
range(end)	for i in range(5): print(i) **Output :** 0,1,2,3,4	This is begins at 0. Increments by 1. End just before the value of end parameter.
range(begin, end)	for i in range(2,5): print(i) **Output :** 2,3,4	Starts at begin, End before end value, Increment by 1
range(begin,end,step)	for i in range(2,7,2) print(i) **Output :** 2,4,6	Starts at begin, End before end value, increment by step value

Example: To compute Harmonic Sum (ie: 1 + ½ + 1/3 + ¼ + …..1/n)

```
# harmonic.py
total = 0
n= int(input("Enter a Positive Integer:"))
for i in range(1,n+1):
   total+= 1/i
print("The Sum of range 1 to ",n, "is", total)
```

Output:

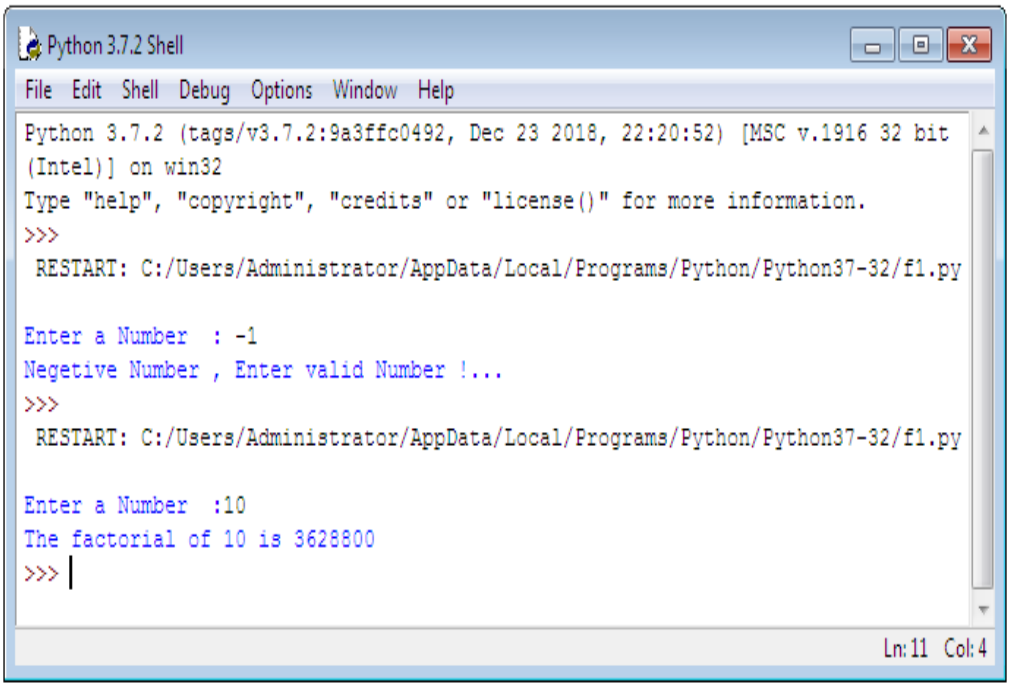


Example:

Factorial of a number “n"

```
n= int(input("Enter a Number :"))
factorial = 1
# Initialize factorial value by 1
# To verify whether the given number is negative / positive / zero
if n < 0:
   print("Negative Number , Enter valid Number !...")
elif n == 0:
   print("The factorial of 0 is 1")
else:
   for i in range(1, n + 1):
      factorial = factorial*i
   print("The factorial of" ,n, "is", factorial)
```

Output:



1.7.3 The while Loop

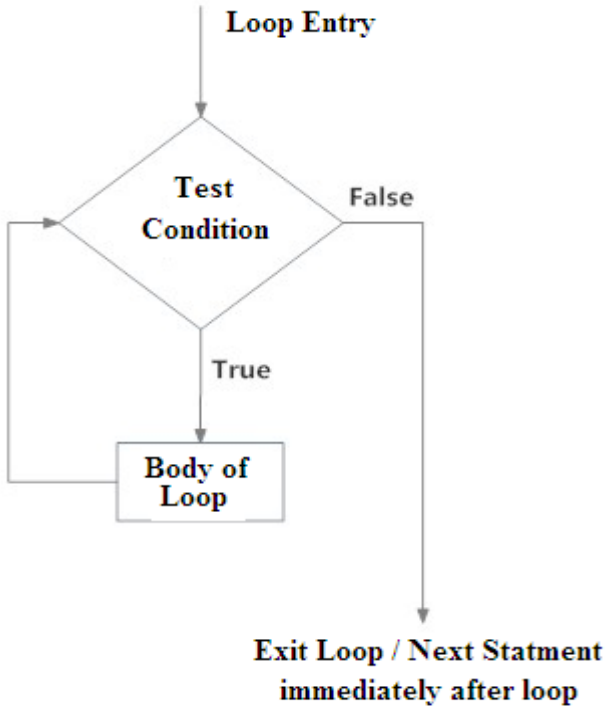
The **while** loop allows the program to repeat the body of a loop, any number of times, when some condition is true.

The drawback of **while** loop is that, if the condition is not proper it may lead to infinite looping.

So the user has to carefully choose the condition in such a way that it will terminate at a particular stage.

Introduction

Flow Chart:



Syntax:

```
while (condition):
        <body of the loop>
```

Introduction

In this type of loop, The execution of the loop body is purely based on the output of the given condition. As long as the condition is TRUE or in other words until the condition becomes FALSE the program will repeat the body of loop.

Valid Example	**Invalid Example**

i = 10
while i<15 :
  print(i)
  i = i + 1
Output :
10,11,12,13,14
	i = 10
	while i<15 :
	  print(i)
	
	Output :
	10,10,10,10…….. Indeterminate number of times

Example: Program to display Fibonacci Sequence

```
# Program to Display Fibonacci Sequence based on number of terms n
n = int(input("Enter number of terms in the sequence you want to display"))
# n1 represents -- > first term and n2 represents --> Second term
n1 = 0
n2 = 1
count = 0
# count -- To check number of terms
if n <= 0:                  # To check whether valid number of terms
  print ("Enter a positive integer")
elif n == 1:
  print("Fibonacci sequence up to ", n,":")
  print(n2)
else:
  print("Fibonacci sequence of  ",n, " terms :" )
  while count < n:
    print(n1, end=' , ')
    nth = n1 + n2
    n1 = n2
```

```
    n2 = nth
    count = count + 1
```

1.8 INPUT / OUTPUT STATEMENT:

Programmer often has a need to interact with users, either to get data or to provide some sort of result.

For Example: In a program to add two numbers, first the program needs to have an input of two numbers (The numbers which they prefer to add) and after processing, the output should be displayed. So to get the input of two numbers, the program need to have an Input Statement and in order to display the result i.e. the sum of two numbers, it needs to have an Output Statement.

1.8.1 Input Statement:

Helpful to take input from the user through input devices like keyboard. In Python, the standard input function is 'input()'

The syntax for input function is as follows:

input( )

However, to get an input by prompting the user, the following form is used:

input('prompt')

where prompt is the string, which programmer wish to display on the screen to give more clarity about the input data. It is optional.

Example:

```
>>> num = input('Enter a number: ')
```

The above statement will wait till the user, enters the input value.

Output:

```
Enter a number:
>>> num
'10'    # Input data entered by the user
```

1.8.2 Output Statement:

The output statement is used to display the output in the standard output devices like monitor (screen). The standard output function “print()” is used.

Syntax:

```
print(‘prompt’)
```

where prompt is the string, which programmer wish to display on the screen

Example 1:

```
print('Welcome to the Python World!')
```

Output:

Welcome to the Python World !

Example 2:

```
X = 5
print ('The value of a is', X)
```

Output:

The value of X is 5

Example 3:

```
print(1,2,3,4)
```

Output:

1 2 3 4

Example 4:

```
print(100,200,300,4000,sep='*')
```

Output:

100*200*300*4000

Example 5:

print(1,2,3,4, sep='#', end='&')

Output:

1#2#3#4&

1.9 OBJECT ORIENTED PROGRAMMING:

Python supports object oriented programming concepts. The basic entities in object oriented programming are Class, Objects, and Methods. It also supports some of the techniques in real world entities like inheritance, Data hiding, Polymorphism, Encapsulation, Method Overloading etc., in programming. Object orientation helps to utilize GUI environment efficiently. Some of the other programming languages which support OOPS concepts are C++, JAVA, C#.net, VB.net etc.

Need for Object Oriented Programming:

The object oriented programming is having certain advantage when compared to the normal procedure oriented programming. The main advantage is to provide access specifiers like Public, Private and Protected. Oops provide data hiding technique which is more secured than procedure oriented programming. Code reusability is one of the key features of OOPs Concept.

Class: It is a template or blue print created by the programmer – which defines how the object's data field and methods are represented. Basically class consists of two parts: data member and function member (methods).

Object: It is an instance of a Class; Any number objects can be created.

Class Name: Student
Data Fields:
Name, Mark1, Mark2, Mark3
Methods:
Average ()
Rank ()

Introduction

A Class is a template for creating an object.

Python provides a special method, __init__, called as initializer, to initialize a new object when it is created.

Example :

```
class Student:
  def __init__(self, name, regno):
              self.name = name
              self.regno = regno
  s1 = Student("John", 36)
print(s1.name)
print(s1.regno)
```

In the above example “Student” is the class name, name and regno are the data fields and s1 is the created object,

Note :

__init__ is a method or constructor in Python. This method is automatically called to allocate memory when a new object/ instance of a class is created. All classes have the __init__ method.

Output :

>>> John

36

Let us create a method (Function member) for the above class

```
class Student:
   def __init__(self, name, regno):
      self.name = name
      self. regno = regno
   def display(self):
      print("Name of the student is " + self.name )
s1 = Student("James", 43)
s1.display()
```

In the above example ***"display( )"*** is the method used to display the student name.

1.9.1 Inheritance

Inheritance allows to create a new class (Child Class) from the existing class (Parent Class).

The child class inherits all the attributes of its parent class.

Parent class is the class, whose properties are being inherited by subclass. Parent class is also called as Base class or Super Class.

Child class is the class that inherits properties from another class. The child class is also called as Sub class or Derived Class.

Introduction

Example :

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname
  def printdetails(self):
    print(self.firstname, self.lastname)
```

#Use the Person class to create an object and then execute the printdetails method:

```
x = Person("John", "Doe")
x.printdetails()
class Employee(Person):
  pass
y = Employee("Mike", "Olsen")
y.printdetails()
```

Output :

```
>>>

 RESTART:
C:/Users/Administrator/AppData/Local/Programs/Python/Python37-32/f1.py

John Doe

Mike Olsen

>>>
```

In the above example the base class is Person. The first object “x” is created through the base class ***“Person”*** and the method ***printdetails( )*** is invoked with that object which produces an output “John Doe”. Again, another object “y” is created through derived class ***“Employee”*** and the same method ***printdetails( )*** (belongs to base class) is invoked to produce the output “Mike Olsen”. Thus, the derived class is having the ability to invoke the method from base class just because of the inheritance property which reduces the code length or in other words it is helpful for reusability of code.

Note: Use the pass keyword when the programmer does not wish to add any other properties or methods to the derived class.

Example 2:

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname
  def printdetails(self):
    print(self.firstname, self.lastname)
#Object For Base Class
x = Person("Paul", "Benjamin")
x.printdetails()

class Employee(Person):
  def __init__(self, fname, lname):
    Person.__init__(self, fname, lname)
    self.doj = 2019
  def greetings(self):
    print("Welcome", self.firstname, self.lastname, "who joined in the year ", self.doj)
```

```
# Object for derived class
y = Employee("Samuel", "Ernest")
y.printdetails()
y.greetings()
```

In the above example a new method ***greetings( )*** is included in the derived class, Thus the derived class object is capable of invoking the method present inside base class as well as its own methods.

printdetails( ) -- method present inside base class Person.

greetings( ) -- method present inside derived class Employee.

The object “y” is able to invoke both the methods ***printdetails( )*** and ***greetings( ).***

Introduction

Questions :

1. Compare a) List and Tuple b) List and Set
2. What is type conversion in Python?
3. Is indentation required in python?
4. What is __init__?
5. How can you randomize the items of a list in place in Python?
6. How do you write comments in python?
7. What is a dictionary in Python?
8. Does Python have OOps concepts?
9. Write a program in Python to check if a sequence is a Palindrome.
10. Write a program in Python to check if a number is prime.
11. How to create an empty class in Python?
12. Write a sorting algorithm for a numerical dataset in Python.

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF BIO AND CHEMICAL ENGINEERING

Common to : Biotech, BioMed, Chemical, EEE

UNIT – II - SCSA1102 - FUNDAMENTALS OF PYTHON PROGRAMMMING

UNIT II

FILES AND EXCEPTIONS HANDLING, MODULES, PACKAGES

2.1 FILE OPERATIONS

An object that stores data, settings or programming commands in a computer system is called as a file. There are three major file operations:

- Opening a file
- Performing file operations using Read or Write
- Closing the file

2.1.1. File Open

Method: open()

Purpose: To open a file

Syntax:

File_object=open(filename,Access_mode,buffering)

Attributes:

i. Filename – Name of the file

ii. Access_mode- Mode of Access (Read, Write, Append)

iii. Buffering – 0 (no buffer), 1 (buffer)

Example:

```
f= open('abc.txt')    (or)

f=open("D:/Mypython/abc.txt")
```

2.1.1.1 File Access Modes

File Mode	Description
r	Read mode
w	Write mode
x	Create and open a file
a	Appending at end of file
t	Text mode
b	Binary mode
+	Update mode

Example:

```
f= open('abc.txt', r)
```

The above statement opens the file 'abc.txt' in read mode.

2.1.1.2 Example for File Access modes and Properties

```
fo=open('aa.txt','w')
print('Filename: ', fo.name)
print('Filemode: ', fo.mode)
print('File closed: ', fo.closed)
fo.close()
print('Fileclosed: ', fo.closed)


'''
output:
Filename:  aa.txt
Filemode:  w
File closed:  False
Fileclosed:  True
'''
```

The above code is a sample snippet for understanding the file modes and file properties.

2.1.2.File Reading and Writing

2.1.2.1. File write:

write() method is used to write the contents to a file. The following code is for writing the contents to the file aa.txt.

```
fo=open('aa.txt','w')

fo.write('hai \n how are you?')

fo.close()
```

Output:

In the above example, the contents of the file can be viewed by opening the file 'aa.txt'.

2.1.2.2. Reading a file:

read() method is used to read the contents from a file. The following code is for reading the first 10 bytes of the file 'aa.txt'.

```
fo=open('aa.txt','r')
print(fo.read())
#reading 10 bytes
fo.read(10)
fo.close()
'''
output:
Hai
 How are you?

 '''
```

2.1.3. File Positions

To know about the file offset positions in Python, the following methods are used:

- seek()
- tell()

seek():

Syntax: seek(offset, from)

Description: Sets the file's current position at the offset. The offset values are as follows:

0 : reference (beginning of file(default))

1 : current (current file position)

2 : end (end of file)

tell() :

Description: Prints the current position of file pointer.

2.1.3.1.File Offset

'h'	'a'	'i'		,		'h'	'o'	'w'		'a'	'r'	'e'		'y'	'o'	'u'	'?'
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

```
fo=open('aa.txt','r')
print('current position',fo.tell())
print(fo.read(10))
print('current position',fo.tell())
fo.seek(2,0) # to skip first 2
print(fo.read())
'''
output
current position 0
Hai , How
current position 10
i , How are you?
 '''
```

In the above code, initially the position of the file pointer is at 0. After reading the contents, the position of the file pointer is moved to 10 (from 0 to 9). Now up on giving the command seek(2,0), the file will be read from the beginning after skipping the first 2 positions.

Detailed Example:

```
f=open('aa.txt','r')
pos=f.tell()
print(pos)
#output : 0
line=f.readline()
print(line)
#output: prints first line  Hai , How are you?
pos=f.tell()
print(pos)

#20
line=f.readline()
print(line)
#
f.seek(0,0)
pos=f.tell()
print(pos)
line=f.readline()

print(line)
pos=f.tell()
print(pos)
line=f.readline()
print(line)

''' output
0
Hai , How are you?

20

Welcome to Sathyabama

0
Welcome to Sathyabama

43
School of Computing
'''
```

The contents of the file aa.txt is now:

```
Hai , How are you?
Welcome to Sathyabama
School of Computing
Department of Computer Science & Engineering
```

2.1.3.2. Reading a file Line by line

In order to read a file till the End of File(EoF), while loop is used.

```
f=open('f8.txt','r')
line=f.readline()
while line!='':
    print(line)
    line=f.readline()
f.close()
'''output
kdskfa

dsafldk

kdafsljf'''
```

2.1.3.3. Modifying a file

```
f=open('aa.txt','a')
f.write('aa bb cc dd')
f.close()
f=open('aa.txt','r')
print(f.read())
#prints the entire file

#go to 5th position using seek(5)
f.seek(5,0)
print('from 5th posn',f.read())
f.seek(30)
line=f.readline()
#prints posn of line from 30th posn
print('line at 30', line)
f.seek(0)
#print(f.read())#prints full file
print('current posn before reading',f.tell())
f.close()
'''
output
Hai , How are you?
Welcome to Sathyabama
School of Computing
Department of Computer Science & Engineeringaa bb cc ddaa bb cc dd
from 5th posn  How are you?
Welcome to Sathyabama
School of Computing
Department of Computer Science & Engineeringaa bb cc ddaa bb cc dd
line at 30  Sathyabama

current posn before reading 0
'''
```

2.1.4. Alternate way for opening and closing a file:

Syntax:

with open('filename') as file object:

- No need to close the file

```
with open('aa.txt') as f:
    for line in f:
        print(line)


'''output
Hai , How are you?

Welcome to Sathyabama

School of Computing

Department of Computer Science & Engineeringaa bb cc ddaa bb cc dd'''
```

2.1.5. read() &readline()

- read() – read entire file content from current position
- readline() – read the particular line of file pointer

2.2 ITERATORS

Iterator in Python is a type which could be implemented in for loops. An iterator is an object that returns data one at a time.

For example if we have a list A=[1,2,3] , then iterator is used to return the items in the list one at a time.

There are two special Methods:

- __iter__() : returns iterator from list
- __next__(): returns next element in the list

Iterable objects in Python are:

- List
- Tuple
- String

2.2.1. Example Iterator:

```
mylist=[4,7,0,3]
myiter=iter(mylist)
print(next(myiter))
print(next(myiter))
print(next(myiter))
print(next(myiter))
print(next(myiter))
'''
output
4
7
0
3
Error
'''
```

In the above code the list items of mylist object are retrieved one by one using ‘next()’ method. When the list reaches its end and if next() method is used , it shows error in the output.

2.2.2. Example for _ _next_ _()

Alternate way for retrieving the items is to use for loop and retrieve the item using __next__() inside the for loop. To find the length of the list 'len()' method is used.

```
list=[3,4,5,6]
iterobj=iter(list)
print()
for i in range(0,len(list)):
    print(iterobj.__next__())
'''
3
4
5
6
'''
```

2.2.3. Building User defined iterators

We can also build our own iterators. The following code is for implementing user defined iterators for finding powers of two.

```
class pow2:
    #To implement an iterator of powers of two
    def __init__(self,max=0):
        self.max=max
    def __iter__(self):
        self.n=0
        return self
    def __next__(self):
        try:
            if self.n<=self.max:
                res=2**self.n
                self.n+=1
                return res
            else:
                raise StopIteration
        except StopIteration:
            quit(0)
a=pow2(4)
i=iter(a)
print(next(i))
while True:
    print(next(i))
'''output
1
2
4
8
16
'''
```

2.2.4. Python Infinite Iterators:

Ther are two Arguments in infinite iterators:

- Callable Object: A built in function
- Sentinels: The terminating value

The following is an example for infinite iterator. next(inf) will always return 0, since the sentinel 1 not at all reaches.

```
>>> int()
0
>>> inf=iter(int,1)
>>> next(inf)
0
>>> next(inf)
0
```

Similarly , the following code uses while loop to print the odd numbers starting from 1 to infinite number of times. The execution is manually terminated by providing keyboard interrupt(Ctrl+c).

```
class infin:
    def __iter__(self):
        self.num=1
        return self
    def __next__(self):
            num=self.num
            self.num+=2
            return num
a=iter(infin())
while True:
    print(next(a))
 '''output
1
3
5
7
9
11
13
15
17
19...................'''
```

2.2.5. Python Generators

Generator functions are alternates for iterators that contain one or more **yield()** statements. Methods like __iter__(), __next__() are implemented and are iterated using next() automatically. Local variables are remembered between successive calls. When function terminates, StopIterator exception is raised automatically.

2.2.5.1.Example

In the following code, n value is initiated to 1 in the first step. In the second step n is incremented by two and the value yielded now is 3. In the last step n is incremented by 1 and now the value is 4.

```
def my_gen():
    n=1
    print('first')
    yield n
    n+=2
    print('second')
    yield n
    n+=1
    print('last')
    yield n
for i in my_gen():
    print(i)

'''output
first
1
second
3
last
4'''
```

The following is an example for reversing a String using python Generator. Here the string ‘hello’ is passed to the function ‘rev()’. Using for loop, the string is yielded from the last character(len-1) to -1(0^{th} position minus 1) as per the syntax.

```
def rev(mystr):
    len1=len(mystr)
    for i in range(len1-1,-1,-1)
        yield mystr[i]

for c in rev('hello'):
        print(c)
'''output
o
l
l
e
h'''
```

2.2.5.2. Advantages of Generators

- Easy to implement
- Memory efficient
- Represents infinite stream
- Generators can be pipelined

2.3. EXCEPTION HANDLING

Exception is an event that occurs during execution of a Python program disrupting the normal flow of execution. Exceptions are handled using try and except blocks in Python. There are built in exception classes for handling common exceptions. BaseException is the parent class for all built in Exception classes. Fig 2.1 represents the Standard Exception class hierarchy.

