F.Y.B.Sc.
(Computer Science)
SEMESTER - II (CBCS)

PROGRAMMING WITH PYTHON - II

SUBJECT CODE : USCS202

Prof. Suhas Pednekar
Vice-Chancellor,
University of Mumbai,

Prof. Ravindra D. Kulkarni
Pro Vice-Chancellor,
University of Mumbai,

Prof. Prakash Mahanwar
Director,
IDOL, University of Mumbai,

Programme Co-ordinator : **Shri Mandar Bhanushe**
Head, Faculty of Science and Technology,
IDOL, University of Mumbai, Mumbai

Course Co-ordinator : **Mr. Sumedh Shejole**
Asst. Professor,
IDOL, University of Mumbai, Mumbai

Editor : **Mr. Patilshirke Santosh**
Smt. Janakibai Rama Salvi College,
Manisha Nagar, Kalwa west, Thane

Course Writers : **Mlind Thorat**
Asst. Professor,
KJSIEIT, Sion, ,Mumbai

: **Vandana Maurya**
Asst. Professor,
B. K. Birla College (Autonomous), Kalyan

February 2022, Print - I

Published by : Director
Institute of Distance and Open Learning ,
University of Mumbai,
Vidyanagari, Mumbai - 400 098.

DTP Composed and Printed by : Mumbai University Press
Vidyanagari, Santacruz (E), Mumbai - 400098

CONTENTS

F.Y.B.Sc. (Computer Science)
SEMESTER - I (CBCS)
Programming with Python – II
(Credits : 2 Lectures/Week: 3)

Objective:

The objective of this paper is to explore the style of structured programming to give the idea to the students how programming can be used for designing real-life applications by reading/writing to files, GUI programming, interfacing database/networks and various other features.

Expected Learning Outcomes

1) Students should be able to understand how to read/write to files using python.
2) Students should be able to catch their own errors that happen during execution of programs.
3) Students should get an introduction to the concept of pattern matching.
4) Students should be made familiar with the concepts of GUI controls and designing GUI applications.\Students should be able to connect to the database to move the data to/from the application.
5) Students should know how to connect to computers, read from URL and send email.

Unit I	**Python File Input-Output:** Opening and closing files, various types of file modes, reading and writing to files, manipulating directories. Iterables, iterators and their problem solving applications. **Exception handling:** What is an exception, various keywords to handle exceptions such try, catch, except, else, finally, raise. **Regular Expressions:** Concept of regular expression, various types of regular expressions, using match function.	15 L
Unit II	**GUI Programming in Python (using Tkinter/wx Python/Qt)** What is GUI, Advantages of GUI, Introduction to UI library. Layout management, events and bindings, fonts, colours, drawing on canvas (line, oval, rectangle, etc.) Widgets such as : frame, label, button, check button, entry, list box, message, radio button, text, spin box etc	15 L

Unit III	**Database connectivity in Python**: Installing mysql connector, accessing connector module module, using connect, cursor, execute & close functions, reading single & multiple results of query execution, executing different types of statements, executing transactions, understanding exceptions in database connectivity. **Network connectivity**: Socket module, creating server-client programs, sending email, reading from URL	15 L

Text books:

1. Paul Gries , Jennifer Campbell, Jason Montojo, *Practical Programming: An Introduction to Computer Science Using Python 3*, Pragmatic Bookshelf, 2/E 2014

Additional References:

1. James Payne , *Beginning Python: Using Python 2.6 and Python 3,* Wiley India, 2010
2. A. Lukaszewski, MySQL for Python: Database Access Made Easy, Pact Publisher, 2010

❖❖❖❖

UNIT 1

1

PYTHON FILE INPUT- OUTPUT

Unit Structure

1.1 INTRODUCTION

Files are named locations on disk to store information. They are used to permanently store data in a non-volatile memory e.g. hard disk.

Random Access Memory (RAM) is volatile memory means it loses its data when the computer is turned off, we use files for future use of the data by permanently storing them.

When we want to read from or write to a file, we need to open it first.

When we have finished our work, it needs to be closed so that the resources that are attached with the file are freed.

Hence, in Python, a file operation takes place in the following sequence:

1. Open a file

2. Read or write (perform operation)

3. Close the file

1.2 OPENING FILES IN PYTHON

Python has a built-in **open()** function to open a file.

The `open()` function returns a file object, which has a `read()` method for reading the content of the file:

>>> f = open("test.txt") # open file in current directory

>>> f = open("C:/Python38/README.txt") # specifying full path

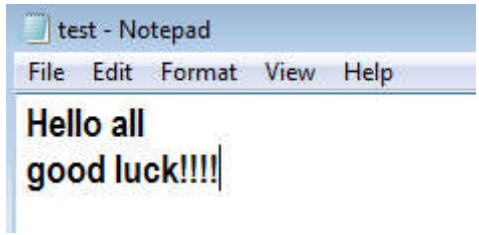
We can specify the mode while opening any file.

In mode, we specify whether we want to **read r, write w** or **append a** to the file. We can also specify if we want to open the file in **text mode** or **binary mode.**

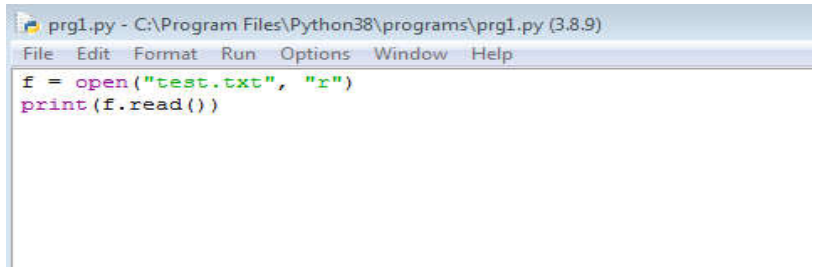
- The default is reading in text mode. In this mode, we get strings when reading from the file.
- On the other hand, binary mode returns bytes, and this is the mode to be used when dealing with non-text files like images or executable files.

Mode	Description
r	It opens a file for reading. (By default)
w	It opens a file for writing. Creates a new file if it does not exist or truncates the file if it exists.
x	It opens a file for exclusive creation. If the file already exists, the operation fails.
a	It opens a file for appending at the end of the file without truncating it. Creates a new file if it does not exist.
t	Opens in text mode. (By default)
b	Opens in binary mode
+	Opens a file for updating (reading and writing)

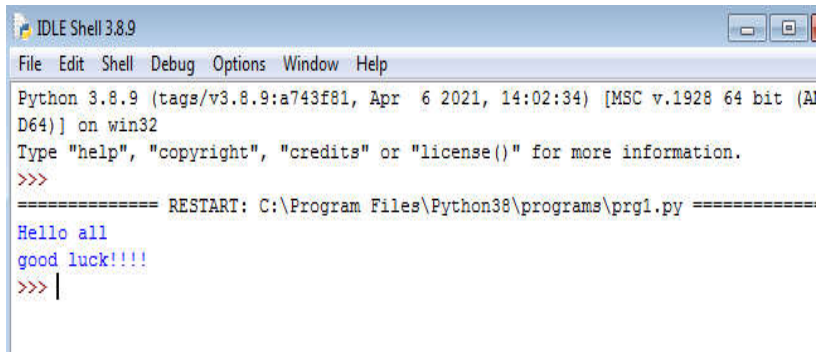
This is the test.txt file we have



Example 1.1

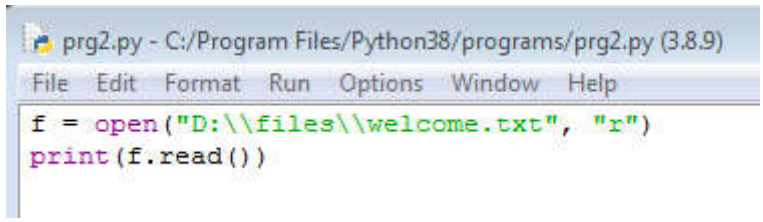


Output:



If the file is located in a different location, you will have to specify the file path, as follows:

Example 1.2



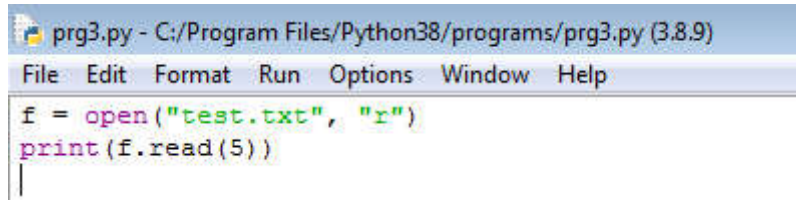
Output:

```
========== RESTART: C:/Program Files/Python38/programs/prg2.py ========
Welcome to this text file!
This file is located in a folder named "files", on the D drive.
Good Luck!
...
```

❖ *Read Only Parts of the File*

By default the **`read()`** method returns the whole text, but you can also specify how many characters you want to return:

Example 1.3Return the 5 first characters of the file



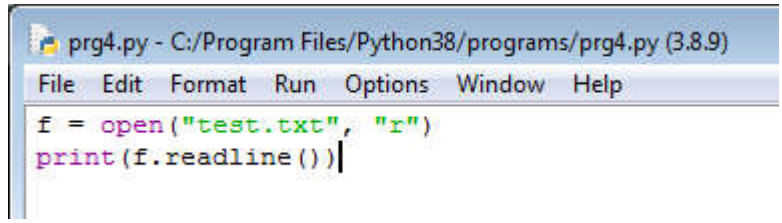
Output:

```
============= RESTART: C:/Program Files/Python38/programs/prg3.py =
Hello
```

❖ *Read Lines*

You can return one line by using the `readline()` method:

Example 1.4Read one line of the file:

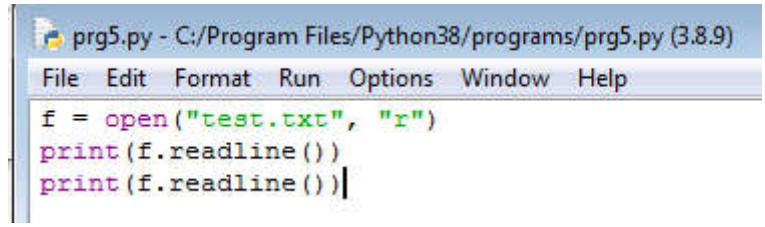


Output:

```
============= RESTART: C:/Program Files/Python38/programs/prg4.py
Hello all
```

By calling readline() two times, you can read the two first lines:

Example 1.5Read two lines of the file:

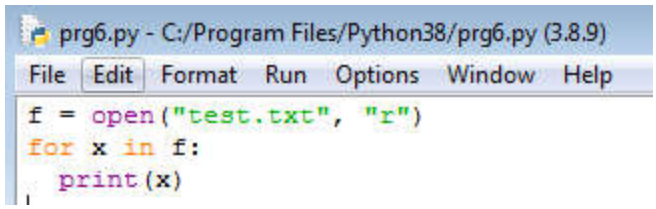


Output:

```
============== RESTART: C:/Program Files/Python38/programs/prg5.py
Hello all

good luck!!!!
>>>
```

By looping through the lines of the file, we can read the whole file, line by line:

*Example 1.6***Loop through the file line by line**



Output:

```
================== RESTART: C:/Program Files/Python38/prg6.py =
Hello all
```

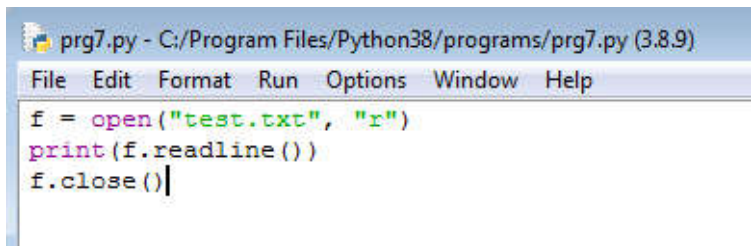
1.3 CLOSING FILES IN PYTHON

When we have completed performing operations on the file, we need to properly close the file.

Closing a file will free up the resources that were attached with the file. It is done using the **close()** method available in Python.

Python has a garbage collector to clean up unreferenced objects, but we must not rely on it to close the file.

Example 1.7



Output:

```
============= RESTART: C:/Program Files/Python38/programs/prg7.py ==
Hello all
```

Note:

You should always close your files, in some cases, due to buffering, changes made to a file may not show until you close the file.

1.4 WRITE TO AN EXISTING FILE

To write to an existing file, we must add a parameter to the `open()` function:

`"a"` - Append

It will append to the end of the file.

`"w"` - Write

It will overwrite any existing content.

Example 1.8

Open the file "test.txt" and append content to the file:

prg8.py - C:/Program Files/Python38/programs/prg8.py (3.8.9)

File Edit Format Run Options Window Help

```
f = open("test.txt", "a")
f.write("All the best!")
f.close()

#open and read the file after appending:
f = open("test.txt", "r")
print(f.read())
```

Output:

```
============== RESTART: C:/Program Files/Python38/programs/prg8.py
Hello all
good luck!!!!All the best!
>>>
```

Example 1.9

Open the file "test.txt" and overwrite its content:

prg9.py - C:/Program Files/Python38/programs/prg9.py (3.8.9)

File Edit Format Run Options Window Help

```
f = open("test.txt", "w")
f.write("Python Programming")
f.close()

#open and read the file after overwriting:
f = open("test.txt", "r")
print(f.read())
```

Output:

```
============== RESTART: C:/Program Files/Python38/programs/prg9.py
Python Programming
>>>
```

1.5 CREATE A NEW FILE

To create a new file in Python, use the open() method, with one of the following parameters:

"x" - Create

It will create a file, returns an error if the file exist.

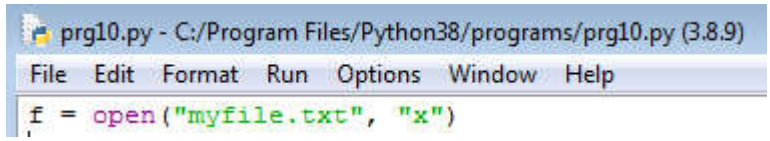
"a" - Append

It will create a file if the specified file does not exist.

"w" - Write

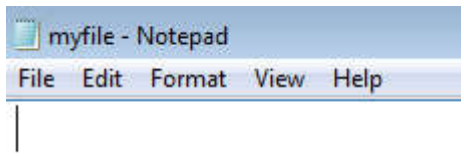
It will create a file if the specified file does not exist.

Example 1.10 Create a file called "myfile.txt"



Output:

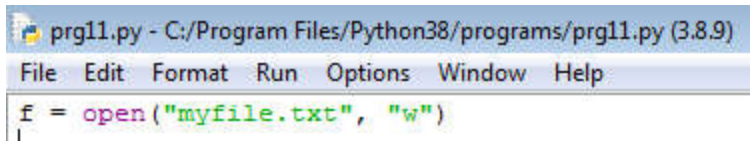
Result: a new empty file is created!



Returns an error if the file exist

```
============ RESTART: C:/Program Files/Python38/programs/prg10.py ========
Traceback (most recent call last):
  File "C:/Program Files/Python38/programs/prg10.py", line 1, in <module>
    f = open("myfile.txt", "x")
FileExistsError: [Errno 17] File exists: 'myfile.txt'
```

Example 1.11Create a new file if it does not exist:

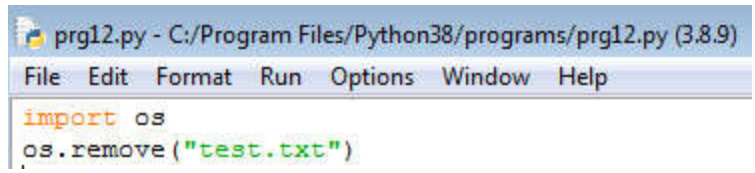


Note: the "w" method will overwrite the entire file.

