Unit 4

Network Programming and GUI Using Python

❖Network Programming :

Interconnection of computers is called a network. A simple network can be formed by connecting two computers using a cable. Thus, a network can have two computers or two thousand computers. For example, Internet is the largest network on the earth where millions of computers are connected.

Python provide two levels of access to network programming. These are –

- **Low Level Access :** At the low level, you can access the basic socket support of the operating system. You can implement client and server for both connection-oriented and connectionless protocols.

- **High Level Access :** At the high level allows to implement protocols like HTTP, FTP etc.

There are three requirements to establish a network:

- **Hardware :** Includes the computers, cables, modems, routers, hubs, etc.
- **Software :** Includes the programs to communicate between servers and clients.
- **Protocol :** Represents a way to establish connection and helps in sending receiving data in a standard format.

Let's now discuss Protocol in some detail.

Protocol

A protocol represents a set of rules to be followed by every computer on the network. Protocol is useful to physically move data from one place to another place on a network. While sending data or receiving data, the computers in the network should follow some rules. For example, if a computer wants to send a file on a network, it is not possible to send the entire file in a single step. The file should be broken into small pieces and then only they can be sent to other computer. The following are two types of protocol models based on which other protocols are developed.

- TCP / IP Protocol
- UDP

TCP/IP Protocol :

TCP (Transmission Control Protocol) / IP (Internet Protocol) is the standard protocol model used on any network, including Internet.

TCP/IP model has got the following five layers:

- Application layer
- TCP
- IP
- Data link layer
- Physical layer

Application layer is the topmost layer of the TCP/IP model that directly in interacts with an application (or data). This layer receives data from the application and formats the data. Then it sends that data to the next layer called TCP in the form of continues stream of bytes. The TCP, upon receiving the data from the application layer, will divide it into small segments called ‘packets’. A packet contains a group of bytes of data. These packets are then sent to the next IP layer. IP layer inserts the packets into envelopes called ‘frames’. Each frame contains a packet, the IP address of destination computer, the IP address of source computer, and some

additional bits useful in error detection and correction. These frames are then sent to data link layer which dispatches them to correct destination computer on the network. The last layer, which is called the Physical layer, is used to Physically transmit data on the network using the appropriate hardware.

Consider Figure :

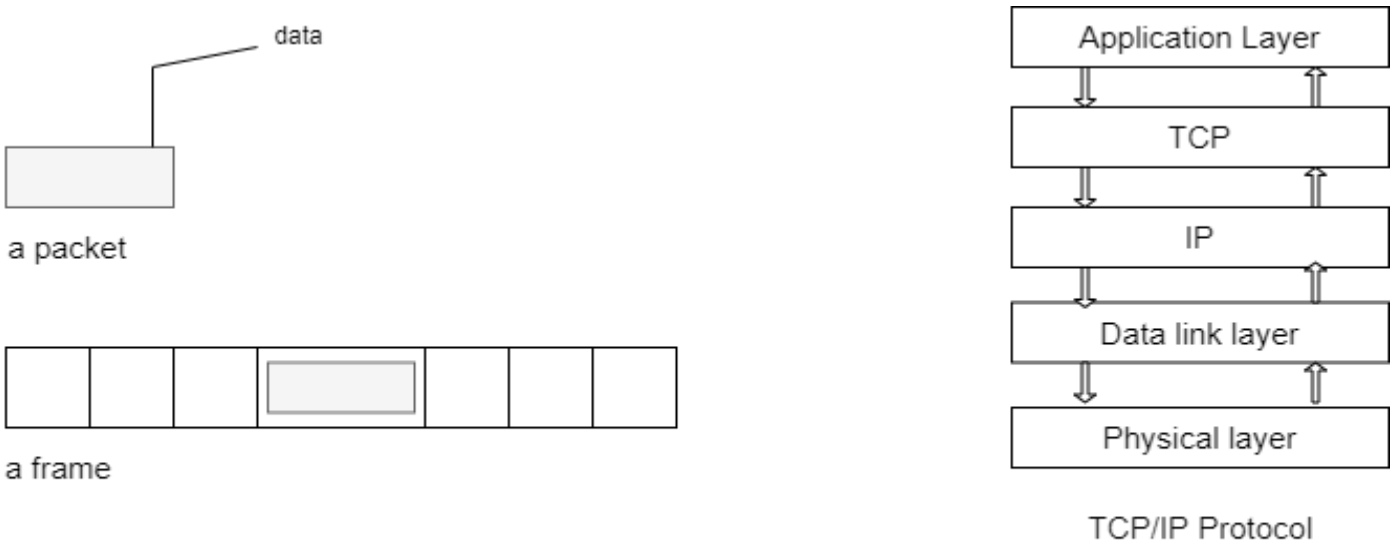

Figure : Packet, Frame and TCP/IP Protocol Layers

Of course, to send data from one place to another, first of all the computers should be correctly identified on the network. This is done with the help of IP addresses. An IP address is a unique identification number given to every computer on the network. It contains four integer numbers in the range of 0 to 255 and separated by a dot as:

For example : 216.58.194.197

This IP address may represent, for example a website on a server machine on Internet as:

For example : www.gmail.com

Therefore, to open ‘gmail.com’ site we can type the site address as ‘www.gmail.com’ or its IP address as ‘216.58.194.197’. But when we type the IP address in numeric form, that number is mapped to the website

automatically. This mapping service is available on Internet, which is called 'DNS' (Domain Naming Service or System). So, Domain Naming System (DNS) is a service on Internet that maps the IP addresses with corresponding website names. On Internet, IP addresses of 4 bytes are used and this version is called IP address version 4. The next new version of IP address is version 6, which uses 16 bytes to identify a computer.

TCP/IP takes care of number of bits send and whether all the bits are received duly by the destination computer. So it is called 'connection oriented reliable protocol'. Every transmitted bit is accountable in this protocol. Almost all the protocols on Internet use TCP/IP model internally.