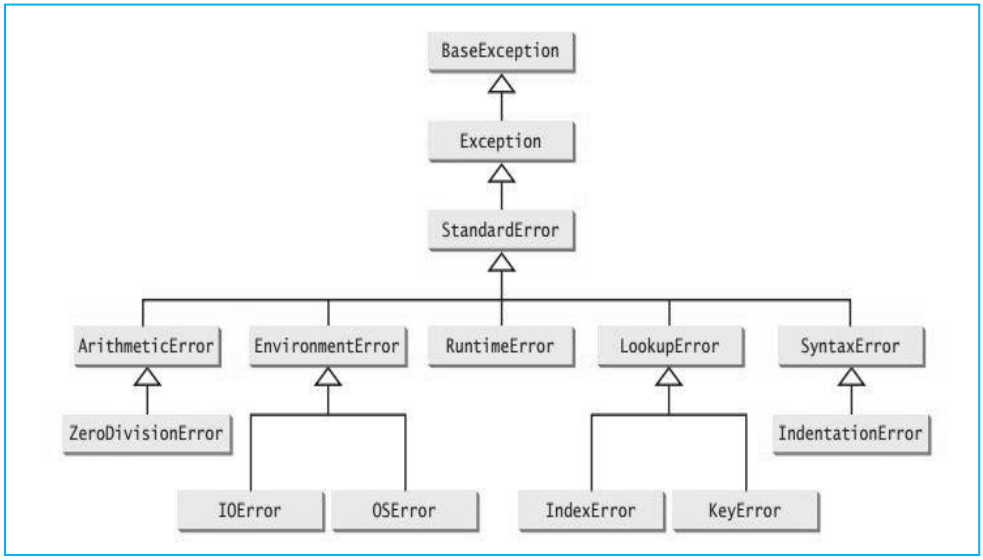


Fig 2.1 Standard Exception class hierarchy

2.3.1. Exception Handling Syntax and Examples

While handling exception, keep the suspicious code in try block and following the try block, include except: statement

```
try:
    suspicious block
except Exception1:
    #statement1
except Exception2:
    #statement2
.....
else:
    no exception
```

The following code raises exception when a run time error occurs upon writing the file 'aa.txt'. In case of normal program flow, the else clause will be invoked and the statements in else block will be executed.

```
try:
    fo=open('aa.txt','w')
    fo.write('Exception for exception')
except IOError:
    print('cant write')
else:
    print('written successfully')


#output:
''' written successfully'''
#content has been written to file aa.txt
```

IOError exception is also invoked when we intend to write a file when it is opened in 'read' mode. The following code depicts this case.

```
try:
    fo=open('aa.txt','r')
    fo.write('Exception handling example')
except IOError:
    print('cant write in read mode')
else:
    print('written successfully')


#output:
''' cant write in read mode'''
```

2.3.1.1. Except Clause without specifying any exception

In python, we can also have except clause with no specific exception. In this case any type of exception can be handled. The following is the syntax for except statement with no specific exception type.

Syntax:

```
try:
    #Error code
except:
    #Execute block with Any exception
else:
   #No exception
```

Example:

In the following code, except clause is alone given, without mentioning the type of exception. In the sample runs when the value of 'b' is given as 0, exception is caught and 'divide by zero error' is printed. Whereas, in case of normal run, the result obtained after dividing two numbers, is displayed as the output.

```
a,b=eval(input('Enter two nos.'))
try:
    c=a/b
except:
    print('divide by zero error')
else:
    print('Normal execution & the value is',c)

'''Sample outputs:
Run1:

Enter two nos.2,0
divide by zero error

Run 2:

Enter two nos.3,6
Normal execution & the value is 0.5
'''
```

2.3.1.2. Except Clause with Multiple exceptions:

There is another way of specifying multiple exceptions in the single except clause. When multiple exceptions are thrown, the first exception which is being caught will alone be handled. The syntax is given as follows.

Syntax:

```
try:
    #Error code
except(Exception 1, Exception2,....):
    #Execute block with Any exception
else:
   #No exception
```

Example:

```
a=input('Enter the value of a')
b =input('Enter the value of b')
try:
    c=a/b
except (TypeError, ZeroDivisionError):
   if TypeError:
       print('Type error')
   elif ZeroDivisionError:
        print('Divide by zero error')

else:
    print('Normal execution & the value is',c)

'''
Sample output:
Enter the value of a : 6
Enter the value of b : a
Type error
'''
```

2.3.1.3 Optional finally clause

Like other object oriented programming languages, try has optional finally clause. The statements given in finally block will be executed even after the exceptions are handled.

```
try:
    f = open("aa.txt",'r')
    f.write('exception handling')
except:
     print('file write exception')
finally:
    f.close()
    print('normal flow')
'''
sample output:
file write exception
normal flow
'''
```

2.3.2. Raising Exceptions

Exception can be raised from a function:
raise ExceptionClass('Something Wrong')

Example:

```
ex=RunTimeError('Something Wrong')

raise ex

        OR

Raise RunTimeError('Something Wrong')
```

```
try:
    a = int(input("Enter a positive integer: "))
    if a<= 0:
        raise ValueError("That is not a positive number!")
except ValueError as er:
    print(er)

'''Sample output:
Enter a positive integer: -7
That is not a positive number!
'''
```

2.3.3. Custom Exception/User Defined Exception

In Python custom exception or otherwise called as user defined exception can be handled by creating a new user defined class which is a derived class from Exception class.

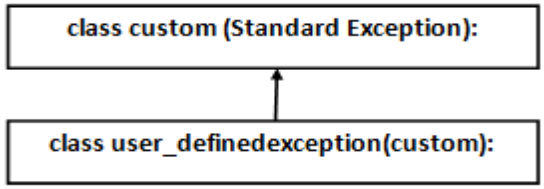


Fig. 2.2: Inheriting the Standard Exception class

In the following example two user defined exception classes are derived from the parent class Error which inherits the standard Exception class. The number guessed is 10. When any number greater than 10 is given as input TooLargeErr exception is thrown and when the number is less than 10, TooSmallErr exception is thrown.

```
class Error(Exception):
    pass
class TooSmallErr(Error):
    pass
class TooLargeErr(Error):
    pass

n=10
while True:
    try:
        x=int(input('enter a number'))
        if x<n:
            raise TooSmallErr
        elif x>n:
            raise TooLargeErr
        break
    except TooSmallErr:
        print('value is small, try again!..')
        print()
    except TooLargeErr:
        print('value is large, try again!..')
        print()
print('Wow! Guess is correct!')


'''output
enter a number23
value is large, try again!..

enter a number1
value is small, try again!..

enter a number10
Wow! Guess is correct!'''
```

2.4 REGULAREXPRESSIONS

Regular Expressions can also be called as RE/regex/regex patterns .RE's are specialized programming languages embedded inside Python. RE's are available by importing **re** module. RE patterns are compiled into a series of bytecodes when executed by a matching engine written in C language. REs could not perform all string

processing tasks. REs are applicable in Pattern recognition problems. RE module has to imported for calling re methods like: split(), findall(), search() etc.

Syntax:

import re

2.4.1 RE matching characters

Character matching is very important for identifying patterns and matching them with the given input. The following table describes some of the important matching characters used in Python REs.

Table: 2.1 Python Character Matching

Matching Character	Description
[ ]	Finding a range of characters [a-z]
\w	Alphanumeric character [a-zA-Z0-9]
\W	Non alpha numeric characters :^ [a-zA-Z0-9]
*	Repeating a character [0] or more times
()	Grouping or including
+	1 or more
?	0 or 1
{x}	Exact number of matches
{a,b}	In range from a to b
\any_number	Matching the group of same number.
\A	Only at the start of the string.

\Z	Only at the end of the string
\b	Empty string only at the beginning or end of a word.
\B	Empty string match not at the beginning or end of a word
\d	[0-9]
\D	^[0-9]
\s	Space
\S	Non space

2.4.2. RE Methods

2.4.2.1. The search() method

Method: search()

Description: Returns true if the search string is found.

Example:

```
import re
m = re.search('info','information')
if m:
    print(m,"is found")
else:
    print('not found')

''' output
<re.Match object; span=(0, 4), match='info'> is found
'''
```

The above code returns the Match object with a span position from 0 to n-1 when the search information is found.

2.4.2.2. The split() method

Method: split()

Description: For creating space in the string.

Example:

```
import re
print(re.split(r'(\s)','This is a string'))
print()
print(re.split(r'[a-i]','This is a string'))
'''
output:
   ['This', ' ', 'is', ' ', 'a', ' ', 'string']

['T', '', 's ', 's ', ' str', 'n', '']
'''
```

In the above code, split() method is applied twice on the string, 'This is a string'. When the matching character \s is applied, the spaces in the string are split up. When the regular expression **r'([a-i])** is applied, the string is split ignoring the range of characters from a to i.

2.4.2.3. The findall () method

Method:findall()

Description: Finds all the matches and returns them as a list of strings.

Example:

```
import re
n='123 1234 12345 636525 1478523'
print(re.findall('\d{5,7}',n))
'''output
returns digits of length from 5 to 7
['12345', '636525', '1478523']
'''
```

2.4.2.4. The match() method

Method:match()

Description: To match the RE pattern to string with optional flags.

Example:

```
import re
list=['csea','cseb','cse a and b']
for e in list:
    z=re.match('(c\w+)',e)
    if z:
        print(z.groups())

'''
Sample output:
The first word of the list items matching the letter c is grouped up
('csea',)
('cseb',)
('cse',)
'''
```

2.4.2.5. The finditer() method

Method: finditer()

Description: Generating an iterator.

Example:

```
import re
str='welcome to cse dept and it dept of Soc'
for i in re.finditer('dept',str):
    localtuple=i.span()
    print(localtuple)

'''output:
returns start index and end index of the string
'dept' which occurs in 2 places:

(15, 19)
(27, 31)
'''
```

2.4.2.6. The compile() method

Method:compile()

Description: Compiling a pattern without rewriting it.

Example:

```
import re
pattern=re.compile('Python')
result=pattern.findall('Welcome to Python programming. Python is Object Oriented.')
print(result)
result2=pattern.findall('Learning Python is Simple')
print(result2)
'''output
['Python', 'Python']
['Python']
'''
```

In the above code the compiled pattern is 'Python'. The result objects return each and every occurrence of the matched pattern line by line. Other Regular Expression methods are given in Table 2.2 and RE Compilation flags are given in Table 2.3.

Table 2.2 Other RE methods

Method/Attribute	Purpose
group()	Returns the string matched by the RE
start()	Returns the starting position of the match
end()	Returns the ending position of the match
span()	Returns a tuple containing the starting and ending positions of the match
sub()	Replaces the RE pattern and returns the modified string

Table 2.3. RE Compilation Flags

Flag	Syntax	Description
ASCII	re.A	Makes several escapes like \w,\b,\s and \d and match only on ASCII characters
DOTALL	re.S	Match any character including newline
IGNORECASE	re.I	Case insensitive matches
MULTILINE	re.M	Multiline matching affecting ^ and $
LOCALE	re.L	Locale aware match(Localization API)
VERBOSE	re.X	Enables verbose RE

Example:

```
import re
list='''csea
nseb
dsea1 and b'''

m1=re.findall(r'^\w',list)
m2=re.findall(r'^\w',list,re.MULTILINE)

print(m1)
print()
print(m2)

'''output
['c']  <-  returns only the first character of first line

['c', 'n', 'd'] <-returns all first characters since it is multiline
'''
```

2.4.3. Case Studies on Pattern Matching:

Case Study 1: Phone number verification

```
import re
ph='412-555-342-4533'
if re.search('\w{3}-\w{3}-\w{3}-\w{4}',ph):
    print('valid phone no')
else:
    print('invalid phone no')
'''
output:
valid phone no'''
```

Case Study 2: Validating First name & Last name

```
import re
name='arthi rathna'
if re.search('\w',name):
    print('valid full name')
else:
    print('invalid name')
'''
output:
valid  full name'''
```

Case Study 3: Email Address Verification

```
import re
n='abc@gmail.com, x3@,@abc.com,az2@abc.in'
print(re.findall('[\w._/+]{1,20}@[\w.-]{2,20}.[A-Za-z]{2,3}',n))
'''output
returns valid emailaddresses:
['abc@gmail.com', 'az2@abc.in']
'''
```

Case Study 4: Web Scrapping

```
import urllib.request
from re import findall
url='http://www.sathyabama.ac.in/sitepagethree.php?mainref=23/'
resp=urllib.request.urlopen(url)
html=resp.read()
htmlstr=html.decode()
pdata=findall('\d{4}\s-\s\d{3}\s-\s\d{4}',htmlstr)
for iterm in pdata:
    print(iterm)

'''
output:
1800 - 425 - 1770
'''
```

2.5 .PYTHON MODULES

2.5.1. Definition

A module is a library of functions used to provide any service. To incorporate the service provided by any module, ‘import’ statement should be used in Python. Modules can be built in or user defined. Modules can be imported in the current program using the import statement.

Syntax:

import*module_name*

Example*:* Time module , Math module

2.5.2.Sample Programs on Built in modules

2.5.2.1. The time module

```
import time
ct_time=time.time()
print(ct_time))

'''Output:
    1559160028.4655905
'''
```

2.5.2.2. The math module

```
import math
print(math.sqrt(9))

'''
output:
3.0
'''
```

2.5.3. Building Custom modules by Modularising functions

Files, containing the Python definitions and statements, can be created by the user, and the same file can be imported on another Python program using import statement. The following example explains importing a python module(File1) over another python code(File 2).

Example:

Let us have two different files File1 & File 2. If we want to import any module of File1 into File2 , then we need to import File1 module in File2 using 'import' statement.

File1.py

```
def max(n1,n2):
    if n1>n2:
        result=n1
    else:
        result=n2
    return result
```

File 2.py

```
import File1
x,y=eval(input('enter x and y'))
z=max(x,y)
print("the max is",z)
```

On running File2.py, we get the maximum of two values as output.

2.6 INTRODUCTION TO PIP

In order to manage and install software packages Python use PIP as Package Management System. PIP is written in Python and available in PyPI(Python Package Index). PIP is otherwise known as PiP Installs Python or PIP installs Packages.

2.6.1. Installing Packages via PIP

2.6.1.1 Steps for installing PIP

Step 1: Download get-pip.py and save this folder in the system's local drive to a folder on your computer.

Step 2: Open the command prompt and explore the folder containing get-pip.py.

Step 3: Run the command: python get-pip.py.

2.6.1.2. Using online python compiler

Python codes can also be executed online without installing Python IDLE or PIP packages. One of the weblink used for running python codes online is: 'https://www.onlinegdb.com/online_python_compiler#'.

2.7. USING PYTHON PACKAGES FOR ADVANCED PROGRAMMING

2.7.1. Python editors for Advanced Python Programming

The following are some of the python editors where Python libraries necessary for advanced scientific programming are almost readily available. If the Python library is not available then the command 'pip install *lib_name*', could be given for installing the specific library.

- JuPYter Notebook
- Pycharm Community Edition & Professional Edition
- Wing IDE
- NINJA IDE
- Spyder
- Pyzo

2.7.2. Python Libraries for running real time projects

2.7.2.1.Numpy

Numpy is a package supporting multidimensional arrays and it is designed for scientific computation purpose. Simple code to create a 3×5 array using numpy is given as follows:

```
import NumPy as np

a = np.arange(15).reshape(3, 5)

print(a)

print ('type of a', type(a))



#Output:

[[ 0  1  2  3  4]
                     [ 5  6  7  8  9]
                                          [10 11 12
13 14]]   type of a <class 'numpy.ndarray'>
```

Table 2.4: Universal Functions in Numpy

Function name	Purpose	Example
np.array()	For creating arrays	a = np.array([0, 1, 2, 3])
np.arrange()	For formatting the array. Start index, end index, step which are the optional attributes.	b = np.arange(1, 9, 2) output: [1,3,5,7]
np.linspace()	For array line spacing with attributes start, end and num-points.	c = np.linspace(0, 1, 6)
np.reshape()	To specify the array dimensions	np.reshape(3,5) : forms a 3* 5 array

Table 2.4: Universal Functions in Numpy(Contd…)

Function name	Purpose	Example
np.sqrt()	Finding square root of an array	d=np.array([[100, 144, 256],[144, 4, 81]]) print(np.sqrt(d)) Output: [[10. 12. 16.] [12. 2. 9.]]
np.exp()	Finding exponential power	np.exp(2)
np.add()	Adding values to an array	np.add(a,10) [[10 11 12 13 14] [20 21 22 23 24]]

2.7.2.2. SciPy

Scipy library is used for performing mathematical and scientific calculations. Scipy can also be used for Engineering applications.

Syntax:

from scipy import module_name

Example:

```
import SciPy

fromscipy.constants import pi

print("sciPy - pi = %.16f"%scipy.constants.pi)



Output:

sciPy - pi = 3.1415926535897931
```

The following are the real time applications which can be implemented using Scipy:

- Signal Processing
- Image manipulation
- Interpolation
- Optimization and fit
- Statistics and random numbers

Files and Exceptions Handling, Modules Packages

- File input/output
- Special Function
- Linear Algebra Operation
- Numerical Integration
- Fast Fourier transforms

2.7.2.2. Matplotlib

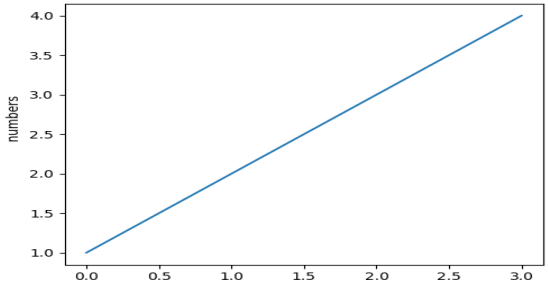
Matplotlib library is used for plotting graphs. The basic methods in matplotlib are:

- Plot()- To plot X, Y axes.
- Show()- To display the plotted graph.

Example:

```
%matplotlib inline

Import matplotlib.pyplot as myplt

myplt.plot([1,2,3,4])

myplt.ylabel('numbers')

myplt.show()
```

Output:



2.8. EXERCISES

1. What is the output of the following code?

```
f1= None
for i in range (5):
    with open("data.txt", "w") as f1:
        if i > 2:
            break
print(f1.closed)
```

2. Write a Python code to read a String, character by character and print the String as a whole using iterators.
3. Write a Python program that matches any string that has an *a* followed by one or more t's.
4. Write a Python program to insert spaces between words starting with capital letters.
5. Write a Python program to remove the parenthesis area in a string using REs.
   Sample data : ["abc (.com)", "w3schools", "google (.com)"]
   Expected Output:
   abc
   w3schools
   google

6. Write a Python program to do a case-insensitive string replacement.

7. Write a Python code to print the given list in reverse order.

8. What is the output of the snippet of code shown below?

```
import numpy as np
a = np.array([[ 1, 2, 3],[4,5,6],[7,8,9]])
print(a[1])
```

9. Write a Python code to append a file 'aa.txt' and then read and display the contents of the file line by line.

10. Check whether the methods today() and now() of datetime library are same or not. Prove the same using a Python code.

REFERENCES:

1. Timothy A.Budd, Exploring Python, Tata McGraw Hill Education Private Limited, New Delhi, 2011.

2. Python basics: https://www.tutorialspoint.com/python , Accessed on May 2019.

3. Y. Daniel Liang, Introduction to Programming Using Python, Pearson, 2013.

4. Python Libraries: http://cs231n.github.io/python-numpy-tutorial/, Accessed on May 2019.

5. Scipy: https://www.guru99.com/scipy-tutorial.html, Accessed on May 2019.

6. Python Exercises: https://www.w3resource.com/python-exercises/re/ , Accessed on May 2019.

7. Python Modules: https://www.sanfoundry.com/python-questions-answers-datetime-module-2/, Accessed on May 2019.

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF BIO AND CHEMICAL ENGINEERING

Common to : Biotech, BioMed, Chemical, EEE

UNIT – III - SCSA1102 - FUNDAMENTALS OF PYTHON PROGRAMMMING

(GUI PROGRAMMING WITH PYTHON)

UNIT III

GUI PROGRAMMING WITH PYTHON

In python text only programs can be created using Command line Interface. Graphical user interface(GUI) can be created using tkinter module in python.

3.1 Introduction To GUI Library In Python

Tkinter is a module in the Python standard library which serves as an interface to Tk (ie) simple *toolkit*. There are many other toolkits also available to create GUI.

Tkinter provides the following widgets:

- button
- canvas
- checkbutton
- combobox
- entry
- frame
- label
- listbox
- menu
- message
- progressbar

- radiobutton
- scrollbar
- spinbox
- text

Tkinter also provides three layout managers:

- place - It positions widgets at absolute locations
- grid - It arranges widgets in a grid
- pack - It packs widgets into a cavity

3.2 Layout Management

The Layout Managers are used to arrange components in a particular manner. It is used to organize the conponents. There are three Layout Management in python:

1. Pack Layout
2. Grid Layout
3. Place Layout

3.2.1 Pack Layout Manager

It is a simple layout manager. Here widgets can be organized in horizontal and vertical boxes. It is used to place each widget next to previous widget. It will be called without any arguments and it will position and size the widgets in a reasonable way. Whenever the user wants to have a series of

widgets in a vertical or horizontal row, the pack layout manager is fairly simple to use. The layout is controlled with the fill, expand, and side options.

