



**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](http://www.sathyabama.ac.in)

**SCHOOL OF COMPUTING**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**UNIT-I Python Programming–SCSA1204**

*SCSA1204- Python Programming*



**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](http://www.sathyabama.ac.in)

**SCHOOL OF COMPUTING**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

## **UNIT-I Python Programming – SCSA1204**

## INTRODUCTION

History of Python- Introduction to the IDLE interpreter (shell) -Expressions – Data Types - Built-in function - Conditional statements - Iterative statements- Input/output -Compound Data Types - Nested compound statements – Introduction to Object Oriented Concepts

### 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.

#### a. About Python

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

## SCSA1204- Python Programming

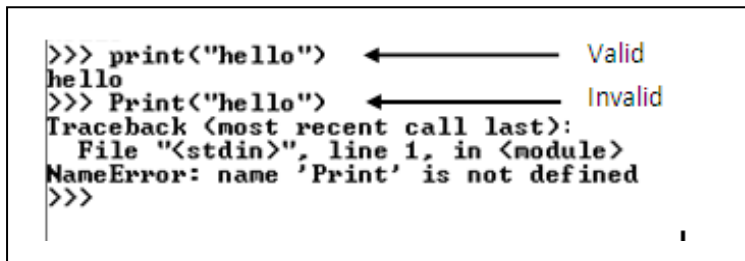
“[http:// python.org/](http://python.org/)”

- 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)



```
>>> print("hello")  ← Valid
hello
>>> Print("hello") ← Invalid
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'Print' is not defined
>>>
```

Fig1. Print Statement

- 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.

## SCSA1204- Python Programming

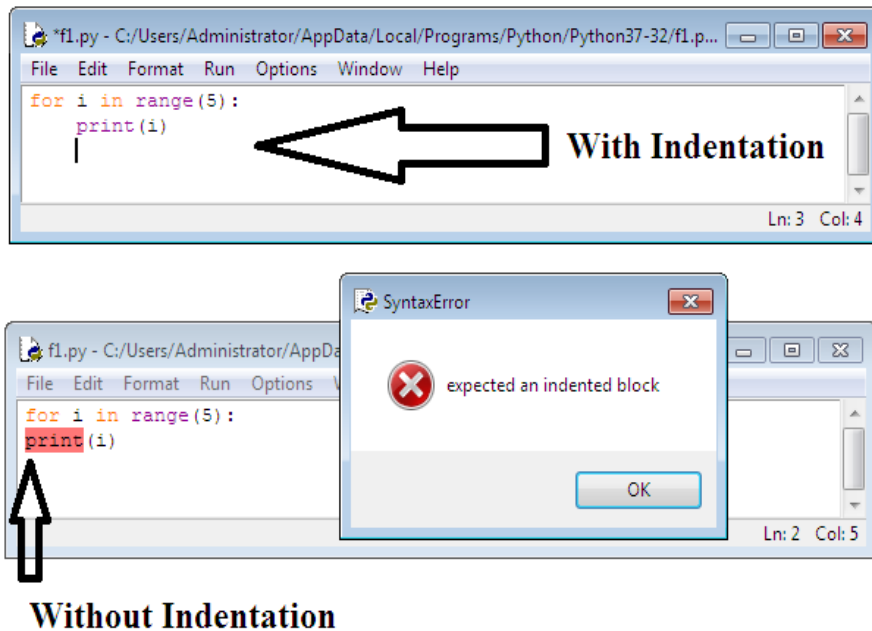


Fig 2 indentation

- 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

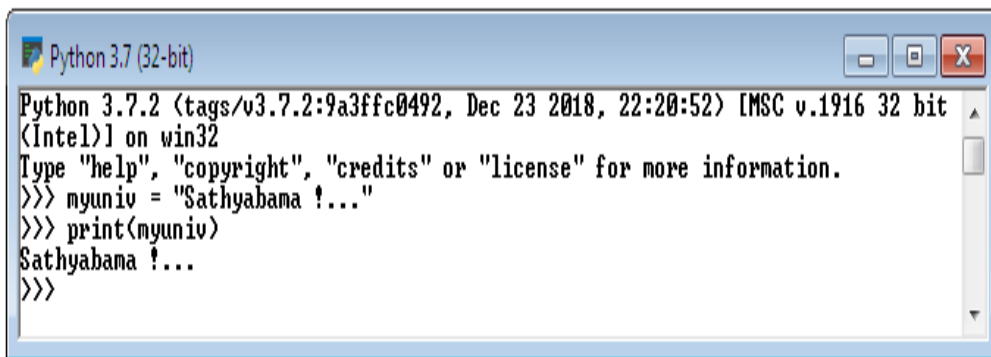
Python Program can be executed in two different modes:

- Interactive mode programming
- Script mode programming

## ***SCSA1204- Python Programming***

### **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.



```
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit  
<Intel>] on win32  
Type "help", "copyright", "credits" or "license" for more information.  
>>> myuniv = "Sathyabama !..."  
>>> print(myuniv)  
Sathyabama !...  
>>>
```

Fig 4 sample code is executed using interactive mode

Interactive mode can also be opened using the following ways:

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

## SCSA1204- Python Programming

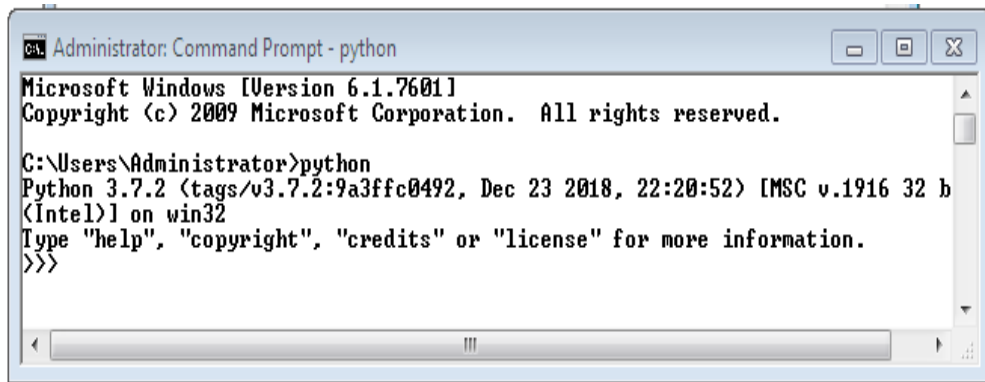


Fig 5 interactive mode

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)

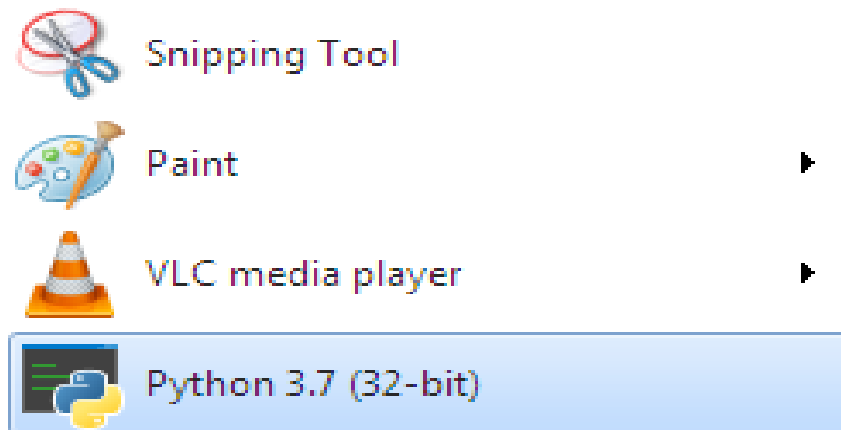


Fig 6 steps in opening

## SCSA1204- Python Programming

### 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

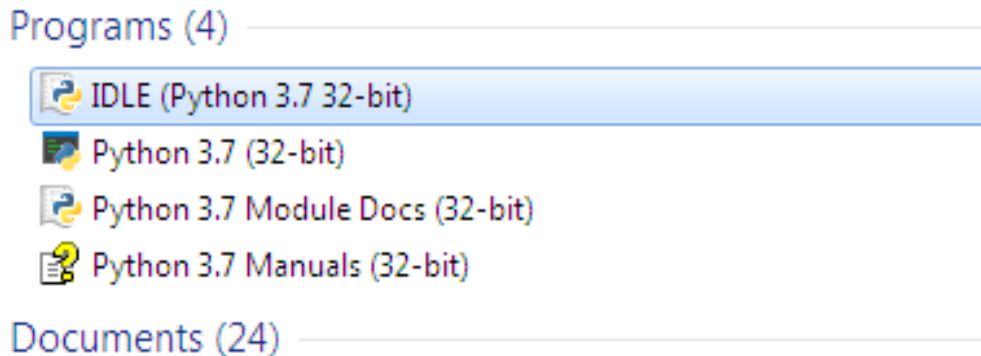


Fig 7 IDLE (Python 3.7 32-bit)

- ii) After clicking on the menu “***IDLE (Python 3.7 32-bit)***”, a new window with the text Python 3.7.2 Shell will be opened as shown below:



## SCSA1204- Python Programming

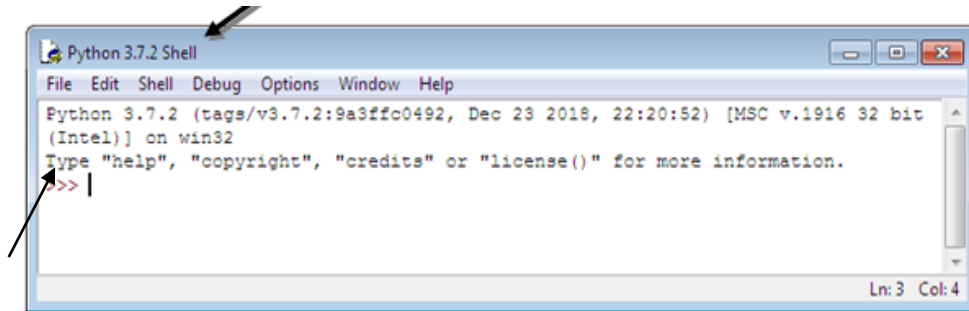


Fig 8 Python 3.7.2 Shell

- 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*)

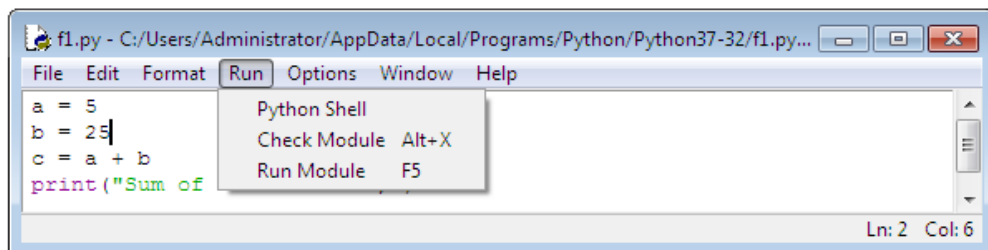


Fig 9. Run Module or Short Cut Key **F5**

The output of the program will be displayed as below:

## ***SCSA1204- Python Programming***

```
>> Sum of a and b is: 30
```

### **VARIABLES:**

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

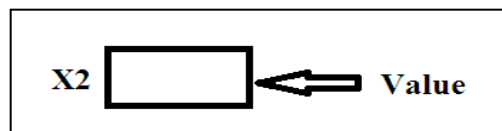
#### ***Need for variable:***

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***

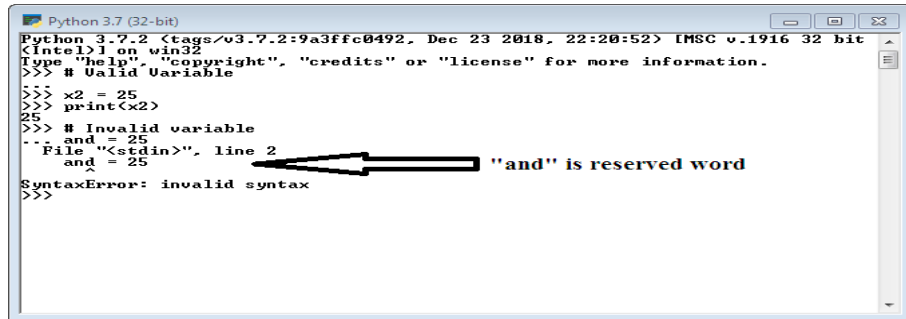
1. A variable name is made up of alphabets (Both upper and lower cases) and digits
  2. No reserved words
  3. Initialize before calling
  4. Multiple variables initialized
  5. Dynamic variable initialization
- i. Consist of upper and lower case alphabets, Numbers (0-9). E.g. X2