1.6 PYTHON DELETE FILE

To delete a file, we must import the OS module, and run its **os.remove()** function.

Example 1.12Remove the file "test.txt":

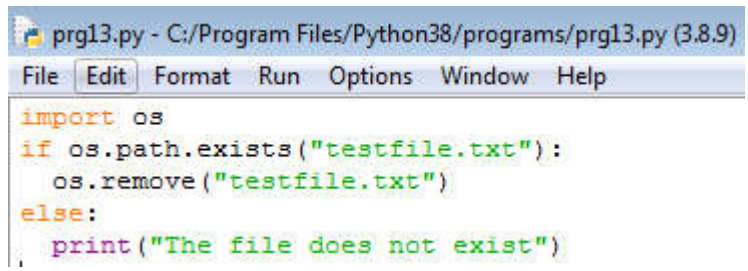


Note: test.txt got deleted from programs folder.

Check if File exist:

To avoid getting an error, you might want to check if the file exists before you try to delete it:

Example 1.13Check if file exists, then delete it:



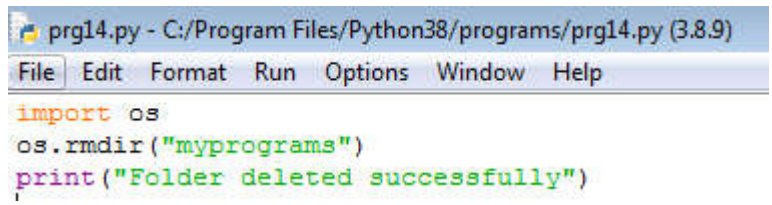
Output:

```
============= RESTART: C:/Program Files/Python38/programs/prg13.py
The file does not exist
```

Delete Folder

To delete an entire folder, use the os.rmdir() method:

Example 1.14Remove the folder "myprograms":



Output:

```
============= RESTART: C:/Program Files/Python38/programs/prg14.py
Folder deleted successfully
```

Note: You can only remove empty folders.

1.7 PYTHON DIRECTORY

If there are many files in our Python program, we can arrange our code within different directories to make things more manageable.

A directory or folder is a collection of files and subdirectories.

Python has the **os** module that provides us with many useful methods to work with directories (and files as well).

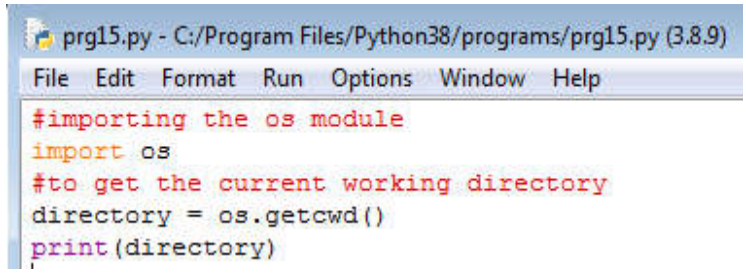
1.7.1 GET CURRENT DIRECTORY

We can get the present working directory by using the **getcwd()** method of the os module.

We use the OS module to interact with the operating system.

This method returns the current working directory in the form of a string.

Example 1.15



Output:

```
============ RESTART: C:/Program Files/Python38/programs/prg15.p
C:\Program Files\Python38\programs
```

1.7.2 CHANGING DIRECTORY

In case if we want to change the current working directory, we can do so by using the chdir() method .

Syntax of chdir()

os.chdir(path)

Parameters:

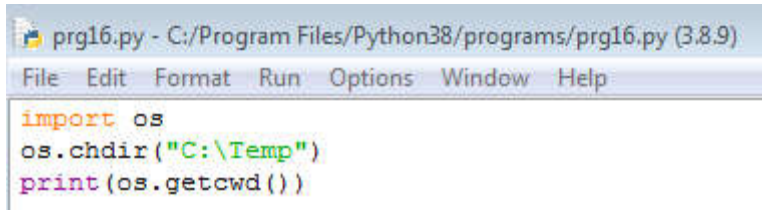
path - The path to the new directory

The new path that we want to change into must be supplied as a string to this method.

We can use both the forward-slash / or the backward-slash \ to separate the path elements.

It is safer to use an escape sequence when using the backward slash.

Example 1.16



Output:

```
============ RESTART: C:/Program Files/Python38/programs/prg16.py
C:\Temp
```

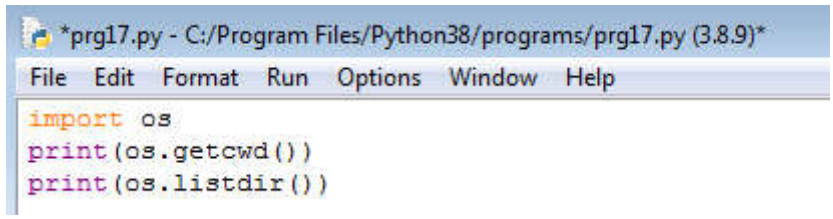
1.7.3 LIST DIRECTORIES AND FILES

All files and sub-directories inside a directory can be listed using the listdir() method.

This method takes in a path and returns a list of subdirectories and files in that path.

If no path is specified, it returns the list of subdirectories and files from the current working directory.

Example 1.17



Output:

```
============ RESTART: C:/Program Files/Python38/programs/prg17.py ============
C:\Program Files\Python38\programs
['fileopen.py', 'myfile.txt', 'prg1.py', 'prg10.py', 'prg11.py', 'prg12.py', 'pr
g13.py', 'prg14.py', 'prg15.py', 'prg16.py', 'prg17.py', 'prg2.py', 'prg3.py', '
prg4.py', 'prg5.py', 'prg7.py', 'prg8.py', 'prg9.py', 'subprograms']
```

1.7.4 MAKING A NEW DIRECTORY

We can make a new directory by usingmkdir() method.

This method takes in the path of the new directory.

If the full path is not specified, the new directory is created in the current working directory.

Example 1.18

```
prg18.py - C:/Program Files/Python38/programs/prg18.py (3.8.9)
File  Edit  Format  Run  Options  Window  Help
import os
os.mkdir('test')  # It will create test folder
```

Result: a new folder is created!

1.7.5 REMOVING DIRECTORY OR FILE

A file can be removed (deleted) using the **remove()** method.

Similarly, the **rmdir()** method removes an empty directory.

Example 1.19

```
prg19.py - C:/Program Files/Python38/programs/prg19.py (3.8.9)
File  Edit  Format  Run  Options  Window  Help
import os
directory = os.listdir()
print(directory)

os.remove('myfile.txt')

directory1 = os.listdir()
print(directory1)

os.rmdir('subprograms')
print(os.listdir())
```

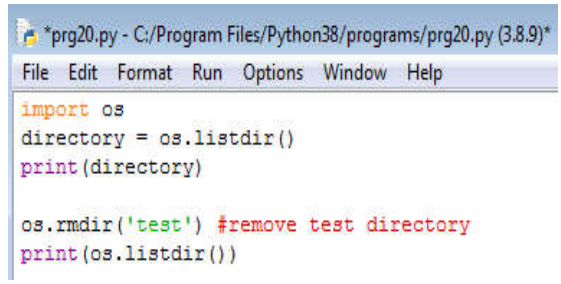
Output:

```
============ RESTART: C:/Program Files/Python38/programs/prg19.py ============
['fileopen.py', 'myfile.txt', 'prg1.py', 'prg10.py', 'prg11.py', 'prg12.py', 'pr
g13.py', 'prg14.py', 'prg15.py', 'prg16.py', 'prg17.py', 'prg18.py', 'prg19.py',
 'prg2.py', 'prg3.py', 'prg4.py', 'prg5.py', 'prg7.py', 'prg8.py', 'prg9.py', 's
ubprograms', 'test']
['fileopen.py', 'prg1.py', 'prg10.py', 'prg11.py', 'prg12.py', 'prg13.py', 'prg1
4.py', 'prg15.py', 'prg16.py', 'prg17.py', 'prg18.py', 'prg19.py', 'prg2.py', 'p
rg3.py', 'prg4.py', 'prg5.py', 'prg7.py', 'prg8.py', 'prg9.py', 'subprograms', '
test']                                                          ⇦myfile.txt deleted
['fileopen.py', 'prg1.py', 'prg10.py', 'prg11.py', 'prg12.py', 'prg13.py', 'prg1
4.py', 'prg15.py', 'prg16.py', 'prg17.py', 'prg18.py', 'prg19.py', 'prg2.py', 'p
rg3.py', 'prg4.py', 'prg5.py', 'prg7.py', 'prg8.py', 'prg9.py', 'test']
                                                                ⇦subprograms folder
>>>                                                               deleted
```

Note: The **rmdir()** method can only remove empty directories.

We can notremove a non-empty directory.

Example 1.20

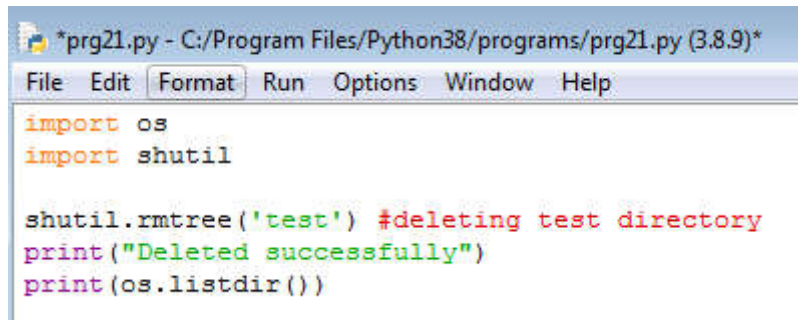


Output:

```
============ RESTART: C:/Program Files/Python38/programs/prg20.py ============
['fileopen.py', 'prg1.py', 'prg10.py', 'prg11.py', 'prg12.py', 'prg13.py', 'prg1
4.py', 'prg15.py', 'prg16.py', 'prg17.py', 'prg18.py', 'prg19.py', 'prg2.py', 'p
rg20.py', 'prg3.py', 'prg4.py', 'prg5.py', 'prg7.py', 'prg8.py', 'prg9.py', 'tes
t']
Traceback (most recent call last):
  File "C:/Program Files/Python38/programs/prg20.py", line 5, in <module>
    os.rmdir('test')
OSError: [WinError 145] The directory is not empty: 'test'
```

In order to remove a non-empty directory, we can use the **rmtree()** method inside the **shutil** module.

Example 1.21



Output:

```
============ RESTART: C:/Program Files/Python38/programs/prg21.py ============
Deleted successfully
['fileopen.py', 'prg1.py', 'prg10.py', 'prg11.py', 'prg12.py', 'prg13.py', 'prg1
4.py', 'prg15.py', 'prg16.py', 'prg17.py', 'prg18.py', 'prg19.py', 'prg2.py', 'p
rg20.py', 'prg21.py', 'prg3.py', 'prg4.py', 'prg5.py', 'prg7.py', 'prg8.py', 'pr
g9.py']
```

❖❖❖❖

2

EXCEPTION HANDLING

Unit Structure

2.1 INTRODUCTION

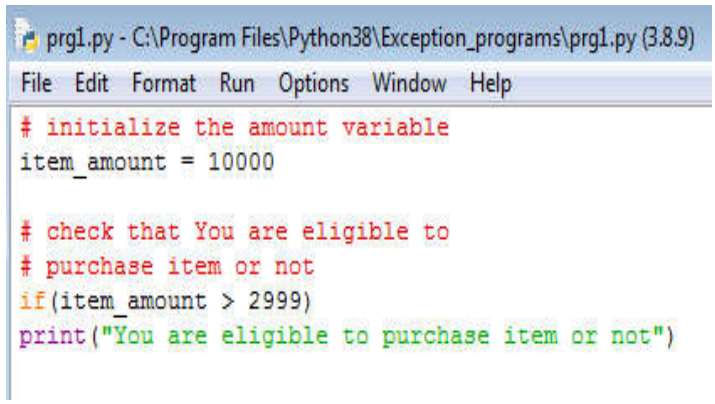
Error in Python can be of two types **i.e. Syntax errors and Exceptions**.

Errors are the problems in a program due to which the program will stop the execution.

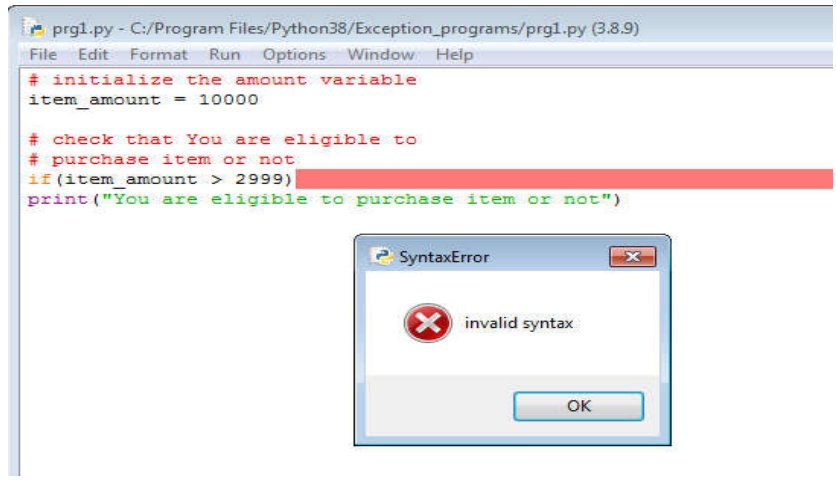
2.1.1 Syntax Error

This error is caused bywrong syntax in the code. It leads to the termination of the program.

Example 2.1



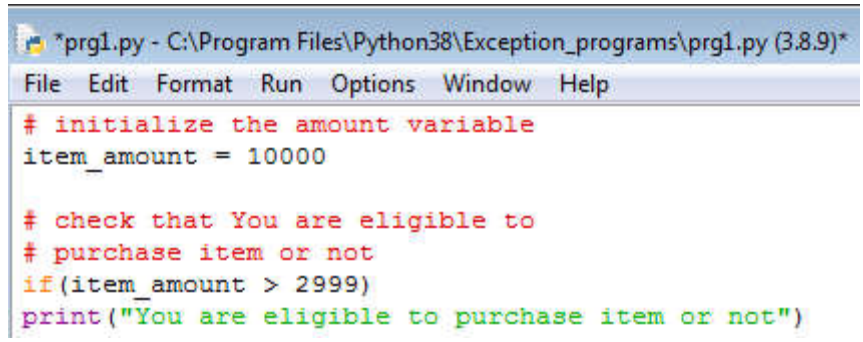
Output:



2.1.2 Exceptions

Exceptions are raised when the program is syntactically correct, but the code resulted in an error. This error does not stop the execution of the program; however, it changes the normal flow of the program.

Example 2.2



Output:

```
========= RESTART: C:/Program Files/Python38/Exception_programs/prg2.py ========
Traceback (most recent call last):
  File "C:/Program Files/Python38/Exception_programs/prg2.py", line 5, in <modul
e>
    a = marks / 0
ZeroDivisionError: division by zero
```

Here, we are trying to divide a number by 0 hence it raised **ZeroDivisionError**Exception.

Python hasbuilt-in exceptions that are raised when your program encounters an errori.esomething in the program goes wrong.

When these exceptions occur, the Python interpreter stops the current flow and passes it to the calling process until it is handled. If not handled, the program may crash.

For example:

Let us consider a program where we have a function **A** that calls function **B**, which in turn calls function **C**.If an exception occurs in function **C** but is not handled in **C**, an exception passes to **B** and then to **A**.

If it is never handled, an error message is displayed, and program comes to a sudden unexpected halt.

Note: Exception is the base class for all the exceptions in Python.