Example:

```
from tkinter import *
top=Tk()
l1=Label(top,text="Label1",bg="blue")
l2=Label(top,text="Label2",bg="red" )
l1.pack(fill=X,side=TOP,expand=True)
l2.pack(fill=X,side=RIGHT)
top.mainloop()
```

Output:

The output is shown in Fig 3.1.

Fig 3.1

Explanation: Label l1 has been placed in top position, it is filled in X axis. Label l2 has been placed in Right Position and it is also filled in X axis. Since expand attribute has the value True for Label l1,it can be stretched.

Padding Option in Pack Layout:
The pack() manager has four padding options:

1. Internal Padding
2. External padding
3. Padding in X Direction.
4. Padding in Y Direction.

External Padding in Horizontal direction (padx)

Example:

```
from tkinter import *
top=Tk()
l1=Label(top,text="Label1",bg="blue")
l2=Label(top,text="Label2",bg="red" )
l1.pack(fill=X,side=TOP,expand=True,padx=10)
l2.pack(fill=X,side=TOP,padx=10)
top.mainloop()
```

Output:

The output is shown in Fig 3.2.

Fig 3.2

External Padding in Vertical direction (pady)

Example:

```
from tkinter import *
top=Tk()
l1=Label(top,text="Label1",bg="blue")
l2=Label(top,text="Label2",bg="red" )
l1.pack(fill=X,side=TOP,expand=True,pady=10)
l2.pack(fill=X,side=TOP,pady=10)
top.mainloop()
```

Output:

The output is shown in Fig 3.3.

Fig 3.3

Internal Padding in Horizontal direction(ipadx)

Example:

```
from tkinter import *

top=Tk()

l1=Label(top,text="Label1",bg="blue")

l2=Label(top,text="Label2",bg="red" )

l1.pack(fill=X,side=TOP,expand=True,ipadx=10)

l2.pack(fill=X,side=TOP,ipadx=10)

top.mainloop()
```

Output:

The output is shown in Fig 3.4.

Fig 3.4

Internal Padding in Y Direction(ipady):

Example:

```
from tkinter import *

top=Tk()

l1=Label(top,text="Label1",bg="blue")

l2=Label(top,text="Label2",bg="red" )

l1.pack(fill=X,side=TOP,expand=True,ipadx=10)

l2.pack(fill=X,side=TOP,ipady=10)

top.mainloop()
```

Output:

The output is shown in Fig 3.5.

Fig 3.5

3.2.2 Place Layout:

Place is the most complex manager out of the 3 managers. It uses absolute positioning, when we choose place lay out in our design, then we need to specify the position of the widgets using x and y coordinates. The size and position of the widgets will not be changed when we resize the window.

Example:

```
from tkinter import *

top=Tk()

l1=Label(top,text="Label1",bg="blue")

l2=Label(top,text="Label2",bg="red" )

l1.place(x=10,y=50)

l2.place(x=10,y=100)

top.mainloop()
```

Output:

The output is shown in Fig 3.6.

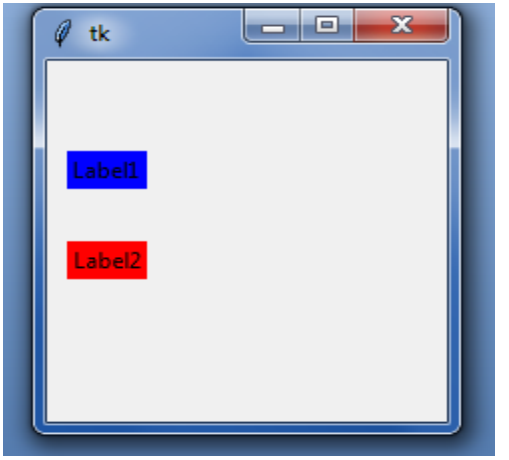


Fig 3.6

Explanation:

Here Label1 is placed in the position (10,50) and label2 is placed in the position (10,100).

3.2.3 Grid Layout

Pack Layout is not easy to understand and it is difficult to change the existing design. By using place layout, we can control the positioning of widgets but it is complex than pack. Grid is one of the most versatile layout manager out of the three layout managers in python. By using Grid layout, the widgets can be placed in rows and columns.

Example:

```
from tkinter import *
top=Tk()
l1=Label(top,text="Label1",bg="blue")
l2=Label(top,text="Label2",bg="red" )
l3=Label(top,text="Label2",bg="green" )
l1.grid(row=0,column=0)
l2.grid(row=0,column=1)
l3.grid(row=1,column=1)
top.mainloop()
```

Output:

The output is shown in Fig 3.7.

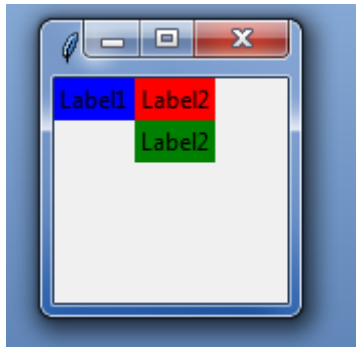


Fig 3.7

Explanation:

Here Label 1 is placed in 0^{th} row and 0^{th} column. Label 2 is placed in 0^{th} row and 1^{st} column and Label 3 is placed in 1^{st} row and 1^{st} column.

3.3 FONT

There are three ways to specify font in python.

1.By using Font Tuple

2.By using Font Object

3.By using XFont

3.3.1 Simple Font Tuple:

Font can be specified using tuple.Herethe font tuple consists of threeelements.First element specifies font family ,second element specifies font size and third element specifies font style.

Ex: t =(“Arial”,14,”Bold”)

Example:

```
from tkinter import *
top=Tk()
b1=Button(text="submit",font=("Arial","16","bold"))
b1.pack()
top.mainloop()
```

Output:

The output is shown in Fig 3.8.

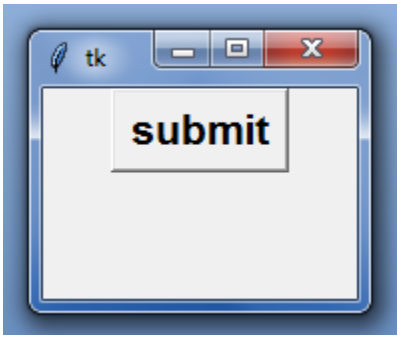


Fig 3.8

Explanation:

Text for the Button has been set in the Arial font with size 16 and Bold style.

3.3.2 Font Object

Font object can be created by importing tkFont module.

Syntax for Font class constructor is:

Import tkFont

Font f1=tkFont.Font(parameters,…..)

Here is the list of parameters:

Family − The font family name as a string.

size − The font height as an integer in points. To get a font n pixels high, use -n.

weight − "bold" for boldface, "normal" for regular weight.

Slant − "italic" for italic, "roman" for unslanted.

underline − 1 for underlined text, 0 for normal.

Overstrike − 1 for overstruck text, 0 for normal

Example:

```
from tkinter import *
from tkFont import *
top=Tk()
f1=Font(family="Helvetica",size=20,weight="bold",slant="italic",underline=1
,overstrike=1)
l1=Label(top,text="Label1",bg="blue",font=f1)
l1.pack()
top.mainloop()
```

3.3.3 X Window Fonts:

If you are running under the X Window System, you can use any of the X font names.

3.4 COLORS

Tkinter represents colors with strings. There are two general ways to specify colors in Tkinter :

- We can use a string specifying the proportion of red, green and blue in hexadecimal digits. For example,
    - "#fff" -- white,
    - "#000000" -- black,
    - "#000fff000" -- pure green
    - "#00ffff" -- pure cyan
- We can also use any locally defined standard following color names.
    - "white"
    - "black"
    - "red"
    - "green"
    - "blue"

- "cyan"
- "yellow"
- "magenta"

The common color options are :

Active background	−	Specifies Background color for the widget when the widget is active.
activeforeground	−	Specifies Foreground color for the widget when the widget is active.
background	−	Specifies Background color for the widget. This can also be represented as bg.
disabledforeground	−	Specifies Foreground color for the widget when the widget is disabled.
foreground	−	Specifies Foreground color for the widget. This can also be represented as fg.
highlightbackground	−	Specifies Background color of the highlight region when the widget has focus.
highlightcolor	−	Specifies Foreground color of the highlight region when the widget has focus.

selectbackground – Specifies Background color for the selected items of the widget.

selectforeground – Specifies Foreground color for the selected items of the widget.

Example:

```
from tkinter import *
top=Tk()
b1=Button(text="submit",bg="red",fg="white")
b1.pack()
top.mainloop()
```

Output:

The output is shown in Fig 3.9.

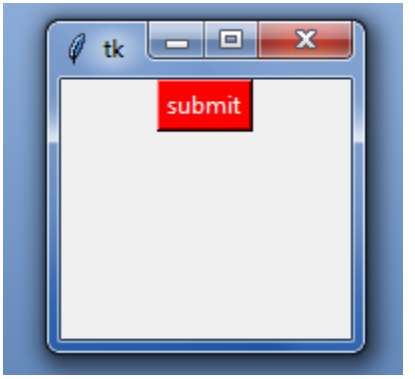


Fig 3.9

Explanation:

Here the back ground of the button is red in color and foreground color of the button is white in colour.

3.5 CANVAS

The Canvas is a rectangular area used for drawing pictures or other complex layouts. Graphics, text, widgets or frames can be placed on a Canvas.

Syntax:

w = Canvas (top, option=value, ...)

top – It represents the parent window.

Options – commonly used options for this widget. These options can be used as key-value pairs separated by commas.

Commonly used Options are:

bd - Border Width of the canvas

bg - Background color of the canvas

cursor - Cursor used in the canvas like circle,arrow and dot.

relief - Type of the border

width - Width of the canvas

Items supported by canvas:

1.Arc

2.Image

3.Line

4.Oval

5.Polygon

3.5.1 ARC

Creates an arc item, which can be a chord or a simple arc.

Syntax:

create_arc(x0, y0, x1, y1, options…..)

x0,y0,x1,y1-Top Left and Bottom Right coordinates of Bounding Rectangle

Commonly used Options:

start,extend-Specifies which section to draw

Example:

```
from tkinter import *
root=Tk()
w = Canvas(root, width=500, height=500)
coord = 10, 50, 240, 210
arc = w.create_arc(coord, start=0, extent=150, fill="blue")
w.pack()
```

Output:

The output is shown in Fig 3.10.

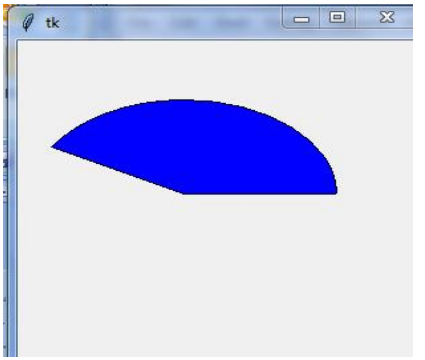


Fig 3.10

Explanation:

Here Arc is drawn with blue color and within the bounded rectangle with top left(10,50)position and bottom right(240,210) position and started from angle 0 and extended till 150 degree.

3.5.2 Image

Creates an image , which can be an instance of either the BitmapImage or the PhotoImage classes.

Syntax:

Create_image(x,y,options….)

x,y-Specifies the position of the image

commonly used options:

anchor=Where to place the image relative to the given position. Default is CENTER.

image=image object

Example:

```
from tkinter import *
root=Tk()
w = Canvas(root, width=500, height=500)
w.create_image("F:\img2",50,50)
w.pack()
root.mainloop()
```

3.5.3 Line

Creates a line item.

Syntax:

canvas.create_line(x0, y0, x1, y1, ...,xn, yn, options)

x0,y0,x1,y1->coordinates of line

Commonly used options:

activefill-Color of the line when it is active

width -Width of the line

Example:

```
from tkinter import *
root=Tk()
w = Canvas(root, width=500, height=500)
w.create_line(10,10,100,100,activefill="red")
w.pack()
root.mainloop()
```

Output:

The output is shown in Fig 3.11.

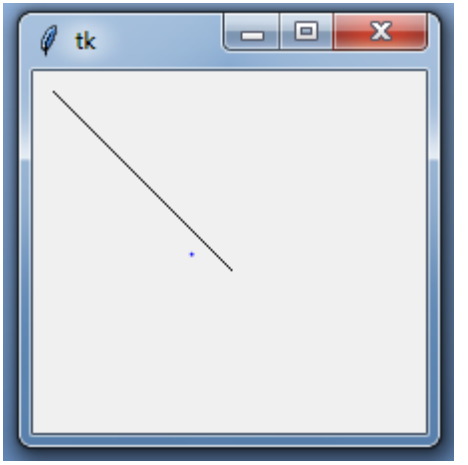


Fig 3.11

3.5.4 OVAL

Creates a circle or an ellipse at the given coordinates. It takes two pairs of coordinates; the top left and bottom right corners of the bounding rectangle for the oval.

Syntax:

canvas.create_oval(x0, y0, x1, y1, options)

x0, y0, x1, y1- the top left and bottom right corners of the bounding rectangle

Options:

activefill-Color of the oval when it is active

width -Width of the line

Example:

```
from tkinter import *
root=Tk()
w = Canvas(root, width=500, height=500)
w.create_oval(10,10,100,100,activefill="red")
w.pack()
root.mainloop()
```

Output:

The output is shown in Fig 3.12.

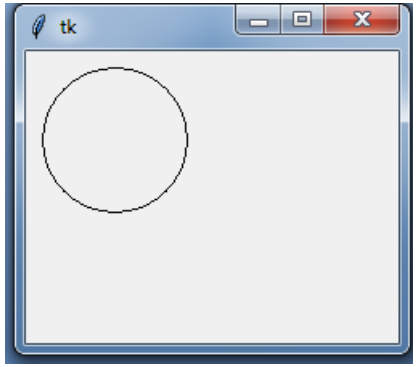


Fig 3.12

3.5.5 Polygon

Creates a polygon item that must have at least three vertices.

Syntax:

canvas.create_polygon(x0, y0, x1, y1,...xn, yn, options)

x0, y0, x1, y1,...xn, yn-Coordinates of polygon

Options:

Activefill-Color of the oval when it is active

width -Width of the line

Example

```
from tkinter import *
root=Tk()
w = Canvas(root, width=500, height=500)
w.create_polygon(50,50,20,20,100,100,activefill="red")
w.pack()
root.mainloop()
```

3.6 WIDGETS IN PYTHON

Widgets are standard graphical user interface (GUI) elements, like different kinds of buttons and menus.

3.6.1 Label

A Label widget shows text to the user about other widgets used in the application. The widget can be updated programmatically.

Syntax to create Label:

w=Label (root ,options)

root - Parent Window

List of commonly used options are given in Table 3.1.

Table 3.1

Option	Description
anchor	It specifies the exact position of the text within the size provided to the widget. The default value is CENTER, which is used to center the text within the specified space.
bg	Specifies background color of the widget
bd	Specifies border width. Default is 2 pixels

cursor	Specifies type of cursor.eg:dot,arrow,circle
font	Specifies font type of the text written inside the widget
fg	Foreground color of the widget
height	Height of the widget
width	Width of the widget
image	Specifies image to be displayed in the widget
padx	Horizontal padding of text
pady	Vertical padding of text
relief	Specifies type of border
text	Text to be displayed in the widget
undeline	Underline the label text

Example:

```
from tkinter import *
root=Tk()
l1=Label(root,text="Enter User Name",bg="green",fg="white")
l1.pack()
root,mainloop()
```

Output:

The output is shown in Fig 3.13.

Fig 3.13

Explanation:

Here Label has been created with green background color and white foreground color with the text “Enter User Name”.

3.6.2 ENTRY

The Entry widget is used to create the single line text-box to the user to accept a value from the user. It can accept the text strings from the user. It can receive one line of text from the user. For multiple lines of text, the text widget will be used.

Syntax for creating Entry Widget:

```
w=Entry(root, options)
```

root-Main Window

List of commonly used options are given in Table 3.2

Table 3.2

Option	Description
bg	Specifies background color of the widget
bd	Specifies border width. Default is 2 pixels
cursor	Specifies type of cursor.eg:dot,arrow,circle
font	Specifies font type of the text written inside the widget
fg	Foreground color of the widget
height	Height of the widget
width	Width of the widget
image	Specifies image to be displayed in the widget
padx	Horizontal padding of text
pady	Vertical padding of text
relief	Specifies type of border
text	Text to be displayed in the widget
undeline	Underline the label text
selectbackground	Background color of the selected text
selectforeground	Foreground color of the selected text
show	Specifies the character used to mask characters in the text box

Example:

```
from tkinter import *
root=Tk()
l1=Label(root,text="Enter User Name",bg="green",fg="white")
e1=Entry(root,show="*")
l1.pack(side=LEFT)
e1.pack(side=RIGHT)
root.mainloop()
```

Output:

The output is shown in Fig 3.14.

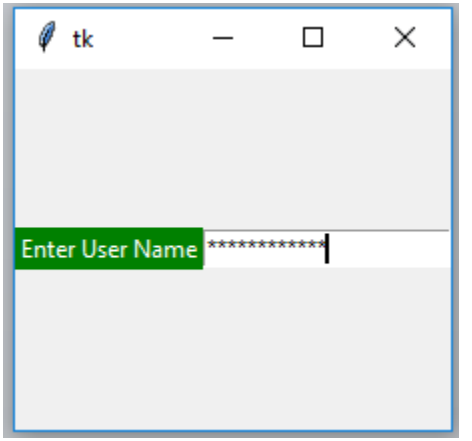


Fig 3.14

Explanation:

Here Label and entry widgets are created.Since the show attribute value is *,the characters entered in the text box appeared as "*".

3.6.3 Button

Button Widget is used to create various kinds of buttons.The user can interact with the button.They can contain text or images.

Syntax for creating Button:

b=Button(root,options)

root-main window

List of commonly used options are given in Table 3.3

Table 3.3

Option	Description
bg	Specifies background color of the widget
bd	Specifies border width. Default is 2 pixels
cursor	Specifies type of cursor.eg:dot,arrow,circle
font	Specifies font type of the text written inside the widget
fg	Foreground color of the widget
height	Height of the widget
width	Width of the widget

Option	Description
image	Specifies image to be displayed in the widget
padx	Horizontal padding of text
pady	Vertical padding of text
relief	Specifies type of border
text	Text to be displayed in the widget
underline	Underline the label text
command	It is set to function name which will be called the button is clicked

Example:

```
from tkinter import *
root=Tk()
b1=Button(root,text="Submit",bg="blue",fg="white")
b1.pack()
root.mainloop()
```

Output:

The output is shown in Fig 3.15.

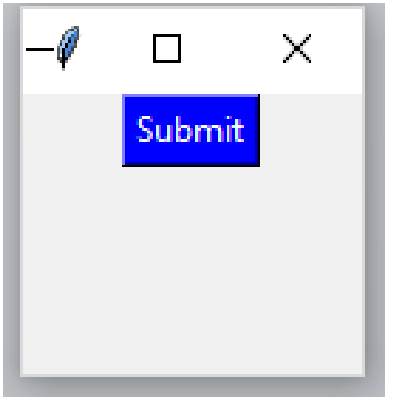


Fig 3.15

3.6.4 Checkbutton

The Checkbutton is used to track the user's choices provided to the application. Checkbutton is used to implement the on/off selections. TheCheckbutton can contain the or images or text. The Checkbutton is mostly used to provide many choices to the user among which, the user needs to choose the one.

Syntax for creating Check Button:

b=CheckButton(root,options)

root-main window

List of commonly used options are given in Table 3.2

Table 3.2

Option	Description
bg	Specifies background color of the widget
bd	Specifies border width. Default is 2 pixels
cursor	Specifies type of cursor.eg:dot,arrow,circle
font	Specifies font type of the text written inside the widget
fg	Foreground color of the widget
height	Height of the widget
width	Width of the widget
image	Specifies image to be displayed in the widget
padx	Horizontal padding of text
pady	Vertical padding of text
relief	Specifies type of border
text	Text to be displayed in the widget
undeline	Underline the label text
command	It is set to function name whicjh will be called the button is clicked
offvalue	Set value to the control variable if the button is checked.Default Value is 1
onvalue	Set value to the control variable if the button is unchecked.Default Value is 0
selectcolor	Set color of the check button when it is checked.
selectimage	Set the image to be shown when it is checked.

Example:

```
from tkinter import *
root=Tk()
c1 = Checkbutton(root, text = "C", onvalue = 1, offvalue = 0, height = 2, width = 10)
c2 = Checkbutton(root, text = "C++", onvalue = 1, offvalue = 0, height = 2, width = 10)
c3 = Checkbutton(root, text = "JAVA", onvalue = 1, offvalue = 0, height = 2, width = 10)
c1.pack()
c2.pack()
c3.pack()
root.mainloop()
```

Output:

The output is shown in Fig 3.16.

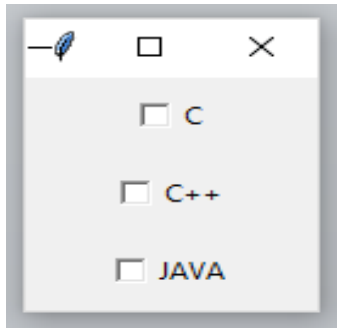


Fig 3.16

3.6.5 Radiobutton

The Radiobutton widget is used to implement one-of-many selection. It shows multiple options to the user out of which, the user can select only one option. It is possible to display the multiple line text or images on the radiobuttons. To keep track the user's selection ,theradiobutton is associated with a single variable.EachRadio button displays a single value for that particular variable.