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



## SCSA1204- Python Programming

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

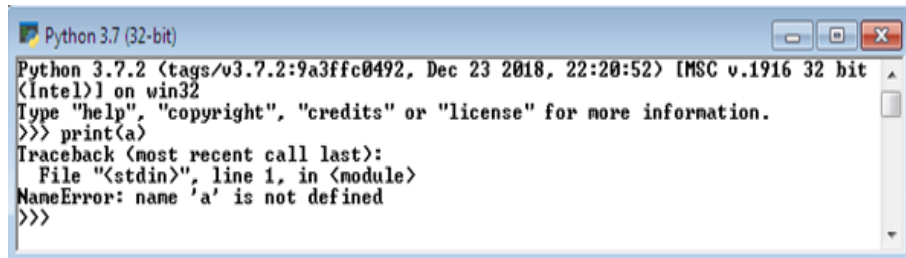


```
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel>] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> # Valid Variable
>>> x2 = 25
>>> print(x2)
25
>>> # Invalid variable
>>> and = 25
File "<stdin>", line 2
    and = 25
SyntaxError: invalid syntax
>>>
```

Fig 10. “and” is a reserved word

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)



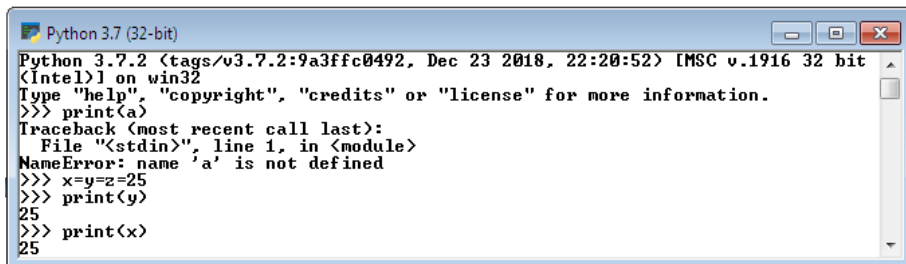
```
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel>] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print(a)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'a' is not defined
>>>
```

Fig 11 “a” is called before it initialized

## SCSA1204- Python Programming

In the above example “a” is called before it is 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`

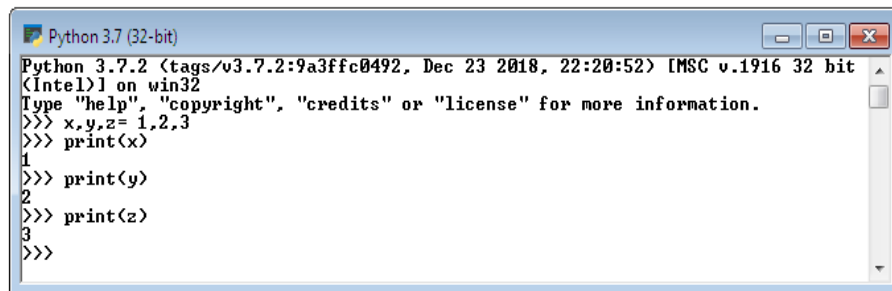


```
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit  
<Intel>] on win32  
Type "help", "copyright", "credits" or "license" for more information.  
>>> print(a)  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
NameError: name 'a' is not defined  
>>> x=y=z=25  
>>> print(y)  
25  
>>> print(x)  
25
```

Fig 12 Multiple variables

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`



```
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit  
<Intel>] on win32  
Type "help", "copyright", "credits" or "license" for more information.  
>>> x,y,z=1,2,3  
>>> print(x)  
1  
>>> print(y)  
2  
>>> print(z)  
3  
>>>
```

Fig 13 dynamic variable initialization

Proper spacing should be given

## SCSA1204- Python Programming

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

### 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...

In order to find area of circle, the expression  $\pi * r * r$  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.

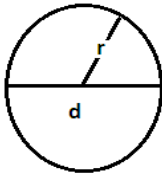
### Circle

Solution For Area ▾

$$A = \pi r^2$$

$r$  Radius

Enter value



### Rectangle

Solve for area ▾

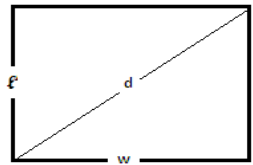
$$A = w l$$

$l$  Length

Enter value

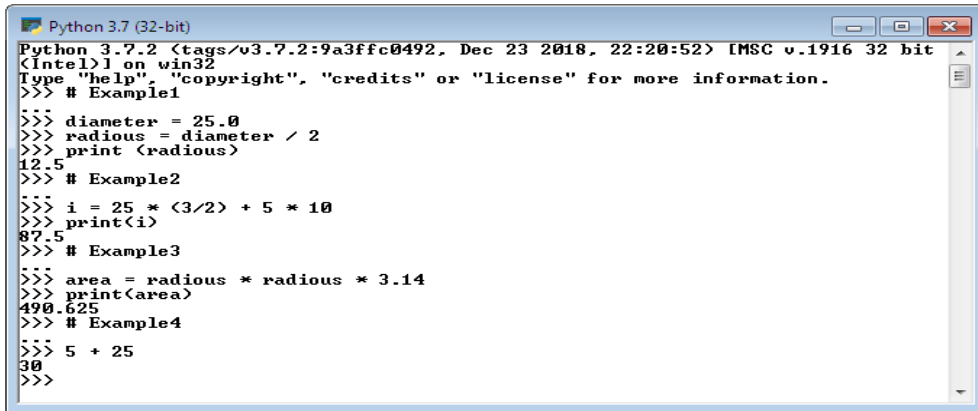
$w$  Width

Enter value



## SCSA1204- Python Programming

Fig 14 expression



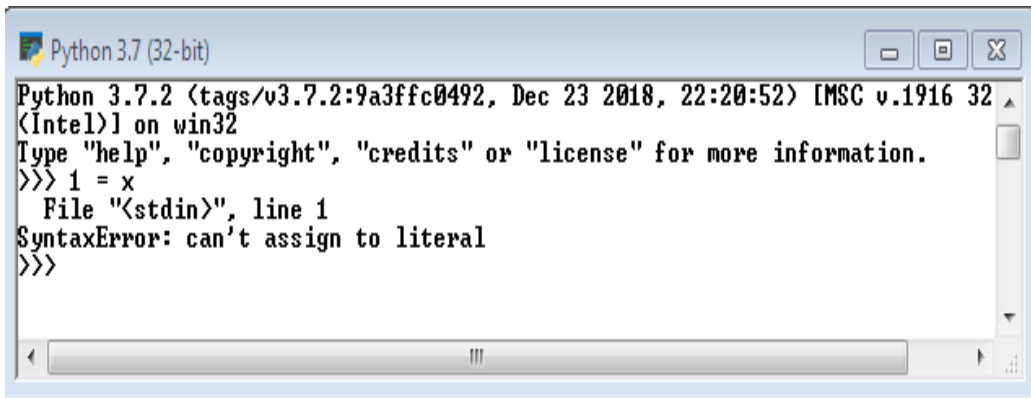
```
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel>] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> # Example1
>>> diameter = 25.0
>>> radius = diameter / 2
>>> print (radius)
12.5
>>> # Example2
>>> i = 25 * (3/2) + 5 * 10
>>> print(i)
87.5
>>> # Example3
>>> area = radius * radius * 3.14
>>> print(area)
490.625
>>> # Example4
>>> 5 + 25
30
>>>
```

Fig 15 expression

### 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.

## SCSA1204- Python Programming



```
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32
(Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 1 = x
      File "<stdin>", line 1
SyntaxError: can't assign to literal
>>>
```

Fig 16 invalid Expression

### 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 !.."

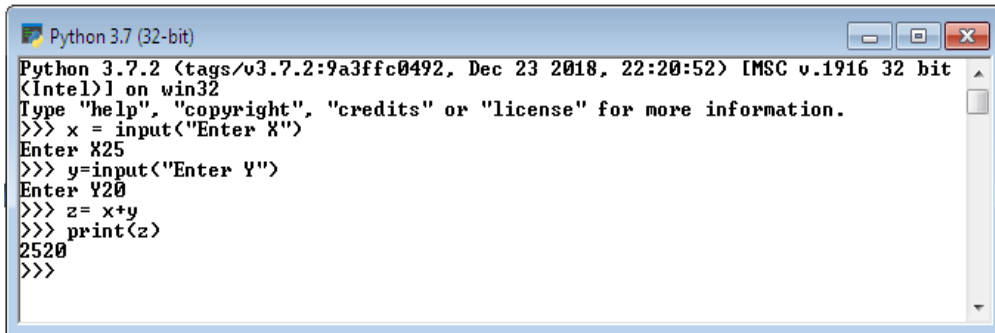
Integers: Whole number values such as 50, 100,-3

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

## SCSA1204- Python Programming

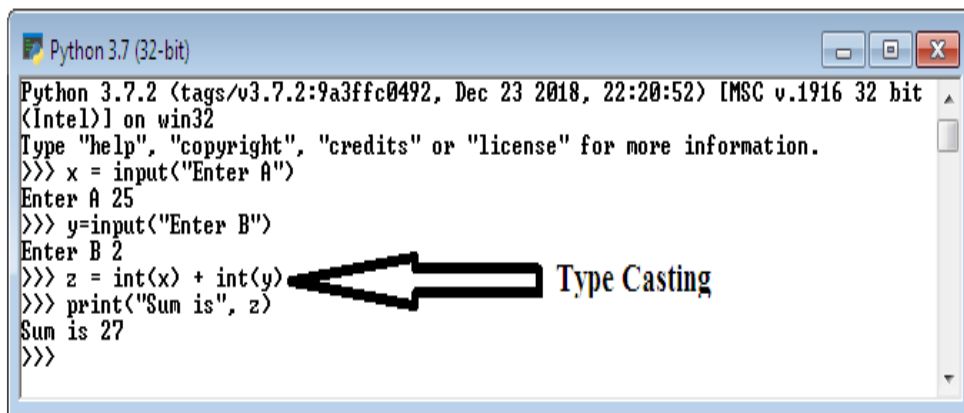
concatenates (joins together) the values together rather performing addition. For example : '25' + '20' = '45' (As in the below Example)



```
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel>] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> x = input("Enter X")
Enter X25
>>> y=input("Enter Y")
Enter Y20
>>> z= x+y
>>> print(z)
2520
>>>
```

Fig 17 arithmetic operation add on string values

Fortunately python have an option of converting one data 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)



```
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
<Intel>] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> x = input("Enter A")
Enter A 25
>>> y=input("Enter B")
Enter B 2
>>> z = int(x) + int(y)
>>> print("Sum is", z)
Sum is 27
>>>
```

➡ Type Casting

Fig 18 type casting



## ***SCSA1204- Python Programming***

### **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.

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']
```

**Table 1 Methods used in list**

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.

## SCSA1204- Python Programming

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

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

**Table 2 Tuple**

List	Tuple
<pre>&gt;&gt;&gt; list1[12,45,27] &gt;&gt;&gt; list1[1] = 55 &gt;&gt;&gt; print(list1) &gt;&gt;&gt; [12,55,27]</pre>	<pre>&gt;&gt;&gt; t1 = (12,45,27) &gt;&gt;&gt; t1[1] = 55 &gt;&gt;&gt; Generates Error Message # Because Tuples are immutable</pre>

### i) 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 3 Set**

List	Set
<pre>&gt;&gt;&gt; L1 = [1,20,25] &gt;&gt;&gt; print(L1[1]) &gt;&gt;&gt; 20</pre>	<pre>&gt;&gt;&gt; S1= {1,20,25,25} &gt;&gt;&gt; print(S1) &gt;&gt;&gt; {1,20,25} &gt;&gt;&gt; print(S1[1])</pre>

### ***SCSA1204- Python Programming***

	>>>Error , Set object does not support indexing.
--	--

**List Vs Set**

**ii) Dictionary**

The Python Dictionary is an unordered collection of key-value pairs. Dictionaries are 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)
```