HTTP (Hyper Text Transfer Protocol) is the most widely used protocol on Internet, which is used to transfer web pages (.html files) from one computer to another computer on Internet. FTP (File Transfer Protocol) is useful to download or upload files from and to the server. SMTP (Simple Mail Transfer Protocol) is useful to send mails on network. POP (Post Office Protocol) is useful to receive mails into the mail boxes. NNTP (Network News Transfer Protocol) is used to transfer news articles on Internet.

User Datagram Protocol (UDP) :

UDP is another protocol that transfer data in a connection less and unreliable manner. It will not check how many bits are sent or how many bits are actually received at the other side. During transmission of data, there may be loss of some bits. Also, the data sent may not be received in the same order. Hence UDP is generally not used to send text. UDP is used to send images, audio files, and video files. Even if some bits are lost, still the image or audio file can be composed with a slight variation that will not disturb the original image or audio.

Sockets

It is possible to establish a logical connecting point between a server and a client so that communication can be done through that point. This point is called ‘socket’. Each socket is given an identification number, which is called ‘port number’. Port number takes 2 bytes and can be from 0 to 65,535. Establishing communication between a server and a client using sockets is called ‘socket programming’.

We should use a new port number for each new socket. Similarly, we should allot a new port number depending on the services provided on a socket. Every new service on the net should be assigned a new port number. Please have a look at some already allotted port numbers for the services shown in Table. The port numbers from 0 to 1023 are used by our computer system for various applications (or services) and hence we should avoid using this port number in our networking programs.

Some Reserved Port Number and Associated Services:

Port Number	Application or Service
13	Date and time services
21	FTP which transfer files
23	Telnet, which provides remote login
25	SMTP, which delivers mails
67	BOOTP, which provides configuration at boot time
80	HTTP, which transfers web pages
109	POP2, which is a mailing service
110	POP3, which is a mailing service
119	NNTP, which is for transferring news articles
443	HTTPS, which transfer web pages securely

The point of the story is that a server on a network uses socket to connect to client. When the server wants to communicate with several clients simultaneously, several sockets are needed. Every socket will have a different port number, as shown in Figure.

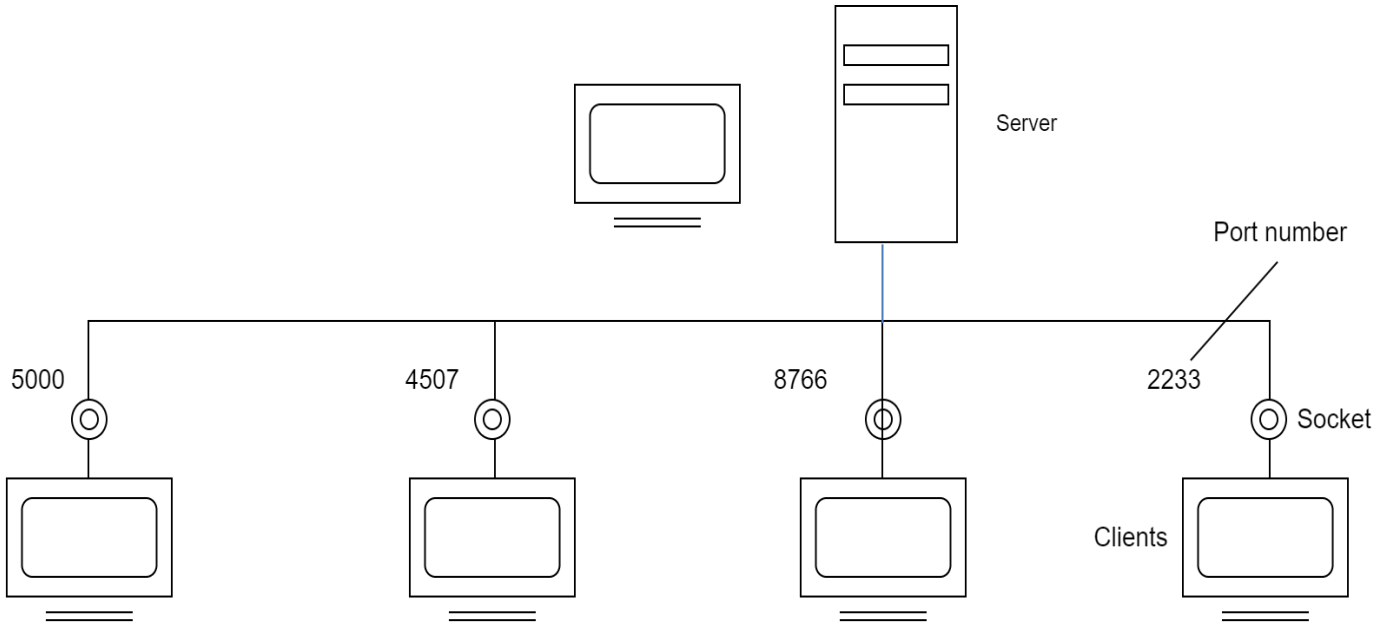

Figure : A Server Connected with Clients through Socktes

To create a socket, Python provides *socket* module. The socket() function of socket module is useful to create a socket object as:

s = socket.socket(address_family, type)

Here, the first argument 'address_family' indicates which version of the IP address should be used, whether IP address version 4 or version 6. This argument can take either or the following two values.

socket.AF_INET # Internet protocol (IPv4) – this is default

socket.AF_INET6 # Internet protocol (IPv6)

The second argument is 'type' which represents the type of the protocol to be used, whether TCP/IP or UDP. This can assume one of the following two values:

socket.SOCK_STREAM # for TCP – this is default

socket.SOCK_DGRAM # for UDP

Knowing IP Address

To know the IP Address of a website on Internet, we can use gethostbyname() function available in socket module. This function takes the website name and returns its IP Address. For example,

addr = socket.gethostbyname('www.google.co.in')

will return the IP Address of google.co.in website into 'addr' variable. If there is no such website on Internet, then it returns 'gaierror' (Get Address Information Error).