Syntax for creating Radio Button:

b=RadioButton(root,options)

root-main window

List of commonly used options are given in Table 3.3

Table 3.3

Option	Description
bg	Specifies background color of the widget
bd	Specifies border width. Default is 2 pixels
cursor	Specifies type of cursor.eg:dot,arrow,circle
font	Specifies font type of the text written inside the widget
fg	Foreground color of the widget
height	Height of the widget
width	Width of the widget

Option	Description
image	Specifies image to be displayed in the widget
padx	Horizontal padding of text
pady	Vertical padding of text
relief	Specifies type of border
text	Text to be displayed in the widget
underline	Underline the label text
command	It is set to function name whicjh will be called the button is clicked
value	Set value to the control variable if the button is selected.
selectcolor	Set color of the check button when it is checked.
selectimage	Set the image to be shown when it is checked.
variable	It is used to keep track of user choices.

***Example*:**

```
from tkinter import *
root=Tk()
r1 = Radiobutton(root, text = "C",   value = 1, height = 2, width = 10)
r2 = Radiobutton(root, text = "C++", value = 2, height = 2, width = 10)
r3 = Radiobutton(root, text = "JAVA",value = 3, height = 2, width = 10)
r1.pack()
```

```
r2.pack()
r3.pack()
root.mainloop()
```

Output:

The output is shown in Fig 3.17.

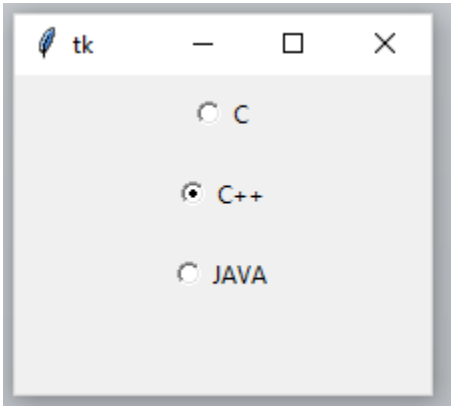


Fig 3.17

3.6.6 Listbox

The Listbox widget is used to display the list items to the user. The user can choose one or more items from the list depending upon the configuration.

Syntax for creatingListBox:

b=Listbox(root,options)

root-main window

List of commonly used options are given in Table 3.4.

Table 3.4

Option	Description
bg	Specifies background color of the widget
bd	Specifies border width. Default is 2 pixels
cursor	Specifies type of cursor.eg:dot,arrow,circle
font	Specifies font type of the text written inside the widget
fg	Foreground color of the widget
height	Height of the widget
width	Width of the widget
image	Specifies image to be displayed in the widget
padx	Horizontal padding of text
pady	Vertical padding of text
relief	Specifies type of border
value	Set value to the control variable if the button is selected.
selectbackground	Set back ground color of the selected text.
xscrollcommand	User can scroll the list box horizontally

yscrollcommand	User can scroll the list box vertically

Example:

```
from tkinter import *
top = Tk()
lbl = Label(top,text = "A list of favourite countries...")
listbox = Listbox(top)
listbox.insert(1,"India")
listbox.insert(2, "USA")
listbox.insert(3, "Japan")
listbox.insert(4, "Austrelia")
lbl.pack()
listbox.pack()
top.mainloop()
```

Output:

The output is shown in Fig 3.18.

Fig 3.18

3.6.7 Message

Its functionality is very similar to Label widget, except that it can automatically wrap the text, maintaining a given width.

Syntax for creating Message:

m=Message(root,options)

root-main window

List of commonly used options are given in Table 3.5.

Table 3.5

Option	Description
bg	Specifies background color of the widget
bd	Specifies border width. Default is 2 pixels
cursor	Specifies type of cursor.eg:dot,arrow,circle
font	Specifies font type of the text written inside the widget
fg	Foreground color of the widget
height	Height of the widget
width	Width of the widget

Option	Description
image	Specifies image to be displayed in the widget
padx	Horizontal padding of text
pady	Vertical padding of text
relief	Specifies type of border

Example:

```
from tkinter import *
top = Tk()
msg = Message( top, text = "Welcome to Javatpoint")
msg.pack()
top.mainloop()
```

Output:

The output is shown in Fig 3.19.

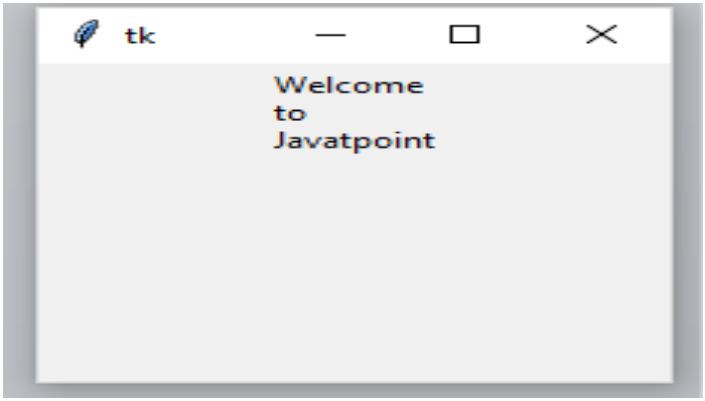


Fig 3.19

3.6.8 Text

Tkinter provides us the Entry widget which is used to implement the single line text box. Text widget provides advanced capabilities that allow us to edit a multiline text and format the way it has to be displayed, such as changing its color and font. We can also use the structures like tabs and marks to locate specific sections of the text, and apply changes to those areas.

Syntax for creating Message:

T=Text(root,options)

root-main window

List of commonly used options are given in Table 3.6.

Table 3.6

Option	**Description**
bg	Specifies background color of the widget
bd	Specifies border width. Default is 2 pixels
cursor	Specifies type of cursor.eg:dot,arrow,circle
font	Specifies font type of the text written inside the widget
fg	Foreground color of the widget
height	Height of the widget
width	Width of the widget
image	Specifies image to be displayed in the widget
padx	Horizontal padding of text
pady	Vertical padding of text
relief	Specifies type of border
xscrollcommand	User can scroll the text widget horizontally
yscrollcommand	User can scroll the text widget vertically
selectbackground	Background color of the selected text

General Methods are given in Table 3.7.

Table 3.7

Method	**Description**

delete(startindex, endindex)	This method is used to delete the characters of the specified range
get(startindex,endindex)	It returns the characters present in the specified range.
insert(index, string)	It is used to insert the specified string at the given index.

Mark Handling Methods :

Marks are used to bookmark the specified position between the characters of the associated text.List of Mark handling methods are given in Table 3.8.

Table 3.8

Method	**Description**
mark_set(mark,index)	It is used to create mark at the specified index.
mark_unset(mark)	It is used to clear the given mark
mark_names()	It is used to return names of all the marks

Tag Handling Methods:

The tags are the names given to the specific areas of the text. The tags are used to configure the different areas of the text separately. The list of tag-handling methods are given in Table 3.9.

Table 3.9

Method	**Description**
tag_add(tagname, startindex, endindex)	It is used to tag the characters in the given range
tag_config()	It is used to configure the tag properties
tag_delete(tagname)	It is used to delete the given tag
tag_remove(tagname, startindex, endindex)	It is used to remove the tag from the specified range

Example:

```
from tkinter import *
top = Tk()
text = Text(top)
text.insert(INSERT, "Name.....")
text.insert(END, "Salary.....")
text.pack()
text.tag_add("Write Here", "1.0", "1.4")
text.tag_add("Click Here", "1.8", "1.13")
text.tag_config("Write Here", background="yellow", foreground="black")
text.tag_config("Click Here", background="black", foreground="white")
```

Output:

The output is shown in Fig 3.20.

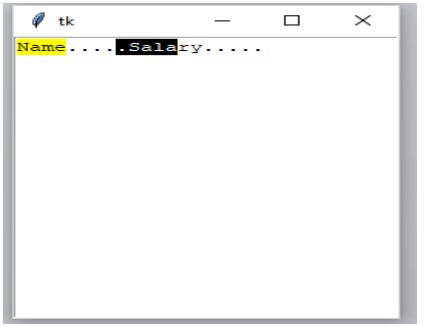


Fig 3.20

Explanation:

The tag “Write Here” tags the characters from the index 0 to 4.The tag “Click Here” tags the characters from the index 8 to 13.These tags are configured using the method tag_config().

3.6.9 SPINBOX

The Spinbox control is an alternative to the Entry control. It provides the range of values to the user, out of which, the user can select only one value.It is used in the case where a user is given some fixed number of values to choose from.

Syntax for creating Message:

S=Spinbox(root,options)

root-main window

List of commonly used options are given in Table 3.10

Table 3.10

Option	Description
bg	Specifies background color of the widget
bd	Specifies border width. Default is 2 pixels
cursor	Specifies type of cursor.eg:dot,arrow,circle
font	Specifies font type of the text written inside the widget
fg	Foreground color of the widget
height	Height of the widget
width	Width of the widget
padx	Horizontal padding of text
pady	Vertical padding of text
relief	Specifies type of border
xscrollcommand	User can scroll the text widget horizontally
from_	It is used to show the starting range of the widget.
to	It specify the maximum limit of the widget value. The other is specified by the from_ option.
values	It represents the tuple containing the values for this widget.

Example:

```
from tkinter import *
top = Tk()
spin = Spinbox(top, from_= 0, to = 25)
spin.pack()
top.mainloop()
```

Output:

The output is shown in Fig 3.21.

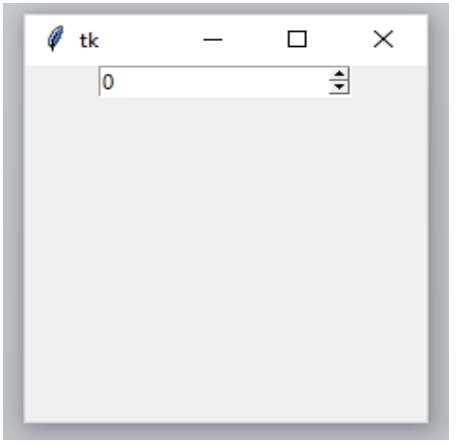


Fig 3.21

3.6.10 Frame

Frame widget is used to organize the group of widgets. It acts like a container which can be used to hold the other widgets. The rectangular areas of the screen are used to organize the widgets to the python application.

Syntax for creating Frame:

S=Frame(root,options)

root-main window

List of commonly used options are given in Table 3.11.

Table 3.11

Option	Description
bg	Specifies background color of the widget
bd	Specifies border width. Default is 2 pixels
cursor	Specifies type of cursor.eg:dot,arrow,circle
height	Height of the widget
width	Width of the widget
Relief	Specifies type of border

Example:

```
from tkinter import *
top = Tk()
Topframe = Frame(top)
Topframe.pack(side = TOP)
Bottomframe = Frame(top)
Bottomframe.pack(side =BOTTOM)
btn1 = Button(Topframe, text="Submit", fg="red",activebackground = "red")
btn1.pack(side = LEFT)
btn2 = Button(Topframe, text="Remove", fg="brown", activebackground = "brown")
btn2.pack(side = RIGHT)
btn3 = Button(Bottomframe, text="Add", fg="blue", activebackground = "blue")
btn3.pack(side = LEFT)
btn4 = Button(Bottomframe, text="Modify", fg="black", activebackground = "white")
btn4.pack(side = RIGHT)
top.mainloop()
```

Output:

The output is shown in Fig 3.23.

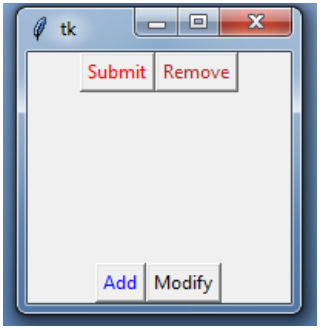


Fig 3.23

Explanation:

Here two frames (Top Frame and Bottom Frame) have been created.Topframe contains submit and remove buttons and Bottom frame contains Add and modify buttons .

3.7 EVENTS AND BINDINGS IN PYTHON

Binding function is used to deal with the events. We can bind Python's Functions and methods to an event as well as we can bind these functions to any particular widget. Events can come from various sources, including key presses and mouse operations by the user. Tkinter provides a powerful mechanism to let you deal with events yourself. For each widget, you can bind Python functions and methods to events.

```
widget.bind(event, handler)
```

If an event matching the event description occurs in the widget, the given handler is called with an object describing the event.

3.7.1 Handling Mouse Button event in Python

Example:

```
from tkinter import *
from tkinter.ttk import *
# creates tkinter window or root window
root = Tk()
# function to be called when button-2 of mouse is pressed
def pressed2(event):
print('Button-2 pressed at x = % d, y = % d'%(event.x, event.y))
# function to be called when button-3 of mouse is pressed
def pressed3(event):
print('Button-3 pressed at x = % d, y = % d'%(event.x, event.y))
```

```
 ## function to be called when button-1 is double clocked
defdouble_click(event):
print('Double clicked at x = % d, y = % d'%(event.x, event.y))
frame1 = Frame(root, height = 100, width = 200)
# Binding mouse  buttons with the Frame widget
frame1.bind('<Button-2>', pressed2)
frame1.bind('<Button-3>', pressed3)
frame1.bind('<Double 1>', double_click)
frame1.pack()
root.mainloop()
```

Output:

The output is shown in Fig 3.24.

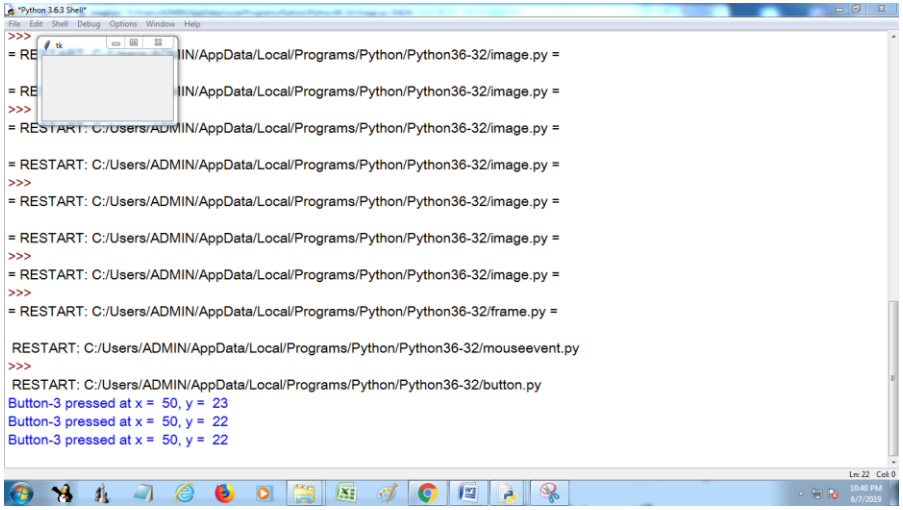


Fig 3.24

3.7.2 Handling Key Press Event in Python

Example:

```
from tkinter import *
from tkinter.ttk import *

# function to be called when
# keyboard buttons are pressed
defkey_press(event):
    key = event.char
    print(key, 'is pressed')
```

```
# creates tkinter window or root window
root = Tk()
root.geometry('200x100')

# here we are binding keyboard
# with the main window
root.bind('<Key>', lambda a : key_press(a))

mainloop()
```

Output:

The output is shown in Fig 3.25.

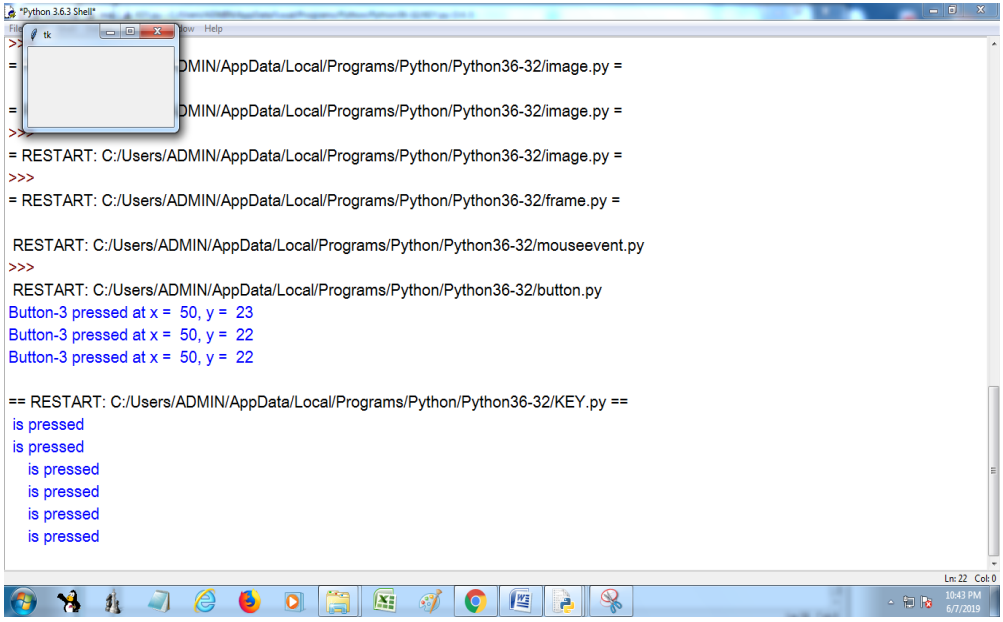
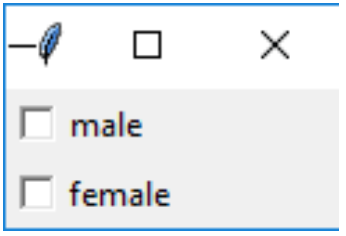

Python 3.6.3 Shell
tk
DMIN/AppData/Local/Programs/Python/Python36-32/image.py =
DMIN/AppData/Local/Programs/Python/Python36-32/image.py =
= RESTART: C:/Users/ADMIN/AppData/Local/Programs/Python/Python36-32/image.py =
>>>
= RESTART: C:/Users/ADMIN/AppData/Local/Programs/Python/Python36-32/frame.py =
RESTART: C:/Users/ADMIN/AppData/Local/Programs/Python/Python36-32/mouseevent.py
>>>
RESTART: C:/Users/ADMIN/AppData/Local/Programs/Python/Python36-32/button.py
Button-3 pressed at x = 50, y = 23
Button-3 pressed at x = 50, y = 22
Button-3 pressed at x = 50, y = 22
== RESTART: C:/Users/ADMIN/AppData/Local/Programs/Python/Python36-32/KEY.py ==
is pressed
is pressed
is pressed
is pressed
is pressed
is pressed
Ln: 22 Col: 0
10:43 PM
6/7/2019

Fig 3.25

QUESTIONS

1. Write the Pyhton Program to create simple window.
2. Write a Python Program to create label, entry and button components and arrange the components using Grid Layout.
3. Write a Python Program to validate user name and password.
4. Write a Python Program to display the basic shapes.
5. Write a Python program to create a following GUI design



6. Write the GUI program to create List Box for shopping cart.
7. Write a pyhton Program to create simple calculator.
8. Write a Python Program to add image on the button.
9. Write a Python progam to create simple application form.
10. Wrtite a Pyhton program to create check button for selecting multiple hobbies.

SCHOOL OF BIO AND CHEMICAL ENGINEERING

Common to : Biotech, BioMed, Chemical, EEE

UNIT – IV- SCSA1102 - FUNDAMENTALS OF PYTHON PROGRAMMMING

(Database and Network)

UNIT IV

DATABASE AND NETWORK

Data is very important for any organization to continue its operations. The data may be related to employees in the organization or the operational data like products information, raw material prices, sales information, profits and losses. Without data, no organization will survive. Hence, data is very important and it should never be lost.

4.1 DATABASE MANAGEMENT SYSTEM (DBMS)

To store data, a file or database can be used. A file stores data in the secondary storage device like hard disk, either in the text format or binary format.

A database represents collection of data. Data is stored in the database. Once the data is stored in the database, various operations can be performed on the data. For example, modifying the existing data, deleting the unwanted data, or retrieving the data from the database and etc. To perform such operations, a database comes with software. This is called a database management system.

DBMS= Database + Software to manage the data

Example DBMS are MySQL, Oracle, Sybase,, SQL server etc.

Types of databases used with Python

1. **Database support**
    - SQL
    - NoSQL

Database and Network

As more and more data become available as unstructured or semi-structured, the need of managing them through NoSql database increases. Python can also interact with NoSQL databases in a similar way as it interacts with Relational databases. In this chapter we will use python to interact with MongoDB as a NoSQL database.

4.2 MONGO DB

MongoDB stores data in JSON-like documents, which makes the database very flexible and scalable.

Where to Use MongoDB?

- Big Data
- Content Management and Delivery
- Mobile and Social Infrastructure
- User Data Management
- Data Hub

download a free MongoDB database at https://www.mongodb.com.

4.2.1 PyMongo

Python needs a MongoDB driver to access the MongoDB database.

In this tutorial we will use the MongoDB driver "PyMongo".

We recommend that you use PIP to install "PyMongo".

PIP is most likely already installed in your Python environment.

Navigate your command line to the location of PIP, and type the following:

Database and Network

Download and install "PyMongo":

C:\Users*Your Name*\AppData\Local\Programs\Python\Python36-32\Scripts>python -m pip install pymongo

Now you have downloaded and installed a mongoDB driver.