**Table 4 Python's built-in data type conversion functions**

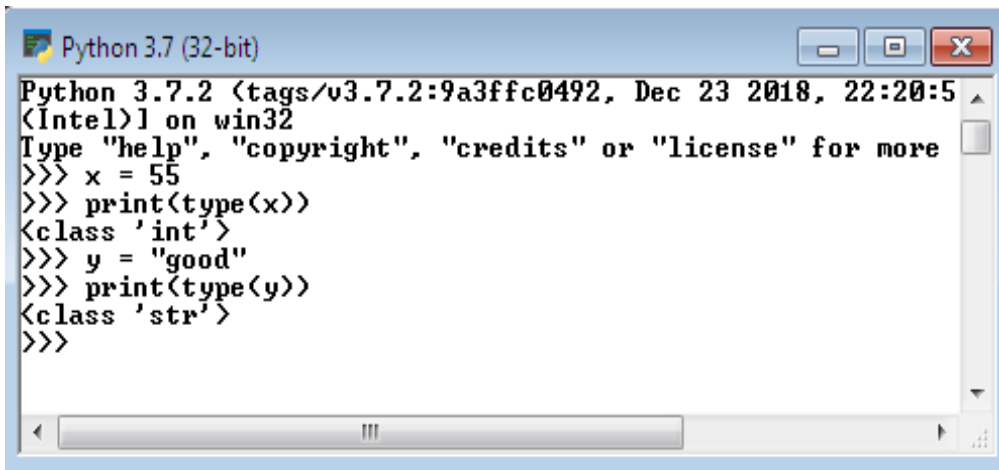
Function	Description	Out Put
int(x)	Converts x into integer whole number	<pre>&gt;&gt;&gt;a = int(input("Enter a")) &gt;&gt;&gt;b = int(input("Enter b")) &gt;&gt;&gt;c = a + b &gt;&gt;&gt;print("Sum is ",c)</pre>

### ***SCSA1204- Python Programming***

<b>Function</b>	<b>Description</b>	<b>Out Put</b>
float(x)	Converts x into floating-point number	>>> x = 5 >>> print(float(5)) >>> 5.0
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)

## SCSA1204- Python Programming



```
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:5
<Intel>] on win32
Type "help", "copyright", "credits" or "license" for more
>>> x = 55
>>> print(type(x))
<class 'int'>
>>> y = "good"
>>> print(type(y))
<class 'str'>
>>>
```

Fig 19 in-built python function “type ()”

### Table5 Python 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 <b>rounds</b> off the number to the nearest integer.	# E.g. 1: >> round(4.5)

## SCSA1204- Python Programming

		>> 5 # Eg 2 >> round(4.567,2) >> 4.57
<b>Mathematical Functions (Using math module)</b>		
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
<b>Function</b>	<b>Description</b>	<b>Example</b>
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

## ***SCSA1204- Python Programming***

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

```
import math
```



## ***SCSA1204- Python Programming***

### **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

***Table 6 Indentation (Structure of C- Program Vs Python)***

<b>C Program</b>	<b>Python</b>
<pre>x = 500 y = 200 if (x &gt; y) {     printf("x is greater than y") } else if(x == y) {     printf("x and y are equal") } else {</pre>	<pre>x = 500 y = 200 if x &gt; 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</pre>

## ***SCSA1204- Python Programming***

<code>printf("x is less than y") }</code>	instead of curly bracket)
---	---------------------------

### ***Structure of C- Program Vs Python***

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.

#### ***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.

#### **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 firststatement after the block.

Syntax:

```
if<Boolean>:
```

```
    <block>
```

### ***SCSA1204- Python Programming***

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

## ***SCSA1204- Python Programming***

### **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

## SCSA1204- Python Programming

### Elif

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

#### Example

```
mark = 55

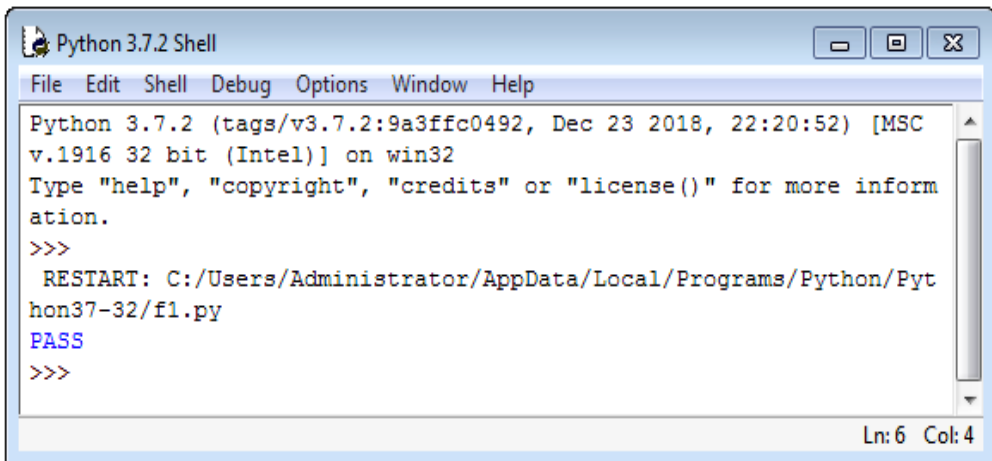
if (mark >=75):

print("FIRST CLASS")

elif mark >= 50:

print("PASS")
```

#### Output :



```
Python 3.7.2 Shell
File Edit Shell Debug Options Window Help
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC
v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more inform
ation.
>>>
RESTART: C:/Users/Administrator/AppData/Local/Programs/Python/Pyt
hon37-32/f1.py
PASS
>>>
Ln: 6 Col: 4
```

**Fig20** **elif** keyword

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.

### **Else**

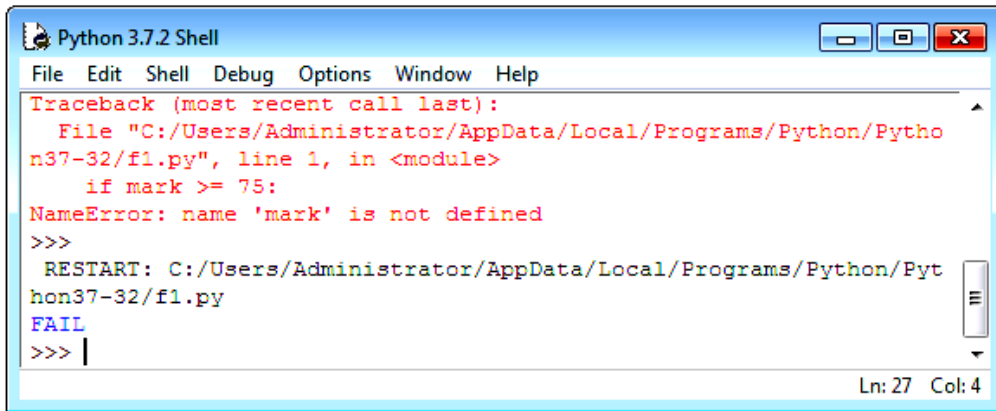
The **else** keyword will be used as a default condition. i.e. When there are many conditions, when the **if-condition** is not true and all **elif-conditions** are also not true, then **else** part will be executed..

#### Example

```
mark = 10

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

## SCSA1204- Python Programming



```
Python 3.7.2 Shell
File Edit Shell Debug Options Window Help
Traceback (most recent call last):
  File "C:/Users/Administrator/AppData/Local/Programs/Python/Python37-32/f1.py", line 1, in <module>
    if mark >= 75:
NameError: name 'mark' is not defined
>>>
RESTART: C:/Users/Administrator/AppData/Local/Programs/Python/Python37-32/f1.py
FAIL
>>> |
```

Ln: 27 Col: 4

**Fig 21** else keyword

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

### 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.

## ***SCSA1204- Python Programming***

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

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  
.....")
```

Types of looping statements

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

### **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 >
```



## SCSA1204- Python Programming

### Flow Chart:

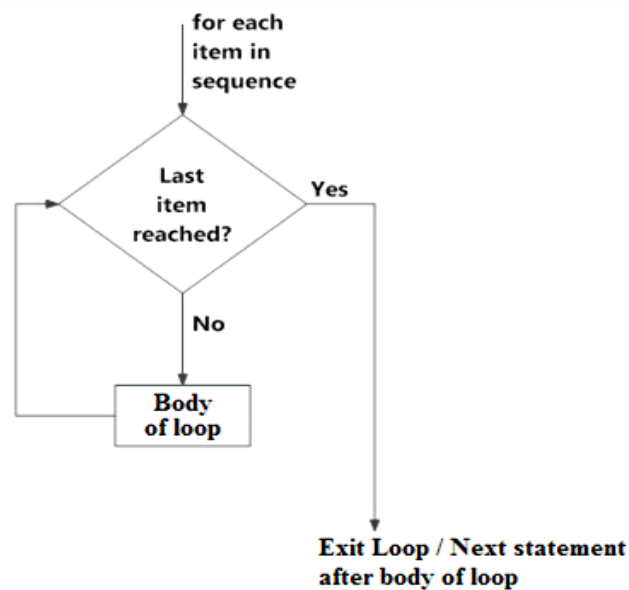


Fig 22 for Flow Chart

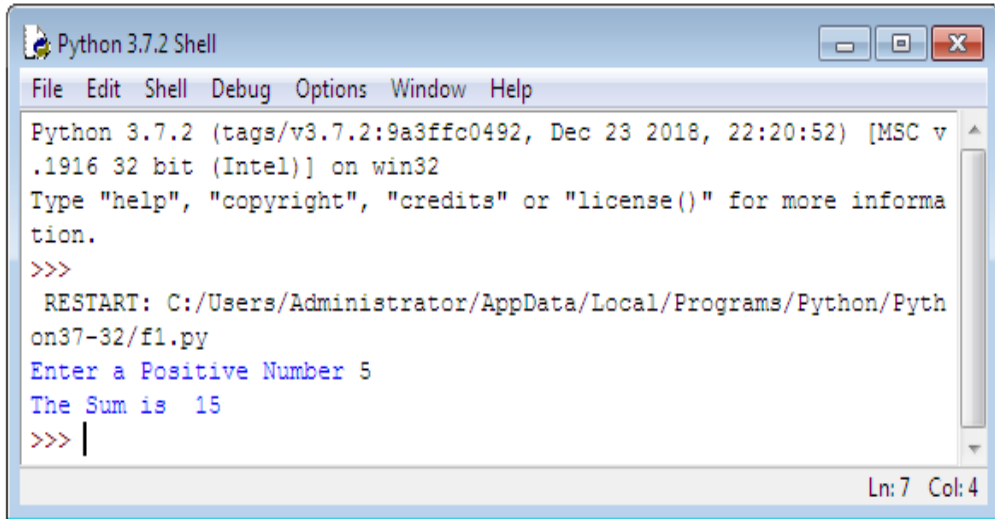
**Example 1:** To compute the sum of first n numbers (i.e.  $1 + 2 + 3 + \dots + n$ )

```
# 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.

## SCSA1204- Python Programming

### Output:

A screenshot of a Python 3.7.2 Shell window. The window has a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area shows the following output:

```
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v
.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more informa
tion.
>>>
  RESTART: C:/Users/Administrator/AppData/Local/Programs/Python/Pyth
on37-32/f1.py
Enter a Positive Number 5
The Sum is 15
>>> |
```

The status bar at the bottom right indicates 'Ln: 7 Col: 4'.

Fig 23statement total = total + i

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.