Program 1: A python program to find the IP address of a website.

```
# knowing IP Address of a website
import socket

# take the sever name
host = 'www.facebook.com'

try:
    # know the ip address of the website
    addr = socket.gethostbyname(host)
    print('IP Address = ' + addr)

except socket.gaierror: # if get address info error occurs
    print('The website does not exist')
```

URL

URL (Uniform Resource Locator) represents the address that is specified to access some information or resource on internet. Look at the example URL:

http://www.dreamtechpress.com:80/index.html

The URL contains four parts:

- The protocol to use (http://).
- The server name or IP address of the server (www.dreamtechpress.com).
- The third part represents port number, which is optional(:80).
- The last part is the file that is referred. This would be generally index.html or home.html file (/index.html).

When a URL is given, we can parse the URL and find out all the parts of the URL with the help of urlparse() function of *urllib.parse* module in python. We can pass the URL to urlparse() function, it returns a tuple containing the parts of the URL.

```
tpl = urllib.parse.urlparse('urlstring')
```

We can retrieve the individual parts of the URL from the tuple 'tpl' using the following attributes:

- scheme = this gives the protocol name used in the URL.
- netloc = gives the website name on the net with port number if present.
- path = gives the path of the web page.
- port = gives the port number.

We can also get the total URL from the tuple by calling geturl() function. These details are shown in Program 2.

Program 2: A python program to retrieve different parts of the URL and display them.

```
# to find different parts of a URL
import urllib.parse
#take any url
url = 'http://www.dreamtechpress.com:80/engineering/computer-science.html'

# get a tuple with parts of the url
tpl = urllib.parse.urlparse(url)

#display the contents of the tuple
print(tpl)

#display different parts of the url
print('scheme = ',tpl.scheme)
print('Net location = ', tpl.netloc)
print('path = ', tpl.path)
print('parameters = ', tpl.params)
print('port number = ', tpl.port)
print('total url = ', tpl.geturl())
```

Reading the Source Code of a Web Page:

If we know the URL of a web page, it is possible to get the source code of the web page with the help of urlopen() function. This function belongs to *urllib*, request module. When we provide the URL of the web page, this function stores its source code into a file-like object and returns it as:

file = urllib.request.urlopen(https://www.python.org/)

Now, using read() method on 'file' object, we can read the data of that object. This is shown in Program 3. Please remember that while executing this program, we should have Internet connection switched on in our computer.

Program 3 : A python program that reads the source code of Web page.

```
# reading source code of a website page from Internet

import urllib.request

# store the url of the page into the file object
file = urllib.request.urlopen("https://www.python.org/")

# read data from file and display
print(file.read())
```

Downloading a Web Page from Internet:

It is also possible to download the entire web page from Internet and save it into our computer system. Of course, the images of the web page may not be downloaded. To download the web page, we should have Internet connection switched on in our computer.

First we should open the web page using the urlopen() function. This function returns the content of the web page into a file like object. Now, we can read from this object using read() method as:

file = urllib.request.urlopen("https://www.python.org/")

content = file.read()

The urlopen() function can raise 'urllib.error.HTTPError' if the web page is not found. The next step is to open a file and write the 'content' data into that file. Since web pages contains binary data, to store the web pages, we have to open the file in binary write mode as:

f = open('myfile.html', 'wb') # we want to write data into myfile.html

f.write(content) # write it

Program 4 : A python program to download a web page from internet and save it into our computer.

```
# reading a websote page from Internet and saving it into our computer.
import urllib.request

try:
    #store the url of the page into file object
    file = urllib.request.urlopen("https://www.python.org/")

    # read data from file and store into content object
    content = file.read()

except urllib.error.HTTPError:
    print("The web page does not exist")
    exit()

# open a file for writing
f = open('myfile.html', 'wb')
# write content into the file
f.write(content)
#close the file
f.close()
```

Downloading an Image from Internet:

We can connect our computer to Internet and download images like .jpg, .gif or .png files from Internet into our computer system by writing a simple python program. For this purpose, we can use urlretrive() function or *urllib.request* module. This function takes the URL of the image file and our own file name as arguments.

download = urllib.request.urlretrive(url, “myimage.jpg”)

Here, ‘download’ is a tuple object. ‘url’ represents the URL string of the image location on Internet. “myimage.jpg” is the name of the file on which the image will be downloaded.

Program 5 : A python program to download an image from the Internet into our computer system.

```
# downloading an image from Internet
import urllib.request

#copy the image url
url = "https://cdn.pixabay.com/photo/2020/02/06/09/39/summer-4823612_960_720.jpg"

#download the image as myimage.jpg in curent directory
download = urllib.request.urlretrieve(url, "myimage1.jpg")
```

A TCP/IP Server :

A server is a program that provides services to other computers on the network or Internet. Similarly a client is a program that receives services from the servers. When a sever wants to communicate with a client, there is a need of a socket. A socket is a point of connection between the server and client. The following are the general steps to be used at server side:

1. Create a TCP/IP socket at server side using socket() function
   s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
   Here, socket.AF_INET represents IP Address version 4 and socket.SOCK_STREAM indicates that we are using TCP/IP protocol for communication with client. Anyhow, a socket uses IP address version 4 and TCP/IP protocol by default. Hence, we can create a socket without giving protocol version and type of the protocol as:
   s = socket.socket() # this uses TCP/IP protocol by default

2. Bind the socket with host name and port number using bind() method. Here, host name can be an IP Address or a web site name. If we want to run this program in an individual computer that is not connected in any network, then we can take host name as ‘localhost’. This represents that the server is running in the local system and not on any network. The IP Address of the ‘localhost’ is 127.0.0.1. As an alternative to ‘localhost’, we can also mention this IP Address. bind() method is used in the following format.
   s.bind((host, port)) # here, (host, port) is a tuple.

3. We can specify maximum number of connection using listen() method as :
   s.listen(5); # maximum connection allowed are 5

4. The server should wait till a client accepts connection. This is done using accept() method as:
   c, addr = s.accept() # this method returns c and addr
   Here, 'c' is connection object that can be used to send messages to the client. 'addr' is the address of the client that has accepted the connection.