2.1.3 BUILT-IN EXCEPTIONS

The table below shows built-in exceptions that are usually raised in Python:

Exception	Description
ArithmeticError	Raised when an error occurs in numeric calculations.
AssertionError	Raised when an assert statement fails.
AttributeError	Raised when attribute reference or assignment fails.
Exception	Base class for all exceptions.
EOFError	Raised when the input() method hits an "end of file" condition (EOF).
FloatingPointError	Raised when a floating point calculation fails.
GeneratorExit	Raised when a generator is closed (with the close() method).
ImportError	Raised when an imported module does not exist.
IndentationError	Raised when indendation is not correct.
IndexError	Raised when an index of a sequence does not exist.
KeyError	Raised when a key does not exist in a dictionary.
KeyboardInterrupt	Raised when the user presses Ctrl+c, Ctrl+z or Delete.

LookupError	Raised when errors raised cant be found.
MemoryError	Raised when a program runs out of memory.
NameError	Raised when a variable does not exist.
NotImplementedError	Raised when an abstract method requires an inherited class to override the method.
OSError	Raised when a system related operation causes an error.
OverflowError	Raised when the result of a numeric calculation is too large.
ReferenceError	Raised when a weak reference object does not exist.
RuntimeError	Raised when an error occurs that do not belong to any specific expections.
StopIteration	Raised when the next() method of an iterator has no further values.
SyntaxError	Raised when a syntax error occurs.
TabError	Raised when indentation consists of tabs or spaces.
SystemError	Raised when a system error occurs.
SystemExit	Raised when the sys.exit() function is called.
TypeError	Raised when two different types are combined.
UnboundLocalError	Raised when a local variable is referenced before assignment.
UnicodeError	Raised when a unicode problem occurs.
UnicodeEncodeError	Raised when a unicode encoding problem occurs.
UnicodeDecodeError	Raised when a unicode decoding problem occurs.
UnicodeTranslateError	Raised when a unicode translation problem occurs.

ValueError	Raised when there is a wrong value in a specified data type.
ZeroDivisionError	Raised when the second operator in a division is zero.

2.2 CATCHING EXCEPTIONS

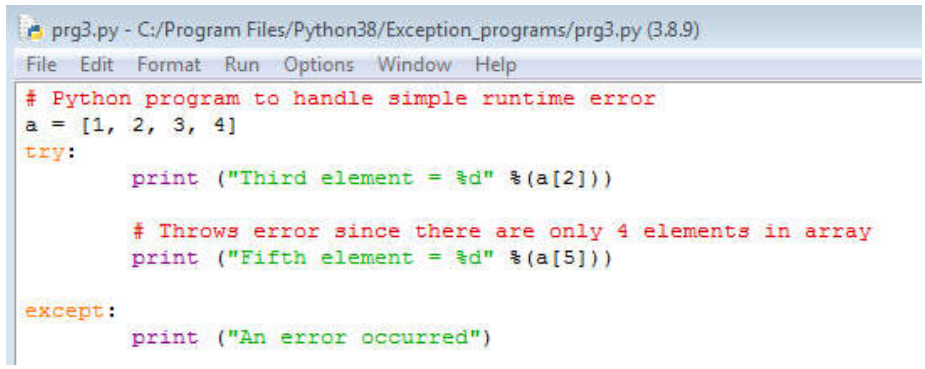
Try and Except Statement

Try and except statements are used to catch and handle exceptions in Python. Statements that may raise exceptions are kept inside the **try** clause and the statements that handle the exception are written inside **except** clause.

Example:

Let us try to access the array element which doesn't exist i.e whose index is out of bound and handle the corresponding exception.

Example2.3



Output:

```
========= RESTART: C:/Program Files/Python38/Exception_programs/prg3.py
Third element = 3
An error occurred
```

In the above example,statements that may cause the error are placed inside the try statement (second print statement in our case).

The second print statement tries to access the fifth element; which is not there in the list;so it throws an exception. This exception is then caught by the except statement.

2.3 CATCHING SPECIFIC EXCEPTION

A **try** statement can have more than one **except** clause, to specify handlers for different exceptions. Please note that at most one handler will be executed.

For example, we can add **IndexError** in the above code.

The general syntax for adding specific exceptions are –

```
try:
 # statement(s)
except IndexError:
 # statement(s)
except ValueError:
# statement(s)
```

Example2.3Catching specific exception in Python

prg4.py - C:\Program Files\Python38\Exception_programs\prg4.py (3.8.9)

File Edit Format Run Options Window Help

```
# Program to handle multiple errors with one
# except statement

def fun(a):
        if a < 4:

                # throws ZeroDivisionError for a = 3
                b = a/(a-3)

        # throws NameError if a >= 4
        print("Value of b = ", b)

try:
        fun(3)
        #fun(5)

# note that braces () are necessary here for
# multiple exceptions
except ZeroDivisionError:
        print("ZeroDivisionError Occurred and Handled")
except NameError:
        print("NameError Occurred and Handled")
```

Here fun(5) is commented.

Output:

```
========= RESTART: C:\Program Files\Python38\Exception_programs\prg4.py
ZeroDivisionError Occurred and Handled
```

If you comment the line fun(3), the output will be :

prg4.py - C:\Program Files\Python38\Exception_programs\prg4.py (3.8.9)

File Edit Format Run Options Window Help

```
# Program to handle multiple errors with one
# except statement

def fun(a):
        if a < 4:

                # throws ZeroDivisionError for a = 3
                b = a/(a-3)

        # throws NameError if a >= 4
        print("Value of b = ", b)

try:
        fun(3)
        #fun(5)

# note that braces () are necessary here for
# multiple exceptions
except ZeroDivisionError:
        print("ZeroDivisionError Occurred and Handled")
except NameError:
        print("NameError Occurred and Handled")
```

Here fun(3) is commented.

Output:

```
========= RESTART: C:\Program Files\Python38\Exception_programs\prg4.py
NameError Occurred and Handled
```

In the above example, python tries to access the value of b, so **NameError** occurs.

❖ TRY WITH ELSE CLAUSE

In python, you can also use the **else** clause on the **try-except** block which must be present after all the except clauses. The code will enter**else** block only if the **try** clause does not raise an exception.

Example 2.4 Try with else clause

prg5.py - C:/Program Files/Python38/Exception_programs/prg5.py (3.8.9)

File Edit Format Run Options Window Help

```
# Program to depict else clause with try-except
# Function which returns a/b
def AdivB(a , b):
        try:
                c = ((a+b) / (a-b))
        except ZeroDivisionError:
                print ("a/b result in 0")
        else:
                print (c)

# Driver program to test above function
AdivB(2.0, 3.0)
AdivB(3.0, 3.0)
```

Output:

```
========= RESTART: C:/Program Files/Python38/Exception_programs/prg5.py
-5.0
a/b result in 0
```

❖ FINALLY KEYWORD IN PYTHON

Python provides a keyword **finally**, which is always executed after the **try** and **except** blocks. The finally block always executes after normal termination of try block or after try block terminates due to some exception.

Syntax:

try:

    # Some Code....

except:

    # optional block

    # Handling of exception (if required)

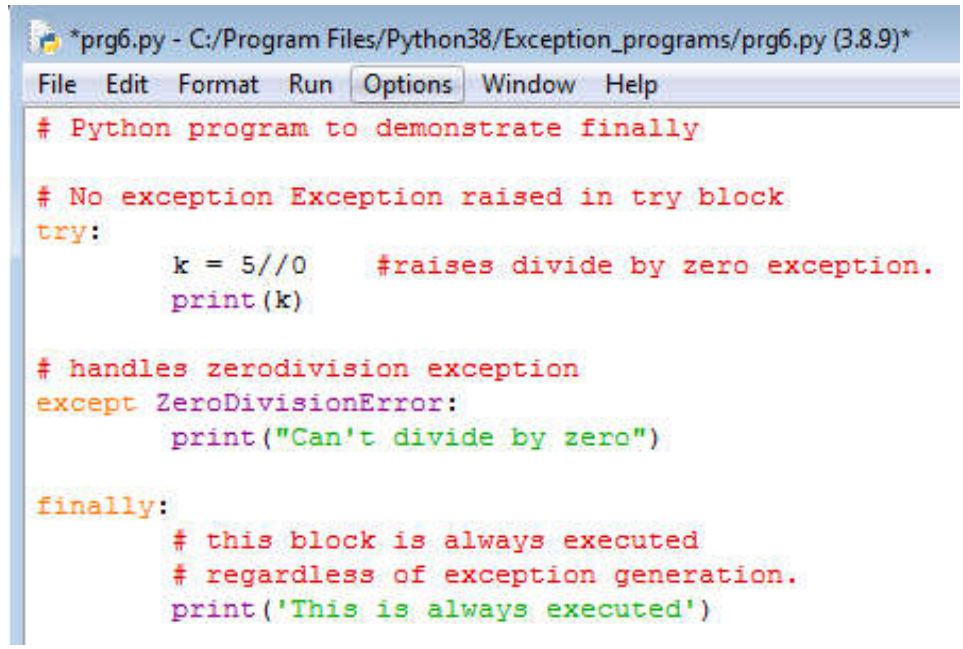
else:

    # execute if no exception

finally:

    # Some code .....(always executed)

Example 2.5 **finally keyword in python**



Output:

```
========= RESTART: C:/Program Files/Python38/Exception_programs/prg6.py
Can't divide by zero
This line is always executed
```

Example 2.6

prg7.py - C:\Program Files\Python38\Exception_programs\prg7.py (3.8.9)

File Edit Format Run Options Window Help

```
#The try block will raise an error when trying to write to a read-only file:

try:
  f = open("demofile.txt")
  try:
    f.write("Hello all")
  except:
    print("Something went wrong when writing to the file")
  finally:
    f.close()
except:
  print("Something went wrong when opening the file")
```

Output:

```
========= RESTART: C:\Program Files\Python38\Exception_programs\prg7.py
Something went wrong when writing to the file
```

RAISING EXCEPTION

The raise statement allows the programmer to force a specific exception to occur.This must be either an exception instance or an exception class (a class that derives from Exception).

Example 2.8

prg8.py - C:\Program Files\Python38\Exception_programs\prg8.py (3.8.9)

File Edit Format Run Options Window Help

```
# Program to depict Raising Exception

try:
        raise NameError("Hello all...")      #Raise Error
except NameError:
        print ("An exception")
        raise       #To determine whether the exception was raised or not
```

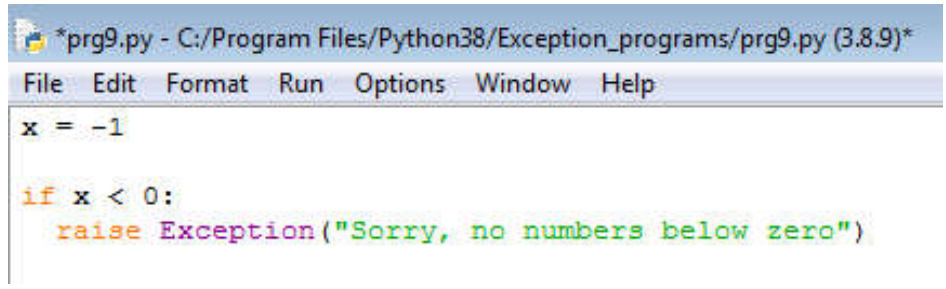
The output of the above code is simply "**An exception"** but a Runtime error will also occur at the end due to raise statement in the last line. So, the output on your command line will look like this:

Output:

```
========= RESTART: C:\Program Files\Python38\Exception_programs\prg8.py ========
An exception
Traceback (most recent call last):
  File "C:\Program Files\Python38\Exception_programs\prg8.py", line 4, in <modul
e>
    raise NameError("Hello all...")     #Raise Error
NameError: Hello all...
```

Example 2.9

Raise an error and stop the program if value of x is lower than 0



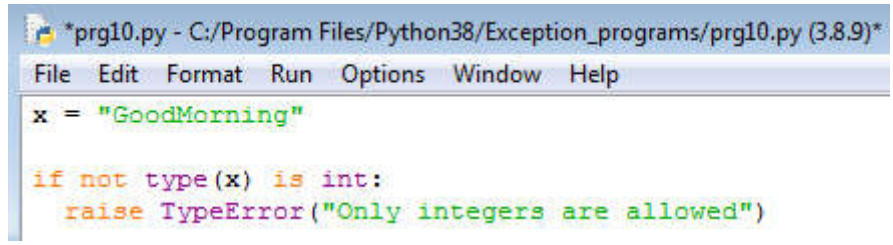
Output:

```
========= RESTART: C:/Program Files/Python38/Exception_programs/prg9.py ========
Traceback (most recent call last):
  File "C:/Program Files/Python38/Exception_programs/prg9.py", line 4, in <modul
e>
    raise Exception("Sorry, no numbers below zero")
Exception: Sorry, no numbers below zero
```

The **raise** keyword is used to raise an exception.

You can define what kind of error to raise, and the text to print to the user.

Example 2.10 Raise a TypeError if x is not an integer



Output:

```
======== RESTART: C:/Program Files/Python38/Exception_programs/prg10.py ========
Traceback (most recent call last):
  File "C:/Program Files/Python38/Exception_programs/prg10.py", line 4, in <modu
le>
    raise TypeError("Only integers are allowed")
TypeError: Only integers are allowed
```

❖❖❖❖

3

REGULAR EXPRESSIONS

Unit Structure

3.1 Introduction

3.2 Reg Ex functions

3.3 Metacharacters

3.4 Special sequences

3.5 Sets

3.1 INTRODUCTION

Regular Expression, or aRegEx, is a sequence of characters that forms a search pattern.

RegEx can be used to check if a string contains the specified search pattern.

❖ **Reg Ex Module**

Python has a built-in package called **re**, which can be used to work with Regular Expressions.

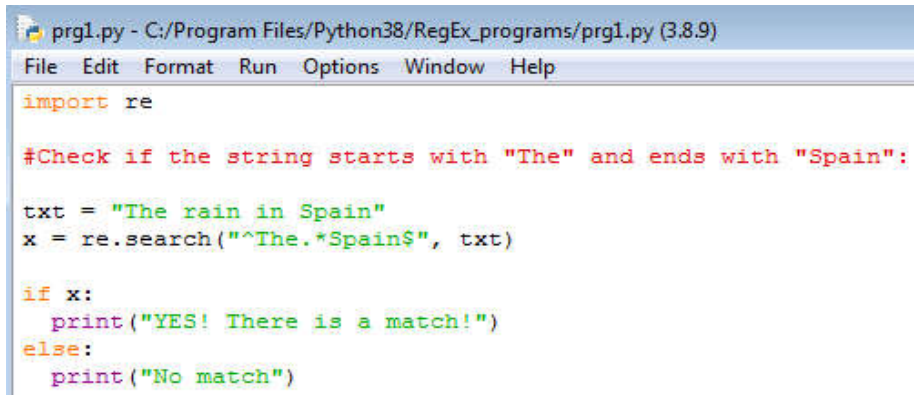
Import the **re** module:

Import re

❖ **Reg Ex in Python**

Once you have imported**re** module, you can start using regular expressions:

Example 3.1Search the string to see if it starts with "The" and ends with "Spain":



Output:

```
========== RESTART: C:/Program Files/Python38/RegEx_programs/prg1.py
YES! There is a match!
```

3.2 REGEXFUNCTIONS

The re module offers a set of functions that allows us to search a string for a match.

Function	Description
Findall	Returns a list containing all matches.
Search	Returns a Match object if there is a match anywhere in the string.
Split	Returns a list where the string has been split at each match.
Sub	Replaces one or many matches with a string.

3.3 METACHARACTERS

Meta characters are characters with a special meaning.

Character	Description	Example
[]	A set of characters	"[a-m]"
	Signals a special sequence (can also be used to escape special characters)	"\d"
.	Any character (except newline character)	"he..o"
^	Starts with	"^hello"
$	Ends with	"world$"
*	Zero or more occurrences	"aix*"
+	One or more occurrences	"aix+"
{}	Exactly the specified number of occurrences	"al{2}"
|	Either or	"falls|stays"
()	Capture and group	

Let’s discuss each of these metacharacters in detail.