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 7 range()**

Type	Example	Explanation
------	---------	-------------

### ***SCSA1204- Python Programming***

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

**Example:**To compute Harmonic Sum (ie:  $1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \dots + \frac{1}{n}$ )

```
# harmonic.py
```

```
total = 0
```

```
n= int(input("Enter a Positive Integer:"))
```

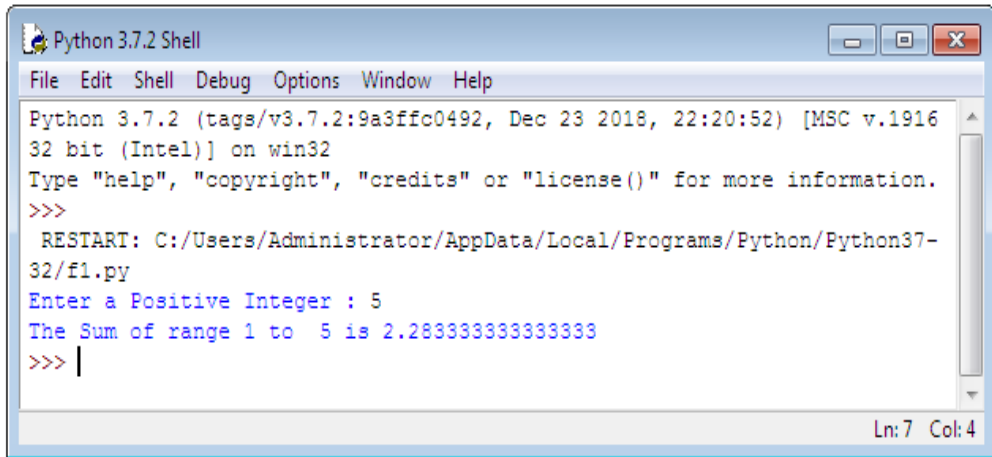
```
for i in range(1,n+1):
```

```
total+= 1/i
```

```
print("The Sum of range 1 to ",n, "is", total)
```

**Output:**

## SCSA1204- Python Programming



```
Python 3.7.2 Shell
File Edit Shell Debug Options Window Help
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916
32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
RESTART: C:/Users/Administrator/AppData/Local/Programs/Python/Python37-
32/f1.py
Enter a Positive Integer : 5
The Sum of range 1 to 5 is 2.2833333333333333
>>> |
```

Fig 24compute Harmonic Sum

### 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:
```

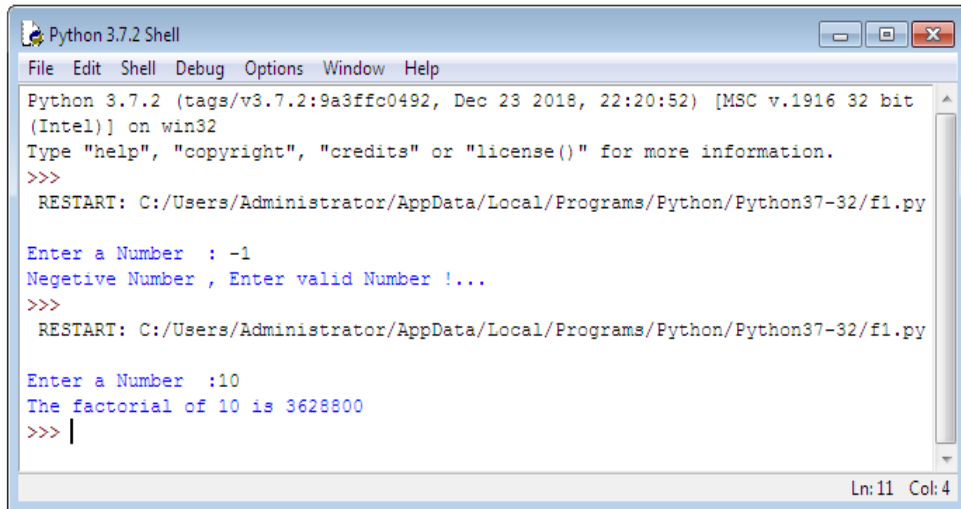
```
print("The factorial of 0 is 1")
```

```
else:
```

## ***SCSA1204- Python Programming***

```
for i in range(1,n + 1):  
    factorial = factorial*i  
print("The factorial of" ,n, "is", factorial)
```

### **Output:**



```
Python 3.7.2 Shell  
File Edit Shell Debug Options Window Help  
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit  
(Intel)] on win32  
Type "help", "copyright", "credits" or "license()" for more information.  
>>>  
RESTART: C:/Users/Administrator/AppData/Local/Programs/Python/Python37-32/f1.py  
  
Enter a Number : -1  
Negative Number , Enter valid Number !...  
>>>  
RESTART: C:/Users/Administrator/AppData/Local/Programs/Python/Python37-32/f1.py  
  
Enter a Number :10  
The factorial of 10 is 3628800  
>>> |  
Ln:11 Col:4
```

Fig 25factorial

### **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 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.

**Flow Chart:**

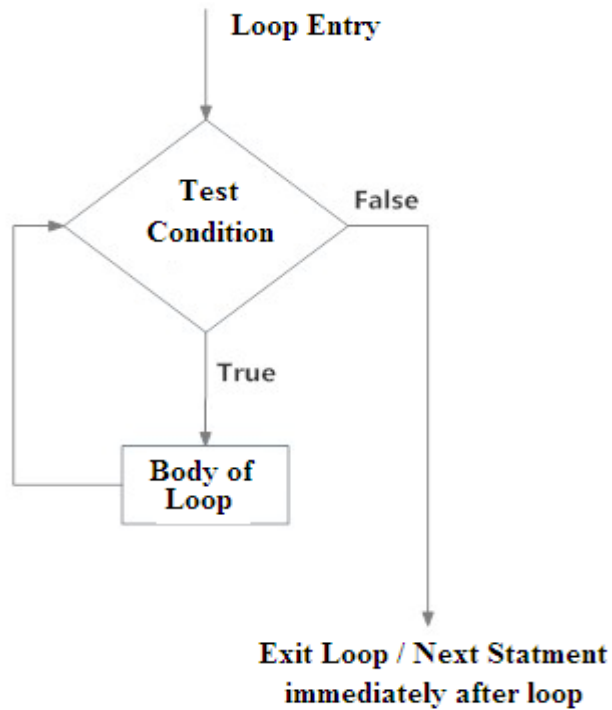


Fig 26 Flow chart while Loop

**Syntax:**

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

## SCSA1204- Python Programming

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.

Table 8 Example

Valid Example	Invalid Example
<pre>i = 10 while i&lt;15 :     print(i)     i = i + 1</pre> <p><b>Output :</b> 10,11,12,13,14</p>	<pre>i = 10 while i&lt;15 :     print(i)</pre> <p><b>Output :</b> 10,10,10,10..... Indeterminate number of times</p>

**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
```

## ***SCSA1204- Python Programming***

```
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
```

### **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



## ***SCSA1204- Python Programming***

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.

***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
```

### ***Output Statement:***

## ***SCSA1204- Python Programming***

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)
```

## *SCSA1204- Python Programming*

**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&
```

**Output formatting**

Sometimes we would like to format our output to make it look attractive. This can be done by using the `str.format()` method. This method is visible to any string object.

```
>>>x = 5; y = 10
>>>print('The value of x is {} and y is {}'.format(x,y))
The value of x is5and y is10
```

## ***SCSA1204- Python Programming***

Here, the curly braces {} are used as placeholders. We can specify the order in which they are printed by using numbers (tuple index).

```
print('I love {0} and {1}'.format('bread','butter'))
```

```
print('I love {1} and {0}'.format('bread','butter'))
```

### **Output**

I love bread and butter

I love butter and bread

We can even use keyword arguments to format the string.

```
>>> print('Hello {name}, {greeting}'.format(greeting =  
'Goodmorning', name = 'John'))
```

Hello John, Goodmorning

### **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, MethodOverloading 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.

## ***SCSA1204- Python Programming***

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.

<b>Class Name: Student</b>
<b>Data Fields:</b>
Name,
Mark1, Mark2, Mark3
<b>Methods:</b>
Average ()
Rank ()

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.

## ***SCSA1204- Python Programming***

### **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):
```

## ***SCSA1204- Python Programming***

```
        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.

### **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.

### **Example :**

## ***SCSA1204- Python Programming***

```
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

```
>>>
```



## ***SCSA1204- Python Programming***

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()
```

## ***SCSA1204- Python Programming***

```
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()`.

### **Questions :**

1. Compare a) List and Tuple b) List and Set
2. What is type conversion in Python?

### ***SCSA1204- Python Programming***

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.

***SCSA1204- Python Programming***



**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](http://www.sathyabama.ac.in)

**SCHOOL OF COMPUTING**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**UNIT-II Python Programming – SCSA1204**

## UNIT II

### FILES AND EXCEPTIONS HANDLING, MODULES, PACKAGES

FileOperations–Iterators-Exceptionhandling-RegularExpressions-Creating  
Modules-Import Statement-Introduction to PIP-Installing Packages via PIP-  
Using Python 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:**

## ***SCSA1204- Python Programming***

- 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***

**Table 2.1:FileAccess 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.

## ***SCSA1204- Python Programming***

### ***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.

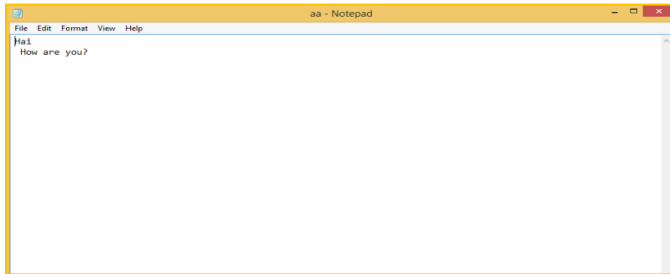
```
fo=open('aa.txt','w')

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

fo.close()
```

## SCSA1204- Python Programming

### 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?
...
```



## ***SCSA1204- Python Programming***

### **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

## SCSA1204- Python Programming

```
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
'''
```

## ***SCSA1204- Python Programming***

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  
kdsdfa  
dsafldk  
kdafsljff'''
```

### ***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  
...
```

## ***SCSA1204- Python Programming***

### **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

## SCSA1204- Python Programming

- `__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.

## SCSA1204- Python Programming

```
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:

There are two Arguments in infinite iterators:

## SCSA1204- Python Programming

- 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

## ***SCSA1204- Python Programming***

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 is now 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



## ***SCSA1204- Python Programming***

string is yielded from the last character(len-1) to -1(0<sup>th</sup>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)

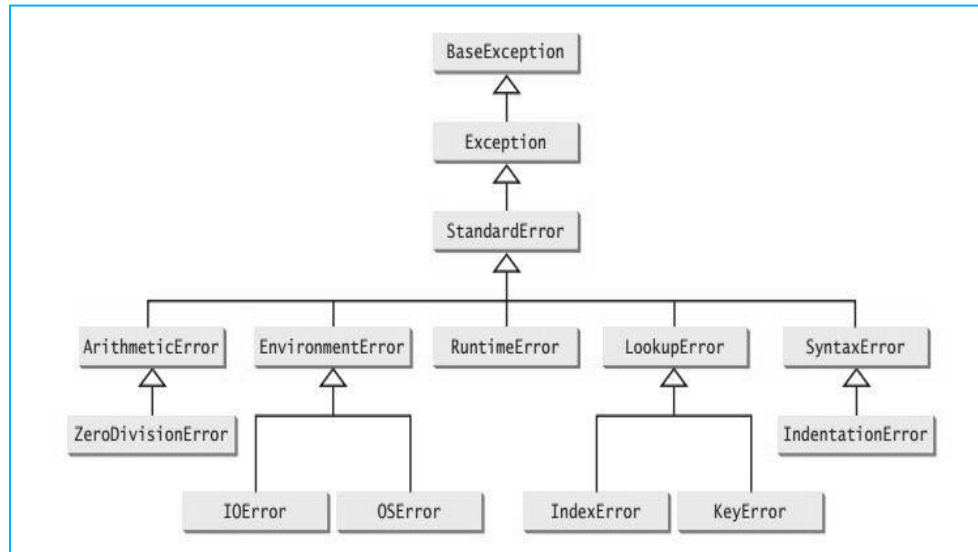
'''output
o
l
l
e
h'''
```

### ***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
```

## ***SCSA1204- Python Programming***

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.

## SCSA1204- Python Programming

### 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 output is displayed.

```
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
'''
```

## ***SCSA1204- Python Programming***

### ***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***

## SCSA1204- Python Programming

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

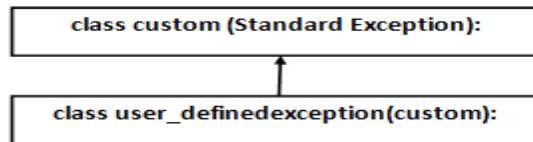
```
Raise RuntimeError('Something Wrong')
```

```
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.

## SCSA1204- Python Programming

```
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
```



## ***SCSA1204- Python Programming***

### **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: 2.2 Python Character Matching**

<b>Matching Character</b>	<b>Description</b>
[ ]	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 no. of match
{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
\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:**

## SCSA1204- Python Programming

```
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.