5. Finally, using send() method, we can send message string to client. The message strings should be sent in the form of byte streams as they are used by the TCP/IP Protocol.
   c.send(b"message string")
   Observe the 'b' prefixed before the message string. This indicates that the string is a binary string. The other way to convert a string into binary format is using encode() method, as:
   string.encode()

6. After sending the messages to the client, the server can be disconnected by closing the connection object as:
   c.close()

Program 6 : A python to create a TCP/IP server program that sends message to a client.

```
# a TCP/IP server that sends message to client
import socket

#take the server name and port number
host = 'localhost'
port = 5000

# create a socket at server side using TCP/IP protocol
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# bind the socket with server and port number
s.bind((host, port))

# allow maximum 1 connection to the socket
s.listen(1)

# wait till a client accepts connection
c, addr = s.accept()

# display client address
print("Connection form : ",str(addr))

# send message to the client after encoding into binary string
c.send(b"Hello client, how are U")
msg = "Bye!"
c.send(msg.encode())

# disconnect the server
c.close()
```

Save this program as server1.py. Let's not run this server program now. We will run it after developing the client program since the server needs a client that receives data.

A TCP/IP Client :

A client is a program that receives the data or services from the server. We use the following general steps in the client program.

1. At the client side, we should first create the socket object that uses TCP/IP protocol to receive data. This is done as:
   s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

2. Connect the socket to the server and port number using connect() method.
   s.connect((host, port))

3. To receive the message from the server, we can use recv() method as :
   msg = s.recv(1024)
   Here, 1024 indicates the buffer size. This is the memory used while receiving the data. It means, at a time, 1024 bytes of data can be received from the server. We can change this limit in multiples of 1024. For example, we can use 2048 or 3072 etc. The received strings will be in binary format. Hence they can be converted into normal strings using decode() method.

4. Finally, we should disconnect the client by calling close() method on the socket object as:
   s.close()

Program 7 : A python program to create TCP/IP client program that receives messages from the server.

```
# a TCP/IP client that receives messages from server
import socket

# take the server name and port number
host = 'localhost'
port = 5000

# create a client side socket using TCP/IP protocol
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# connect it to server and port number
s.connect((host, port))

# receiving message string from server, at a time 1024 B
msg = s.recv(1024)

# repeat as long as message strings are not empty
while msg:
    print("Received : " + msg.decode())
    msg = s.recv(1024)

# disconnected the client
s.close()
```

Save the above program as client1.py. The programs 6 and 7 should be opened in two separate DOS windows and then executed. First we run the server1.py program in the left side DOS window and then we run the client1.py program in the right side DOS window. We can see the client's IPAddress and the port number being displayed at the server side window.

The messages sent by the server are displayed at the client side window, as shown in the output.

A UDP Server :

If we want to create a server that uses UDP protocol to send message, we have to specify socket.SOCK_DGRAM while creating the socket object, as :

```
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

It means the server will send data in the form of packets called 'datagrams'. Now, using sendto() function, the server can send data to the client. Since UDP is a connection-less protocol, server does not know where the data should be sent. Hence we have to specify the client address in sendto() function, as:

```
s.sendto("message string", (host, port))
```

Here, the "message string" is the binary string to be sent. The tuple (host, port) represents the host name and port number of the client.

In program 8, we are creating a server to send data to the client. The server will wait for 5 second after running it, and then it sends two messages to the client.

Program 8 : A python program to create a UDP server that sends messages to the client.

```
# a UDP server that sends messages to client
import socket
import time

# take the server name and port number
host = 'localhost'
port = 5000

# Create a socket at server side to use UDP protocol
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# let the server waits for 5 seconds
time.sleep(5)

# send messages to the client after encoding into binary string
s.sendto(b"Hello client, how are U", (host, port))
msg = "Bye!"
s.sendto(msg.encode(), (host, port))

# disconnected the server
s.close()
```

Save this program as server2.py. we will run this program only after the client program is also ready.

A UDP Client :

At the client side, the socket should be created to use UDP protocol as :

```
s = socket.socket(socket.AF_INET,  socket.SOCK_DGRAM)
```

The socket should be bound to the server using bind() method as :

```
s.bind((host, port))
```

Now, the client can receive messages with the help of recvfrom() method as :

```
msg, addr = s.recvfrom(1024)
```

This method receives 1024 bytes at a time from the server which is called buffer size. This method returns the received messages into ‘msg’ and the address of the server into ‘addr’. Since the client does not know how many messages (i.e. strings) the server sends, we can recvfrom() method inside a while loop and receive all the messages as :

```
while msg:
    Print('Received : ', + msg.decode())
    msg, addr = s.recvfrom(1024)
```

But the problem here is that the client will hang when the server disconnects. To rectify this problem, we have to set some time for the socket so that the client will automatically disconnect after that time elapses. This is done using settimeout() method.

```
s.settimeout()
```

This method instructs the socket to block when 5 seconds time is elapsed. Hence if the server disconnects, the client will wait for another 5 seconds time and disconnects. The settimeout() method can raise socket.timeout exception.

Program 9 : A python program to create a UDP client that receives message from the server.

```
# a UDP client that receives messages from server
import socket

#take the server name and port number
host = 'localhost'
port = 5000

# create a client side socket that uses UDP protocol
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# connect it to server with host name and port number
s.bind((host, port))

# receive message string from server, at a time 1024 B
msg, addr = s.recvfrom(1024)

try:
    # let the socket blocks after 5 seconds if the server disconnects
    s.settimeout(5)

    # repeat as long as message strings are not empty
    while msg:
        print('Received : ' + msg.decode())
        msg, addr = s.recvfrom(1024)
except socket.timeout:
    print("Time is over and hence terminating...")

# disconnect the client
s.close()
```

Save this program as client2.py. Now, run the client2.py program in a DOS window and the server2.py program in another DOS window. It is better to run the client2.py program first so that it will wait for the server. Then run the server2.py program. This program will send two strings to the client after 5 seconds from the time of its execution. The client receives them. Once the server is disconnected, the client will wait for another 5 seconds and then terminates, as shown in the output.

Remove the settimeout() method from client2.py program and re-run both the client and server. You can observe that the client hangs and it will never terminate.

File Server :

A file server is a sever program that accepts a file name from a client, searches for the file on the hard disk and sends the content of the file to the client. When the requested file is not found at server side, the server sends a message to the client saying 'file does not exist'. We can create a file server using TCP/IP type of socket connection.

Program 10 : A python program to create file server that receives a file name from a client and sends the contents of the file.