- ***[] Square Brackets***

Square Brackets ([]) represents a character class consisting of a set of characters that we wish to match.

For example, the character class **[abc]** will match any single a, b, or c.

We can also specify a range of characters using – inside the square brackets.

For example:

- [0, 3] is sample as [0123]
- [a-c] is same as [abc]

We can also invert the character class using the caret(^) symbol.

For example,

[^0-3] means any number except 0, 1, 2, or 3.

[^a-c] means any character except a, b, or c.

- ***. Dot***

Dot(.) symbol matches only a single character except for the newline character (\n).

For example:

a.b will check for the string that contains any character at the place of the dot such as acb, acbd, abbb, etc.

It will check if the string contains at least 2 characters.

- ***^ Caret***

Caret (^) symbol matches the beginning of the string i.e. checks whether the string starts with the given character(s) or not.

For example:

^g will check if the string starts with g such asglobe, girl, g, etc.

^ge will check if the string starts with ge such as geeks, geeksandgeek etc.

- ***$ Dollar***

Dollar($) symbol matches the end of the string i.e checks whether the string ends with the given character(s) or not.

For example :

s$ will check for the string that ends with geeks, ends, s, etc.

ks$ will check for the string that ends with ks such as geeks, geeksandgeeks, ks, etc.

- ***| Or***

Or symbol works as the or operator meaning it checks whether the pattern before or after the or symbol is present in the string or not.

For example:

a|b will match any string that contains a or b such as acd, bcd, abcd, etc.

- ***? Question Mark***

Question mark(?) checks if the string before the question mark in the regex occurs at least once or not at all.

For example:

ab?c will be matched for the string ac, acb, dabc but will not be matched for abbc because there are two b.

Similarly, it will not be matched for abdc because b is not followed by c.

- ***Star***

Star (*) symbol matches zero or more occurrences of the regex preceding the * symbol.

For example:

ab*c will be matched for the string ac, abc, abbbc, dabc, etc. but will not be matched for abdc because b is not followed by c.

- **+ *Plus***

Plus (+) symbol matches one or more occurrences of the regex preceding the + symbol.

For example:

ab+c will be matched for the string abc, abbc, dabc, but will not be matched for ac, abdc because there is no b in ac and d is not followed by c in abdc.

- ***{m, n} – Braces***

Braces matches any repetitions preceding regex from m to n both inclusive.

For example :

a{2, 4} will be matched for the string aaab, baaaac, gaad, but will not be matched for strings like abc, bc because there is only one a or no a in both the cases.

- ***(<regex>) – Group***

Group symbol is used to group sub-patterns.

For example :

(a|b)cd will match for strings like acd, abcd, gacd, etc.

3.4 SPECIAL SEQUENCES

A special sequence is a \ followed by one of the characters in the list below, and has a special meaning:

Character	Description	Example
\A	Returns a match if the specified characters are at the beginning of the string.	"\AThe"
\b	Returns a match where the specified characters are at the beginning or at the end of a word (the "r" in the beginning is making sure that the string is being treated as a "raw string")	r"\bain" r"ain\b"
\B	Returns a match where the specified characters are present, but NOT at the beginning (or at the end) of a word (the "r" in the beginning is making sure that the string is being treated as a "raw string")	r"\Bain" r"ain\B"
\d	Returns a match where the string contains digits (numbers from 0-9)	"\d"
\D	Returns a match where the string DOES NOT contain digits	"\D"
\s	Returns a match where the string contains a white space character	"\s"
\S	Returns a match where the string DOES NOT contain a white space character	"\S"

\w	Returns a match where the string contains any word characters (characters from a to Z, digits from 0-9, and the underscore _ character)	"\w"
\W	Returns a match where the string DOES NOT contain any word characters	"\W"
\Z	Returns a match if the specified characters are at the end of the string	"Spain\Z"

3.5 SETS

A set is a set of characters inside a pair of square brackets [] with a special meaning:

Set	**Description**
[arn]	Returns a match where one of the specified characters (a, r, or n) are present.
[a-n]	Returns a match for any lower-case character, alphabetically between a and n.
[^arn]	Returns a match for any character EXCEPT a, r, and n.
[0123]	Returns a match where any of the specified digits (0, 1, 2, or 3) are present.
[0-9]	Returns a match for any digit between 0 and 9.
[0-5][0-9]	Returns a match for any two-digit numbers from 00 and 59.
[a-zA-Z]	Returns a match for any character alphabetically between a and z, lower case OR upper case.
[+]	In sets, +, *, ., |, (), $,{} has no special meaning, so [+] means: return a match for any + character in the string.

❖ *The findall() Function*

The **findall()** function returns a list containing all matches.

Example 3.2 **Print a list of all matches**

prg2.py - C:/Program Files/Python38/RegEx_programs/prg2.py (3.8.9)

File Edit Format Run Options Window Help

```
import re

#Return a list containing every occurrence of "ai":
text = "The rain in Spain"
y = re.findall("ai", text)
print(y)
```

Output:

```
========== RESTART: C:/Program Files/Python38/RegEx_programs/prg2.py
['ai', 'ai']
```

The list contains the matches in the order they are found.

If no matches are found, an empty list is returned:

Example 3.3 Return an empty list if no match was found

prg3.py - C:/Program Files/Python38/RegEx_programs/prg3.py (3.8.9)

File Edit Format Run Options Window Help

```
import re

text = "The rain in Spain"
y = re.findall("Portugal", text)
print(y)

if (y):
  print("Yes, there is at least one match!")
else:
  print("No match")
```

Output:

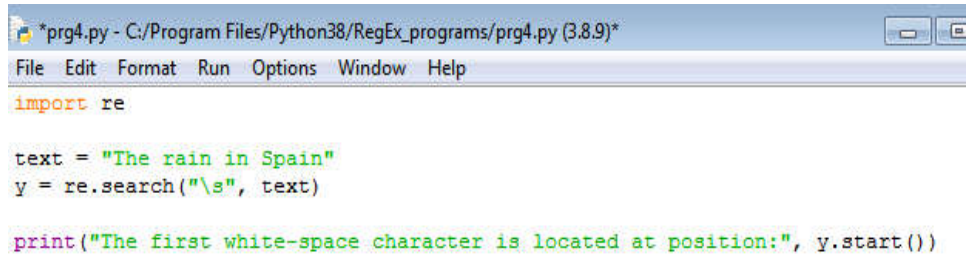
```
========== RESTART: C:/Program Files/Python38/RegEx_programs/prg3.py
[]
No match
```

❖ *The search() Function*

The **search()** function searches the string for a match, and returns a **Match object** if there is a match.

If there is more than one match, only the first occurrence of the match will be returned.

Example 3.4 **Search for the first white-space character in the string**

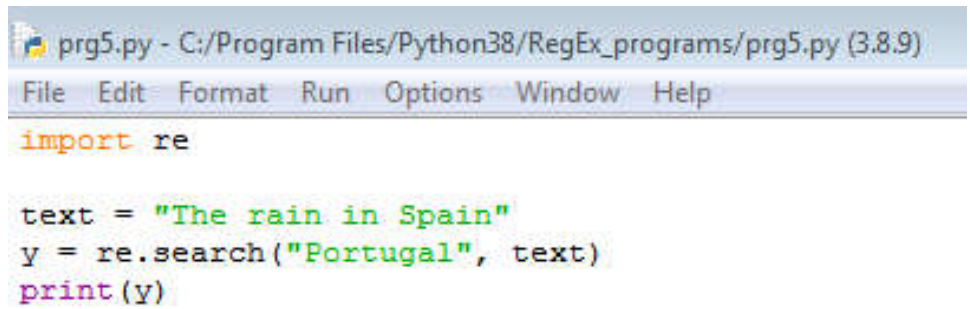


Output:

```
========== RESTART: C:/Program Files/Python38/RegEx_programs/prg4.py
The first white-space character is located at position: 3
```

If no matches are found, the value **None** is returned:

Example 3.5 **Make a search that returns no match**



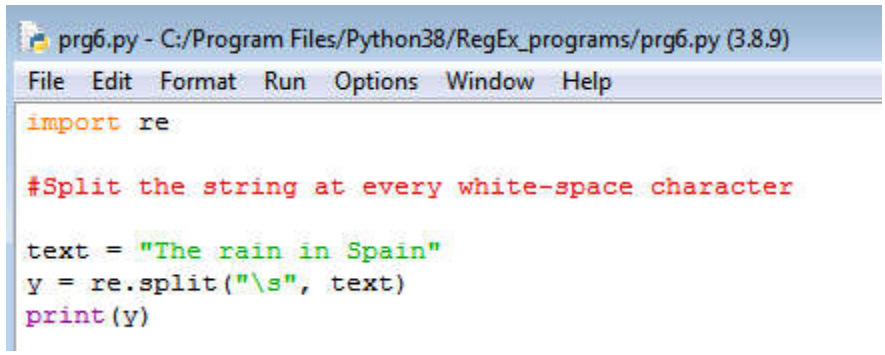
Output:

```
========== RESTART: C:/Program Files/Python38/RegEx_programs/prg5.py
None
```

❖ ***The split() Function***

The **split()** function returns a list where the string has been split at each match.

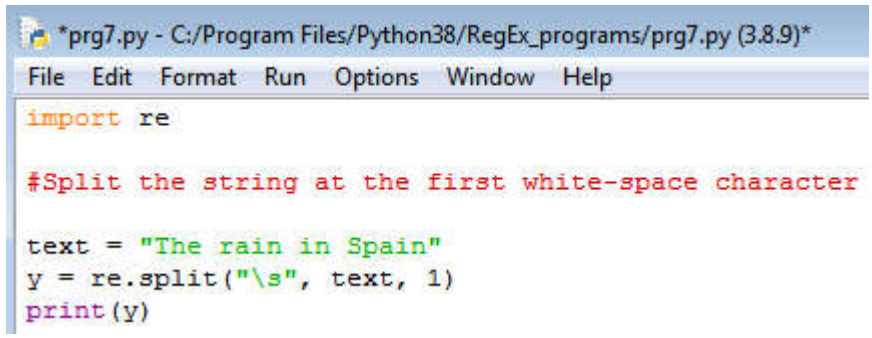
Example 3.6 Split at each white-space character



Output:

```
========== RESTART: C:/Program Files/Python38/RegEx_programs/prg6.py
['The', 'rain', 'in', 'Spain']
```

*Example 3.7***Split the string only at the first occurrence**



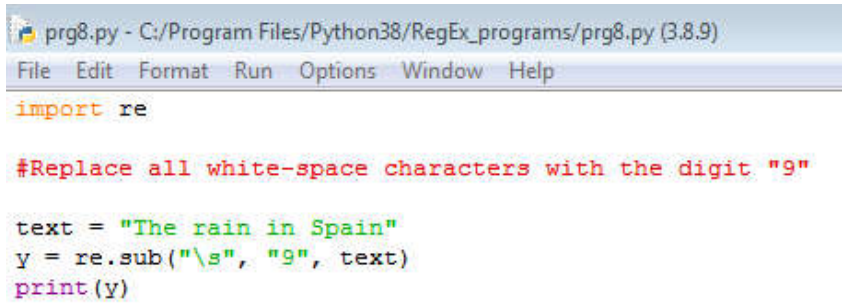
Output:

```
========== RESTART: C:/Program Files/Python38/RegEx_programs/prg7.py
['The', 'rain in Spain']
```

❖ ***The sub() Function***

The sub() function replaces the matches with the text of your choice.

Example 3.8 Replace every white-space character with the number 9

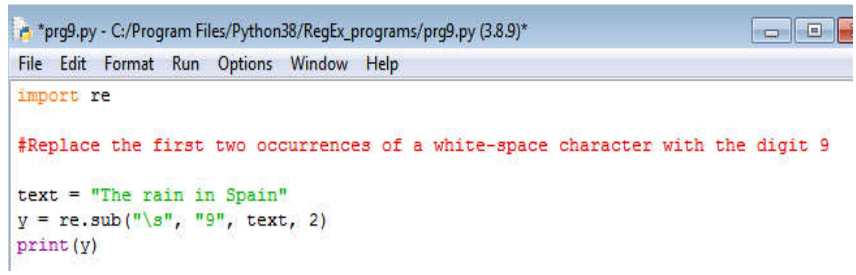


Output:

```
========== RESTART: C:/Program Files/Python38/RegEx_programs/prg8.py
The9rain9in9Spain
```

You can control the number of replacements by specifying the **count** parameter:

Example 3.9 **Replace the first 2 occurrences**



Output:

```
========== RESTART: C:/Program Files/Python38/RegEx_programs/prg9.py
The9rain9in Spain
```

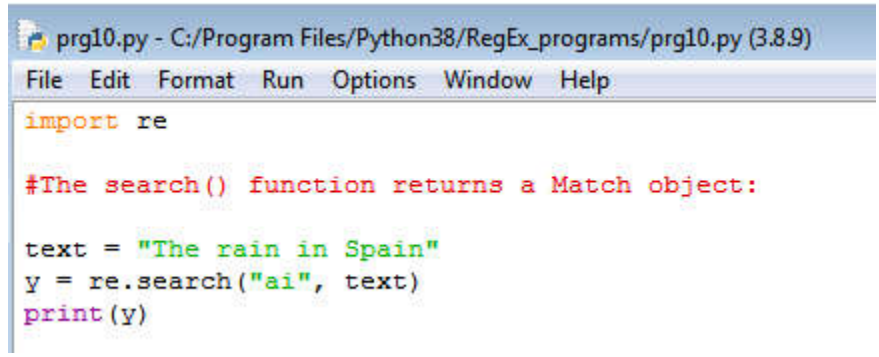
❖ **Match Object**

A Match Object is an object containing information about the search and the result.

Note:

If there is no match, the value **None** will be returned, instead of the **Match Object**.

Example 3.10 Do a search that will return a Match Object



Output:

```
========== RESTART: C:/Program Files/Python38/RegEx_programs/prg10.py
<re.Match object; span=(5, 7), match='ai'>
```

The Match object has properties and methods used to retrieve information about the search, and the result:

.span() - returns a tuple containing the startand end positions of the match.

.string- returns the string passed into the function.

.group()-returns the part of the string where there was a match.

Example 3.11 Print the position (start- and end-position) of the first match occurrence.

The regular expression looks for any words that starts with an upper case "S".

prg11.py - C:/Program Files/Python38/RegEx_programs/prg11.py (3.8.9)

File Edit Format Run Options Window Help

```
import re

#Search for an upper case "S" character in the beginning of a word, and print it

text = "The rain in Spain"
y = re.search(r"\bS\w+", text)
print(y.span())
```

Output:

```
========== RESTART: C:/Program Files/Python38/RegEx_programs/prg11.py
(12, 17)
```

Example 3.12 **Print the string passed into the function**

prg12.py - C:/Program Files/Python38/RegEx_programs/prg12.py (3.8.9)

File Edit Format Run Options Window Help

```
import re

#The string property returns the search string:

text = "The rain in Spain"
y = re.search(r"\bS\w+", text)
print(y.string)
```

Output:

```
========== RESTART: C:/Program Files/Python38/RegEx_programs/prg12.py
The rain in Spain
```

Example 3.13 Print the part of the string where there was a match.

The regular expression looks for any words that starts with an upper case "S":

prg13.py - C:/Program Files/Python38/RegEx_programs/prg13.py (3.8.9)

File Edit Format Run Options Window Help

```
import re

text = "The rain in Spain"
y = re.search(r"\bS\w+", text)
print(y.group())
```

Output:

```
========== RESTART: C:/Program Files/Python38/RegEx_programs/prg13.py
Spain
```

❖❖❖❖

UNIT-II

4

GUI PROGRAMMING IN PYTHON-I

Unit Structure

4.0 OBJECTIVE

- Understand GUI Programming
- Understand concept of window and main loop
- Understand different widget
- Understand menu driven programming

4.1 INTRODUCTION

The graphical user interface (GUI), developed in the late 1970s by the Xerox Palo Alto research laboratory and Apple's Macintosh and Microsoft's Windows, was designed.