Where to Use MongoDB?

- Big Data
- Content Management and Delivery
- Mobile and Social Infrastructure
- User Data Management
- Data Hub

Test PyMongo

To test if the installation was successful, or if you already have "pymongo" installed, create a Python page with the following content:

demo_mongodb_test.py:

import pymongo

Creating a Database

To create a database in MongoDB, start by creating a MongoClient object, then specify a connection URL with the correct ip address and the name of the database you want to create.

Database and Network

MongoDB will create the database if it does not exist, and make a connection to it.

Example

Create a database called mydatabase

Program

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")

mydb = myclient["mydatabase"]
```

MongoDB waits until you have created a collection (table), with at least one document (record) before it actually creates the database (and collection).

4.3 CREATING A COLLECTION

To create a collection in MongoDB, use database object and specify the name of the collection you want to create.

MongoDB will create the collection if it does not exist.

Program

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]
```

MongoDB waits until you have inserted a document before it actually creates the collection.

4.3.1 Python MongoDB Insert Document

Insert Into Collection

To insert a record, or *document* as it is called in MongoDB, into a collection, we use the insert_one() method.

The first parameter of the insert_one() method is a dictionary containing the name(s) and value(s) of each field in the document you want to insert.

Example

Insert a record in the “Customers” Collection:

```
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]
```

mydict = { "name": "John", "address": "Highway 37" }
x = mycol.insert_one(mydict)

Insert Multiple Documents

To insert multiple documents into a collection in MongoDB, we use theinsert_many() method.

The first parameter of the insert_many() method is a list containing dictionaries with the data you want to insert:

```
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

mylist = [
  { "name": "Amy", "address": "Apple st 652"},
  { "name": "Hannah", "address": "Mountain 21"},
  { "name": "Michael", "address": "Valley 345"},
  { "name": "Sandy", "address": "Ocean blvd 2"},
  { "name": "Betty", "address": "Green Grass 1"},
  { "name": "Richard", "address": "Sky st 331"},
  { "name": "Susan", "address": "One way 98"},
  { "name": "Vicky", "address": "Yellow Garden 2"},
  { "name": "Ben", "address": "Park Lane 38"},
  { "name": "William", "address": "Central st 954"},
```

```
  { "name": "Chuck", "address": "Main Road 989"},
  { "name": "Viola", "address": "Sideway 1633"}
]

x = mycol.insert_many(mylist)
```

4.3.2 Python MongoDB Find

In MongoDB we use the **find** and **findOne** methods to find data in a collection.

Just like the **SELECT** statement is used to find data in a table in a MySQL database.

Find One

To select data from a collection in MongoDB, we can use the find_one()method.

The find_one() method returns the first occurrence in the selection.

Example

Find the first document in the customers collection:

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

x = mycol.find_one()
```

```
print(x)
```

Output

{'_id': 1, 'name': 'John', 'address': 'Highway37'}

Find All

To select data from a table in MongoDB, we can also use the find() method.

The find() method returns all occurrences in the selection.

The first parameter of the find() method is a query object. In this example we use an empty query object, which selects all documents in the collection.

Example

Return all documents in the "customers" collection, and print each document:

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

for x in mycol.find():

print(x)
```

{'_id': 1, 'name': 'John', 'address': 'Highway37'}
{'_id': 2, 'name': 'Peter', 'address': 'Lowstreet 27'}
{'_id': 3, 'name': 'Amy', 'address': 'Apple st 652'}
{'_id': 4, 'name': 'Hannah', 'address': 'Mountain 21'}
{'_id': 5, 'name': 'Michael', 'address': 'Valley 345'}
{'_id': 6, 'name': 'Sandy', 'address': 'Ocean blvd 2'}
{'_id': 7, 'name': 'Betty', 'address': 'Green Grass 1'}
{'_id': 8, 'name': 'Richard', 'address': 'Sky st 331'}
{'_id': 9, 'name': 'Susan', 'address': 'One way 98'}
{'_id': 10, 'name': 'Vicky', 'address': 'Yellow Garden 2'}
{'_id': 11, 'name': 'Ben', 'address': 'Park Lane 38'}
{'_id': 12, 'name': 'William', 'address': 'Central st 954'}
{'_id': 13, 'name': 'Chuck', 'address': 'Main Road 989'}
{'_id': 14, 'name': 'Viola', 'address': 'Sideway 1633'}

4.3.3. Filter the Result

When finding documents in a collection, you can filter the result by using a query object.

The first argument of the find() method is a query object, and is used to limit the search.

Example

Find document(s) with the address "Park Lane 38":

```
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]
myquery = { "address": "Park Lane 38" }
mydoc = mycol.find(myquery)
```

```
for x in mydoc:
  print(x)
```

output

{'_id': 11, 'name': 'Ben', 'address': 'Park Lane 38'}

Example

Find documents where the address starts with the letter "S" or higher:

```
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]
myquery = { "address": { "$gt": "S" } }
mydoc = mycol.find(myquery)
for x in mydoc:
  print(x)
```

Output

```
{'_id': 5, 'name': 'Michael', 'address': 'Valley 345'}
{'_id': 8, 'name': 'Richard', 'address': 'Sky st 331'}
{'_id': 10, 'name': 'Vicky', 'address': 'Yellow
Garden 2'}
{'_id': 14, 'name': 'Viola', 'address': 'Sideway
1633'}
```

Return Only Some Fields

The second parameter of the find() method is an object describing which fields to include in the result.

This parameter is optional, and if omitted, all fields will be included in the result.

Example

Return only the names and addresses, not the _ids:

```
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]

mycol = mydb["customers"]


for x in mycol.find({},{ "_id": 0, "name": 1, "address": 1}):
  print(x)
```

Output

```
{'name': 'John', 'address': 'Highway37'}
{'name': 'Peter', 'address': 'Lowstreet 27'}
{'name': 'Amy', 'address': 'Apple st 652'}
{'name': 'Hannah', 'address': 'Mountain 21'}
{'name': 'Michael', 'address': 'Valley 345'}
{'name': 'Sandy', 'address': 'Ocean blvd 2'}
{'name': 'Betty', 'address': 'Green Grass 1'}
{'name': 'Richard', 'address': 'Sky st 331'}
{'name': 'Susan', 'address': 'One way 98'}
```

```
{'name': 'Vicky', 'address': 'Yellow Garden 2'}
{'name': 'Ben', 'address': 'Park Lane 38'}
{'name': 'William', 'address': 'Central st 954'}
{'name': 'Chuck', 'address': 'Main Road 989'}
{'name': 'Viola', 'address': 'Sideway 1633'}
```

4.3.4. Sort the Result

Use the sort() method to sort the result in ascending or descending order.

The sort() method takes one parameter for "fieldname" and one parameter for "direction" (ascending is the default direction).

Example

Sort the result alphabetically by name:

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

mydoc = mycol.find().sort("name")

for x in mydoc:
  print(x)
```

OUTPUT

```
{'_id': 3, 'name': 'Amy', 'address': 'Apple st 652'}
{'_id': 11, 'name': 'Ben', 'address': 'Park Lane 38'}
{'_id': 7, 'name': 'Betty', 'address': 'Green Grass 1'}
{'_id': 13, 'name': 'Chuck', 'address': 'Main Road 989'}
{'_id': 4, 'name': 'Hannah', 'address': 'Mountain 21'}
{'_id': 1, 'name': 'John', 'address': 'Highway37'}
{'_id': 5, 'name': 'Michael', 'address': 'Valley 345'}
{'_id': 2, 'name': 'Peter', 'address': 'Lowstreet 27'}
{'_id': 8, 'name': 'Richard', 'address': 'Sky st 331'}
{'_id': 6, 'name': 'Sandy', 'address': 'Ocean blvd 2'}
{'_id': 9, 'name': 'Susan', 'address': 'One way 98'}
{'_id': 10, 'name': 'Vicky', 'address': 'Yellow Garden 2'}
{'_id': 14, 'name': 'Viola', 'address': 'Sideway 1633'}
{'_id': 12, 'name': 'William', 'address': 'Central st 954'}
```

Sort Descending

Use the value -1 as the second parameter to sort descending.

```
sort("name", 1) #ascending
sort("name", -1) #descending
```

Example

Sort the result reverse alphabetically by name:

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]
```

```
mydoc = mycol.find().sort("name", -1)

for x in mydoc:
  print(x)
```

Output

```
{'_id': 12, 'name': 'William', 'address': 'Central st 954'}
{'_id': 14, 'name': 'Viola', 'address': 'Sideway 1633'}
{'_id': 10, 'name': 'Vicky', 'address': 'Yellow Garden 2'}
{'_id': 9, 'name': 'Susan', 'address': 'One way 98'}
{'_id': 6, 'name': 'Sandy', 'address': 'Ocean blvd 2'}
{'_id': 8, 'name': 'Richard', 'address': 'Sky st 331'}
{'_id': 2, 'name': 'Peter', 'address': 'Lowstreet 27'}
{'_id': 5, 'name': 'Michael', 'address': 'Valley 345'}
{'_id': 1, 'name': 'John', 'address': 'Highway37'}
{'_id': 4, 'name': 'Hannah', 'address': 'Mountain 21'}
{'_id': 13, 'name': 'Chuck', 'address': 'Main Road 989'}
{'_id': 7, 'name': 'Betty', 'address': 'Green Grass 1'}
{'_id': 11, 'name': 'Ben', 'address': 'Park Lane 38'}
{'_id': 3, 'name': 'Amy', 'address': 'Apple st 652'}
```

4.3.5 Python MongoDB Delete Document

To delete one document, we use the delete_one() method.

The first parameter of the delete_one() method is a query object defining which document to delete.

Note: If the query finds more than one document, only the first occurrence is deleted.

Example

Delete the document with the address "Mountain 21":

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

myquery = { "address": "Mountain 21" }

mycol.delete_one(myquery)
```

Delete Many Documents

To delete more than one document, use the delete_many() method.

The first parameter of the delete_many() method is a query object defining which documents to delete.

Example

Delete all documents were the address starts with the letter S:

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]
```

```
myquery = { "address": {"$regex": "^S"} }

x = mycol.delete_many(myquery)

print(x.deleted_count, " documents deleted.")
```

output

```
2 documents deleted.
```

Delete All Documents in a Collection

To delete all documents in a collection, pass an empty query object to the delete_many() method:

Example

Delete all documents in the "customers" collection:

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

x = mycol.delete_many({})

print(x.deleted_count, " documents deleted.")
```

Output:

```
11 documents deleted
```

4.3.6 Python MongoDB Drop Collection

Delete Collection

You can delete a table, or collection as it is called in MongoDB, by using the drop() method.

Example

Delete the "customers" collection:

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

mycol.drop()
```

The drop() method returns true if the collection was dropped successfully, and false if the collection does not exist.

4.3.7 Python MongoDB Update

You can update a record, or document as it is called in MongoDB, by using the update_one() method.

The first parameter of the update_one() method is a query object defining which document to update.

Note: If the query finds more than one record, only the first occurrence is updated.

Example

Change the address from "Valley 345" to "Canyon 123":

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

myquery = { "address": "Valley 345" }
newvalues = { "$set": { "address": "Canyon 123" } }

mycol.update_one(myquery, newvalues)

#print "customers" after the update:
for x in mycol.find():
  print(x)
```

OUTPUT

```
{'_id': 1, 'name': 'John', 'address': 'Highway37'}
{'_id': 2, 'name': 'Peter', 'address': 'Lowstreet 27'}
{'_id': 3, 'name': 'Amy', 'address': 'Apple st 652'}
{'_id': 4, 'name': 'Hannah', 'address': 'Mountain 21'}
{'_id': 5, 'name': 'Michael', 'address': 'Canyon 123'}
{'_id': 6, 'name': 'Sandy', 'address': 'Ocean blvd 2'}
{'_id': 7, 'name': 'Betty', 'address': 'Green Grass 1'}
{'_id': 8, 'name': 'Richard', 'address': 'Sky st 331'}
{'_id': 9, 'name': 'Susan', 'address': 'One way 98'}
{'_id': 10, 'name': 'Vicky', 'address': 'Yellow Garden 2'}
{'_id': 11, 'name': 'Ben', 'address': 'Park Lane 38'}
{'_id': 12, 'name': 'William', 'address': 'Central st 954'}
{'_id': 13, 'name': 'Chuck', 'address': 'Main Road 989'}
{'_id': 14, 'name': 'Viola', 'address': 'Sideway'}
```

Update Many

To update *all* documents that meets the criteria of the query, use the update_many() method.

Example

Update all documents where the address starts with the letter "S":

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]
```

```
myquery = { "address": { "$regex": "^S" } }
newvalues = { "$set": { "name": "Minnie" } }

x = mycol.update_many(myquery, newvalues)

print(x.modified_count, "documents updated.")
```

Output

```
2 documents updated.
```

4.3.8 Python MongoDB Limit

To limit the result in MongoDB, we use the limit() method.

The limit() method takes one parameter, a number defining how many documents to return.

Consider you have a "customers" collection:

```
{'_id': 1, 'name': 'John', 'address': 'Highway37'}
{'_id': 2, 'name': 'Peter', 'address': 'Lowstreet 27'}
{'_id': 3, 'name': 'Amy', 'address': 'Apple st 652'}
{'_id': 4, 'name': 'Hannah', 'address': 'Mountain 21'}
{'_id': 5, 'name': 'Michael', 'address': 'Valley 345'}
{'_id': 6, 'name': 'Sandy', 'address': 'Ocean blvd 2'}
{'_id': 7, 'name': 'Betty', 'address': 'Green Grass 1'}
{'_id': 8, 'name': 'Richard', 'address': 'Sky st 331'}
{'_id': 9, 'name': 'Susan', 'address': 'One way 98'}
{'_id': 10, 'name': 'Vicky', 'address': 'Yellow Garden 2'}
{'_id': 11, 'name': 'Ben', 'address': 'Park Lane 38'}
```

```
{'_id': 12, 'name': 'William', 'address': 'Central st 954'}
{'_id': 13, 'name': 'Chuck', 'address': 'Main Road 989'}
{'_id': 14, 'name': 'Viola', 'address': 'Sideway'}
```

Example

Limit the result to only return 5 documents:

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

myresult = mycol.find().limit(5)

#print the result:
for x in myresult:
  print(x)
```

OUTPUT

```
{'_id': 1, 'name': 'John', 'address': 'Highway37'}
{'_id': 2, 'name': 'Peter', 'address': 'Lowstreet 27'}
{'_id': 3, 'name': 'Amy', 'address': 'Apple st 652'}
{'_id': 4, 'name': 'Hannah', 'address': 'Mountain 21'}
{'_id': 5, 'name': 'Michael', 'address': 'Valley 345'}
```

4.4 CURSOR CLASS

To work with MySQL in python, connector sub module of mysql module.

```
import mysql.connector;
```

to establish connection with MySQL database, we use the connect() method of mysql.connector module as:

conn=mysql.connector.connect(host='localhost',database='university',user='root', password='***')

The connect() method returns MySQLConnection class object 'conn'.

The next step is to create cursor class object by calling the cursor() method on 'conn' object as:

```
cursor=con.cursor()
```

Cursor object is useful to execute SQL commands on the database.

it is done by execute() method of cursor object.

```
cursor.execute( sql querry)
```

example: cursor.execute("select * from emptab")

The resultant rows retirieved from the table are stored in cursor object. the result can be fetched using fetchone() or fetchall() methods.

```
example:    row = cursor.fetchone()  # get 1 row
```

```
row = cursor.fetchall()   # get all rows
```

Finally, the connection with MySQL can be closed by closing the cursor and connection objects as:

```
cursor.close()

conn.close()
```

Program: A python program to retrieve and display all rows from the student table:

```
import mysql.connector;

conn=mysql.connector.connect(host='localhost',database='university',user='root',
password='***')

cursor=con.cursor()

cursor.execute("select * from stutab")

row = cursor.fetchone()

while row is not None:

    print(row)

  row=cursor.fetchone()

cursor.close()

conn.close()
```

Output:

(1001, ‘Ajay’, 8.5)

(1002, ‘Alan’, 7.5)

(1001, ‘Joe’, 9.00)

4.5 EXCEPTIONS CLASSES

Interacting with a database is an error prone process, so we must always implement some mechanism to handle errors.

Built in Exceptions

Exception	**Description**
Warning	Used for non-fatal issues. Must subclass StandardError.
Error	Base class for errors. Must subclass StandardError.
InterfaceError	Used for errors in the database module, not the database itself. Must subclass Error.
DatabaseError	Used for errors in the database. Must subclass Error.
DataError	Subclass of DatabaseError that refers to errors in the data.
OperationalError	Subclass of DatabaseError that refers to errors such as the loss of a connection to the database. These errors are generally outside of the control of the Python scripter.
Exception	Description

IntegrityError	Subclass of DatabaseError for situations that would damage the relational integrity, such as uniqueness constraints or foreign keys.
InternalError	Subclass of DatabaseError that refers to errors internal to the database module, such as a cursor no longer being active.
ProgrammingError	Subclass of DatabaseError that refers to errors such as a bad table name and other things that can safely be blamed on you.

4.6 NETWORKING

For a specific purpose if things are connected together, are referred as a NETWORK. A network can be of many types, like a telephone network, television network, computer network or even a people network.

Similarly, a COMPUTER NETWORK is also a kind of setup, where it connects two or more devices to share a range of services and information in the form of e-mails and messages, databases, documents, web-sites, audios and videoes, Telephone calls and video conferences etc among them.

A PROTOCOL is nothing but set of defined rules, which has to be followed by every connected devices across a network to communicate and share information among them. To facilitates End to End communication, a number of protocols worked together to form a Protocol Suites or Stacks.

Some basic Protocols are:

- IP : Internet Protocol
- FTP : File Transfer Protocol
- SMTP : Simple Mail Transfer Protocol
- HTTP : Hyper Text Transfer Protocol

The Network reference models were developed to allow products from different manufacturers to interoperate on a network. A network reference model serves as a blueprint, detailing standards for how protocol communication should occur. The most widely recognized reference models are, the Open Systems Interconnect (OSI) Model and Department of Defense (DoD, also known as TCP/IP) model.

Network Types are often categorized by their size and functionality. According to the size, the network can be commonly categorized into Three types.

- **LANs (Local Area Networks)**
- **MANs (Metropolitan Area Networks)**
- **WANs (Wide Area Networks)**

An **Internetwork** is a general term describing multiple networks connected together. The Internet is the largest and most well-known internetwork.

Some networks are categorized by their function, as opposed to their size.

For example:

- **SAN (Storage Area Network)**: A SAN provides systems with high-speed, lossless access to high-capacity storage devices.

- **VPN (Virtual Private Network)**: A VPN allows for information to be securely sent across a public or unsecure network, such as the Internet. Common uses of a VPN are to connect branch offices or remote users to a main office.

In a network, any connected device is called as ***host***. A host can serve as following ways:

- A host can acts as a ***Client***, when he is requesting information.
- A host can acts as a ***Server***, when he provides information.
- A host can also request and provide information, is called ***Peer***.

4.7 SOCKET MODULE

What Are Sockets?

A socket is a link between two applications that can communicate with one another (either locally on a single machine or remotely between two machines in separate locations).

Basically, sockets act as a communication link between two entities, i.e. a server and a client. A server will give out information being requested by a client. For example, when you visited this page, the browser created a socket and connected to the server.

The socket Module

In order to create a socket, you use the socket.socket() function, and the syntax is as simple as:

```
import socket

s= socket.socket (socket_family, socket_type, protocol=0)
```

Here is the description of the arguments:

- **socket_family**: Represents the address (and protocol) family. It can be either AF_UNIX or AF_INET.
- **socket_type**: Represents the socket type, and can be either SOCK_STREAM or SOCK_DGRAM.
- **protocol**: This is an optional argument, and it usually defaults to 0.

After obtaining your socket object, you can then create a server or client as desired using the methods available in the socket module.

- s.recv() –It receives TCPmessage
- s.send() – It transmits TCP message
- s.recvfrom() – It receives UDPmessage
- s.sendto() – It transmits UDP message
- s.close() – It closes socket
- socket.gethostname() – It returns thehostname

4.8 CREATE A SIMPLE CLIENT

Before we get started, let's look at the client socket methods available in Python.

s= socket.socket(socket.AF_INET, socket.sock_STREAM)

s.connect()Initiates a TCP server connection.

To create a new socket, you first import the socket method of the socket class.

import socket

Next, we'll create a stream (TCP) socket as follows:

stream_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

The AF_INET argument indicates that you're requesting an Internet Protocol (IP) socket, specifically IPv4. The second argument is the transport protocol type SOCK_STREAM for TCP sockets. Additionally, you can also create an IPv6 socket by specifying the socket AF_INET6 argument.

Specify the server.

server = "localhost"

Specify the port we want to communicate with.

port =80

Connect the socket to the port where the server is listening.

server_address = ((host, port))

```
stream_socket.connect(server_address)
```

It's important to note that the host and port must be a tuple.

Send a data request to the server:

```
message = 'message'

stream_socket.sendall(message)
```

Get the response from the server:

```
data = sock.recv(10)

print data
```

To close a connected socket, you use the close method:

```
stream_socket.close()
```

Below is the full code for the Client/Server.