```
# a server that sends file contents to client
import socket

# take server name and port number
host = 'localhost'
port = 6767

# create TCP socket
s = socket.socket()

# bind socket to host and port number
s.bind((host, port))

# maximum 1 connection is accepted
s.listen(1)

# wait till client accepts a connection
c, addr = s.accept()
print("A client accepted connection")
```

```
# accepts file name from client
fname = c.recv(1024)

# convert file name into normal string
fname = str(fname.decode())
print("File name received from client: " + fname)

try:
    # open the file at sever side
    f = open(fname, 'rb')

    # read content of the file
    content = f.read()

    # send file content to client
    c.send(content)
    print("File content send to client")

    # close the file
    f.close()
except FileNotFoundError:
    c.send(b'File does not exist')
# disconnect server
c.close()
```

File Client :

A file client is a client side program that sends a request to the server to search for a file and send the file content. We have to type the file name from the keyboard at the file client. This file name is sent to the file server and the file contents are received by the client in turn. We can create a file client program using TCP/IP type of socket connection.

Program 11 : A python program to create a file client program that sends a file name to the server and receives the file contents.

```
# a client that sends and receives data
import socket

# take server name and port number
host = 'localhost'
port = 6767

# create a tcp socket
s = socket.socket()

# connect to server
s.connect((host, port))

# type file name from the keyboard
filename = input("Enter file name : ")
```

```
# send the file name to the server
s.send(filename.encode())

# receive file content from server
content = s.recv(1024)
print(content.decode())

# disconnect the client
s.close()
```

Two-Way Communication between Server and Client :

It is possible to send data from client to the server and make the server respond to the client's request. After receiving the response from server, the client can again ask for some information from the server. In this way both the server and client can establish two-way communication with each other. It means sending and receiving data will be done by both server and client. The following program will give you an idea of how to achieve this type of communication. These programs are at very fundamental level and need refinement. When refined, we can get programs for server and client which can take part in real chatting.

Program 12 : A python program to create a basic chat server program in python.

```
# a server that receives and sends messages - save this as chatserver.py
import socket

host = '127.0.0.1'
port = 9000

# create server side socket
s = socket.socket()
s.bind((host, port))

# let maximum number of connections are 1 only
s.listen(1)

# wait till a client connects
c, addr = s.accept()
print("A Client connected")

# server runs continuously
while True:
    # receive data from client
```

```
    data = c.recv(1024)

    # if client sends empty string, come out
    if not data:
        break
    print("From client: " + str(data.decode()))

    # enter response data from server
    data1 = input("Enter Response : ")

    # send that data to client
    c.send(data1.encode())

# close connection
c.close()
```

Program 13 : A python program creating a basic chat client program in python.

```
# a client that sends and receives messages - save this code as chatclient.py
import socket

host = '127.0.0.1'
port = 9000

# create client side socket
s = socket.socket()
s.connect((host, port))

# enter data at client
str = input("Enter data : ")

# continue as long as exit not entered by user
while str != 'exit':
    # send data from client to server
    s.send(str.encode())

    # receive the response data from server
    data = s.recv(1024)
    data = data.decode()
    print("From server: " + data)

    # enter data
    str = input("Enter data: ")

# close connection
s.close()
```

Now run the chatserver.py and chatclient.py programs in separate DOS windows. Start entering data first in the chatclient. This data is received by the chatserver. Then enter response at the chatserver which will be sent to chatclient as shown in the output.

Sending a Simple Mail :

It is possible to develop a Python program that can send a mail to any email address. For this purpose, we need SMTP class of smtplib module. When we create an object of SMTP class, we actually connect to smtp(simple mail transfer protocol) server that is useful to send the mail. For example to connect to gmail.com server we can write:

```
server = smtplib.SMTP('smtp.gmail.com',587)
```

Here, we want to connect to gmail.com website using its port number 587. If we want to connect to any other mail server, we are supposed to provide that server name along with its port number. Once we connect to mail server, we can send the mail using send_message() method as:

```
server.send_message(msg)
```

Here, 'msg' represents the total message string including the address to which the mail should be sent, address from where the mail is sent, subject of the mail and body of the mail. This 'msg' is represented as an object of MIMEText class which belongs to email.mime.text module. The word 'MIME'(Multi-Purpose Internet Mail Extension) indicates an extension of the original Internet e-mail protocol that lets people use the protocol to exchange different kinds of data on the Internet. Now, let's see how to create the 'msg' object:

```
msg = MIMEText(body) # here, body indicates mail body text
```

Once the 'msg' is created, we can full in the details like from address, to address and subject as:

```
msg['From'] = fromaddr

msg['To'] = toaddr

msg['Subject'] = "SUBJECT LINE OF OUR MESSAGE"
```

Of course, before sending the email, we have to log into the server with a valid password.

This is done in two steps as shown below:

1. First put the smtp connection into TLS (Transport Layer Security) mode. This will start the process of verifying authentication of the sender.
   server.starttls()

2. Next, we have to login to the server with our mail address and password. This done using login() method as:
   server.login(fromaddr, "password")
   All these step are shown in program 14 in a proper sequence. Just follow each step one by one and you will be able to send mail easily.

Program 14 : A python program to create a python program to email to any email address.

```
# sending email using a python program
import smtplib
from email.mime.text import MIMEText

# first type the body text for our mail
body = '''This is my text mail. This is sent to you from my Python program. I
think you appreciated me.'''