## SCSA1204- Python Programming

### 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.

## SCSA1204- Python Programming

### 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 2.3 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 2.4 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
'''|
```

## ***SCSA1204- Python Programming***

### **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***

## SCSA1204- Python Programming

```
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 item in pdata:
    print(item)

'''
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:**



## SCSA1204- Python Programming

```
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:**

## ***SCSA1204- Python Programming***

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 IMPORT STATEMENT**

Python import statement enables the user to import particular modules in the corresponding program. Import in Python is similar to #include header file in C/C++. Python modules can get access to code from another module by importing the file/function using import. The import statement is the most

## ***SCSA1204- Python Programming***

common way of invoking the import machinery, but it is not the only way. Import statement consists of the import keyword along with the name of the module.

The import statement involves two operations, it searches for a module and it binds the result of the search to name in local scope. When a module is imported, Python runs all of the code in the module file and made available to the importer file.

When a module is imported then interpreter first searches it in `sys.modules`, which is the cache of all modules which have been previously imported. If it is not found then it searches in all built-in modules with that name, if it is found then interpreter runs all of the code and made available to file. If the module is not found then it searches for a file with the same name in the list of directories given by the variable `sys.path`. `sys.path` is a variable containing a list of paths that contains python libraries, packages, and a directory containing the input script. For example a module named `math` is imported then interpreter search it in a built-in modules, if it is not found then it searches for a file named `math.py` in list of directories given by `sys.path`. It searches for a particular module in its built-in modules section at first. If it's not found, it searches those modules in its current directory.

**A module is loaded only once in a particular program, without being affected by the number of times the module is imported.**

## ***SCSA1204- Python Programming***

### **Syntax:**

```
import module_name
```

### **Example:**

```
import collections
```

### **1. Importing class/functions from a module**

### **Example:**

```
from collections import OrderedDict  
from os import path  
from math import pi  
print(pi)
```

### **Output:**

```
3.141592653589793
```

### **2. The *import \** Statement**

All the methods and constants of a particular module can be imported using `import *` operator.

```
from math import *  
print(pi)  
print(floor(3.15))
```

### **Output:**

```
3.141592653589793  
3
```

## ***SCSA1204- Python Programming***

### **3. Python's *import as* Statement**

The **import as** statement helps the user provide an alias name to the original module name.

```
# python import as
import math as M
print(M.pi)
print(M.floor(3.18))
```

#### **Output:**

```
3.141592653589793
3
```

### **4. Importing user-defined modules**

We can import the functions of one program into another using its name. Initially, we need to create a python code.

#### **test.py:**

```
def sub(a, b):
    return int(a) - int(b)

def lower_case(str1):
    return str(str1).lower()
```

Then create another python script, wherein we need to import the above create test.py script.

#### ***test2.py***

```
import test
print(test.sub(5,4))
```

## ***SCSA1204- Python Programming***

```
print(test.lower_case('SafA'))
```

### **Output:**

```
1  
safa
```

## **5. Importing from another directory**

The **importlib** library is used to import a script from another directory. Initially, we need to create a python script and define functions in it.

### ***test1.py***

```
defsub(a, b):  
    returnint(a) -int(b)
```

```
deflower_case(str1):  
    returnstr(str1).lower()
```

Then, we will create another python script and save it into another directory and then import the functionalities from test1.py (which resides into another directory).

### ***design.py***

```
importimportlib, importlib.util
```

```
defmodule_directory(name_module, path):
```

```
    P =importlib.util.spec_from_file_location(name_module, path)
```

## ***SCSA1204- Python Programming***

```
import_module =importlib.util.module_from_spec(P)

P.loader.exec_module(import_module)

returnimport_module
```

```
result =module_directory("result", "../inspect_module/test1.py")

print(result.sub(3,2))

print(result.lower_case('SaFa'))
```

### **Output:**

```
1
safa
```

Another alternative way is to add the module directory to the **sys.path** list.

## **6. Importing class from another file**

*tests.py*

```
classEmployee:
    designation =""

    def__init__(self, result):
        self.designation =result

    defshow_designation(self):
        print(self.designation)

classDetails(Employee):
```

## ***SCSA1204- Python Programming***

```
id=0
def __init__(self, ID, name):
    Employee.__init__(self, name)
    self.id=name

def get_Id(self):
    return self.id
```

### ***design.py***

```
import importlib, importlib.util
```

```
def module_directory(name_module, path):
    P = importlib.util.spec_from_file_location(name_module, path)
    import_module = importlib.util.module_from_spec(P)
    P.loader.exec_module(import_module)
    return import_module
```

```
result = module_directory("result", "../Hello/tests.py")
```

```
a = result.Employee('Project Manager')
a.show_designation()
```

```
x = result.Details(4001, 'Safa')
x.show_designation()
print(x.get_Id())
```

### **Output:**

```
Project Manager
Safa
Safa
```



## ***SCSA1204- Python Programming***

### **2.7 INTRODUCTION TO PIP**

In order to manage and install software packages Python uses 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.

#### **2.7.1. Installing Packages via PIP**

##### ***2.7.1.1 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.

##### ***2.7.1.2. 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#](https://www.onlinegdb.com/online_python_compiler#)'.

### **2.8. USING PYTHON PACKAGES FOR ADVANCED PROGRAMMING**

#### **2.8.1. 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.

## ***SCSA1204- Python Programming***

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

### **2.8.2. Python Libraries for running real time projects**

#### ***2.8.2.1.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 2.5: Universal Functions in Numpy**

Function name	Purpose	Example
np.array()	For creating arrays	a = np.array([0, 1, 2, 3])
np.arange()	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 2.6: 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)

## ***SCSA1204- Python Programming***

		[[10 11 12 13 14] [15 16 17 18 19] [20 21 22 23 24]]
--	--	--

### ***2.8.2.2. 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
from scipy.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
- File input/output
- Special Function

## ***SCSA1204- Python Programming***

- Linear Algebra Operation
- Numerical Integration
- Fast Fourier transforms

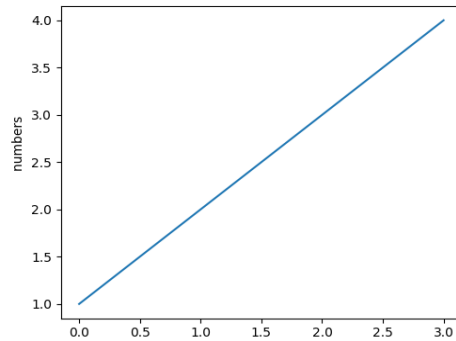
### ***2.8.2.2. matplotlib***

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()
```



**Fig.2.3 Output**

***SCSA1204- Python Programming***



**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](http://www.sathyabama.ac.in)

**SCHOOL OF COMPUTING**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**UNIT-III Python Programming – SCSA1204**

## **GUI PROGRAMMING WITH PYTHON**

Conditional statements: Boolean values and operators, conditional (if), alternative (if-else), chained conditional (if-elif-else); Iterative statements: while, for, break, continue, pass, Applying for critical conditions.

Algorithms: square root, gcd, exponentiation, sum an array of numbers, factorial computation-Sine function computation-Generation of the Fibonacci sequence-Reversing the digits of an integer.

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

### **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

## ***SCSA1204- Python Programming***

- 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

## **Layout Management**



## ***SCSA1204- Python Programming***

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

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

### **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:***

```
fromtkinter 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()
```

## ***SCSA1204- Python Programming***

### ***Output:***



***Explanation:*** Label 11 has been placed in top position, it is filled in X axis. Label 12 has been placed in Right Position and it is also filled in X axis. Since expand attribute has the value True for Label 11, 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:***

```
fromtkinter 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)
```

## ***SCSA1204- Python Programming***

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

```
top.mainloop()
```

***Output:***



**External Padding in Vertical direction (pady)**

***Example:***

```
fromtkinter 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:***

## *SCSA1204- Python Programming*



### **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:*

## *SCSA1204- Python Programming*

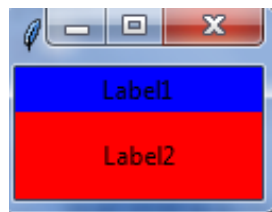


### **Internal Padding in Y Direction(ipady):**

#### *Example:*

```
fromtkinter 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:*



## ***SCSA1204- Python Programming***

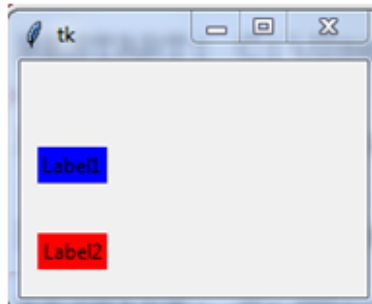
### **Place Layout:**

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:***

```
fromtkinter 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:***



### ***Explanation:***

## ***SCSA1204- Python Programming***

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

### **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:***



## ***SCSA1204- Python Programming***

### ***Explanation:***

Here Label 1 is placed in 0<sup>th</sup> row and 0<sup>th</sup> column. Label 2 is placed in 0<sup>th</sup> row and 1<sup>st</sup> column and Label 3 is placed in 1<sup>st</sup> row and 1<sup>st</sup> column.

## **FONT**

There are three ways to specify font in python.

- 1.By using Font Tuple
- 2.By using Font Object
- 3.By using XFont

### **Simple Font Tuple:**

Font can be specified using tuple. Here the font tuple consists of three elements. First element specifies font family, second element specifies font size and third element specifies font style.

Ex: t=("Arial",14,"Bold")

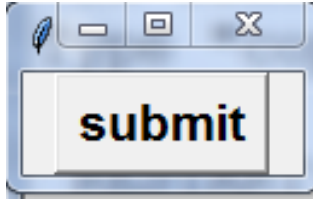
### ***Example:***

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

### ***Output:***



## ***SCSA1204- Python Programming***



Explanation:

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

### **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.                             |

## ***SCSA1204- Python Programming***

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

underline            – 1 for underlined text, 0 for normal.

Overstrike          – 1 for overstruck text, 0 for normal

### ***Example:***

```
fromtkinter import *  
  
fromtkFont 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()
```

### **X Window Fonts:**

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

### **COLORS**

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

## ***SCSA1204- Python Programming***

- 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 :

## ***SCSA1204- Python Programming***

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.

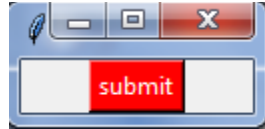
### ***Example:***

```
fromtkinter import *
```

## ***SCSA1204- Python Programming***

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

***Output:***



***Explanation:***

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

## **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:

## ***SCSA1204- Python Programming***

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

### **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:

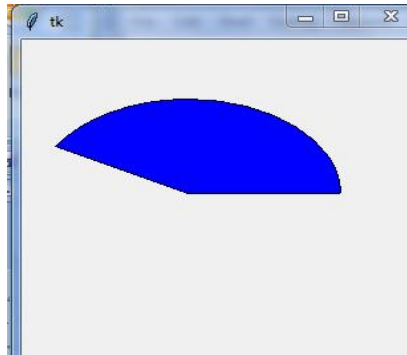
start,extend-Specifies which section to draw

## ***SCSA1204- Python Programming***

### ***Example:***

```
fromtkinter 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:***



### ***Explanation:***

## ***SCSA1204- Python Programming***

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.

### **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:***

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



## ***SCSA1204- Python Programming***

```
root.mainloop()
```

### **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:***

```
fromtkinter import *
```

```
root=Tk()
```

```
w = Canvas(root, width=500, height=500)
```

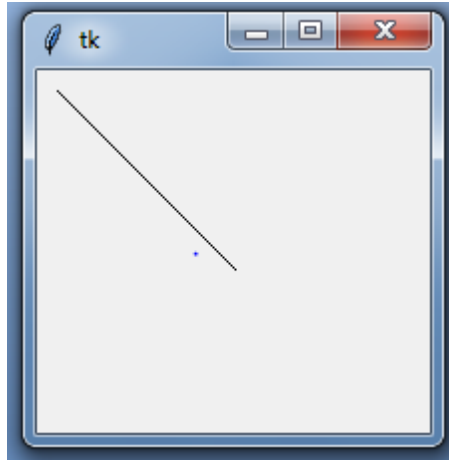
```
w.create_line(10,10,100,100,activefill="red")
```

```
w.pack()
```

```
root.mainloop()
```

#### ***Output:***

## ***SCSA1204- Python Programming***



### **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

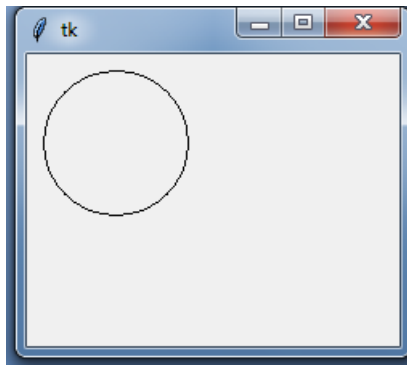
width -Width of the line

## ***SCSA1204- Python Programming***

### ***Example:***

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

### ***Output:***



## **POLYGON**

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

### ***Syntax:***

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

## ***SCSA1204- Python Programming***

$x_0, y_0, x_1, y_1, \dots, x_n, y_n$  - Coordinates of polygon

Options:

Activefill-Color of the oval when it is active

width -Width of the line

### ***Example***

```
fromtkinter 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()
```

## **WIDGETS IN PYTHON**

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

### **LABEL**

## ***SCSA1204- Python Programming***

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 below:

<b>Option</b>	<b>Description</b>
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