```
import socket

import sys

 # Create a TCP/IP socket

stream_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

 # Define host

host = 'localhost'
```

```
 # define the communication port

port = 8080

 # Connect the socket to the port where the server is listening

server_address = ((host, port))

print "connecting"

stream_socket.connect(server_address)

 # Send data

message = 'message'

stream_socket.sendall(message)

# response

data = stream_socket.recv(10)

print data

print 'socket closed'

stream_socket.close()
```

4.9 BUILD A SIMPLE SERVER

Now let's take a look at a simple Python server. The following are the socket server methods available in Python.

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

s.bind(): Binds address (hostname, port number) to socket.

s.listen(): Sets up and starts TCP listener.

s.accept(): Accepts TCP client connection.

We will follow the following steps:

- Create a socket.
- Bind the socket to a port.
- Start accepting connections on the socket.

Here is the server program.

```
import socket

import sys

# Create a TCP/IP socket

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

 # Define host

host = 'localhost'

 # define the communication port

port = 8080
```

```
 # Bind the socket to the port
sock.bind((host, port))
# Listen for incoming connections
sock.listen(1)
# Wait for a connection
print 'waiting for a connection'
connection, client = sock.accept()
print client, 'connected'
# Receive the data in small chunks and retransmit it
data = connection.recv(16)
print 'received "%s"' % data
if data:
    connection.sendall(data)
else:
    print 'no data from', client
 # Close the connection
connection.close()
```

The server is now ready for incoming connections.

Now run the client and server programs in separate terminal windows, so they can communicate with each other.

Server Output

```
$ python server.py
waiting for a connection
('127.0.0.1', 47050) connected
received "message"
```

Client Output

```
$ python client.py
connecting
message
socket closed
```

4.10 SENDING EMAIL USING SMTP

Simple Mail Transfer Protocol (SMTP) is a protocol, which handles sending e-mail and routing e-mail between mail servers.

Python provides **smtplib** module, which defines an SMTP client session object that can be used to send mail to any Internet machine with an SMTP or ESMTP listener daemon.

Here is a simple syntax to create one SMTP object, which can later be used to send an e-mail –

```
import smtplib

smtpObj = smtplib.SMTP( [host [, port [, local_hostname]]] )
```

Here is the detail of the parameters −

- **host** − This is the host running your SMTP server. You can specify IP address of the host or a domain name like tutorialspoint.com. This is optional argument.
- **port** − If you are providing *host* argument, then you need to specify a port, where SMTP server is listening. Usually this port would be 25.
- **local_hostname** − If your SMTP server is running on your local machine, then you can specify just *localhost* as of this option.

An SMTP object has an instance method called **sendmail**, which is typically used to do the work of mailing a message. It takes three parameters −

- The *sender* − A string with the address of the sender.
- The *receivers* − A list of strings, one for each recipient.

- The *message* − A message as a string formatted as specified in the various RFCs.

Example

Here is a simple way to send one e-mail using Python script. Try it once −

```
import smtplib

sender = 'from@fromdomain.com'

receivers = ['to@todomain.com']

message = """From: From Person from@fromdomain.com

To: To Person to@todomain.com

Subject: SMTP e-mail test

This is a test e-mail message.

"""

try:

   smtpObj = smtplib.SMTP('localhost')

   smtpObj.sendmail(sender, receivers, message)

   print "Successfully sent email"

except SMTPException:
```

Here, you have placed a basic e-mail in message, using a triple quote, taking care to format the headers correctly. An e-mail requires a **From**, **To**, and **Subject** header, separated from the body of the e-mail with a blank line.

To send the mail you use *smtpObj* to connect to the SMTP server on the local machine and then use the *sendmail* method along with the message, the from address, and the destination address as parameters (even though the from and to addresses are within the e-mail itself, these aren't always used to route mail).

If you are not running an SMTP server on your local machine, you can use *smtplib* client to communicate with a remote SMTP server. Unless you are using a webmail service (such as Hotmail or Yahoo! Mail), your e-mail provider must have provided you with outgoing mail server details that you can supply them, as follows −smtplib.SMTP('mail.your-domain.com', 25)

4.11 URL ACCESS

URL(Uniform REsource Locator)

- urlib is the module used for fetching URLs

urllib is a Python module that can be used for opening URLs. It defines functions and classes to help in URL actions.

With Python we can also access and retrieve data from the internet like XML, HTML, JSON, etc. We can also use Python to work with this data directly.

Database and Network

```
#Used to make requests
    import urllib.request
    x= urllib.request.urlopen('https://www.google.com/')
    print(x.read())
```

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE

www.sathyabama.ac.in

SCHOOL OF BIO AND CHEMICAL ENGINEERING

Common to : Biotech, BioMed, Chemical, EEE

UNIT – V - SCSA1102 - FUNDAMENTALS OF PYTHON PROGRAMMMING

(Case Study)

UNIT v

CASE STUDY

5.1 WEB PROGRAMMING USING PYTHON

Python is one of the most suitable language for web application development for its efficiency and readability. There are different frameworks supported by python. A framework is a bundle of packages and modules that allow us to create web application very easily without having to handle low-level activities such as thread management, process management and protocol management. We can build our application very effectively with the help of frameworks.

Given below are some of the popular web frameworks in python

1. Django

Django is a popular python web framework and is used for larger applications. It contains everything needed for web development bundled with the framework itself. Users have no need to handle database administration, routing and authentication. Django works well with all important databases like Oracle, MySQL, PostgreSQL, SQLite,etc.

Features

1. Fast- Django is designed to handle the applications from beginning to end as quickly as possible.

2. Fully loaded – Django framework handle all services required for a web application like user authentication, context administration, site maps and many more.
3. Security- It helps the developer to avoid common security mistakes such as SQL injection, cross-site scripting and cross site request forgery.
4. Scalability- It handles the heaviest traffic demands.

2. Flask

Flask is a micro framework for python and good choice for building smaller applications and web services. It implements the commonly used core components of a web application framework such as URL routing, request and response objects and templates. However, built-in functions like Database access, form generation and validation are not supported in Flask.

3. Pyramid

Pyramid is the most flexible python framework and is used for mid-high scale applications. Anyone can start to work with Pyramid without any prior knowledge about it. It comes with only some important tools which are needed for developing application. It is a finishing framework with the ability to start small application and allow us to code a solid foundation for our solution and to scale up as needed.

5.1.1 Developing simple application using Django

The Django web framework provides tools and libraries to simplify the task of web development operations. It solves the issues and it will make our work a lot easier. Django web framework helps in building clean and maintainable web applications very quickly.

5.1.2 Django Architecture

It follows MVC-MVT architecture. MVC stands for Model View Controller. It is used for developing the web applications. It consists of three segments like model, view and controller. The fig 5.1 given below shows the MVC architecture.

- Model: It is used for storing and maintaining our data. It is the backend where our database is defined.
- Views: views are in html. Whatever user is seeing, it is defined as view.
- Controller: Controller is business logic that interacts with the model and the view.

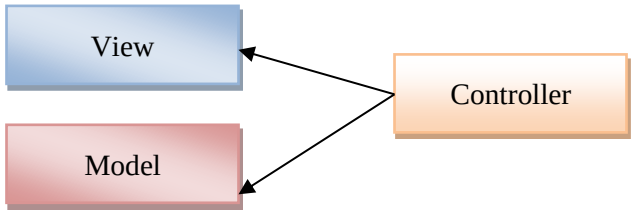


Fig 5.1 MVC Architecture

5.1.3 Django MVT pattern

MVT stands for Model View Template. In MVT, predefined template is used for user interface. User has no need to rewrite the code again by using template. Django will acts as controller in this part. Template is our front end which interacts with the view and the model will be used as the backend. View will access both the model and templates and maps them to a URL. Fig 5.2 describes the MVT pattern.

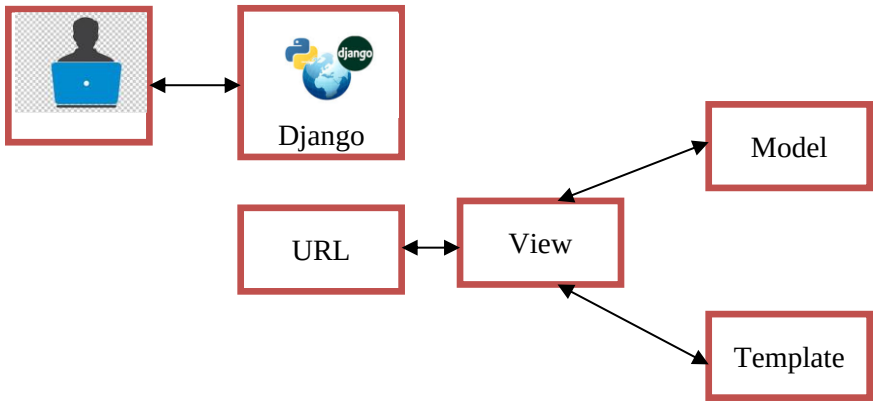


Fig 5.2 MVT Pattern

5.1.4 Django Installation

Step 1: Go to the link: https://www.djangoproject.com/download/. It is described in fig 5.3.

Step 2: Select the command prompt from the start menu, right click and choose the option “run as administrator”. Now the screen displays the command prompt shown in fig 4.

Step 3: Type the pip command on command prompt as follows.

Pip install Django == 1.11.4

Step 4: This creates a project folder in the python environment .The folder name is “myproject”

Step 5: To build a web application, enter into the “myproject” folder. Type the following in command terminal

Django-admin startproject myproject

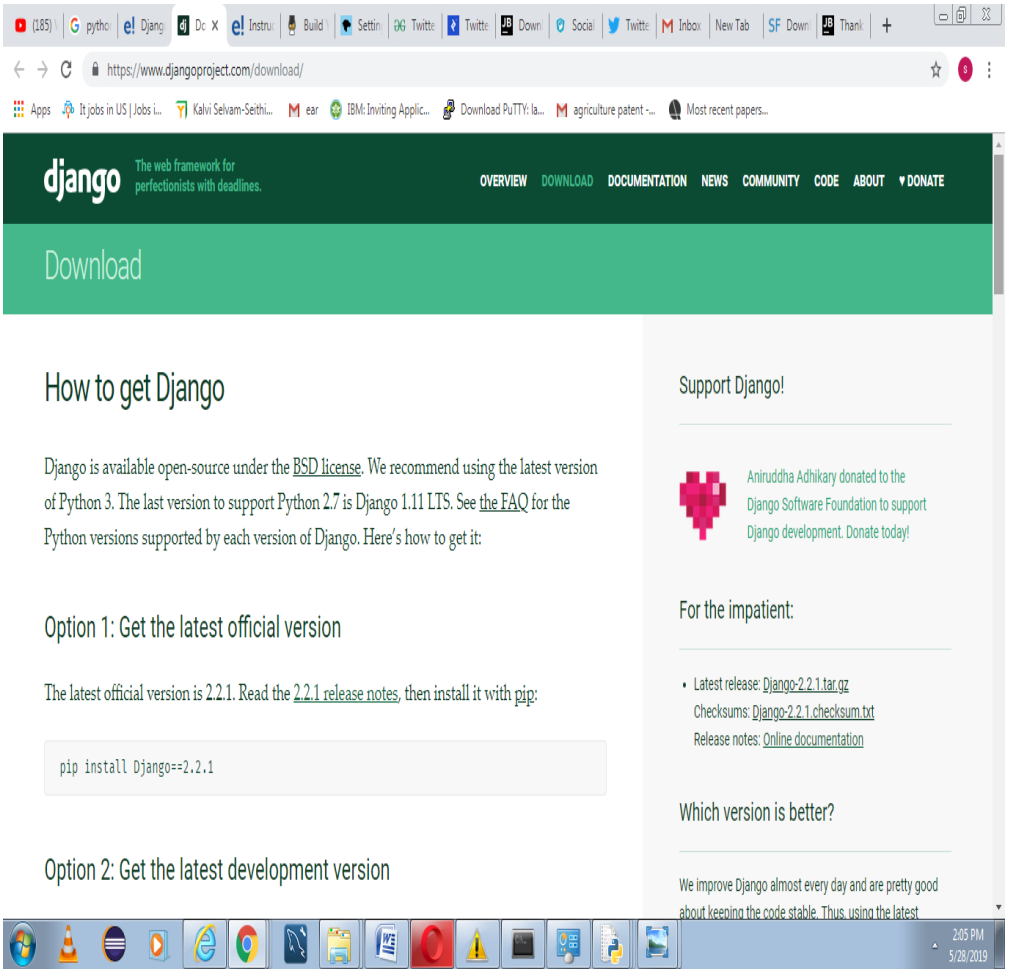


Fig 5.3 Django Website

In this example, we used Django==1.11.4 version.

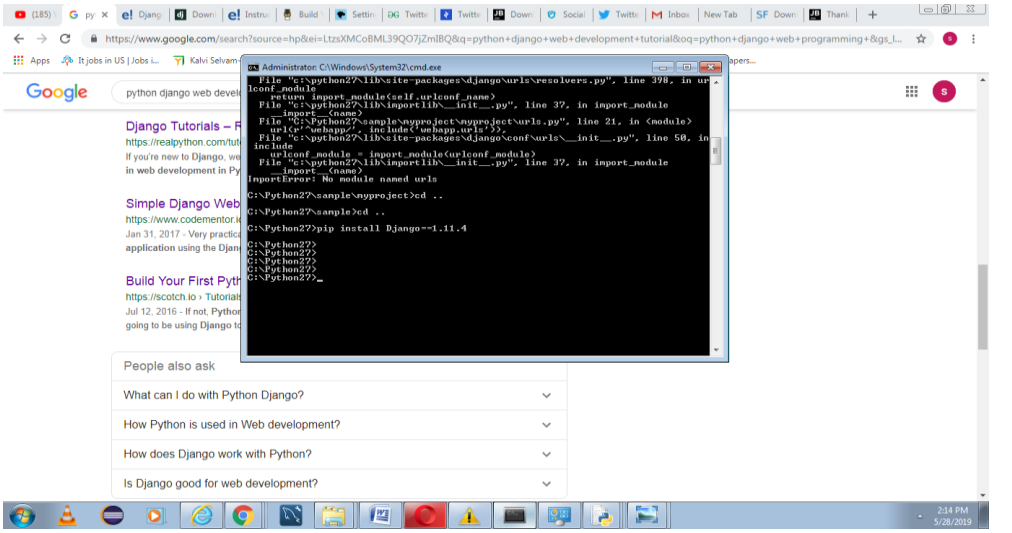


Fig 5.4 Installation of Django

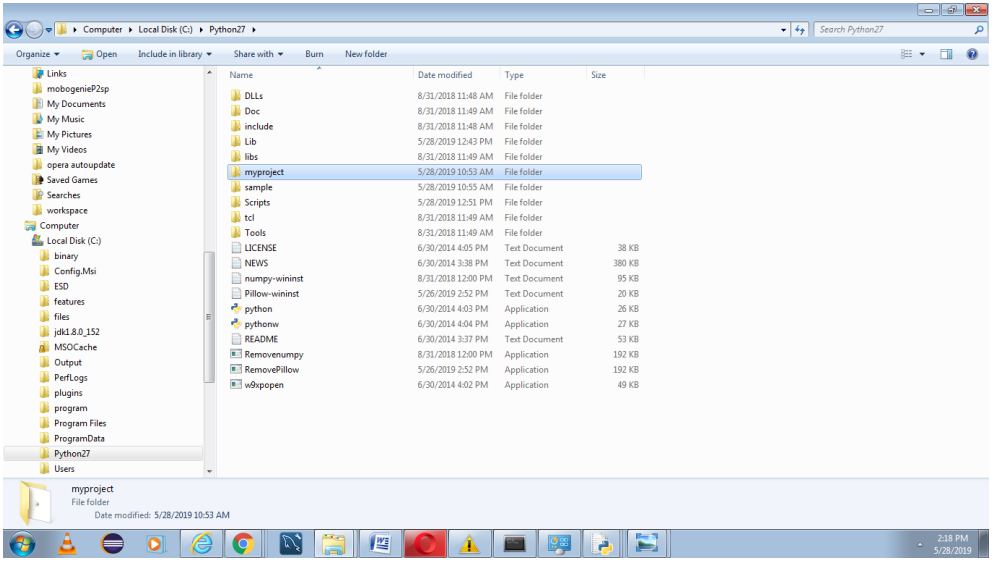


Fig 5.5 Folder Creation in Python Environment

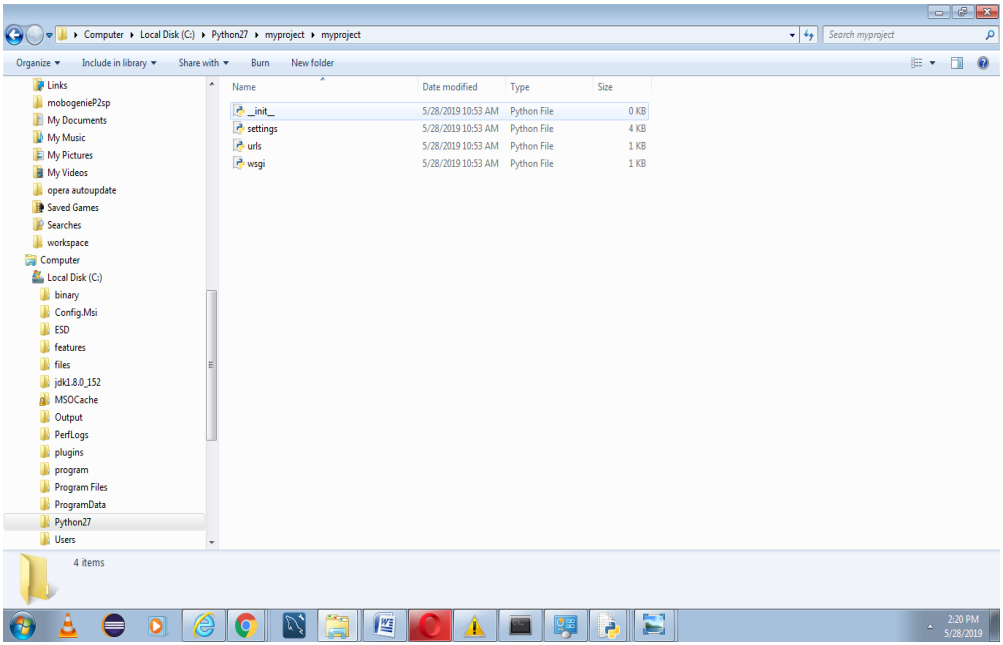


Fig 5.6 Files in Directory

Fig 5.5 and 5.6 describes the folder creation and list of files in directory. Our project is created now. We will see the list of files in directory. Let's discuss about the following files.

1. manage.py- It is a command line utility
2. myproject –It is actual python package in our project.
3. init.py-Python package
4. settings.py- It manages all the settings of our project

5. urls.py-Main controller which maps it to our web site.
6. wsgi.py- It acts as an entry point for WSGI (Web Server Gateway Interface) compatible web servers

Step 6: Create our web application and make sure that we are in the same directory as mangae.py and type the following command in the command terminal

```
python manage.py startappwebapp
```

Now webapp is added in our project folder also few other elements are added in web app like view, test and model. It is shown in fig 5.7.

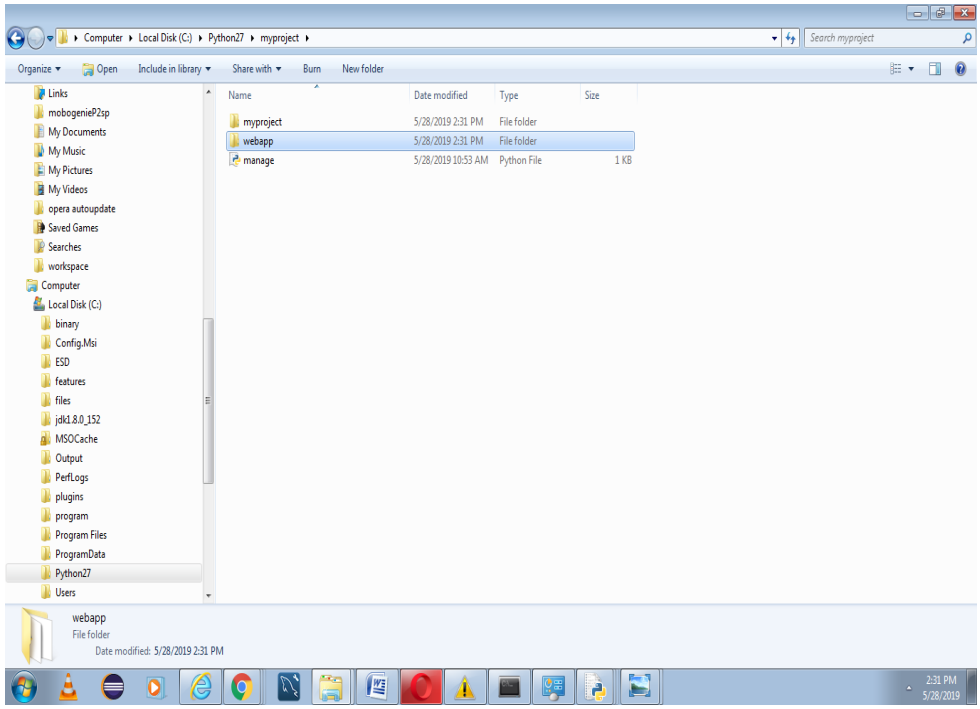


Fig 5.7 Creation of Web App

Step 7: Now open our myproject/myproject/settings.py The following fig 5.8 shows the settings file.

Step 8: In setting.py file, we add the “web app” line in the first statement. By above insertion, we have added our web app.

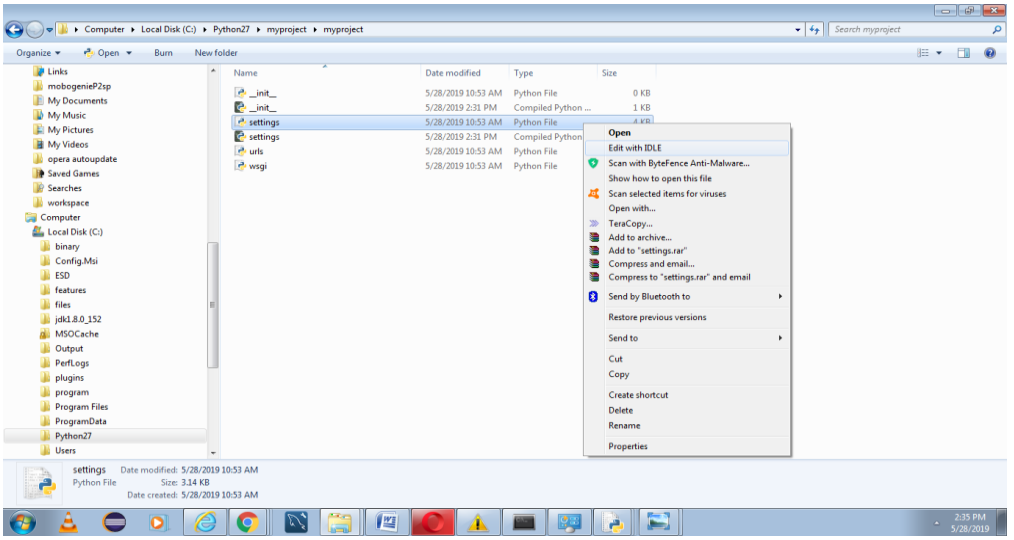


Fig 5.8 Settings File

```
INSTALLED_APPS = [

    'webapp',

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

]
```

Step 9: Once we have added our app, a new file view.py is automatically added in the web app that is shown in fig 5.9. Open our webapp/views.py and enter the following code.