4.2 WHAT IS GUI

Graphical user interfaces (GUI) would become the standard of user-centered design in software application programming, providing users to operate computers through the direct manipulation of graphical icons such as Text box, Buttons, Scroll bars, Spin box, Windows, Radio Button, Menus, and Cursors etc.

4.3 ADVANTAGES OF GUI

1. GUI is very user-friendly
2. GUI is more attractive and multi-colored.
3. It is much easy than the command-driven interface
4. User can switch easily between tasks on the GUI interface

Disadvantages of GUI:

1. It becomes complex if the user needs to communicate with the computer directly
2. It is fully based applications require more RAM .
3. GUI uses more processing power compared to other interface types

4.4 INTRODUCTION TO GUI LIBRARY

Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Python provides various options for developing graphical user interfaces (GUIs) such as Tkinter, wxPython, JPython.

Tkinter − Tkinter is the Python interface to the Tk GUI toolkit shipped with Python.

wxPython − This is an open-source Python interface for wxWindows http://wxpython.org.

JPython − JPython is a Python port for Java which gives Python scripts.
Tkinter Programming:

Tkinter is the standard GUI library for Python. Tkinter provides a easy as well as fast way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.

Creating a GUI application using Tkinter you need to do the following steps −
Import the Tkinter module.
Create the GUI application main window.

Add one or more of the above-mentioned widgets to the GUI application.
Enter the main event loop to take action against each event triggered by the user.

Importing Tkinter:

How to import Tkinter?

To import the Tkinter, use the import statement and write the tkinter and create iGts object. Call the Tk() GUI Kit. Call the mainloop() method from the Tk().

Example:-

```
>>> import tkinter as x
>>> a=x.Tk()
>>>a.mainloop()
```

Output

4.5 LAYOUT MANAGEMENT

Tkinter has three built-in layout managers: the pack , grid , and place managers. The place geometry manager positions widgets using absolute positioning. The pack geometry manager organizes widgets in horizontal and vertical boxes. The grid geometry manager places widgets in a two dimensional grid.

1. Pack method()

   The pack method of geometry manager organizes the widget in blocks before the appear on the parent widget.

Syntax:

Widget.pack(pack option);

There are three option of pack method and they are expand, fill and side

Expand – if the value of expand option is set to true then widget expand to fill any space. If the value of expand option is set to False then it is used in widget's parent.

Fill- The fill option is used to fill any extra space allocated to it by packer. The fill method has its own minimal dimensions and they are NONE, X (horizontally), Y (vertically) or Both (Vertical and Horizontal)

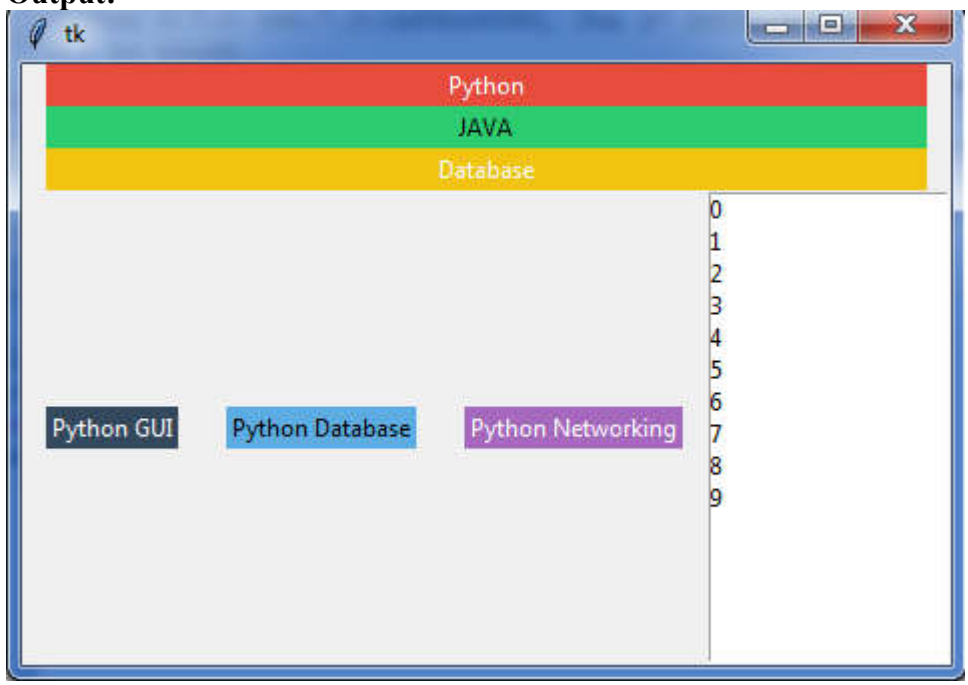
Side- It find out which side of the parent widget packs against : TOP , BOTTOm, LEFT or RIGHT.

Example-

```
fromtkinter import *
r = Tk()
r.geometry('350x300+120+100')
l1 = Label(r, text="Python", bg="#E74C3C", fg="white").pack(fill=X, padx=12)
```

```
l2 = Label(r, text="JAVA", bg="#2ECC71", fg="black").pack(fill=X,
padx=12)
l3 = Label(r, text="Database", bg="#F1C40F", fg="white").pack(fill=X,
padx=12)
l4 = Label(r, text="Python GUI", bg="#34495E", fg="white").pack(fill=X,
padx=12, pady=12, side=LEFT)
l5 = Label(r, text="Python Database", bg="#5DADE2",
fg="black").pack(fill=X, padx=12, side=LEFT)
l6 = Label(r, text="Python Networking", bg="#A569BD",
fg="white").pack(fill=X, padx=12, side=LEFT)
listbox = Listbox()
listbox.pack(fill=BOTH, expand=1)
for i in range(10):
listbox.insert(END, str(i))
mainloop()
```

Output:



Grid method:

Grid method geometry manager is used to organize widgets in a table like structure in the parent widget.

Syntax:

Widget.grid(options)

There are following option

Column : It is a column to put widget in default 0.

Columnspan : This tells that how many columns widget occupies. The default value is 1.

Row : it is a row to put widget in. default the first row that is still empty.

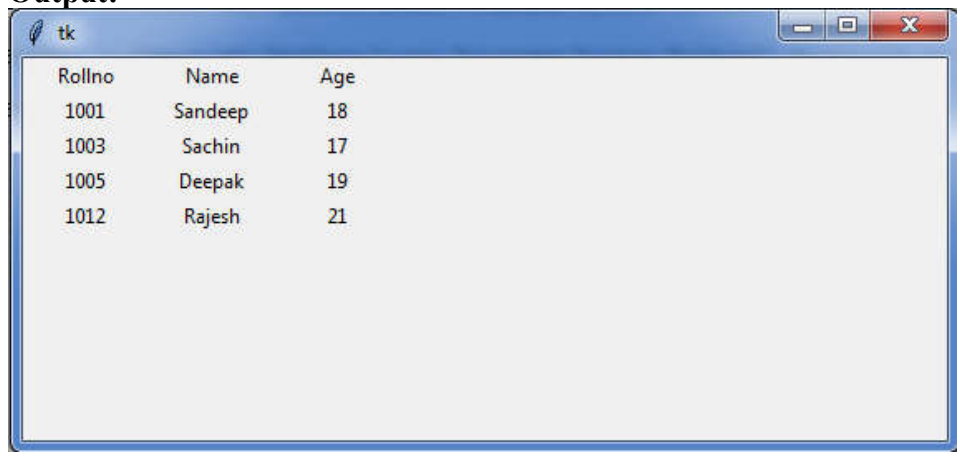
Rowspan : it tells how many row widget have occupied, be default is is 1.

Example:

```
fromtkinter import *
```

```
Label(text="Rollno", width=10).grid(row=0, column=0)
Label(text="Name", width=10).grid(row=0, column=1)
Label(text="Age", width=10).grid(row=0, column=2)
Label(text="1001", width=10).grid(row=1, column=0)
Label(text="Sandeep", width=10).grid(row=1, column=1)
Label(text="18", width=10).grid(row=1, column=2)
Label(text="1003", width=10).grid(row=2, column=0)
Label(text="Sachin", width=10).grid(row=2, column=1)
Label(text="17", width=10).grid(row=2, column=2)
Label(text="1005", width=10).grid(row=3, column=0)
Label(text="Deepak", width=10).grid(row=3, column=1)
Label(text="19", width=10).grid(row=3, column=2)
Label(text="1012", width=10).grid(row=4, column=0)
Label(text="Rajesh", width=10).grid(row=4, column=1)
Label(text="21", width=10).grid(row=4, column=2)
mainloop()
```

Output:



Place()

The place() method of layout manager organizes widgets in the parent widget by placing them in a specific position.

Syntax:

widget.place(place option)

The following option:

Anchor – The exact spot of widget other option refer to may ne N,E, S, W, NE, NW, SE or SW, compass directions indicating the corners and sides od widget: default is NW.

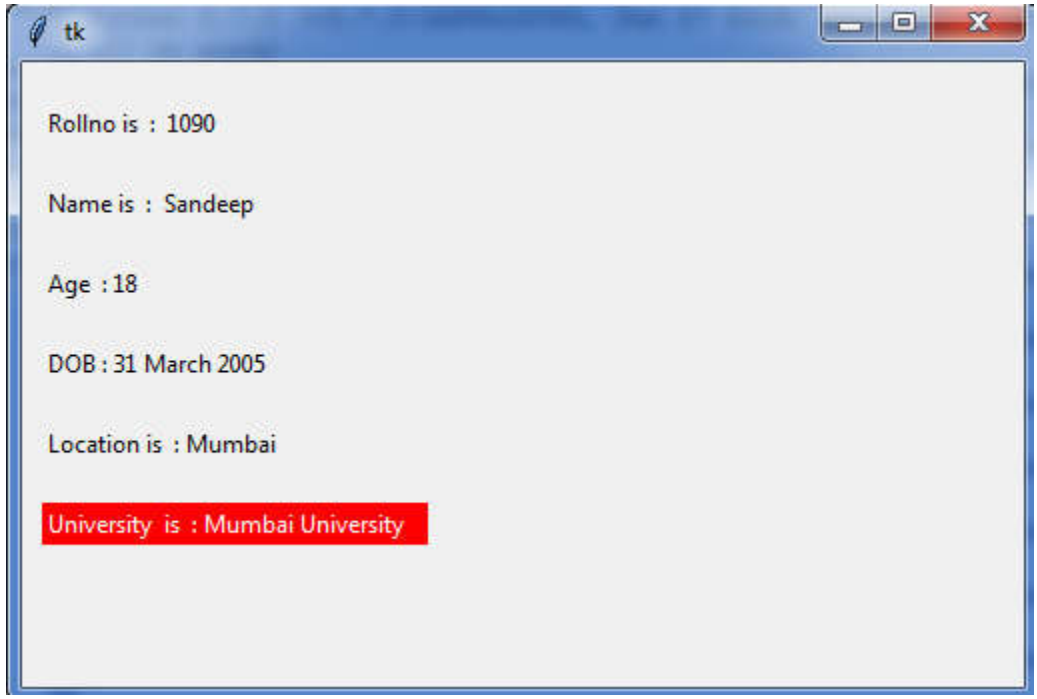
Bordermode – INSIDE to indicate that other option refer to the parent's inside, OUTSIDE otherwise.

Height, width- It determines the height and width in pixel.
X,y – It determines horizontal and vertical offset in pixcel.

```
fromtkinter import *
```

```
root = Tk()

Label(root, text="Rollno is  :  1090 ").place(x=10, y=20)

Label(root, text="Name is  :  Sandeep ").place(x=10, y=60)

Label(root, text="Age  : 18   ").place(x=10, y=100)

Label(root, text="DOB : 31 March 2005  ").place(x=10, y=140)

Label(root, text="Location is  : Mumbai ").place(x=10, y=180)

Label(root, text="University  is  : Mumbai University  ", bg="red",
fg="white").place(x=10, y=220)

mainloop()
```

Output:



4.6 EVENTS AND BINDINGS

Events and bindings plays important role in events handing in python.Widget configuration and styling is also vital in GUI.

To bind an event to any widget in python , bind() is used.

Syntax

widget.bind(event,handler)

Where,

event can be button clicked or key press etc

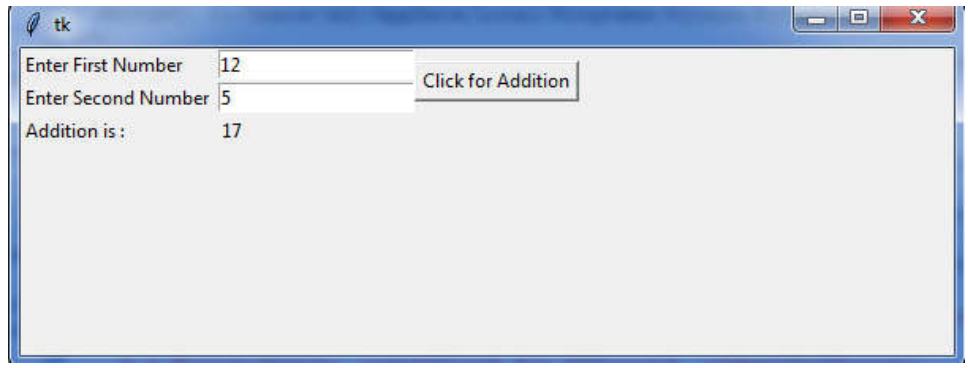
handler can be type of button or key used to handle event

Whenever event occurs the handler is called to execute particular function related to the event.

Example:

```
fromtkinter import *

defadditon():

res=int(n1.get())+int(n2.get())

mt.set(res)

m = Tk()

mt=StringVar()

Label(m, text="Enter First Number ").grid(row=0, sticky=W)

Label(m, text="Enter Second Number ").grid(row=1, sticky=W)

Label(m, text="Addition is :").grid(row=3, sticky=W)

result=Label(m, text="", textvariable=mt).grid(row=3,column=1,
sticky=W)

n1 = Entry(m)

n2 = Entry(m)

n1.grid(row=0, column=1)

n2.grid(row=1, column=1)

b = Button(m, text="Click for Addition", command=additon)

b.grid(row=0, column=2,columnspan=2, rowspan=2)

mainloop()
```

output:



Events

Tkinter provides a powerful mechanism to deal with events. For each widget, you can bind python functions and methods to events. If an event matching the event description occurs in the widget, the given handler is called with an object describing the event.

The event sequence is given as a string, using the following:

Syntax

(modifier-type-detail)

The type field is the essential part of an event specifier, whereas the “modifier” and “detail” fields are not obligatory and are left out in many cases. They are used to provide additional information for the chosen “type”. The event “type” describes the kind of event to be bound, e.g. actions like mouse clicks, key presses or the widget got the input focus.

Events and its Description

<buttton> - A mouse button is pressed with the mouse pointer over the widget. If you press down a mouse button over a widget and keep it pressed.

<motion> (x,y) - The mouse is moved with a mouse button being held down.The current position of the mouse pointer is provided in the x and y members of the event object passed to the callback, i.e. event.x, event.y

<ButtonRelease>- Event, if a button is released.

The current position of the mouse pointer is provided in the x and y members of the event object passed to the callback, i.e. event.x, event.y

<Double-Button>- Similar to the Button event, see above, but the button is double clicked instead of a single click

Class bindings

The bind method we used in the above example creates an instance binding. This means that the binding applies to a single widget only; if you create new frames, they will not inherit the bindings.