## ***SCSA1204- Python Programming***

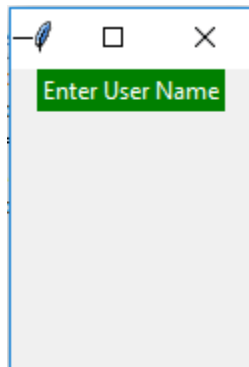
relief	Specifies type of border
text	Text to be displayed in the widget
underline	Underline the label text

*Table 1:Widget*

### ***Example:***

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

### ***Output:***



### ***Explanation:***

## ***SCSA1204- Python Programming***

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

### **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 below:

<b>Option</b>	<b>Description</b>
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

### ***SCSA1204- Python Programming***

<b>Option</b>	<b>Description</b>
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
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

*Table 2: Widget*

#### ***Example:***

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

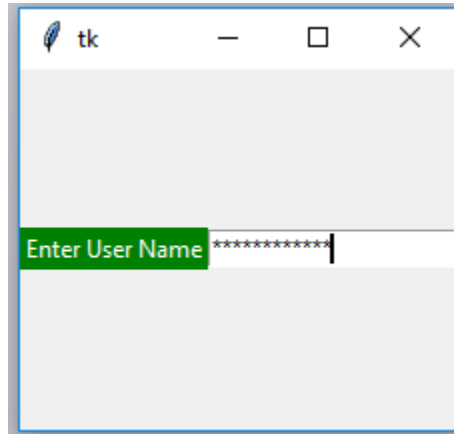


## ***SCSA1204- Python Programming***

```
e1.pack(side=RIGHT)
```

```
root.mainloop()
```

***Output:***



***Explanation:***

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

## **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:

## ***SCSA1204- Python Programming***

b=Button(root,options)

root-main window

List of commonly used options are given below:

<b>Option</b>	<b>Description</b>
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
underline	Underline the label text
command	It is set to function name which will be called the button is clicked

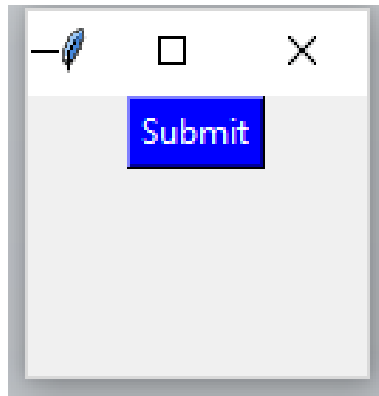
*Table 3: Budget Widget*

## ***SCSA1204- Python Programming***

### ***Example:***

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

### ***Output:***



## **Checkbutton**

The Checkbutton is used to track the user's choices provided to the application. Checkbutton is used to implement the on/off selections. The Checkbutton 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:

## ***SCSA1204- Python Programming***

`b=CheckButton(root,options)`

root-main window

List of possible options are given below:

<b>Option</b>	<b>Description</b>
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
underline	Underline the label text
command	It is set to function name which 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.

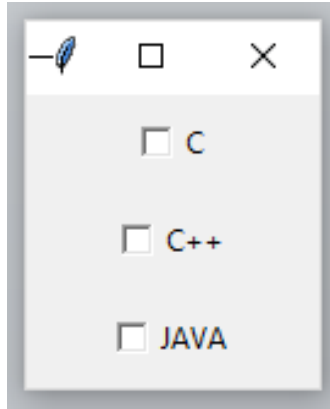
## ***SCSA1204- Python Programming***

### ***Example:***

```
fromtkinter 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:***

## ***SCSA1204- Python Programming***



### **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, the radiobutton is associated with a single variable. Each Radio button displays a single value for that particular variable.

Syntax for creating Radio Button:

```
b=RadioButton(root,options)
```

root-main window

List of possible options are given below:

### ***SCSA1204- Python Programming***

<b>Option</b>	<b>Description</b>
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
underline	Underline the label text
command	It is set to function name which 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.

## ***SCSA1204- Python Programming***

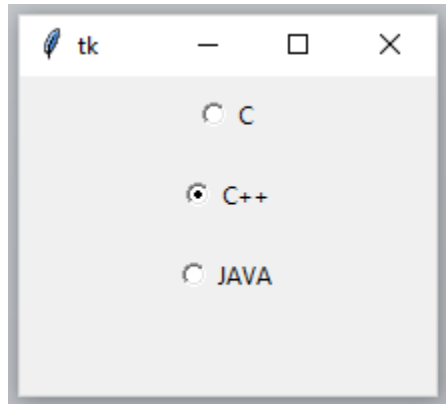
### ***Example:***

```
fromtkinter 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:***



## SCSA1204- Python Programming



### 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 creating Listbox:

```
b=Listbox(root,options)
```

root-main window

List of possible options are given below:

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

### ***SCSA1204- Python Programming***

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:***

```
fromtkinter import *  
  
top = Tk()  
  
lbl = Label(top,text = "A list of favourite countries...")  
  
listbox = Listbox(top)  
  
listbox.insert(1,"India")
```

### ***SCSA1204- Python Programming***

```
listbox.insert(2, "USA")
```

```
listbox.insert(3, "Japan")
```

```
listbox.insert(4, "Austrelia")
```

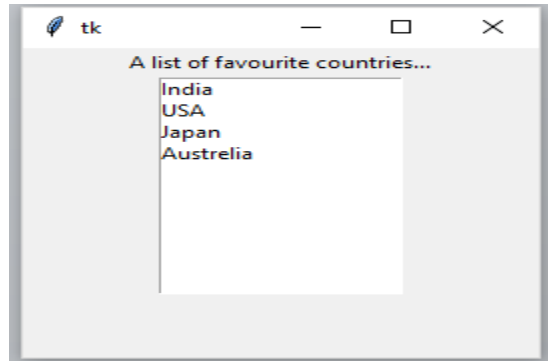
```
lbl.pack()
```

```
listbox.pack()
```

```
top.mainloop()
```

## SCSA1204- Python Programming

**Output:**



### 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 possible options are given below:

Option	Description
bg	Specifies background color of the widget
bd	Specifies border width. Default is 2 pixels

### ***SCSA1204- Python Programming***

<b>Option</b>	<b>Description</b>
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

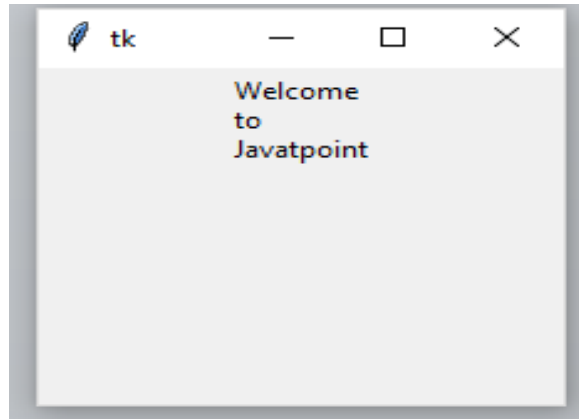
#### ***Example:***

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



## ***SCSA1204- Python Programming***

***Output:***



### **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 possible options are given below:

### ***SCSA1204- Python Programming***

<b>Option</b>	<b>Description</b>
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:

<b>Method</b>	<b>Description</b>
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



## ***SCSA1204- Python Programming***

	range.
<code>insert(index, string)</code>	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 below:

<b>Method</b>	<b>Description</b>
<code>mark_set(mark, index)</code>	It is used to create mark at the specified index.
<code>mark_unset(mark)</code>	It is used to clear the given mark
<code>mark_names()</code>	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 below:

<b>Method</b>	<b>Description</b>
<code>tag_add(tagname, startindex, endindex)</code>	It is used to tag the characters in the given range
<code>tag_config()</code>	It is used to configure the tag properties
<code>tag_delete(tagname)</code>	It is used to delete the given tag

## ***SCSA1204- Python Programming***

<code>tag_remove(tagname, startindex, endindex)</code>	It is used to remove the tag from the specified range
--	---

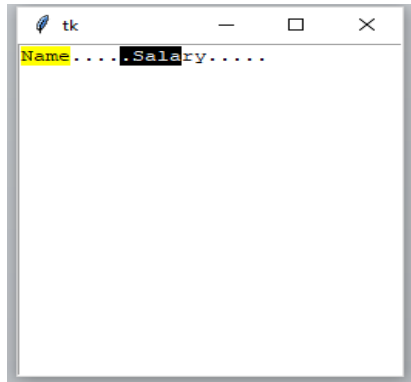
*Table :Tag Handling*

### ***Example:***

```
fromtkinter 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:***

## ***SCSA1204- Python Programming***



### ***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()`.

## **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 possible options are given below:

### ***SCSA1204- Python Programming***

<b>Option</b>	<b>Description</b>
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.

*Table :Spinbox*

#### ***Example:***

```
fromtkinter import *
```

```
top = Tk()
```

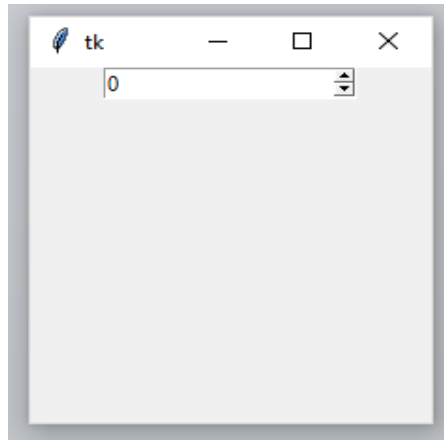
```
spin = Spinbox(top, from_ = 0, to = 25)
```

```
spin.pack()
```

## ***SCSA1204- Python Programming***

`top.mainloop()`

***Output:***



### **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 possible options are given below:

## ***SCSA1204- Python Programming***

<b>Option</b>	<b>Description</b>
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

*Table: Frame Widget*

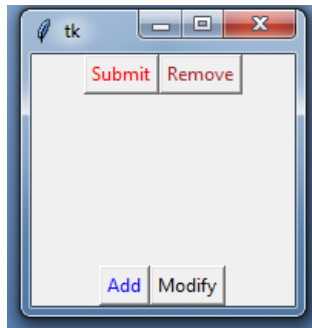
### ***Example:***

```
fromtkinter 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)
```

## ***SCSA1204- Python Programming***

```
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:***



### ***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 .

## **EVENTS AND BINDINGS IN PYTHON**

## ***SCSA1204- Python Programming***

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.

### **HANDLING MOUSE BUTTON EVENT IN PYTHON**

#### ***Example:***

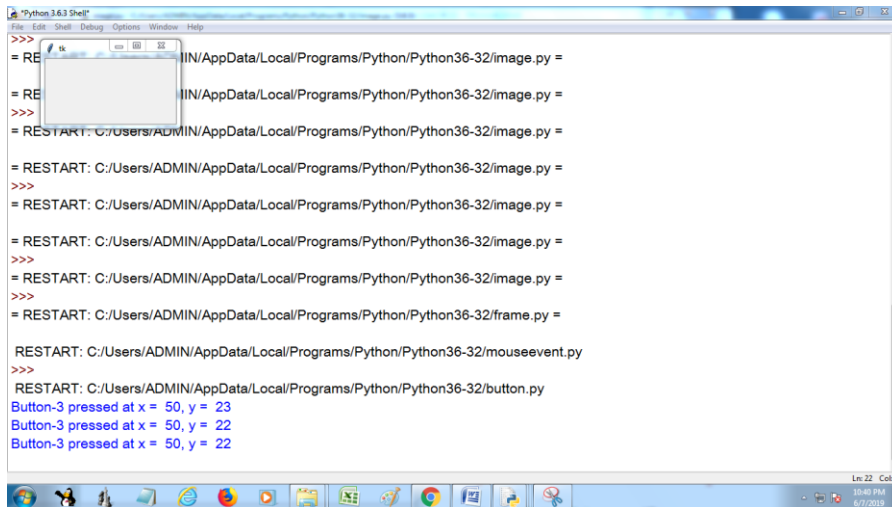
```
fromtkinter import *
fromtkinter.ttk import *
# creates tkinter window or root window
root = Tk()
# function to be called when button-2 of mouse is pressed
def pressed2(event):
    print('Button-2 pressed at x = % d, y = % d'%(event.x, event.y))
# function to be called when button-3 of mouse is pressed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



## ***SCSA1204- Python Programming***

```
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:**



```
>>>  
= RESTART: C:/Users/ADMIN/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/image.py =  
= RESTART: C:/Users/ADMIN/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/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
```