```
fromdjango.shortcuts import render

fromdjango.http import HttpResponse

def index(request):

return  HttpResponse("<H2>! Welcome to Sathyabama! </H2>")
```

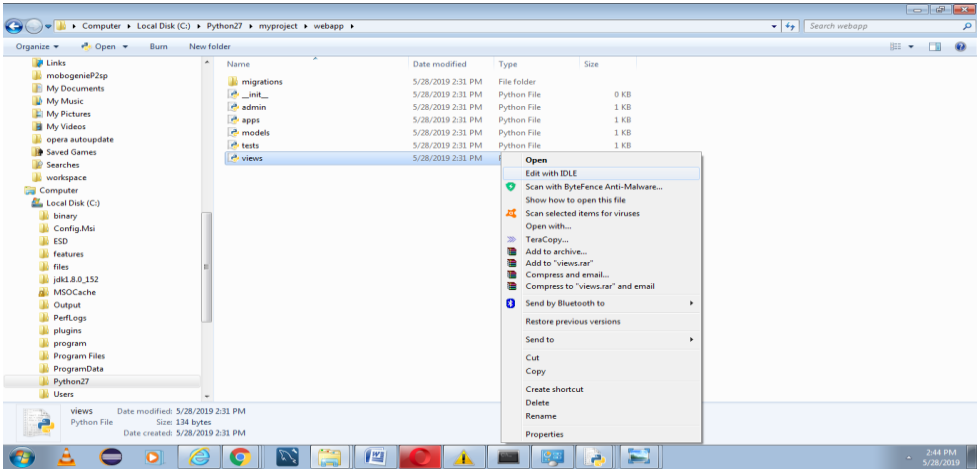


Fig 5.9 View File

Step 10: We have created a view that returns http response and map this view to a URL. We need to create a “url.py” inside our web app and enter the following code.

```
fromdjango.conf.urls import url
from .import views
urlpatterns = [
url(r'^$', views.index, name='index'),
]
```

Step 11: Point the root URLconf at the webapp.urls module. Open our myproject/urls.py file and write the following code.

```
From django.conf.urls import include, url
From django.contrib import admin
url patterns = [
url(r'^admin/', include(admin.site.urls)),
url(r'^webapp/', include('webapp.urls')),
]
```

Step 12: Now start the server by type the following command

```
Python manage.py runserver
```

After running the server, goto http://localhost:8000/webapp/ in our browser and see the "Welcome to Sathyabama" message which we defined in the index view.

5.2 Image Processing

Image processing involves representation, processing and information extraction from images. It can increase the readability of the image and enhance the quality of the image.

Image processing is a part of computer vision. Computer vision is an important field in the area of artificial intelligence.

1. By representation we mean converting an image into digital form.
2. By processing we mean performing operation like smoothing, sharpening, contrasting and stretching on image to get an enhanced image.
3. Information extraction refers to applying techniques for deriving useful information like tumor detection, remote sensing, weather forecasting etc.

Python supports lot of libraries for image processing, including

- **Open-CV**- It is mainly focused on real time computer vision with variety of applications such as two dimensional and three dimensional Open-CV is an open source computer vision library for real time image and video processing. It supports a lot of algorithms related to computer vision. It supports a variety of languages like C++, Python and Java. It is available on different platforms including Windows, Linux, Android and iOS.
- **Numpy and Scipy libraries**- Numpy is a optimized library for numerical operations. Open-CV array structures are converted to Numpy arrays. Both are used for image manipulation and processing.
- **Python Imaging Library (PIL)** – It is mainly used for performing basic operations such as resize, rotation and converts between different file formats.
- **Matplotlib**- It is an optional choice for displaying frames from images or videos.

The following Python packages are needed to be downloaded and installed to their default locations.

- Python3.7
- Numpy
- Matplotlib

Steps for installation of packages:

1. https://www.python.org/downloads/ and download the installer.
2. After installation , open Python IDE and enter the following two commands

   >>>python –m pip install numpy

   >>>python –m pip install opencv-python
3. Open Python IDE and type the following codes in python terminal for verifying the installation of opencv and numpy libraries.

   >>>import cv2

   >>>print cv2._version_

5.2.1 Gray Scale Image

Below are the some of the examples for demonstrating the use of libraries for image processing. The given program shows the image in gray scale. Import the all the libraries and read the image using imread function. Fig 5.10 shows the image in gray scale.

Code:

```
import cv2

import numpy as np
```

```
from matplotlib import pyplot as plt

im = cv2.imread('boat.jpg',cv2.IMREAD_GRAYSCALE)

cv2.imshow('image',im)

cv2.waitKey(0)

cv2.destroyAllWindows()
```

Output:

Fig 5.10 Gray scale Image

To read the original image, simply call the **imread** function of the **cv2**module, passing as input the path to the image, as a string. We used imshow function for receiving the first argument as input string and as second argument the image to show. We used waitkey function for including the delay in the key board event.

5.2.2 Geo metric Transformation of Image

5.2.2.1 Resize Image

Scaling is just resizing of the image. The size of the image can be specified manually or specify with scaling factor. It helps in reducing the number of pixels from an image. We need to either resize the image shrink it or scale up to meet the size requirements.

The following syntax specifies the resize function.

cv2.resize(src,dsize, Interpolation)

where src specifies source image

dsize specifies destination image

Interpolation represents the different function such as cv.INTER_AREA for shrinking and cv.INTER_CUBIC for zooming operation.

Fig 5.11 shows the output of scaling.

Code:

```
import cv2

img = cv2.imread('boat.jpg', cv2.IMREAD_UNCHANGED)
```

```
print('Original Dimensions : ',img.shape)

scale_percent = 60 # percent of original size

width = int(img.shape[1] * scale_percent / 100)

height = int(img.shape[0] * scale_percent / 100)

dim = (width, height)

# resize image

resized = cv2.resize(img, dim, interpolation = cv2.INTER_AREA)

print('Resized Dimensions : ',resized.shape)

cv2.imshow("Resized image", resized)

cv2.waitKey(0)

cv2.destroyAllWindows()
```

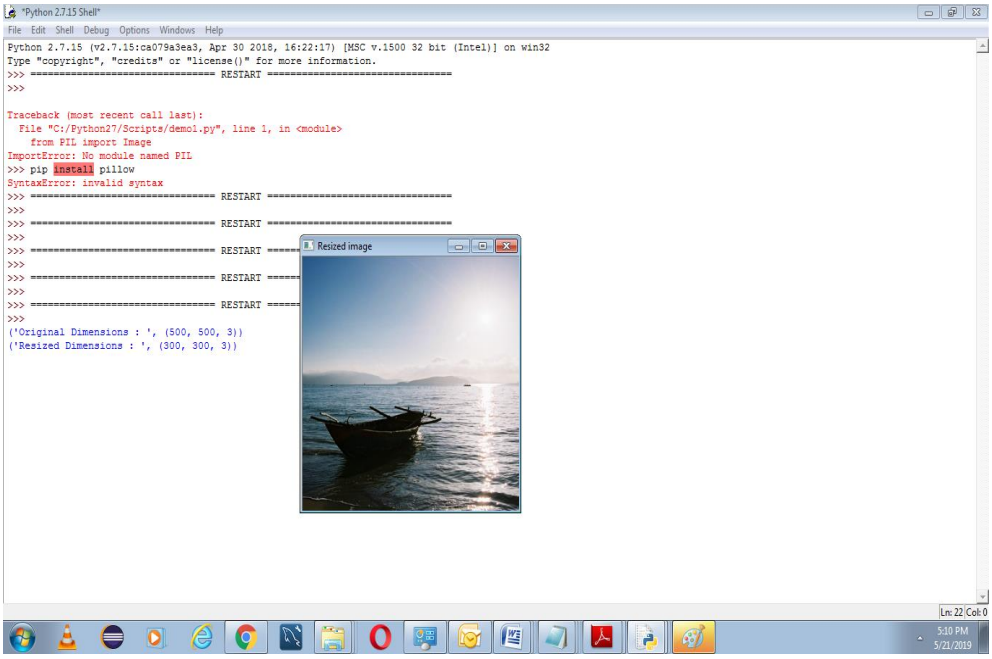


Fig 5.11 Scaling

5.2.2.2 Translation

Translation is the shifting of object's location from one point to another i.e (x,y) to (x_1,y_1).

The transformation matrix M is represented as follows:

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \qquad (1)$$

Code:

```
Import numpy as np

import cv2 as cv

img = cv.imread('boat.jpg',0)

rows,cols = img.shape

M = np.float32([[1,0,100],[0,1,50]])

dst = cv.warpAffine(img,M,(cols,rows))

cv.imshow('img',dst)

cv.waitKey(0)

cv.destroyAllWindows()
```

Fig 5.12 and 5.13 describes the original image and translation result. In code, tx, ty values are the X and Y translation values. The image will be moved x units towards the right and by Y units downwards. cv.warpaffine function specifies size of the output image. It refers the number of rows and columns in the resulting image.

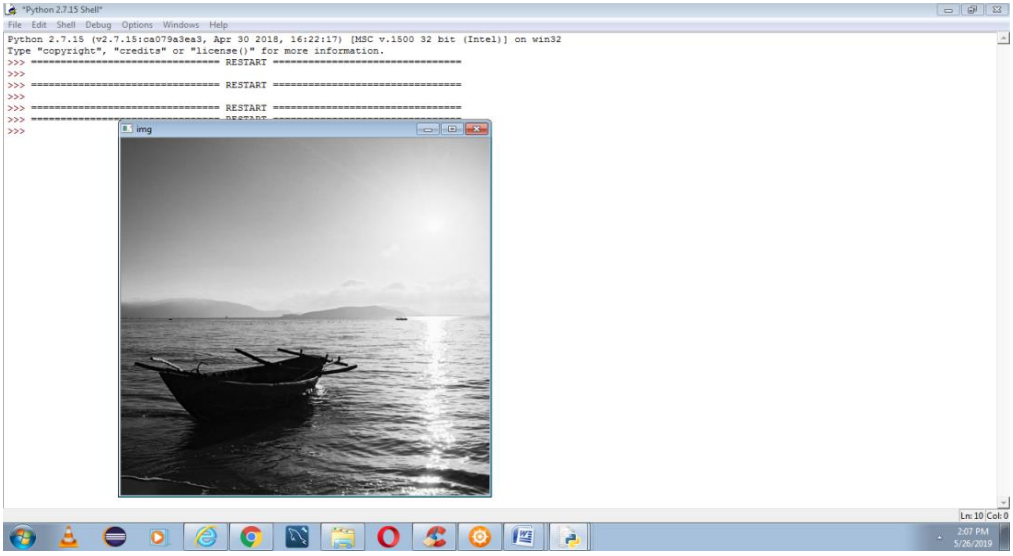


Fig 5.12 Original Image

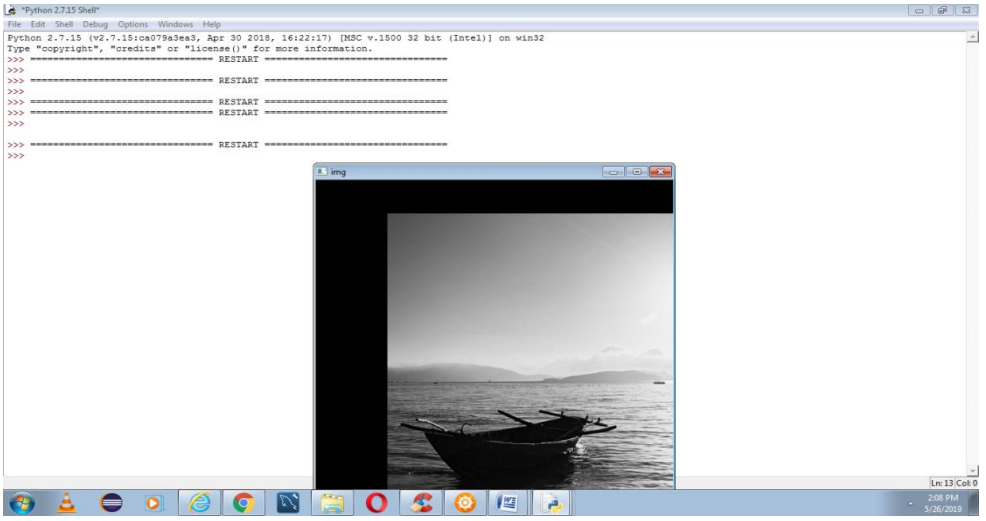


Fig 5.13 Translation

5.2.3 Thresholding

Thresholding is a simplest method for converting a gray scale image into a binary image. If a pixel is greater than a threshold value, it is assigned with one value(White), else it is assigned another value (Black). The algorithm is described as below:

$$If\ I_{i,j} = 1 \quad I_{i,j} > \theta$$

$$Else\ I_{i,j} = 0 \quad I_{i,j} \geq \theta \qquad (2)$$

The threshold function is described as below:

Cv2.threshold (src, thresh, maxval, type[, dst])

This function is used to get a binary image out of a grayscale image for removing a noise.

1. src- Input array. This is the source image.
2. thresh-threshold value which is used for classifying the pixel.
3. maxval- Maxval which represents the value to given if pixel is more than the threshold value.
4. Type- Thresholding type. Different types are mentioned as below:

a. cv2.THRESH_BINARY (Threshold Binary)

b. cv2.THRESH_BINARY_INVY (Threshold Binary Inverted)

c. cv2.THRESH_TRUNCY (Truncate)

d. cv2.THRESH_TOZEROY (Threshold to Zero)

e. cv2.THRESH_TOZERO_INVY(Threshold to Zero Inverted)

The following fig 5.14 shows the outputs for different threshold functions.

Code:

```
import cv2
importnumpy as np
frommatplotlib import pyplot as plt
img = cv2.imread('bloodcells.jpg',0)
ret,thresh1 = cv2.threshold(img,127,255,cv2.THRESH_BINARY)
ret,thresh2 = cv2.threshold(img,127,255,cv2.THRESH_BINARY_INV)
ret,thresh3 = cv2.threshold(img,127,255,cv2.THRESH_TRUNC)
```

```
ret,thresh4 = cv2.threshold(img,127,255,cv2.THRESH_TOZERO)
ret,thresh5 = cv2.threshold(img,127,255,cv2.THRESH_TOZERO_INV)
titles = ['Original
Image','BINARY','BINARY_INV','TRUNC','TOZERO','TOZERO_INV']
images = [img, thresh1, thresh2, thresh3, thresh4, thresh5]
fori in xrange(6):
plt.subplot(2,3,i+1),plt.imshow(images[i],'gray')
plt.title(titles[i])
plt.xticks([]),plt.yticks([])
plt.show()
```

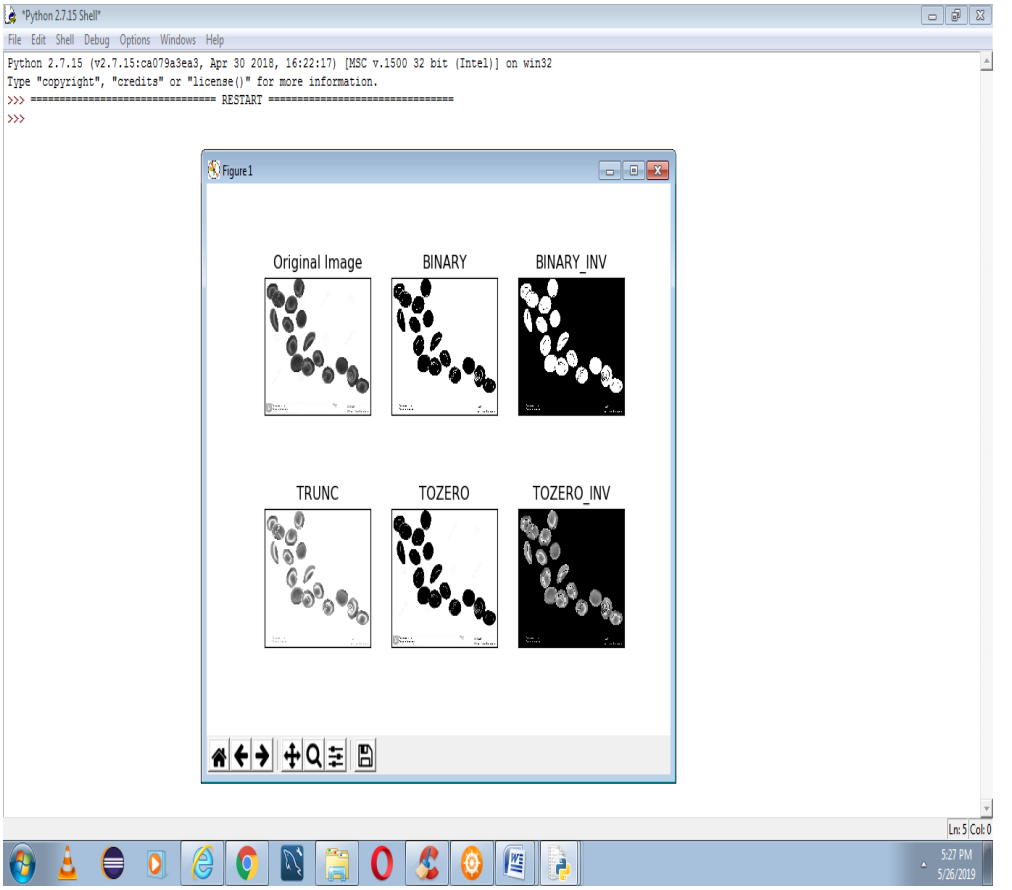


Fig 5.14 Thresholding

The different simple thresholding techniques are :

cv2. THRESH_BINARY: If pixel intensity is greater than the set threshold, value is set to 255, else set to 0 (black).

cv2. _BINARY_INV: Inverted or Opposite case of of cv2.THRESH_BINARY.

cv2.THRESH_TRUNC: If pixel intensity value is greater than threshold, it is truncated to the threshold. The pixel values are set to be same as the threshold. All other values remain same.

Cv2. THRESH_TOZERO: Pixel intensity is set to 0, all the pixels intensity, less than the threshold value.

Cv2. THRESH_TOZERO_INV: Opposite case of cv2. THRESH_TOZERO.

Matplotlib is a visualization library in python for 2D plots of the array. It is a data visualization library built on Numpy arrays. It consists of several plots like line, scatter etc. Ticks are the values used to show specific points on the coordinate axis. Whenever we plot a graph, the axes adjust and take the default ticks.

5.2.4 Image Blurring (Image Smoothing)

Image blurring is achieved by removing the outlier pixels in the image. It removes high frequency content from the image resulting in edges being blurred when the filter is applied. Here the following section describes the examples of blurring techniques.

5.2.4.1 Averaging

It takes the average of all the pixels under kernel area and replaces the central element with this average. This is achieved by using cv2.blur(). A 3×3 filter is described as below: $K = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ (3)

Code:

```
import cv2

importnumpy as np

frommatplotlib import pyplot as plt

img = cv2.imread('bloodcells.jpg')

blur = cv2.blur(img,(5,5))

plt.subplot(121),plt.imshow(img),plt.title('Original')

plt.xticks([]), plt.yticks([])

plt.subplot(122),plt.imshow(blur),plt.title('Blurred')

plt.xticks([]), plt.yticks([])

plt.show()
```

Fig 15 shows the image averaging output.

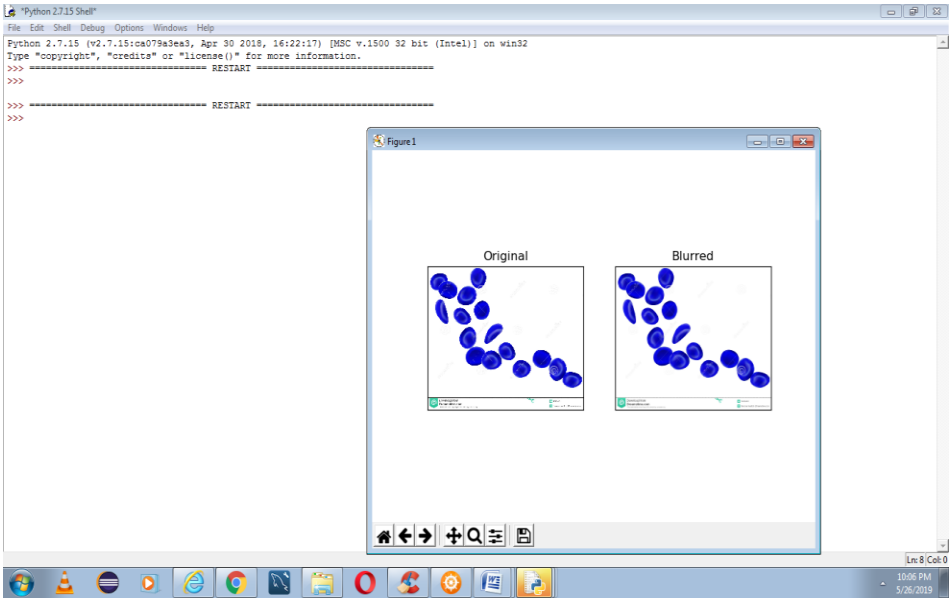


Fig 5.15 Image Averaging

5.2.5 Canny Edge Detection

It is a popular edge detection algorithm and consists of multiple numbers of stages such noise reduction, finding intensity gradient of the image, Non-maximum suppression and hysteresis threshold. In noise reduction, remove the noise from the image. It allows us to find the gradient of the gray scale image to find the edge regions in the x axis and y axis directions. After getting the magnitude and direction, a full scanning is performed to remove unwanted pixels in the edges. In hysteresis thresholding, we decides which are the edges are really edges or not by using two threshold values minval and maxval. Any edges with intensity gradient are more than maxval are

considered as edges and those below minval are considered as non edged and also discarded. The following fig 5.16 shows the result of canny edge detection.

Code

```
import cv2
importnumpy as np
frommatplotlib import pyplot as plt
img = cv2.imread('noise.jpg',0)
edges = cv2.Canny(img,100,200)
plt.subplot(121),plt.imshow(img,cmap = 'gray')
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(edges,cmap = 'gray')
plt.title('Edge Image'), plt.xticks([]), plt.yticks([])
plt.show()
```

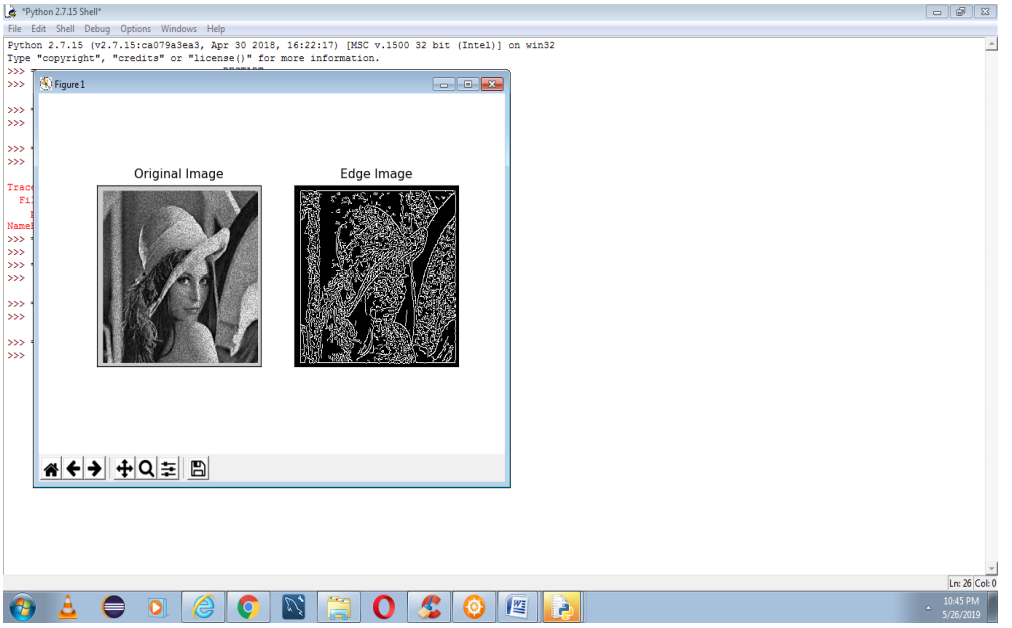


Fig 5.16 Canny Edge Detection

5.3 FACE BOOK DATA ANALYSIS

Face book provides an extensive API to interact with its platform and fetch the required information for analysis. Python is used for extract data from face book. We need to register as developer on face book. Here the steps are listed below.

1. Go to the link developers.facebook.com and create an account there.
2. Go to the link developers.facebook.com/tools/explorer.
3. Go to Myapps drop down in the top right corner and select add a new app. Choose the display name and category and then create APP ID.

4. Again, go to the link developers.facebook.com/tools/explorer. We will see “Graph API Explorer below “Myapps” in the top right corner. From “Graph API Explorer” drop down, select our App.
5. Select “Get Token”. From this menu select “Get user access Token “. Select permissions from the menu that appears and then select “Get access Token”

We can download datasets from other Face book pages and get these statuses for each post:

- Number of likes
- Number of shares
- Number of comments

Then we can analyze this data using Excel or Tableau or Python or any software used for data analysis. Fig 17 shows the login access in face book developer account. App creation details are described in fig 5.18, 5.19 and 5.20.

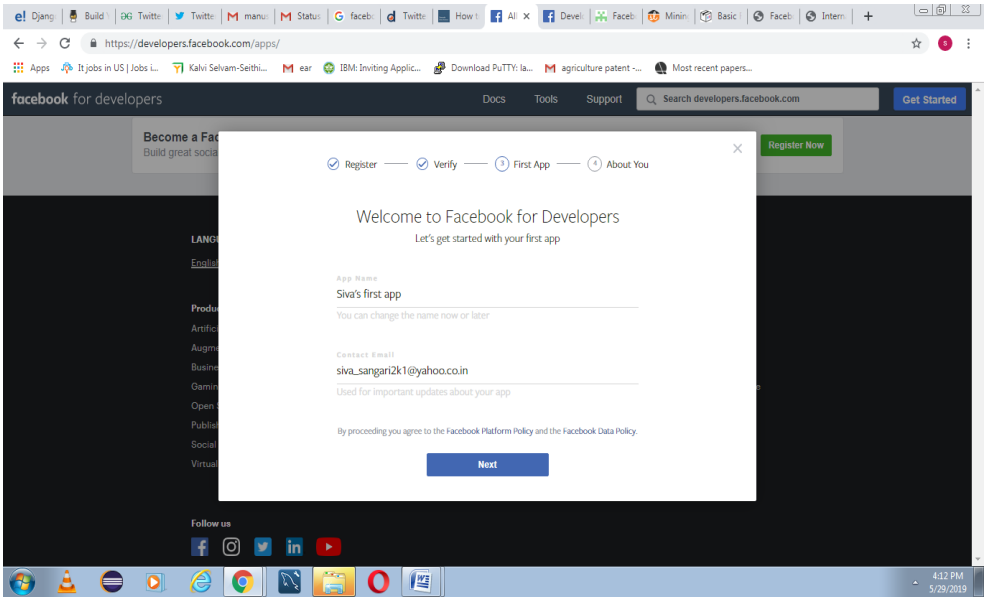


Fig 5.17 Login in face book developer's account

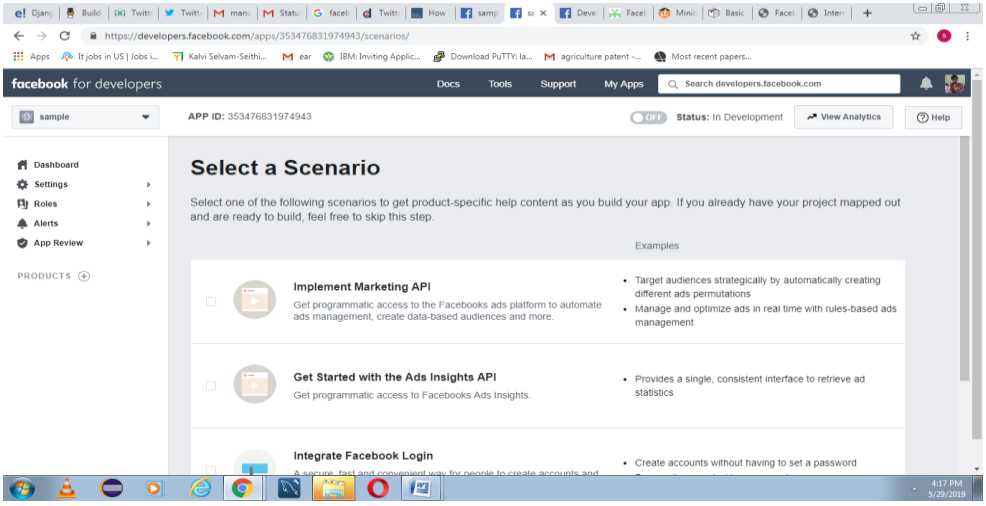


Fig 5.18 Creation of App

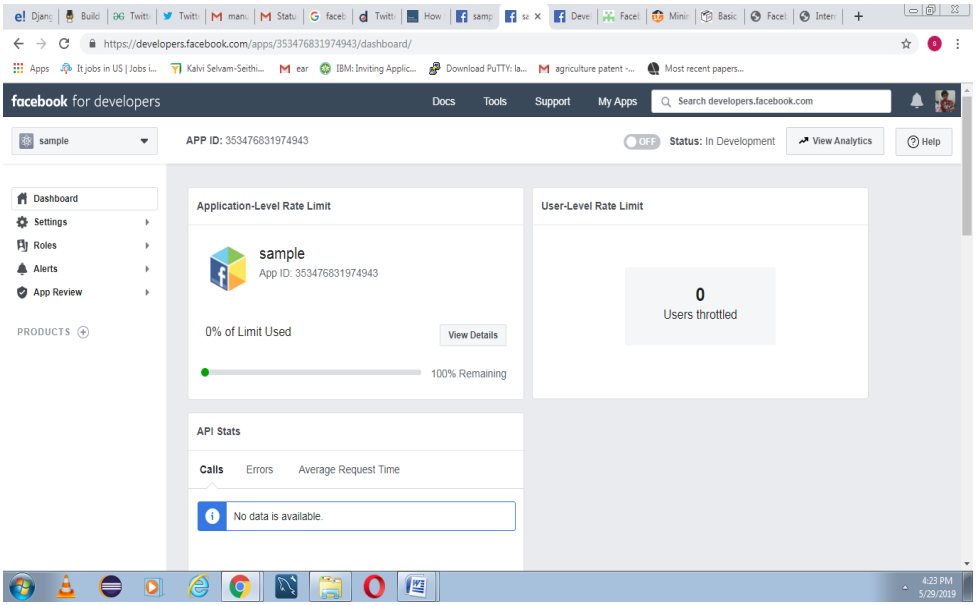


Fig 5.19 App Dashboard

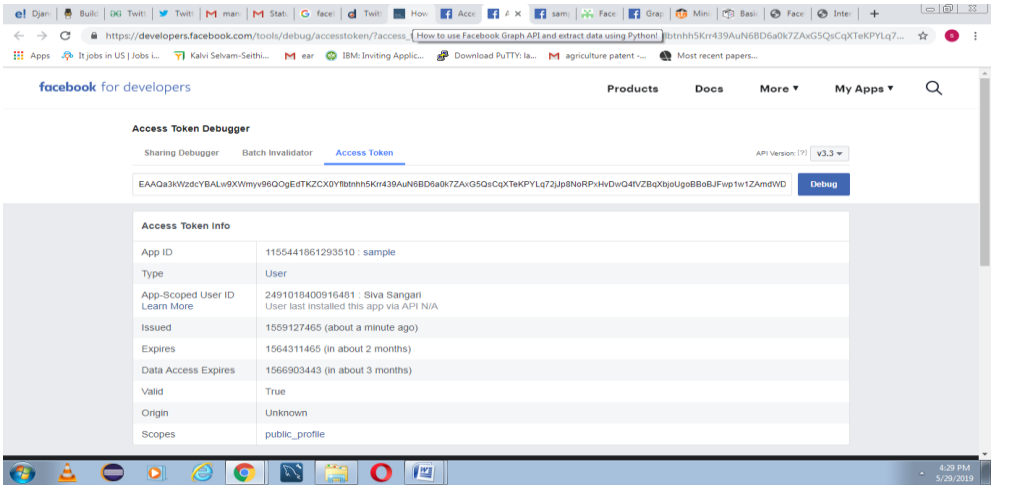


Fig 5.20 Access Token Details

The graph API is called social graph. It is a representation of information in face book. It consists of the following elements.

- Nodes- Individual objects such as user, photo ,page or comment
- Edges- Connection between a collection of objects and a single object such as photos or comments on a photo.
- Fields- Data about an object such as birthday or a page's name.

We can use nodes to get data about a specific object, use edges to get collections of objects on a single object and fields to get about a single object or each object in a collection. Graph API is HTTP based and works with any language.

Google graph API provides us a way by which we can get data from face book. We can put our data in face book platform. It is a REST based API and used to query data,

manage our ads on face book, upload photos, videos and post our new stories to face book automatically. We can this API to get our own face book account data. But, we need to get other users data for this we need to take several permissions from users. We need to implement oAuth protocol to implement this operation. Anyone can authenticate and grant our permissions. Fig 5.21 and 5.22 show the face book graph API and node information.

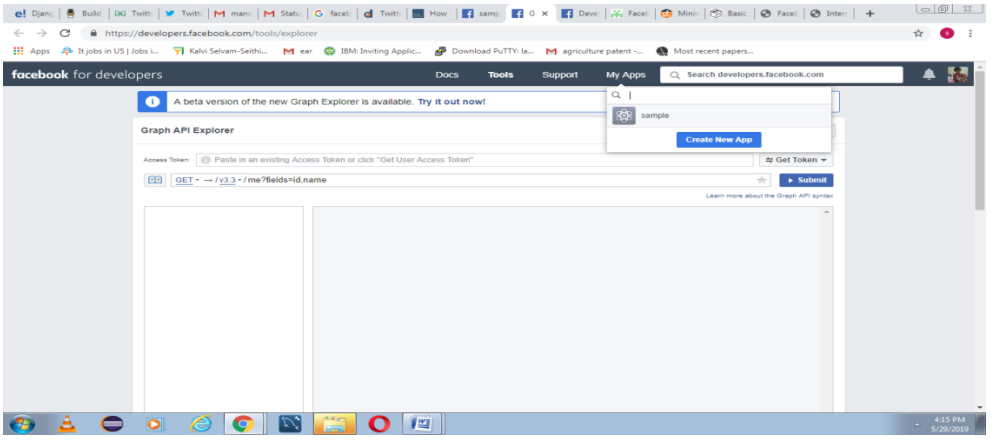


Fig 5.21 Face book Graph API

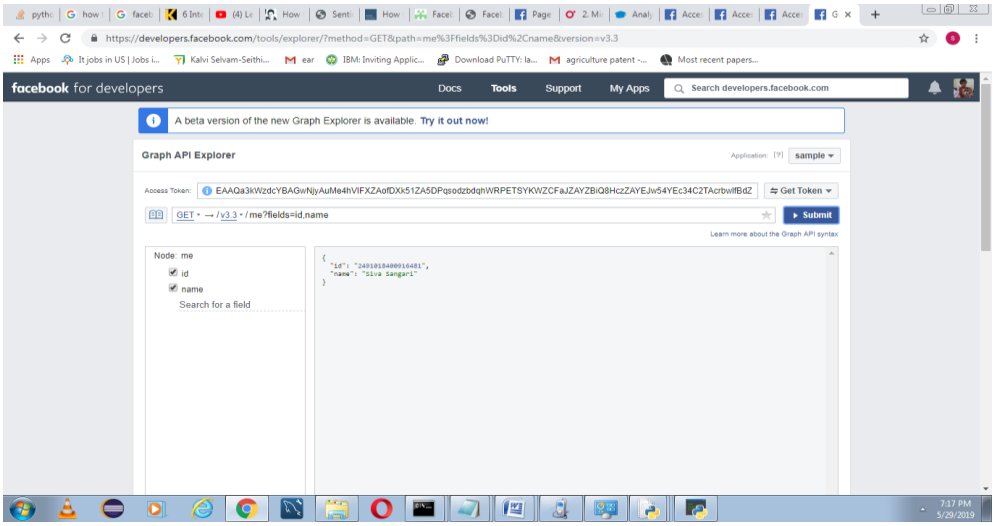


Fig 5.22 Node Information

Code

Import json

Import facebook

def main():

token =
"{EAAQa3kWzdcYBAKdzunCHWEixLKLvLSb5lnd8Ohs5Jh6zBefMCgOP
PJdYq4mTvkpgl15y1th6XpRSO5pxlnijQSCZAHShENSP06xtF4WZAAD0
CPFq988ZBdZAZAG8nx0DrTZAvIZBcfsYskP3JXsg7GN973Q39XwhKO

```
RlmxxR5kZA5GYN3ZCyNM3uL3waUh3dm91HruwWM63ZAtYQZDZD}
"

# Token value get it from access token details in fig 5.20.

        graph = facebook.GraphAPI(token)

        page_name = raw_input("Enter a page name: ")

        # list of required fields

        fields = ['id','name','about','likes']

        fields = ','.join(fields)

        page = graph.get_object(page_name, fields=fields)

        print(json.dumps(page,indent=4))

if __name__ == '__main__':

        main()
```

Output:

Enter page name

Smith

Name: Smith

Id: 13456234578

Likes: 23

5.4 TWITTER ANALYSIS

Twitter is a good resource to collect data. Unlike other social platforms almost every user's tweets are completely public. Twitter's API allows us to do complex queries like pulling every tweet about a particular topic. The pre processing of the text data is an essential step as it makes the raw text ready for mining. The main objective is to clean noise those are less relevant to find the sentiment tweets such as punctuation, characters and terms.

API stands for application programming interface. API is a tool that makes the interaction with computer programs and web services. Twitter streaming API is used to download tweets related to the key words that we specified in the coding.

Installation:

Before we start coding, we need to register for the Twitter API https://apps.twitter.com/. Here we need to register an app to generate various keys associated with our API. The following keys are used for authentication.

- API key
- API secret Key
- Access Token
- Access Token Secret

After creating the app we need to install the following commands.

pip install tweepy

pip install textblob

Tweepy is an easy way to use python library for accessing twitter API. We will extract tweets from twitter stream. TextBlob is used for processing textual data. It provides a simple API for dividing into common natural language processing tasks. Next create a new file called twitter.py and type the following code into it. Make sure to enter your credentials into access_token, access_token_secret, consumer key (API Key) and consumer secret (API secret key). In this code, we will download the scripts related python, java and java script.

Code:

```
from tweepy.streaming import StreamListener

from tweepy import OAuthHandler

from tweepy import Stream

#Variables that contains the user credentials to access Twitter API

access_token = "ENTER YOUR ACCESS TOKEN"

access_token_secret = "ENTER YOUR ACCESS TOKEN SECRET"

consumer_key = "ENTER YOUR API KEY"

consumer_secret = "ENTER YOUR API SECRET"
```

```
#This is a basic listener that just prints received tweets to stdout.
class StdOutListener (Stream Listener):
    def on_data(self, data):
        print data
        return True
    def on_error(self, status):
        print status
if __name__ == '__main__':
    # This handles Twitter authentication and the connection to Twitter Streaming API
    l = StdOutListener()
    auth = OAuthHandler(consumer_key, consumer_secret)
    auth.set_access_token(access_token, access_token_secret)
    stream = Stream(auth, l)
    #This line filter Twitter Streams to capture data by the keywords: 'python', 'java',
'javascript'
    stream.filter(track=['python', 'java', 'javascript'])
```

Next, we run the program in the command terminal using the command

python twitter.py.

If we want to capture this data into a file for future analysis, we can do piping the output to a file using the following command

python twitter.py > twitterdata.txt.

Here, the data we stored twitterdata.txt is a JSON (Java Script Object Notation).This format makes it easy to humans to read the data, and for machines to parse it. We will type the below code for printing the number of tweets.

```
import json

import pandas as pd

import matplotlib.pyplot as pl

tweets_data_path = 'C:python3/scripts/twitter_data.txt'

tweets_data = []

tweets_file = open(tweets_data_path, "r")

for line in tweets_file:

   try:
```

```
        tweet = json.loads(line)
        tweets_data.append(tweet)
    except:
        continue
        print len(tweets_data)
```

QUESTION BANK

1. List out the frame works of python in web programming.
2. Mention the libraries for image processing.
3. Explain different types of threshold function types?
4. Illustrate about canny edge detection algorithm?
5. How do you find the intensity distribution of the image?
6. Describe about the parameters of histogram function?
7. Evaluate the procedure for getting access token in Face Book data analysis?
8. Illustrate the implementation of Django web framework?
9. Elaborate about the method for removing noise from the image?
10. Assess the methods used in geo metric transformation of the image?
11. Analyze the steps involved in face book data analysis?
12. Elaborate about twitter data analysis?