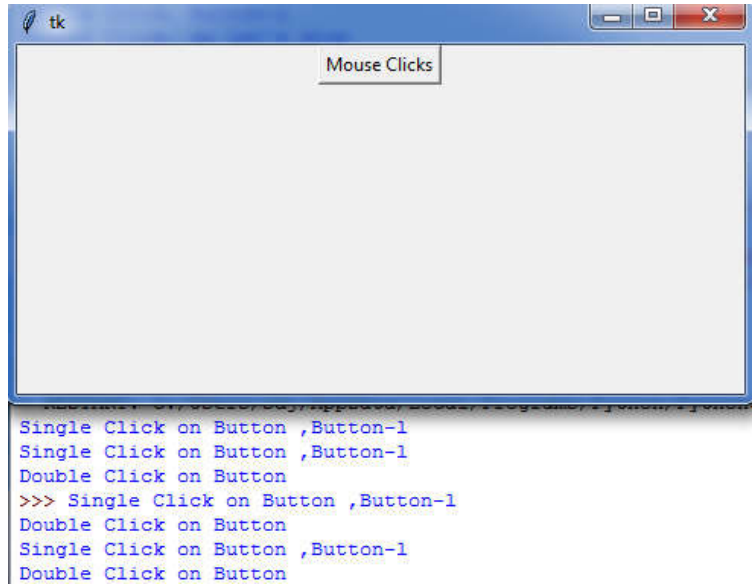
But Tkinter also allows you to create bindings on the class and application level

Example:

```
fromtkinter import *

def function1(event):

print("Single Click on Button ,Button-l")

def function2(event):
```

```
print("Double Click on Button")
import sys; sys.exit()
widget = Button(None, text='Mouse Clicks')
widget.pack()
widget.bind('<Button-1>', function1)
widget.bind('<Double-1>', function2)
widget.mainloop()
```

Output:



4.7 FONTS

The Simple Tuple fonts are commonly use for specify the font. The tuple contains first element font family, Second element size in points and third element style modifiers like underline, bold etc.

Example:

(“Arial”,”28”,”bold”

For Font object Fonts

User can create a font object by importing the tkFont and using its font class constructor.

Import tkFont

Font= tkFont.Font(option…)

Option are:

Family – It is the font family name as a string

Size- It is the font height as an integer in points.

Weight – it uses “bold” for boldface

Underline use 1 for underline, 0 for normal

Example

X=tkFont.Font(family=”Arial”, size=14, weight=”bold”)

4.8 COLORS

Standard attributes and properties of Widgets-

There are some common attributes like color, size, and font.

Color:

Color is represented in the string format. User can specify the color in the following:

Name

User can use any locally defined standard color name like “red”, “green”, “white”, “black”, “green” etc.

Hexadecimal unit

Instead of color name use hexadecimal digit like ‘#fff’ for white, ‘#00000’ for balck, ‘#000fff000’ for pure green.

Color option

The following color option

activebackground - It is used to set Background color for the active widget

activeforeground - It is used to set foreground color for the active widget

Background – bg is used to set background color for the active widget

highlightbackground – bg is used to set background color for the highlight region when the widget has focus.

highlightcolor– It is used to set the foreground color for the highlight region when the widget has focus.

Selectbackground - It is used to set the background color for the selected item of the widget.

Selectforeground - It is used to set the foreground color for the selected item of the widget.

disabledforeground - It is used to set foreground color for the disable widget

Foreground – fg is used to set foreground color for the active widget

4.9 SUMMARY

1. GUI is more attractive and multi-colored.
2. Tkinter is the standard GUI library for Python. Python when combined with
3. Tkinter provides a fast and easy way to create GUI applications. Tkinter has three built-in layout managers: the pack , grid , and place managers.

4.10 REFERENCE FOR FURTHER READING

1. Paul Gries , Jennifer Campbell, Jason Montojo, *Practical Programming: An Introduction to Computer Science Using Python 3*, Pragmatic Bookshelf, 2/E 2014
2. James Payne , *Beginning Python: Using Python 2.6 and Python 3,* Wiley India, 2010

4.11 UNIT END EXERCISES

1. Explain the layout management features.
2. Explain bind with example
3. Explain color in GUI.
4. Explain Font in GUI.
5. Write a program for addition and multiplication of two number using Entry

5

GUI PROGRAMMING IN PYTHON-II

Unit Structure

5.0 OBJECTIVE

- Understand canvas, frame, Label
- Understand concept of Entry, Message, List
- Understand Check button, Button
- Understand Radio Button, Text etc

5.1 INTRODUCTION

Graphical user interfaces (GUI) would become the standard of user-centered design in software application programming, providing

users to operate computers through the direct manipulation of graphical icons such as Text box, Buttons, Scroll bars, Spin box etc.

5.2 DRAWING ON CANVAS:

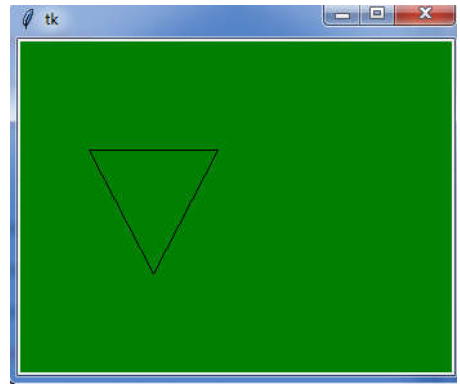
a. line

The create_line method creates a line item on the Canvas.

Example:

```
fromtkinter import *
t = Tk()
C = Canvas(t, bg="green", height=350, width=400)
coord = 10, 50, 240, 210
arc = C.create_line(55, 85, 155, 85, 105, 180, 55, 85)
C.pack()
t.mainloop()
```

output :



b. oval

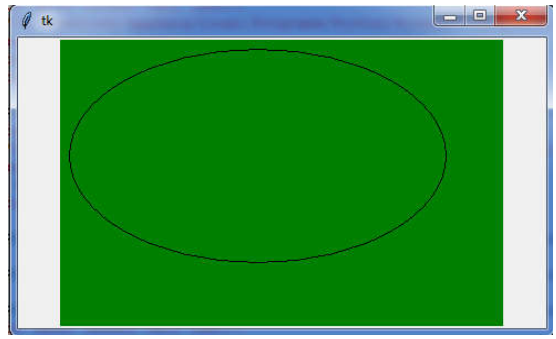
The create_oval() method is used to create a circle item. The first four parameters are the bounding box coordinates of the circle. In other words, they are x and y coordinates of the top-left and bottom-right points of the box, in which the circle is drawn.

Example:

```
fromtkinter import *
t = Tk()
C = Canvas(t, bg="green", height=350, width=400)
coord = 10, 50, 240, 210
```

```
arc = C.create_oval(10, 10, 350, 200, width=1)
C.pack()
t.mainloop()
```

output:



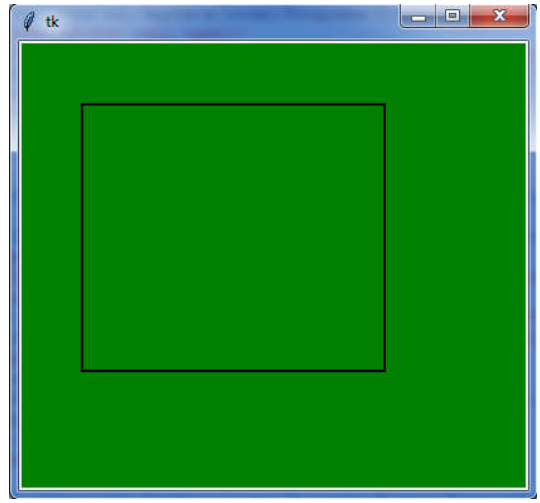
c. rectangle

To create a rectangle create_rectangle() method is used.
This method accepts 4 parameters x1, y1, x2, y2. Here x1 and y1 are the coordinates for the top left corner and x2 and y2 are the coordinates for the bottom right corner.

Example:

```
fromtkinter import *
t = Tk()
C = Canvas(t, bg="green", height=350, width=400)
coord = 10, 50, 240, 210
arc = C.create_rectangle(50, 50, 290, 260, width=2)
C.pack()
t.mainloop()
```

output:



2. Widgets such as :

a. Frame

The frame widget is used to group the widget in a friendly way. Its helps to look the GUI organized. It is like a container which arranges the position of the other widget .

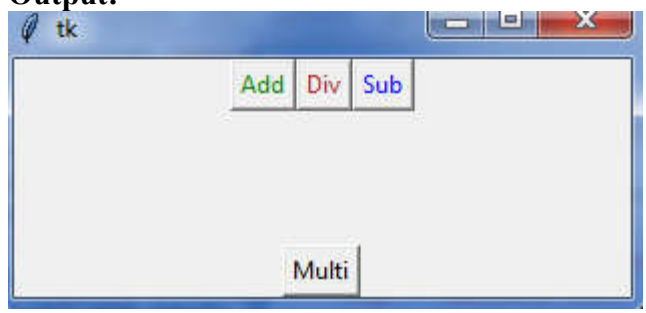
Syntax:

F=Frame (master, option)

Example:

```
fromtkinter import *
root = Tk()
frame = Frame(root)
frame.pack()
bottomframe = Frame(root)
bottomframe.pack( side = BOTTOM )
button1 = Button(frame, text="Add", fg="green")
button1.pack( side = LEFT)
button2 = Button(frame, text="Div", fg="brown")
button2.pack( side = LEFT )
button3 = Button(frame, text="Sub", fg="blue")
button3.pack( side = LEFT )
button4 = Button(bottomframe, text="Multi", fg="black")
button4.pack( side = BOTTOM)
root.mainloop()
```

Output:



b. Label

Labels

The label widget is a display box where we can place text or images. We can change label text any time we want . If we want to underline the text we can do that and also we can span text across multiple lines

Syntax

Simple syntax to create this widget −

x = Label (master, option, ...)

The argument master represents the parent window and the argument option is the option used by label widget as a key-value pairs and they are separated by comma.

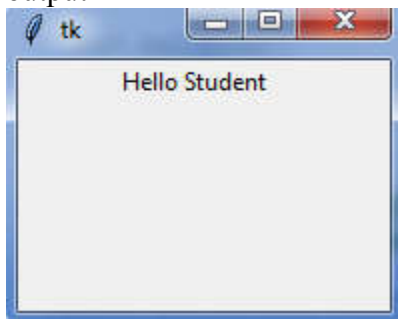
The list of most commonly used options for this widget

Anchor	This options controls where the text is positioned. The default is anchor=CENTER.
Bg	The background color displayed behind the label.
Bd	The size of the border around. Default is 2 pixels.
Font	The font option specifies in what font that text will be displayed.
Image	To display a static image in the label widget.
Width	Width of the label in characters.

Example:

```
importtkinter as tk
r = tk.Tk()
x = tk.Label(r, text="Hello Student   ")
x.pack()
r.mainloop()
```

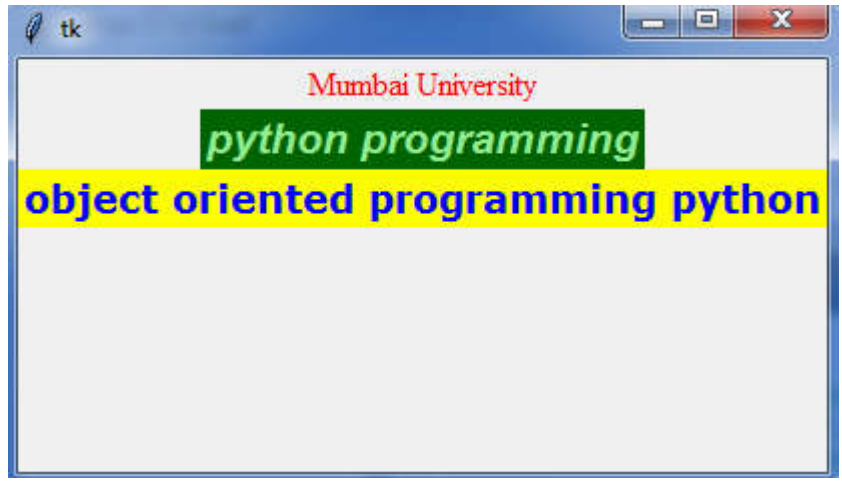
output



Example 2:

```
importtkinter as tk
r = tk.Tk()
tk.Label(r,
                text="Mumbai University",
                fg = "red",
                font = "Times").pack()
tk.Label(r,
                text="python programming",
                fg = "light green",
                bg = "dark green",
                font = "Helvetica 16 bold italic").pack()
tk.Label(r,
                text="object oriented programming python",
                fg = "blue",
                bg = "yellow",
                font = "Verdana 14 bold").pack()
r.mainloop()
```

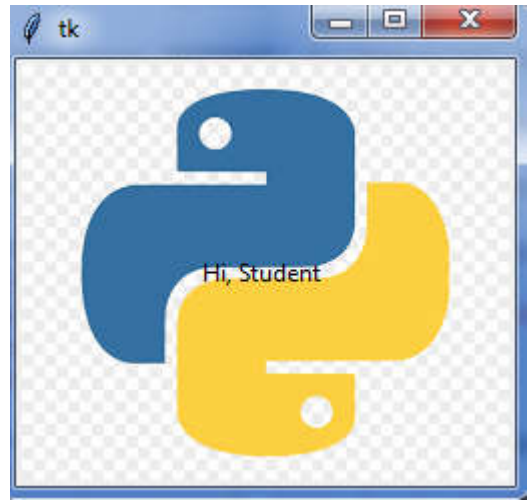
Output:



Example:

```
importtkinter as tk
r = tk.Tk()
img1 = tk.PhotoImage(file="python123.gif")
txt = "Hi, Student "
w = tk.Label(r,   compound = tk.CENTER,
                                text=txt,
image=img1).pack(side="right")
r.mainloop()
```

output:



c. Button

Labels

The label widget is a display box where we can place text or images. We can change label text any time we want . If we want to underline the text we can do that and also we can span text across multiple lines

Syntax

Simple syntax to create this widget −

x = Label (master, option, ...)

The argument master represents the parent window and the argument option is the option used by label widget as a key-value pairs and they are separated by comma.

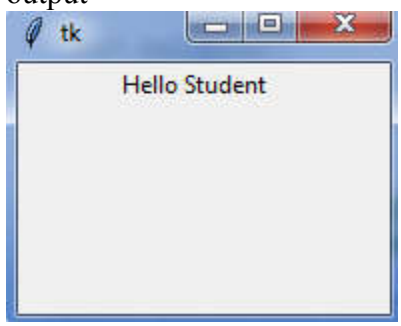
The list of most commonly used options for this widget

Anchor	This options controls where the text is positioned. The default is anchor=CENTER.
Bg	The background color displayed behind the label.
Bd	The size of the border around. Default is 2 pixels.
Font	The font option specifies in what font that text will be displayed.
Image	To display a static image in the label widget.
Width	Width of the label in characters.