## **HANDLING KEY PRESS EVENT IN PYTHON**

## ***SCSA1204- Python Programming***

### ***Example:***

```
fromtkinter import *

fromtkinter.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()

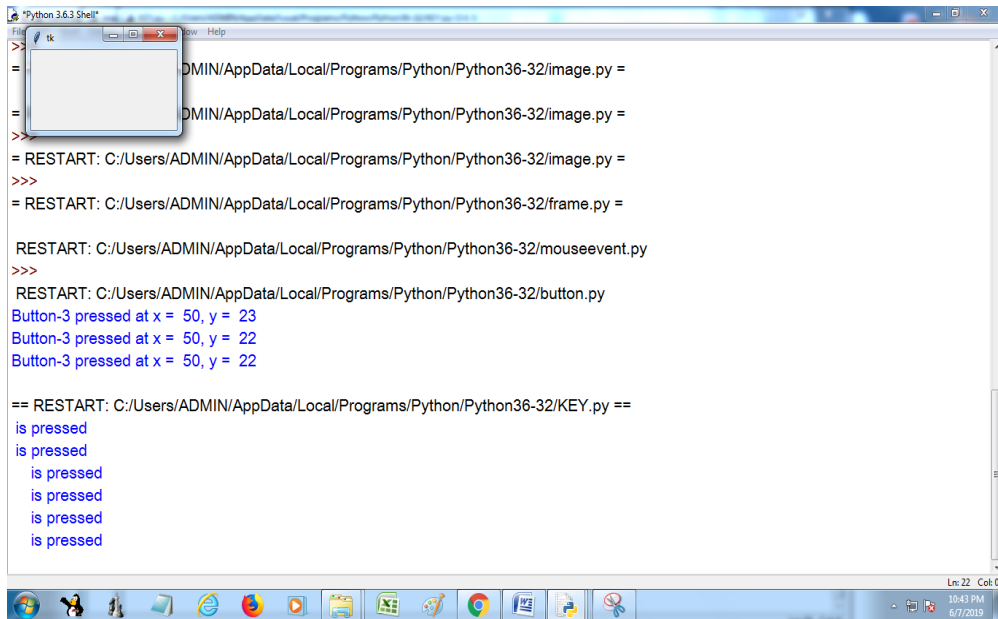
root.geometry('200x100')

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

mainloop()
```

### ***Output:***

## SCSA1204- Python Programming



The screenshot shows a Windows desktop with a taskbar at the bottom. The taskbar contains icons for the Start menu, Task View, File Explorer, Google Chrome, and several other applications. The active window is titled "Python 3.6.3 Shell". It has a menu bar with "File", "Edit", and "Help". The shell window displays a series of commands and their outputs. A small "tk" window is visible in the foreground, partially obscuring the shell. The shell output shows several "RESTART:" commands followed by file names, and a series of "Button-3 pressed" messages with coordinates (50, 23) and (50, 22). The status bar at the bottom right of the shell window shows "Ln 22 Col 0".

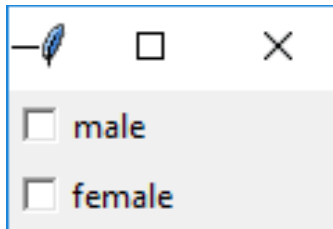
```
>>> tk
= RESTART: C:/Users/ADMIN/AppData/Local/Programs/Python/Python36-32/image.py =
= RESTART: C:/Users/ADMIN/AppData/Local/Programs/Python/Python36-32/image.py =
>>>
= RESTART: C:/Users/ADMIN/AppData/Local/Programs/Python/Python36-32/frame.py =
>>>
RESTART: C:/Users/ADMIN/AppData/Local/Programs/Python/Python36-32/mouseevent.py
>>>
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
```

## ***SCSA1204- Python Programming***

### **QUESTIONS**

1. Write the Python 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 python Program to create simple calculator.
8. Write a Python Program to add image on the button.
9. Write a Python program to create simple application form.
10. Write a Python program to create check button for selecting multiple hobbies.

***SCSA1204- Python Programming***



**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](http://www.sathyabama.ac.in)

**SCHOOL OF COMPUTING**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**UNIT-IV Python Programming – SCSA1204**

## **DATABASE AND NETWORK**

Database(usingNoSQL):ConnectorModule–Cursor–Statements-  
Exceptionsin database.Networkconnectivity:Socketmodule-  
Client–Server–Email–URLAccess.

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.

### **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 can be stores 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

## ***SCSA1204- Python Programming***

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

Types of databases used with Python

### **1. Databasesupport**

- SQL
- NoSQL

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 is interacts with Relational databases. In this chapter we will use python to interact with MongoDB as a NoSQL database.

### **MongoDB**

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>.

## ***SCSA1204- Python Programming***

### **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:

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:



## ***SCSA1204- Python Programming***

```
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.

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

### Example

Create a database called mydatabase

```
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).

### Creating a Collection

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

## ***SCSA1204- Python Programming***

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.**

### 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"]
```

## ***SCSA1204- Python Programming***

```
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 the `insert_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"},
```

## *SCSA1204- Python Programming*

```
{ "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)
```

### 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/")
```

## ***SCSA1204- Python Programming***

```
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"]
```

## ***SCSA1204- Python Programming***

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'}
```

### 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":

## ***SCSA1204- Python Programming***

```
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" } }
```

## ***SCSA1204- Python Programming***

```
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'}
```

**Fig No 1**

### **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:



## ***SCSA1204- Python Programming***

```
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'}
```

### **Sort the Result**

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

## ***SCSA1204- Python Programming***

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'}
```

## *SCSA1204- Python Programming*

```
{'_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'}
```

## *SCSA1204- Python Programming*

```
{'_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'}
```

### 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"]
```

## ***SCSA1204- Python Programming***

```
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 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" } }
```

```
x = mycol.delete_many(myquery)
```

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

output

**2 documents deleted.**

## ***SCSA1204- Python Programming***

### **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**

### **Python MongoDB Drop Collection**

#### **Delete Collection**

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

## *SCSA1204- Python Programming*

### 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.

### 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":

## ***SCSA1204- Python Programming***

```
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"]
```

## ***SCSA1204- Python Programming***

```
myquery = { "address": { "$regex": "^S" } }
newvalues = { "$set": { "name": "Minnie" } }
x = mycol.update_many(myquery, newvalues)

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

Output

**2 documents updated.**

Python MongoDB Limit

o 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' }
```

## ***SCSA1204- Python Programming***

```
{'_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'}
```

## ***SCSA1204- Python Programming***

### **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 query)
```

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

The resultant rows retrieved 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
```

### ***SCSA1204- Python Programming***

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=conn.cursor()
```

```
cursor.execute("select * from stutab")
```

```
row = cursor.fetchone()
```

```
while row is not None:
```

```
    print(row)
```

```
    row=cursor.fetchone()
```

```
cursor.close()
```

```
conn.close()
```

## ***SCSA1204- Python Programming***

Output:

(1001, 'Ajay', 8.5)

(1002, 'Alan', 7.5)

(1001, 'Joe', 9.00)

### **Exceptions Classes**

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

Table No 1

### **Built in Exceptions**

<b>Exception</b>	<b>Description</b>
<b>Warning</b>	Used for non-fatal issues. Must subclass StandardError.
<b>Error</b>	Base class for errors. Must subclass StandardError.
<b>InterfaceError</b>	Used for errors in the database module, not the database itself. Must subclass Error.
<b>DatabaseError</b>	Used for errors in the database. Must subclass Error.
<b>DataError</b>	Subclass of DatabaseError that refers to errors in

## ***SCSA1204- Python Programming***

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

## **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 videos, Telephone calls and video conferences etc among them.

## ***SCSA1204- Python Programming***

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)**



## ***SCSA1204- Python Programming***

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

### **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).

## ***SCSA1204- Python Programming***

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

## ***SCSA1204- Python Programming***

- `s.close()` – It closes socket
- `socket.gethostname()` – It returns thehostname

### **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
```

## ***SCSA1204- Python Programming***

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)
```

### ***SCSA1204- Python Programming***

```
# 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()
```

## ***SCSA1204- Python Programming***

### **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'
```

### ***SCSA1204- Python Programming***

```
# 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
```

## ***SCSA1204- Python Programming***

```
# 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
```

Sending Email using SMTP



## ***SCSA1204- Python Programming***

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.

## ***SCSA1204- Python Programming***

- 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:
```

## ***SCSA1204- Python Programming***

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 *smtpplib* 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 – `smtpplib.SMTP('mail.your-domain.com', 25)`



**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](http://www.sathyabama.ac.in)

**SCHOOL OF COMPUTING**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

## **UNIT-V Python Programming – SCSA1204**

<b>Case Study: Web Programming using Python. Image Processing – Facebook Analysis – Twitter Analysis</b>
--

**What is sentiment analysis?**

Sentiment analysis is one of the best modern branches of machine learning, which is mainly used to analyze the data in order to know one's own idea, nowadays it is used by many companies to their own feedback from customers.

**Why should we use sentiment analysis?**

- **Invaluable Marketing:**

Using sentiment analysis companies and product owners use can use sentiment analysis to know the demand and supply of their products through comments and feedback from the customers.

- **Identifying key emotional triggers:**

In psychology and other medical treatment institutions, sentiment analysis can be used to detect whether the individuals' emotion is normal or abnormal, and based on the data record they can decide person health.

- **Politics:**

In the political field, candidates to be elected can use sentiment analysis to predict their political status, to measure people's acceptance. It can also be used to predict election results for electoral board commissions.

- **Education:**

Universities and other higher institutes like colleges can use sentiment analysis to know their student's feedback and comment, therefore they can take consideration to revise or improve their education curriculum.

**Installations in Anaconda**

**NLTK:is used for understanding of human natural language.**

**Installation Using conda command.**

conda install -c anaconda nltk

✧ Installation Using pip.

pip install nltk

## ***SCSA1204- Python Programming***

**NumPy:** is a python package used for scientific and computational methods in python.

Installation Using conda.

```
conda install -c conda-forge numpy
```

Using pip.

```
pip install numpy
```

**Pandas:** is a python module used for data preprocessing and analysis .

Installation Using conda

```
conda install -c anaconda pandas
```

Installation Using pip.

```
pip install pandas
```

**Matplotlib:** is a python module used for data visualization and 2D plotting for representation of data.

Installation Using conda.

```
conda install -c conda-forge matplotlib
```

Installation Using pip.

```
pip install matplotlib
```

### **Authentication**

There are many ways to fetch Facebook comments those are:

- Facebook graph API
- Direct download from Facebook
- Downloading from another dataset provider sites

Among the above methods, we used downloading the Facebook comment dataset from the Kaggle website which is the best dataset provider. For the code we already used kindle.txt for analysis of kindle amazon facebook comment, you can use your own Facebook comment using this code to analyze your own comments or create a file in text format and try it for simplification.

## ***SCSA1204- Python Programming***

Below is the implementation.

```
importtime
importpandas as pd
importnumpy as np
importmatplotlib.pyplot as plt
importnlk
importio
importunicodedata
importnumpy as np
importre
importstring
fromnumpyimportlinalg
fromnlk.sentiment.vaderimportSentimentIntensityAnalyzer
fromnlk.tokenizeimportsent_tokenize, word_tokenize
fromnlk.tokenizeimportPunktSentenceTokenizer
fromnlk.tokenizeimportPunktSentenceTokenizer
fromnlk.corpusimportwebtext
fromnlk.stem.porterimportPorterStemmer
fromnlk.stem.wordnetimportWordNetLemmatizer
    with open('kindle.txt', encoding ='ISO-8859-2') as f:
        text =f.read()
        sent_tokenizer=PunktSentenceTokenizer(text)
sents=sent_tokenizer.tokenize(text)
    print(word_tokenize(text))
print(sent_tokenize(text))
    porter_stemmer=PorterStemmer()
    nltk_tokens=nltk.word_tokenize(text)
    forinnltk_tokens:
        print("Actual: % s Stem: % s"%(w, porter_stemmer.stem(w)))

wordnet_lemmatizer=WordNetLemmatizer()
nltk_tokens=nltk.word_tokenize(text)
```

```
forwinnltk_tokens:
    print("Actual: % s Lemma: % s"%(w, wordnet_lemmatizer.lemmatize(w)))
    text =nltk.word_tokenize(text)
print(nltk.pos_tag(text))
sid=SentimentIntensityAnalyzer()
tokenizer =nltk.data.load('tokenizers / punkt / english.pickle')
with open('kindle.txt', encoding ='ISO-8859-2') as f:
    fortextinf.read().split("\n"):
        print(text)
        scores =sid.polarity_scores(text)
        forkeyinsorted(scores):
            print('{0}: {1}', '.format(key, scores[key]), end ="")
            print()
```

### **Output:**

here is the sample output of the code:

['i', 'love', 'my', 'kindle']

['i love my kindle']

Actual: i Stem: i

Actual: love Stem: love

Actual: my Stem: my

Actual: kindle Stem: kindl

Actual: i Lemma: i

Actual: love Lemma: love

Actual: my Lemma: my

Actual: kindle Lemma: kindle

[('i', 'NN'), ('love', 'VBP'), ('my', 'PRP\$'), ('kindle', 'NN')]

i love my kindle

compound: 0.6369, neg: 0.0, neu: 0.323, pos: 0.677,

### **We follow these major steps in our program:**

- Downloading(fetching) facebook comment from Kaggle site and save it as text format.