Example:

```
importtkinter as tk
r = tk.Tk()
x = tk.Label(r, text="Hello Student   ")
x.pack()
r.mainloop()
```

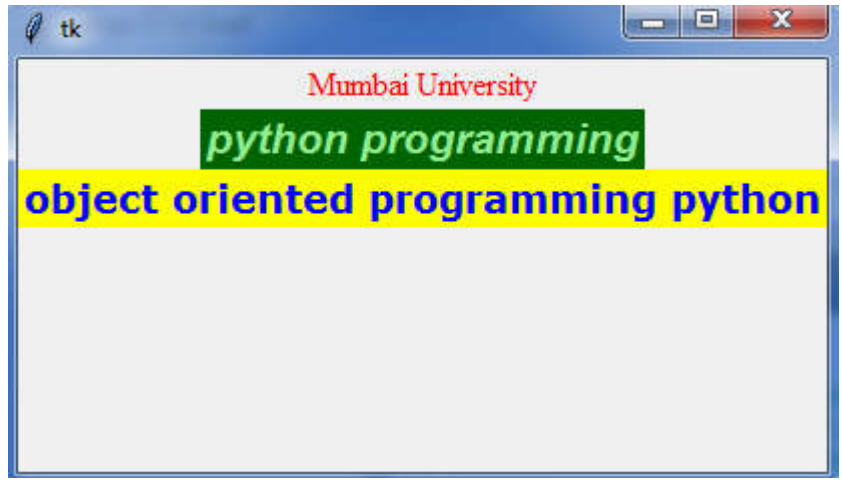
output



Example 2:

```
importtkinter as tk
r = tk.Tk()
tk.Label(r,
                        text="Mumbai University",
                        fg = "red",
                        font = "Times").pack()
tk.Label(r,
                        text="python programming",
                        fg = "light green",
                        bg = "dark green",
                        font = "Helvetica 16 bold italic").pack()
tk.Label(r,
                        text="object oriented programming python",
                        fg = "blue",
                        bg = "yellow",
                        font = "Verdana 14 bold").pack()

r.mainloop()
```

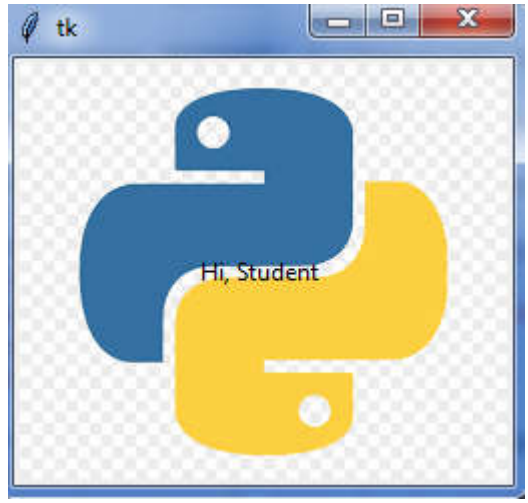
Output:



Example:

```
importtkinter as tk
r = tk.Tk()
img1 = tk.PhotoImage(file="python123.gif")
txt = "Hi, Student "
w = tk.Label(r,   compound = tk.CENTER,
                                text=txt,
image=img1).pack(side="right")
r.mainloop()
```

Output:



d. check button,

The Checkbutton widget is used to display a number of options to a user as toggle buttons. The user can then select one or more options by clicking the button.

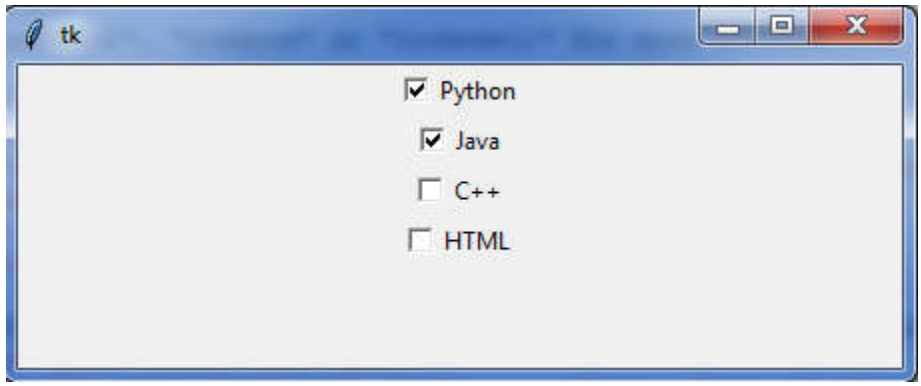
Syntax

w = Checkbutton(master, option, ...)

Example

```
fromtkinter import *

r = Tk()

C1 = Checkbutton(r, text = "Python")

C2 = Checkbutton(r, text = "Java")

C3 = Checkbutton(r, text = "C++")

C4 = Checkbutton(r, text = "HTML")

C1.pack()

C2.pack()

C3.pack()

C4.pack()

r.mainloop()
```

Output:



e. entry

The Entry widget accepts single line text strings.

Syntax

e = Entry(master, option, ...)

Parameters

master − This represents the parent window.

options − These options can be used as key-value pairs separated by commas.

Option	Description
Font	The font used for the text.
Command	A procedure to be called every time the user changes the state of this check button.
Bg	The normal background color displayed behind the label and indicator.
Bd	The size of the border around the indicator. Default is 2 pixels.
exportselection	By default, if you select text within an Entry widget, it is automatically exported to the clipboard. To avoid this exportation, use exportselection=0.
Justify	If the text contains multiple lines, this option controls how the text is justified: CENTER, LEFT, or RIGHT.

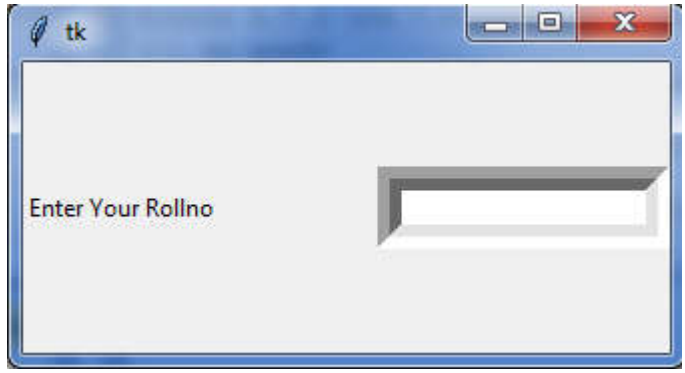
List of methods are commonly used for this widget –

Methods	Description
get()	Returns the entry's current text as a string.
icursor (index)	Set the insertion cursor just before the character at the given index.
insert (index, s)	Inserts string s before the character at the given index.
select_clear()	Clears the selection.
select_range (start, end)	Selects the text starting at the start index, up to but not including the character at the end index.

Example :

```
fromtkinter import *
t = Tk()
l1= Label(t, text="Enter Your Rollno")
l1.pack(side = LEFT)
e1 = Entry(t,bd=12)
e1.pack(side = RIGHT)
t.mainloop()
```

Output:



f. list box

The listbox widget displays a list of items. The user can select the item from the given list. ListBox can display different types of items. These items must be of the same type of font and color. The user can select one or more items from the given list according to the requirement.

Syntax

L=Listbox(master, option,…)

The argument master represents the parent window and the argument option is the option used by Listbox widget as a key-value pairs and they separated by comma.

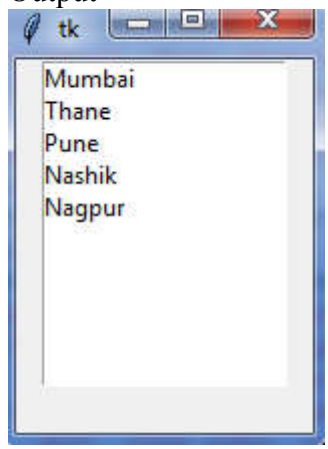
Methods on listbox:

activate (index)	Selects the line specifies by the index.
get (first, last=None)	Returns a values containing the text of the lines with indices from first to last
curselection()	Returns values containing the line numbers of the selected element or elements, counting from zero.
delete (first, last=None)	Deletes the lines whose indices are in the range [first, last]

Example:

```
fromtkinter import *
t = Tk()
Lb1 = Listbox(t)
Lb1.insert(1, "Mumbai")
Lb1.insert(2, "Thane")
Lb1.insert(3, "Pune")
Lb1.insert(4, "Nashik")
Lb1.insert(5, "Nagpur")
Lb1.pack()
t.mainloop()
```

Output



g. message

It is GUI element of tkinter. It is multiline and non-editable object. It displays the static text. If length of message is large, it automatically breaks the long text to multiple lines. It is similar to label widget. The only difference is the message widget automatically wraps the text where the label widget does not do automatically.

Syntax:

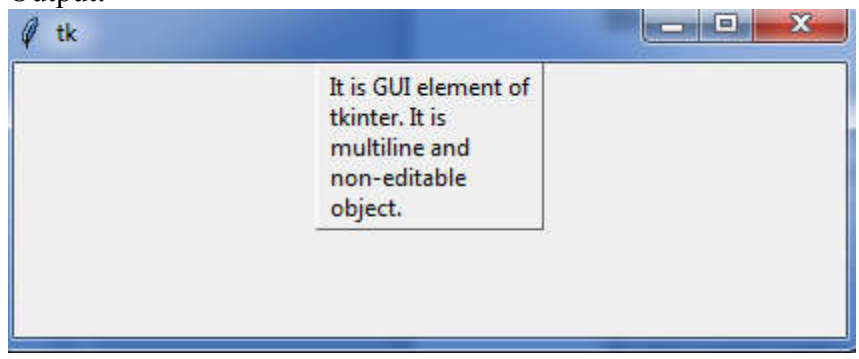
M=Message(master, option)

The argument master represents the parent window and the argument option is the option used by message widget as a key-value pairs and they separated by comma.

Example:

```
fromtkinter import *
t = Tk()
var1 = StringVar()
label1 = Message( t, textvariable=var1, relief=RAISED )
var1.set("It is GUI element of tkinter. It is multiline and non-editable object.")
label1.pack()
t.mainloop()
```

Output:



h. radio button

The radio button is also known as option button. The option button allows user to select values from the predefined set of values. Radio button contains text as well as image. We can associate the function with the option button when we select the option the function is called automatically.

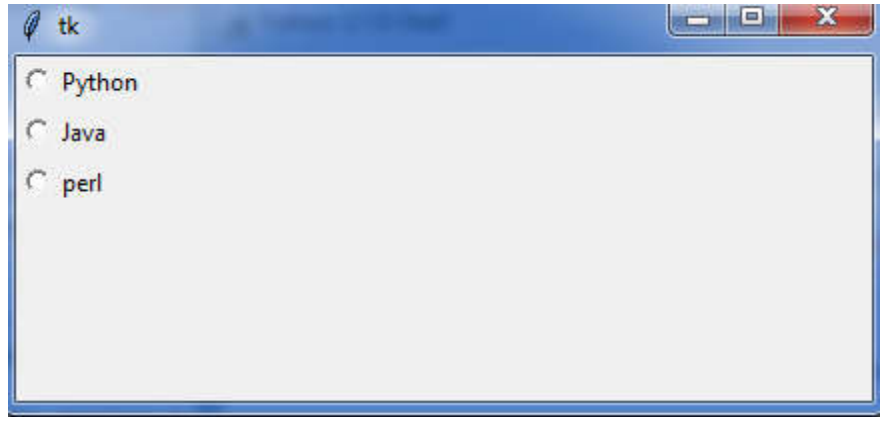
Methods use in Radio Button

Methods	Description
select()	Sets the radiobutton.
deselect()	Clears the radiobutton.
invoke()	You can call this method to get the same actions that would occur if the user clicked on the option button to change its state.

Example:

```
fromtkinter import *
t = Tk()
def function1():
selection = "selected your subject " + str(var1.get())
label.config(text = selection)
var1 = IntVar()
R1 = Radiobutton(t, text="Python", variable=var1, value=1,
command=function1)
R1.pack( anchor = W )
R2 = Radiobutton(t, text="Java", variable=var1, value=2,
command=function1)
R2.pack( anchor = W )
R3 = Radiobutton(t, text="perl", variable=var1, value=3,
command=function1)
R3.pack( anchor = W)
label = Label(root)
label.pack()
t.mainloop()
```

Output:



i. text

Text widget allows user to edit the multiline text. User can also format the text the way user want to it display. User can change the color of the text, foreground color as well as background color for the text and also user can set the font of the text.

Syntax:

T= Text(master, option,…)

Method

Text object have following methods

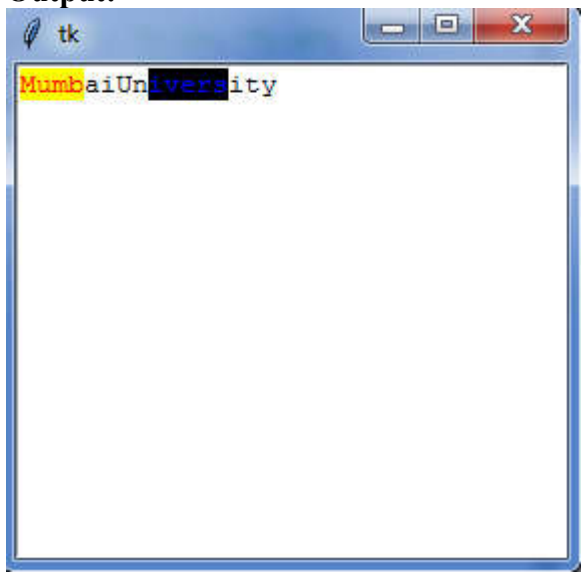
Method	Description
Get()	This method returns a specific character of text.
Insert()	This method inserts strings at the specified index location.
See(index)	This method returns true if the text located at the index position is visible.
Index()	Returns the absolute value of an index based on the given index.

Example:

```
fromtkinter import *
t = Tk()
defonclick():
pass
text = Text(t)
text.insert(INSERT, "Mumbai")
text.insert(END, "University")
text.pack()
```

```
text.tag_add("123", "1.0", "1.4")
text.tag_add("444", "1.8", "1.13")
text.tag_config("123", background="yellow", foreground="red")
text.tag_config("444", background="black", foreground="blue")
t.mainloop()
```

Output:



j. spin box

The spinbox contains the fixed number of values and it allows selecting the value from the given values. It is a standard Tkinter Entry eidget.

Syntax:

s = Spinbox(master, option)

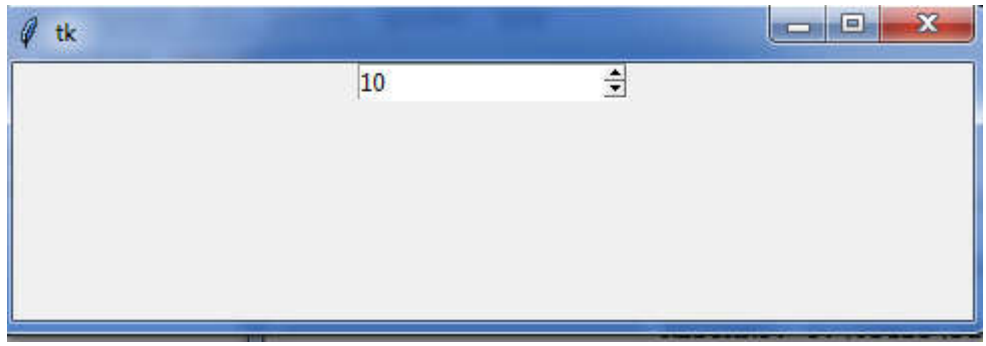
The argument master represents the parent window and the argument option is the option used by message widget as a key-value pairs and they separated by comma.

Option	Description
from_	The minimum value.
Justify	Default is LEFT
State	One of NORMAL, DISABLED, or "readonly".
To	See from.
Validate	Validation mode. Default is none
Width	Widget width, in character units. Default is 20.

Example

```
fromtkinter import *
t = Tk()
s = Spinbox(t, from_=0, to=10)
s.pack()
b.pack()
```

Output



5.4 SUMMARY

- In this chapter we studied the GUI programming with different widget.
- Widget such as Button, Text, List, Radio button etc. call function when user clicks on button and different widget.

5.5 REFERENCE FOR FURTHER READING

1. Mastering GUI Programming with Python: Develop impressive cross-platform.

2. Python GUI Programming with Tkinter: Develop responsive and powerful GUI applications with Tkinter

5.6 UNIT END EXERCISES

1. Explain Menu widget with Example
2. Write a program for Addition of two number using tkinter.
3. Write a program for factorial of given number using tkinter.
4. Explain any three widget.
5. Explain tkMessagebox module
6. Explain PanedWindow and Toplevel widget

❖❖❖❖

UNIT-III

6

DATABASE & NETWORKING CONNECTIVITY

Unit Structure

6.0 OBJECTIVE

- Understand Python and connector installation
- Understand concept of database
- Understand Create table, insert record
- Understand how the record are delete, update etc

6.1 INTRODUCTION

Python supports various databases like MySQL, Oracle, Sybase, Postgre SQL, etc. Python also supports Data Definition Language (DDL), Data Manipulation Language (DML) and Data Query Statements. For database programming, the Python DB API is a widely used module that provides a database application programming interface.

6.2 DATABASE CONNECTIVITY IN PYTHON:

a. Installing mysql connector,

Connector/Python Installation

Connector/Python runs on any platform where Python is installed. Python comes preinstalled on most Unix and Unix-like systems, such as Linux, macOS, and FreeBSD. On Microsoft Windows, a Python installer is available at the Python Download website. If necessary, download and install Python for Windows before attempting to install Connector/Python.

Connector/Python implements the MySQL client/server protocol two ways:

- As pure Python; an implementation written in Python. Its dependencies are the Python Standard Library and Python Protobuf>= 3.0.0.
- As a C Extension that interfaces with the MySQL C client library. This implementation of the protocol is dependent on the client library, but can use the library provided by MySQL Server packages (see MySQL C API Implementations).

Obtaining Connector/Python

Packages are available at the Connector/Python download site. For some packaging formats, there are different packages for different versions of Python; choose the one appropriate for the version of Python installed on your system.

Installing Connector/Python from a Binary Distribution

Connector/Python installers in native package formats are available for Windows and for Unix and Unix-like systems:

- Windows: MSI installer package
- Linux: RPM packages for Oracle Linux, Red Hat, and SuSE;
- macOS: Disk image package with PKG installer

  You may need root or administrator privileges to perform the installation operation.

Binary distributions that provide the C Extension link to an already installed C client library provided by a MySQL Server installation. For those distributions that are not statically linked, you must install MySQL

Server if it is not already present on your system. To obtain it, visit the MySQL download site.

Installing Connector/Python on Microsoft Windows

Managing all of your MySQL products, including MySQL Connector/Python, with MySQL Installer is the recommended approach. It handles all requirements and prerequisites, configurations, and upgrades.

Prerequisite. The Microsoft Visual C++ 2015 Redistributable must be installed on your system.

- MySQL Installer (recommended): When executing MySQL Installer, choose MySQL Connector/Python as one of the products to install. MySQL Installer installs the Windows MSI Installer described in this documentation.

- Windows MSI Installer (.msi file): To use the MSI Installer, launch it and follow the prompts in the screens it presents to install Connector/Python in the location of your choosing.

Like with MySQL Installer, subsequent executions of the Connector/Python MSI enable you to either repair or remove the existing Connector/Python installation.

Connector/Python Windows MSI Installers (.msi files) are available from the Connector/Python download site (see Section 4.1, "Obtaining Connector/Python"). Choose an installer appropriate for the version of Python installed on your system. As of Connector/Python 2.1.1, MSI Installers include the C Extension; it need not be installed separately.

Installing Connector/Python from Source on Microsoft Windows

A Connector/Python Zip archive (.zip file) is available from the Connector/Python download site (see Section 4.1, "Obtaining Connector/Python").

To install Connector/Python from a Zip archive, download the latest version and follow these steps:

1. Unpack the Zip archive in the intended installation directory (for example, C:\mysql-connector\) using WinZip or another tool that can read .zip files.
2. Start a console window and change location to the folder where you unpacked the Zip archive:

   $> cd C:\mysql-connector\
3. Inside the Connector/Python folder, perform the installation using this command:

   $> python setup.py install

To include the C Extension (available as of Connector/Python 2.1.1), use this command instead:

$> python setup.py install --with-mysql-capi="path_name"

The argument to --with-mysql-capi is the path to the installation directory of MySQL Server.

To see all options and commands supported by setup.py, use this command:
$> python setup.py –help

b. accessing connector module,

Configuration file is used to store credentials like permission to database, table and database related commands to specific users. The configuration file is used to the MySql server and then creates your own database on the MySql server.

Establishing a Connection with MySQL Server
MySQL is a server-based database management system. One server might contain multiple databases. To interact with a database, you must first establish a connection with the server. The general workflow of a Python program that interacts with a MySQL-based database is as follows:

- Connect to the MySQL server.
- Create a new database.
- Connect to the newly created or an existing database.
- Execute a SQL query and fetch results.
- Inform the database if any changes are made to a table.
- Close the connection to the MySQL server.

c. using connect,

d. cursor, execute & close functions

To execute a SQL query in Python, you'll need to use a cursor, which abstracts away the access to database records. MySQL Connector/Python provides you with the MySQLCursor class, which instantiates objects that can execute MySQL queries in Python. An instance of the MySQLCursor class is also called a cursor.

cursor objects make use of a MySQLConnection object to interact with your MySQL server. To create a cursor, use the .cursor() method of your connection variable:
cursor = connection.cursor()

Inserting Records in Tables

In the last section, you created three tables in your database: movies, reviewers, and ratings. Now you need to populate these tables with data.

This section will cover two different ways to insert records in the MySQL Connector for Python.

The first method, .execute(), works well when the number of records is small and the records can be hard-coded. The second method, .executemany(), is more popular and is better suited for real-world scenarios.

Using .execute()

The first approach uses the same cursor.execute() method that you've been using until now. You write the INSERT INTO query in a string and pass it to cursor.execute(). You can use this method to insert data into the movies table.

Reading Records Using the SELECT Statement

To retrieve records, you need to send a SELECT query to cursor.execute(). Then you use cursor.fetchall() to extract the retrieved table in the form of a list of rows or records.

Filtering Results Using the WHERE Clause

You can filter table records by specific criteria using the WHERE clause. For example, to retrieve all movies with a box office collection greater than $300 million, you could run the following query:

SELECT empno,ename

FROM emp

WHERE salary > 9000;

UPDATE Command

For updating records, MySQL uses the UPDATE statement

DELETE Command

Deleting records works very similarly to updating records. You use the DELETE statement to remove selected records.

Example for connection:

importmysql.connector

cnx = mysql.connector.connect(user='abc', password='123',

host='122.0.0.1',

database='xyz')

cnx.close()

Example for Connection using try

importmysql.connector

frommysql.connector import errorcode

```
try:
  cnx1 = mysql.connector.connect(user='abc',
database='emp1')
exceptmysql.connector.Error as err1:
if err1.errno == errorcode.ER_ACCESS_DENIED_ERROR:
print("wrong user name or password")
elif err1.errno == errorcode.ER_BAD_DB_ERROR:
print("Database not found")
else:
print(err1)
else:
cnx1.close()
```

Create dictionary to hold connection information

```
dbConfig={
'user':<adminName>, #your Admin Nmae
'password':<adminpwd>,#admin password
'host':122.0.0.1,#local host ip address
}
```

For creating own database use following commands

```
GUID="GuiDB"
Conn=mysql.connect(**guiConf.dbConfig)
Cursor.conn.cursor()
try:
cursor.execute(:CREATE DATABSE {} DEFAULT CHARACTER SET 'utf8'".
Format(GUIDB))
Except mysql.Error as err:
Print("Failed to create database{}".format(err))
Conn.close()
```

In the above code we created the cursor object from connection object to execute commands to MYSQL. A cursor is usually a place in a definite row in a database table.

e. Reading single & multiple results of query execution, executing different types of statements,executing transactions, Understanding exceptions in database connectivity.

Retrieving record from table we used select command. where clause are used with select command for matching particular condition.

Syntax:

Select Field No1, Field No 2,

From <Table Name>

Where Condition.

Example

```
importmysql.connector
try:
connection = mysql.connector.connect(host='localhost',
database='db1',
user='abc',
password='123')
sql = "select * from Student"
cursor = connection.cursor()
cursor.execute(sql)
records = cursor.fetchall()
print("Total number of records ", cursor.rowcount)
print("\nPrinting each row")
for row in records:
print("Student_ID = ", row[0], )
print("Student Name = ", row[1])
print("Phy  = ", row[2])
        print("Chem  = ", row[2])
print("Bio  = ", row[3], "\n")
exceptmysql.connector.Error as e:
print("Error reading data", e)
finally:
ifconnection.is_connected():
connection.close()
cursor.close()
```

Show database command:

```
Import mysql.connector as mysql
Import GuiDBConfig as guiCon
Conn=mysql.connect(**guiConfig.dbConfig)
Cursor=conn.cursor()
Cursor.execute(“SHOW DATABASE” )
```

Print(cursor.fetchall())

Conn.close()

Command for Create table Student

Student table have following fields

Student_Idint

Student_Namevarchar

Phyint

ChemInt

Bio int

Python code

Conn=mysql.connect(**guiConfig.dbConfig)

Cursor=conn.cursor()

Cursor.execute("xyz")

Cursor.execute("CREATE TABLE Student(

Student_Id INT primary key ,

Student_NameVARCHAR(12) not null,

Phy INT,

Chem INT,

Bio INT)ENGINE=innoDB")

Conn.close()

Above code create Student table in Database.

To create table having foreign key constraint

Create student_info table having following columns

Sr No INT not null auto_increment,

Student_IdInt,

MobileNoint,

Age int

Command:

Cursor.execute("USE guidb")

Cursor.execute ("CREATE TABLE Student_Info (

Sr No INT not null auto_increment,

Student_id INT,

MobilrNo INT,

AGE INT,

FOREIGN KEY (Student_Id)

REFERENCES Student(Student_ID)

ON DELETE CASCADE)

ENGINE=Innodb")

INSERT Command:

Below code is use to insert record in Student Table

Table Info

Studentid

Student Name

Phy

Chem

BIO

Insert into tables values(value1,value2,….) query is used to insert new record in table

Example

```
Import MySQLdb
db=MYSQLdb.connect("localhost","abc","123","database1")
Cursor=db.cursor()
Sql="INSERT INTO STUDENT
(STUDENT_ID,STUDENT_NAME,PHY,CHEM,BIO)
Values(101,"xyz",87,99,67)"
try:
cursor.execute(sql)
db.commit()
db.close()
```

UPDATE Command:

Update is used to update one or more records. We can use where with UPDATE.

Syntax for UPDATE

UPDATE <table Name> Set <Field Name> where <Condition>

```
Import MySQLdb
db=MYSQLdb.connect("localhost","abc","123","database1")
Cursor=db.cursor()
Sql="UPDATE STUDENT
Set Phy=67
Where Student_Id=101"
```

```
try:
cursor.execute(sql)
db.commit()
except
db.rollback()
db.close()
```

Example for update all record in table

Update chemistry subject marks by 10% of each student

```
Import MySQLdb
db=MYSQLdb.connect("localhost","abc","123","database1")
Cursor=db.cursor()
Sql="UPDATE STUDENT
Set chem=chem*.010"
try:
cursor.execute(sql)
db.commit()
except
db.rollback()
db.close()
```

DELETE Command

DELETE command is use for delete record from table. To delete particular record we used where with condition.

Syntax:

```
DELETE FROM <TABLE NAME>
Where Condition.
```

Example:

```
Import MySQLdb
db=MYSQLdb.connect("localhost","abc","123","database1")
Cursor=db.cursor()
Sql="DELETE FROM STUDENT
Where Student_Id=101"
try:
cursor.execute(sql)
db.commit()
except
```

```
db.rollback()
db.close()
```

Example for DELETE ALL Record from Table

```
Import MySQLdb
db=MYSQLdb.connect("localhost","abc","123","database1")
Cursor=db.cursor()
Sql="DELETE FROM STUDENT"
try:
cursor.execute(sql)
db.commit()
except
db.rollback()
db.close()
```

This code will delete all record from Student Table.

6.3 NETWORK CONNECTIVITY:

Python provides two levels of access to network services. At a low level, you can access the basic socket support in the underlying operating system, which allows you to implement clients and servers for both connection-oriented and connectionless protocols.

Python also has libraries that provide higher-level access to specific application-level network protocols, such as FTP, HTTP, and so on.

f. Socket module,

Sockets are the endpoints of a bidirectional communications channel. Sockets may communicate within a process, between processes on the same machine, or between processes on different continents.

Sockets may be implemented over a number of different channel types: Unix domain sockets, TCP, UDP, and so on. The socket library provides specific classes for handling the common transports as well as a generic interface for handling the rest.

The socket Module

To create a socket, you must use the socket.socket() function available in socket module, which has the general syntax −

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

Here is the description of the parameters −

socket_family − This is either AF_UNIX or AF_INET, as explained earlier.

socket_type − This is either SOCK_STREAM or SOCK_DGRAM.

protocol − This is usually left out, defaulting to 0.

g. creating server-client programs,

The socket Module

To create a socket, you must use the socket.socket() function available in socket module, which has the general syntax −

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

Here is the description of the parameters −

socket_family − This is either AF_UNIX or AF_INET, as explained earlier.

socket_type − This is either SOCK_STREAM or SOCK_DGRAM.

protocol − This is usually left out, defaulting to 0.

Server Socket Methods

s.bind()

This method binds address (hostname, port number pair) to socket.

s.listen()

This method sets up and start TCP listener.

s.accept()

This passively accept TCP client connection, waiting until connection arrives (blocking).

A Simple Server

To write Internet servers, we use the socket function available in socket module to create a socket object. A socket object is then used to call other functions to setup a socket server.

Now call bind(hostname, port) function to specify a port for your service on the given host.

Example:

```
import socket
s = socket.socket()
host = socket.gethostname()
port = 12345
s.bind((host, port))
s.listen(5)
```

```
while True:
c, addr = s.accept()
print 'Got connection from', addr
c.send('Thank you for connecting')
c.close()
```

A Simple Client

Let us write a very simple client program which opens a connection to a given port 12345 and given host. This is very simple to create a socket client using Python's socket module function.

The socket.connect(hosname, port) opens a TCP connection to hostname on the port. Once you have a socket open, you can read from it like any IO object. When done, remember to close it, as you would close a file.

```
import socket
s = socket.socket()
host = socket.gethostname()
port = 12345
s.connect((host, port))
prints.recv(1024)
s.close()
```

h. sending email,

When you send emails through Python, you should make sure that your SMTP connection is encrypted, so that your message and login credentials are not easily accessed by others. SSL (Secure Sockets Layer) and TLS (Transport Layer Security) are two protocols that can be used to encrypt an SMTP connection. It's not necessary to use either of these when using a local debugging server.

Using SMTP_SSL()

The code example below creates a secure connection with Gmail's SMTP server, using the SMTP_SSL() of smtplib to initiate a TLS-encrypted connection. The default context of ssl validates the host name and its certificates and optimizes the security of the connection.

```
importsmtplib, ssl
  port = 465  # For SSL
  password = input("Type your password and press enter: ")
  # Create a secure SSL context
```

```
context = ssl.create_default_context()

withsmtplib.SMTP_SSL("smtp.gmail.com", port, context=context) as server:

server.login("my@gmail.com", password)
```

i. reading from URL

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

With Python you can also access and retrieve data from the internet like XML, HTML, JSON, etc. You can also use Python to work with this data directly. In this tutorial we are going to see how we can retrieve data from the web. For example, here we used a guru99 video URL, and we are going to access this video URL using Python as well as print HTML file of this URL.

```
import urllib2

def main():

# open a connection to a URL using urllib2

webUrl = urllib2.urlopen("https://www.youtube.com/user/uyyyt")

#get the result code and print it

print "result code: " + str(webUrl.getcode())

# read the data from the URL and print it

data = webUrl.read()

print data

if __name__ == "__main__":

main()
```

6.4 SUMMARY

- In this chapter we studied the use of database and Python.
- Create database using python.
- Create table with constraints such as primary key, not null and foreign key etc., and insert values in table.
- Update record using update command. Delete record from table.
- Retrieve data from table using select command.

6.5 REFERENCE FOR FURTHER READING

1. Python: The Complete Reference
2. python and sql programming by Tony Coding

6.6 UNIT END EXERCISES

1. Explain the UPDATE Command
2. Explain the DELETE Command
3. Explain the INSERT Command
4. How to create database in python.
5. Explain the SELECT Command
6. Explain server client.