## SCSA1204- Python Programming

- Preprocessing the data through SkLearn and nltk libraries .we first tokenize the data and then after tokenizing we stemize and lemmatize.
- Parse the comments using Vader library . Classify each comment as positive, negative or neutral.

**Now, let us try to understand the above piece of code:**

- First we open a file named kindle which is downloaded from Kaggle site and saved in local disk.

*with open('kindle.txt', encoding='ISO-8859-2') as f:*

- After we open a file we preprocess the text through tokenize, stemize and then lemmatize:

- Tokenize the text, i.e split words from text.

```
sent_tokenizer = PunktSentenceTokenizer(text)
```

```
sents = sent_tokenizer.tokenize(text)
```

```
print(word_tokenize(text))
```

```
print(sent_tokenize(text))
```

- Stemize and lematize the text for normalization of the text:

- 1) For stemize we use PorterStemmer() function:

```
from nltk.stem.porter import PorterStemmer
```

```
porter_stemmer = PorterStemmer()
```

```
nltk_tokens = nltk.word_tokenize(text)
```

```
for w in nltk_tokens:
```

```
    print ("Actual: %s Stem: %s" % (w, porter_stemmer.stem(w)))
```

- 2) For lematize we use WordNetLemmatizer() function :

```
from nltk.stem.wordnet import WordNetLemmatizer
```

```
wordnet_lemmatizer = WordNetLemmatizer()
```

```
nltk_tokens = nltk.word_tokenize(text)
```

```
for w in nltk_tokens:
```

```
    print ("Actual: %s Lemma: %s" % (w, wordnet_lemmatizer.lemmatize(w)))
```

- POS( 5t of speech) tagging of the tokens and select only significant features/tokens like adjectives, adverbs, and verbs, etc.

```
text = nltk.word_tokenize(text)
```

```
print(nltk.pos_tag(text))
```

- Pass the tokens to a sentiment intensity analyzer which classifies the Facebook comments as positive, negative or neutral.

**Here is how vader sentiment analyzer works:**

- VADER uses a combination of A sentiment lexicon which is a list of lexical features (e.g., words) which are generally labeled according to their semantic orientation as either positive or negative.
- sentiment analyzer not only tells about the Positivity and Negativity score but also tells us about how positive or negative a sentiment is.
- Then, We used the polarity\_scores() method to obtain the polarity indices for the given sentence.

Then, we build the comment intensity and polarity as:

```
sid = SentimentIntensityAnalyzer()
tokenizer = nltk.data.load('tokenizers/punkt/english.pickle')
with open('kindle.txt', encoding='ISO-8859-2') as f:
    for text in f.read().split('\n'):
        print(text)
        scores = sid.polarity_scores(text)
        for key in sorted(scores):
            print('{0}: {1}, '.format(key, scores[key]), end='')
        print()
```

Let us to understand what the sentiment code is and how VADER performs on the output of the above code:

i love my kindle

compound: 0.6369, neg: 0.0, neu: 0.323, pos: 0.677

- The Positive(pos), Negative(neg) and Neutral(neu) scores represent the proportion of text that falls in these categories. This means our sentence was rated as 67% Positive, 32% Neutral and 0% Negative. Hence all these should add up to 1.
- The Compound score is a metric that calculates the sum of all the lexicon ratings which have been normalized between -1( extreme negative) and +1 ( extreme positive).
- Finally, sentiment scores of comments are returned.

## TWITTER SENTIMENT ANALYSIS

**Tweepy:** tweepy is the python client for the official Twitter API.

Install it using following pip command:

```
pip install tweepy
```

**TextBlob:** textblob is the python library for processing textual data.

Install it using following pip command:

```
pip install textblob.
```

Also, we need to install some NLTK corpora using following command:

```
python -m textblob.download_corpora
```

(Corpora is nothing but a large and structured set of texts.)

### **Authentication:**

In order to fetch tweets through Twitter API, one needs to register an App through their twitter account. Follow these steps for the same:

- Log in to twitter and click the button: 'Create New App'
- Fill the application details. You can leave the callback url field empty.
- Once the app is created, you will be redirected to the app page.
- Open the 'Keys and Access Tokens' tab.
- Copy 'Consumer Key', 'Consumer Secret', 'Access token' and 'Access Token Secret'.

### **Implementation:**

```
import re
import tweepy
from tweepy import OAuthHandler
from textblob import TextBlob

class TwitterClient(object):

    """
    Generic Twitter Class for sentiment analysis.
    """
```

```
def __init__(self):
    """
    Class constructor or initialization method.
    """
    # keys and tokens from the Twitter Dev Console
    consumer_key='XXXXXXXXXXXXXXXXXXXXXXXXXXXX'
    consumer_secret='XXXXXXXXXXXXXXXXXXXXXXXXXXXX'
    access_token='XXXXXXXXXXXXXXXXXXXXXXXXXXXX'
    access_token_secret='XXXXXXXXXXXXXXXXXXXXXXXXXXXX'

    # attempt authentication
    try:
        # create OAuthHandler object
        self.auth=OAuthHandler(consumer_key, consumer_secret)
        # set access token and secret
        self.auth.set_access_token(access_token, access_token_secret)
        # create tweepy API object to fetch tweets
        self.api=tweepy.API(self.auth)
    except:
        print("Error: Authentication Failed")

def clean_tweet(self, tweet):
    """
    Utility function to clean tweet text by removing links, special characters
    using simple regex statements.
    """
    return ' '.join(re.sub("(@[A-Za-z0-9]+)|(^0-9A-Za-z \t)|
                        |(\w+:\w+\S+)", "", tweet).split())

def get_tweet_sentiment(self, tweet):
    """
    Utility function to classify sentiment of passed tweet
    """
```

```
using textblob's sentiment method
'''
# create TextBlob object of passed tweet text
analysis =TextBlob(self.clean_tweet(tweet))
# set sentiment
if analysis.sentiment.polarity>0:
    return'positive'
elif analysis.sentiment.polarity==0:
    return'neutral'
else:
    return'negative'

def get_tweets(self, query, count =10):
    '''
    Main function to fetch tweets and parse them.
    '''
    # empty list to store parsed tweets
    tweets =[]

    try:
        # call twitter api to fetch tweets
        fetched_tweets=self.api.search(q=query, count=count)

        # parsing tweets one by one
        for tweet in fetched_tweets:
            # empty dictionary to store required params of a tweet
            parsed_tweet={}

            # saving text of tweet
            parsed_tweet['text'] =tweet.text
            # saving sentiment of tweet
            parsed_tweet['sentiment'] =self.get_tweet_sentiment(tweet.text)
```

```
# appending parsed tweet to tweets list
if tweet.retweet_count>0:
    # if tweet has retweets, ensure that it is appended only once
    if parsed_tweet not in tweets:
        tweets.append(parsed_tweet)
    else:
        tweets.append(parsed_tweet)

# return parsed tweets
return tweets

except tweepy.TweepError as e:
    # print error (if any)
    print("Error : "+str(e))

def main():
    # creating object of TwitterClient Class
    api=TwitterClient()
    # calling function to get tweets
    tweets =api.get_tweets(query ='Donald Trump', count =200)

    # picking positive tweets from tweets
    ptweets=[tweet for tweet in tweets if tweet['sentiment'] == 'positive']
    # percentage of positive tweets
    print("Positive tweets percentage: { } %".format(100*len(ptweets)/len(tweets)))
    # picking negative tweets from tweets
    ntweets=[tweet for tweet in tweets if tweet['sentiment'] == 'negative']
    # percentage of negative tweets
    print("Negative tweets percentage: { } %".format(100*len(ntweets)/len(tweets)))
    # percentage of neutral tweets
    print("Neutral tweets percentage: { } %\n
    ".format(100*(len(tweets) -(len( ntweets )+len( ptweets)))/len(tweets)))
```

```
# printing first 5 positive tweets
print("\n\nPositive tweets:")
for tweet in ptweets[:10]:
    print(tweet['text'])

# printing first 5 negative tweets
print("\n\nNegative tweets:")
for tweet in ntweets[:10]:
    print(tweet['text'])

if __name__ == "__main__":
    # calling main function
    main()
```

Here is how a sample output looks like when above program is run:

Positive tweets percentage: 22 %

Negative tweets percentage: 15 %

Positive tweets:

RT @JohnGGalt: Amazing—after years of attacking Donald Trump the media managed to turn #InaugurationDay into all about themselves.

#MakeAme...

RT @vooda1: CNN Declines to Air White House Press Conference Live YES!

THANK YOU @CNN FOR NOT LEGITIMI...

RT @Muheeb\_Shawwa: Donald J. Trump's speech sounded eerily familiar...

POTUS plans new deal for UK as Theresa May to be first foreign leader to meet new president since inauguration

.@realdonaldtrump #Syria #Mexico #Russia & now #Afghanistan.

Another #DearDonaldTrump Letter worth a read @AJEnglish

Negative tweets:

RT @Slate: Donald Trump's administration: "Government by the worst men."

RT @RVAwonk: Trump, Sean Spicer, et al. lie for a reason.

Their lies are not just lies. Their lies are authoritarian propaganda.

RT @KomptonMusic: Me: I hate corn

Donald Trump: I hate corn too

Me: <https://t.co/GPgy8R8HB5>

It's ridiculous that people are more annoyed at this than Donald Trump's sexism.

RT @tony\_broach: Chris Wallace on Fox news right now talking crap  
about Donald Trump news conference it seems he can't face the truth either...

RT @fravel: With False Claims, Donald Trump Attacks Media on Crowd Turnout  
Aziz Ansari Just Hit Donald Trump Hard In An Epic Saturday Night Live Monologue  
We follow these 3 major steps in our program:

- Authorize twitter API client.
- Make a GET request to Twitter API to fetch tweets for a particular query.
- Parse the tweets. Classify each tweet as positive, negative or neutral.

Now, let us try to understand the above piece of code:

- First of all, we create a **TwitterClient** class. This class contains all the methods to interact with Twitter API and parsing tweets. We use **\_\_init\_\_** function to handle the authentication of API client.
- In **get\_tweets** function, we use:  

```
fetcher_tweets = self.api.search(q = query, count = count)
```

  
to call the Twitter API to fetch tweets.
- In **get\_tweet\_sentiment** we use textblob module.  

```
analysis = TextBlob(self.clean_tweet(tweet))
```

TextBlob is actually a high level library built over top of NLTK library. First we call **clean\_tweet** method to remove links, special characters, etc. from the tweet using some simple regex.

Then, as we pass **tweet** to create a **TextBlob** object, following processing is done over text by textblob library:



Tokenize the tweet ,i.e split words from body of text.

Remove stopwords from the tokens.(stopwords are the commonly used words which are irrelevant in text analysis like I, am, you, are, etc.)

Do POS( part of speech) tagging of the tokens and select only significant features/tokens like adjectives, adverbs, etc.

Pass the tokns to a **sentiment classifier** which classifies the tweet sentiment as positive, negative or neutral by assigning it a polarity between -1.0 to 1.0 .

Here is how **sentiment classifier** is created:

**TextBlob** uses a Movies Reviews dataset in which reviews have already been labelled as positive or negative.

Positive and negative features are extracted from each positive and negative review respectively.

Training data now consists of labelled positive and negative features. This data is trained on a Naive Bayes Classifier.

Then, we use **sentiment.polarity** method of **TextBlob** class to get the polarity of tweet between -1 to 1.

Then, we classify polarity as:

```
if analysis.sentiment.polarity> 0:
```

```
    return 'positive'
```

```
elif analysis.sentiment.polarity == 0:
```

```
    return 'neutral'
```

```
else:
```

```
    return 'negative'
```

Finally, parsed tweets are returned. Then, we can do various type of statistical analysis on the tweets. For example, in above program, we tried to find the percentage of positive, negative and neutral tweets about a query.