# create MIMEText class object with body text
msg = MIMEText(body)

# from which address the mail is sent
fromaddr = "gohiljay5255@gmail.com"

# to which address the mail is sent
toaddr = "sc3316643@gmail.com"

# store the addresses into msg object
msg['From'] = fromaddr
msg['To'] = toaddr
msg['Subject'] = "Hey Friend"

# connect to gmail.com server using 587 port number
server = smtplib.SMTP('smtp.gmail.com', 587)

# put the smtp connection in TLS mode.
server.starttls()
```

```
# login to the server with your correct password
server.login(fromaddr, "mypassword")

# send the message to the server
server.send_message(msg)
print("Mail sent...")

# close connection with server
server.quit()
```

While running this program, our computer should have been connected to the Internet. This program is sending the mail through gmail.com server. Now-a-days, the gmail.com server people are using more security levels and hence this program may show an error like 'SMTPAuthenticationError' which means that the gmail.com server does not allow this program to communicate. In that case, we have to inform the gmail.com server to allow "less secure apps" also. You can do this by logging into your Gmail account and then visiting the following page: https://www.google.com/settings/security/lesssecureapps

In this page, we should click on 'Turn on' option against 'Access for less secure apps'. Then the less secure application like our Program 14 are allowed by gmail.com server. Then we can run this program successfully. A caution is to 'Turn off' the 'Access for less secure apps' option after execution of this program is over. You can verify in the rediffmail.com's inbox that the mail sent by our program reached nageswara.r@rediffmail.com:

❖ GUI Programming :

- ## Creating Simple GUI :

GUI in Python :

Python offers *tkinter* module to create graphics programs. The *tkinter* represents 'toolkit interface' for GUI. This is an interface for python programmers that enable them to use the classes of TK module of TCL/TK language. Let's see what this TCL/TK is. The TCL (Tool Command Language) is a powerful dynamic programming language, suitable for web and desktop applications, networking, administration, testing and many more. It is open source and hence can be used by any one freely. TCL language uses TK (Tool Kit) language to generate graphics. TK provides standard GUI not only for TCL but also for many other dynamic programming languages like python. Hence, this TK is used by python programmers in developing GUI application through Python's *tkinter* module.

The following are the general steps involved in basic GUI programs:

1. First of all, we should create the root window. The root window is the top level window that provides rectangular space on the screen where we can display text, colors, images, components, etc.

2. In the root window, we have to allocate space for our use. This is done by creating a canvas or frame. So, canvas and frame are child windows in the root window.

3. Generally, we use canvas for displaying drawings like lines, arcs, circles, shapes etc. We use frame for the purpose of displaying components like push buttons, check buttons, menus, etc. These components are also called 'widgets'.

4. When the user clicks on a widget like push button, we have to handle that event. It means we have to respond to the events by performing the desired tasks.

Now, let's proceed further with the top level window or root window and learn to create it.

The Root Window :

To display the graphical output, we need space on the screen. This space that is initially allocated to every GUI program is called 'top level window' or 'root window'. We can say that the root window is the highest level GUI component in any tkinter application. We can reach this root window by creating an object to TK class. This is shown in program 1. The root window will have a title bar that contains minimize, resize and close options. When you click on close 'X' option, the window will be destroyed.

Program : A python program to create root window or top level window.

```
# import all components from tkinter
from tkinter import *

#create the root window
root = Tk()

# wait and watch for any events that may take place in the root window
root.mainloop()
```

Program 1 can be improved to display our own title in the root window's title bar. We can also set the size of the window to something like 400 pixel X 300 pixel. It is also possible to replace the TK's leaf image with an icon of our choice. For this, we need to use the .ico file that contains an image. These improvement are shown in program 2.

Program : A python program to create root window with some options.

```
from tkinter import *

# create top level window
root = Tk()

# set window title
root.title("My window")

# set window size
root.geometry("400x300")

# set window size
root.wm_iconbitmap('image.ico')

# display window and wait for any events
root.mainloop()
```

Fonts and color :

A font represents a type of displaying letters and numbers. In tkinter, fonts are mentioned using a tuple that contains font family name, size and font style as:

fnt = (‘Times’, -40, ‘bold italic underline overstrike’)

Here, the font family name is ‘Times’ and font size is 40 pixels. If the size is a positive number, it indicates size in points. If the size is a negative number, it indicates size in pixels. The style of the font can be ‘bold’, ‘italic’, ‘underline’, ‘overstrike’. We can mention any one or more styles as a string. The following program is useful to know the available font families in your system. All fonts are available in tkinter.font module inside tkinter module.

Program : A python program to know the available font families.

```
from tkinter import *
from tkinter import font

# create root window
root = Tk()

# get all the supported font families
list_fonts = list(font.families())

# display them
print(list_fonts)
```

Colors in tkinter can be displayed directly by mentioning their names as: blue, lightblue, darkblue, red, lightred, darkred, black, white, yellow, magenta, cyan, etc. We can also specify colors using the hexadecimal numbers in the format:

#rrggbb #8 bits per color

#rrrgggbbb # 12 bits per color

For example, #000000 represents black and #ff0000 represents red. In the same way, #000fff000 represents puregreenand#00ffffis c yan(greenplusblue). Table 22.1 gives some standard color names:

AliceBlue	DeepPink to DeepPink4	LemonChiffon1 to LemonChiffon4	RoyalBlue
AntiqueWhite	DeepSkyBlue	Magenta	RayalBlue1 to RoyalBlue4
Azure	Firebrick	MistyRose	SeaGreen1 to SeaGreen4
blue	bray / grey	orange	tomato1 to tomato4
Brown	Green1 to Green4	orangeRed1 to orangeRed4	VioletRed
Chocolate	IndianRed	Pink1 to Pink4	Yellow1 to Yellow4
Cyan	LavenderBlush	Red1 to Red4	Light / dark grey
Aquamarine	DoderBlue	Maroon	Salmon1 to Salmon4

Working with Container :

A container is a component that is used as a place where drawings or widgets can be displayed. In short, a container is a space that displays the output to the user. There are two important containers:

- **Canvas :** This is a container that is generally used to draw shapes like lines, curves, arcs and circles.

- **Frame :** This is a container that is generally used to display widgets like buttons, check buttons or menus.

After creating the root window, we have to create space, i.e. the container in the root window so that we can use this space for displaying any drawings or widgets.

Canvas :

A canvas is a rectangular area which can be used for drawing picture like lines, circles, polygons, arcs, etc. To create a canvas , we should create an object to canvas class as:

```
c = Canvas(root, bg="blue", height=500, width=600, cursor='pencil')
```

Here, 'c' is the canvas class object. 'root' is the name of the parent window. 'bg' represents background color, 'height' and 'width' represent the height and width of the canvas in pixels. A pixel (picture element) is a minute dot with which all the text and picture on the monitor are composed. When the monitor screen resolution is 1360 X 768, it indicates that the screen can accommodate 1360 pixels width-wise and 768 pixels height-wise. Consider Figure 22.4 to understand the pixels and screen coordinates in pixels. The top left corner of the screen will be at (0, 0) pixels as x and y coordinates. When we move from left to right, the x coordinates increases and when we move from top to bottom, the y coordinates increases. Thus the top right corner will be at (1360, 0). The bottom left corner will be at (0, 768) and the bottom right corner will be at (1360, 768).

The option 'cursor' represents the shape of the cursor in the canvas. Important cursor shapes are: arrow, box_spiral, center_ptr, circle, clock, coffee_mug, cross, cross_reverse, crosshair, diamond_cross, dot, double_arrow, exchange, hand1, hand2, heart, left_ptr, mouse, pencil, plus, question_arrow, right_ptr, star, tcross, top_side, umbrella, watch, xterm, X_cursor.

Colors in the canvas can be displayed directly by mentioning their names as: blue, lightblue, darkblue, red, lightred, darkred, black, white, yellow, magenta, cyan, etc. We can also specify color using hexadecimal numbers in the format.

#rrggbb # 8 bits per color

#rrrgggbbb # 12 bits per color

Once the canvas is created, it should be added to the root window. Then only it will be visible. This is done using the pack() method, as:

c.pack()

Program : A GUI program that demonstrates the creation of various shapes in canvas.

```
from tkinter import *

#create root window
root = Tk()

# create canvas as a child to root window
c = Canvas(root, bg="blue", height=700, width=1200, cursor='pencil')

# create a line in the canvas
id = c.create_line(50, 50, 200, 50, 200, 150, width=4, fill="white")

# create an oval in the canvas
id = c.create_oval(100, 100, 400, 300, width=5, fill="yellow", outline="red",
activefill="green")

# create a polygon in the canvas
id = c.create_polygon(10, 10, 200, 200, 300, 200, width=3, fill="green", outline="red",
smooth=1, activefill="lightblue")

# create a rectange in the canvas
id = c.create_rectangle(500, 200, 700, 600, width=2, fill="gray", outline="black",
activefill="yellow")

# create some text in the canvas
fnt = ('Times', 40, 'bold italic underline')
id = c.create_text(600, 100, text="My Canvas", font=fnt, fill="yellow",
activefill="white")

# add canvas to the root window
c.pack()

# wait for any events
root.mainloop()
```

Another important shape that we can draw in the canvas in an arc. An are represents a part of an ellipse or circle. Arcs can be created using the create_arc() method as:

Program 5 : A python program to create arcs in different shape

```
from tkinter import *

#create root window
root = Tk()

# create canvas as a child to root window
c = Canvas(root, bg="blue", height=700, width=1200, cursor='pencil')

# create a line in the canvas
id = c.create_line(50, 50, 200, 50, 200, 150, width=4, fill="white")

# create an oval in the canvas
id = c.create_oval(100, 100, 400, 300, width=5, fill="yellow", outline="red",
activefill="green")

# create a polygon in the canvas
id = c.create_polygon(10, 10, 200, 200, 300, 200, width=3, fill="green", outline="red",
smooth=1, activefill="lightblue")

# create a rectange in the canvas
id = c.create_rectangle(500, 200, 700, 600, width=2, fill="gray", outline="black",
activefill="yellow")

# create some text in the canvas
fnt = ('Times', 40, 'bold italic underline')
id = c.create_text(600, 100, text="My Canvas", font=fnt, fill="yellow",
activefill="white")

# add canvas to the root window
c.pack()

# wait for any events
root.mainloop()
```

Frame :

A frame is similar to canvas that represents a rectangular area where some text or widgets can be displayed. Our root window is in fact a frame. To create a frame, we can create an object of Frame class as:

f = Frame(root, height=400, width=500, bg=”yellow”, cursor=”cross”)

Here, ‘f’ is the object of Frame class. The frame is created as a child of ‘root’ window. The options ‘height’ and ‘width’ represent the height and width of the frame in pixels. ‘bg’ represents the pack ground color to be displayed and ‘cursor’ indicates the type of the cursor to be displayed in the frame.

Once the frame is created, it should be added to the root window using the pack() method as follows:

Program : A GUI program to display a frame in the root window.

```
from tkinter import *

# create root window
root = Tk()

# give a title for root window
root.title("My Frame")

# crete a frame as child to root window
f = Frame(root, height=400, width="500", bg="yellow", cursor="cross")

# attach the frame to root window
f.pack()

# let the root window wait for any events
root.mainloop()
```

Widgets :

A widget is a GUI component that is displayed on the screen and can perform a task as desired by the user. We create widgets as objects. For example, a push button is a widget that is nothing but an object of Button class. Similarly, label is a widget that is an object of label class. Once a widget is created, it should be added to canvas or frame. The following are important widgets in python.

- ❑ Button
- ❑ Label
- ❑ Message
- ❑ Text
- ❑ Scrollbar
- ❑ Checkbutton
- ❑ Radiobutton
- ❑ Entry

- ❑ Spinbox
- ❑ Listbox
- ❑ Menu

In general, working with widgets takes the following four step:

1. Create the widgets that are needed in the program. A widget is a GUI component that is represented as an object of a class. For example, a push button is a widget that is represented as Button class object.

   As an example, suppose we want to create a push button, we can create an object to Button class as:

   ```
   b = Button(f, text='My Button')
   ```

   Here, 'f' is frame object to which the button is added. 'My Button' is the text that is displayed on the button.

2. When the user interacts with a widget, he will generate an event. For example, clicking on a push button is an event. Such events should be handled by writing functions or routines. These functions are called in response to the events. Hence they are called 'callback handlers' or 'event handlers'. Other examples for events are pressing the Enter button, right clicking the mouse button, etc.

   As an example, let's write a function that may be called in response to button click.

   ```
   def buttonClick(self):
           print('You have clicked me')
   ```

3. When the user clicks on the push button, that 'clicking' event should be linked with the 'callback handler' function. Then only the button widget will appear as if it is performing some task.

   As an example, let's bind the button click with the function as:

   ```
   b.bind('<Button-1>', buttonClick)
   ```

   Here, 'b' represents the push button. <Button-1> indicates the left mouse button. When the user presses the left mouse button, the 'buttonClick'

function is called as these are linked by bind() method in the preceding code.

4. The preceding 3 steps make the widgets ready for the user. Now, the user has to interact with the widgets. This is done by entering text from the keyboard or pressing mouse button. These are called events. These events are continuously monitored by our program with the help of a loop, called 'event loop'.

   As an example, we can use the mainloop() method that waits and processes the events as:
   root.mainloop()

   Here, 'root' is the object of root window in python GUI. The events in root window are continuously observed by the mainloop() method.

Button Widget :

A push button is a component that performs some action when clicked. These buttons are created as objects of Button class as:

```
b = Button(f, text='My Button', width=15, height=2, bg='yellow', fg='blue', activebackground='green', activeforeground='red')
```

Here, 'b' is the object of Button class. 'f' represents the frame for which the button is created as a child. It means the button is shown in the frame. The 'text' option represents the text to be displayed on the button. 'width' represents the width of the button in characters. If an image is displayed on the button instead of text, then 'width' represents the width in pixels. 'height' represents the height of the button in textual lines. If an images is displayed on the button, then 'height' represents the height of the button in pixels. 'fg' represents the foreground color and 'bg' represents the back ground color of the button. 'activebackground' represents the background color when the button is clicked. Similarly, 'activeforeground' represents the foreground color when the button is clicked.

We can also display an image on the button as:

```
# first load the image into file1
file1 = PhotoImage(file="cat.gif")

# create a push button with image
b = Button(f, image=file1, width=150, height=100, bg='yellow', fg='blue',
activebackground='green', activeforeground='red')
```

In the preceding statement, observe that the width and height of the button are mentioned in pixels.

b.bind('<Button-1>', buttonClick)

Here, <Button-1> represents the mouse left button that is linked withbuttonClick() method. It means when the mouse left button is clicked, the buttonClick() method is called. This method is called event handler.

Program : A python program to create a push button and bind it with an event handler function.

```
from tkinter import *

# method to be called when the button is clicked
def buttonClick(self):
    print('You have clicked me')

# create root window
root = Tk()

# create frame as child to root window
f = Frame(root, height=200, width=300)

# let the frame will not shrink
f.propagate(0)

# attach the frame to root window
f.pack()

# create a push button as child to frame
b = Button(f, text='My Button', width=15, height=2, bg='yellow', fg='blue',
activebackground='green', activeforeground='red')

# attach button to the frame
b.pack()

# bind the left mouse button with the method to be called
b.bind("<Button-1>", buttonClick)

# the root window handles the mouse click event
root.mainloop()
```

Program : A python program to create three push buttons and change the background of the frame according to the button clicked by the user.

```
from tkinter import *

class MyButton:
    # constructor
    def __init__(self, root):
        # create a frame as child to root window
        self.f = Frame(root, height=400, width=500)

        # let the frame will not shrink
        self.f.propagate(0)

        # attach the frame to root window
        self.f.pack()

        # create 3 push buttons and bind them to buttonClick method and pass a number
        self.b1 = Button(self.f, text='Red', width=15, height=2, command=lambda:
self.buttonClick(1))
        self.b2 = Button(self.f, text='Green', width=15, height=2, command=lambda:
self.buttonClick(2))
        self.b3 = Button(self.f, text='Blue', width=15, height=2, command=lambda:
self.buttonClick(3))

        # attach buttons to the frame
        self.b1.pack()
        self.b2.pack()
        self.b3.pack()

    # method to be called when the button is clicked
    def buttonClick(self, num):
        # set the background color of the frame depending on the button clicked
        if num==1:
            self.f["bg"] = 'red'
        if num==2:
            self.f["bg"] = 'green'
        if num==3:
            self.f["bg"] = 'blue'

# create root window
root = Tk()

# create an object to MyButton class
mb = MyButton(root)

# the root window handles the mouse click event
root.mainloop()
```

Label Widget :

A label represents constant text that is displayed in the frame of container. A label can display one or more lines of text that cannot be modified. A label is created as an object of Label class as:

lbl = Label(f, text=”Welcome to Python”, width=20, height=2, font=(‘Courier’, -30, ‘bold underline’), fg=’blue’, bg=’yellow’)

Here, ‘f’ represents the frame object to which the label is created as a child. ‘text’ represents the text to be displayed. ‘width’ represents the width of the label in number of character and ‘height’ represents the height of the label in number of lines. ‘font’ represents a tuple that contains font name, size and style. ‘fg’ and ‘bg’ represents the foreground and background colors for the text.

In program, we are creating two push buttons. We display ‘Click Me’ on the first button. When this button is clicked, we will display a label “Welcome to Python”. This label is created in the event handler method buttonClick() that is bound to the first button. We display another button ‘Close’ that will close the root window upon clicking. The close button can be created as:

b2 = Button(f, text=’Close’, width=15, height=2, command=quit)

Please observe the ‘command’ option that is set to ‘quit’. This represents closing of the root window.

Program : A python program to display a label upon clicking a push button.

```
from tkinter import *

class MyButtons:
    # constructor
    def __init__(self, root):
        # create a frame as child to root window
        self.f = Frame(root, height=350, width=500)

        # let the frame will not shrink
        self.f.propagate(0)

        # attach the frame to root window
        self.f.pack()

        # create a push button and bind it to buttonClick method
        self.b1 = Button(self.f, text='Click Me', width=15, height=2,
command=self.buttonClick)

        # create another button that closes the root window upon clicking
        self.b2 = Button(self.f, text='Close', width=15, height=2, command=quit)
```

```
        # attach buttons to the frame
        self.b1.grid(row=0, column=1)
        self.b2.grid(row=0, column=2)

    # the event handler method
    def buttonClick(self):
        # Create a label with some text
        self.lbl = Label(self.f, text="Welcome to Python", width=20, height=2,
font=('Courier', -30, 'bold underline'), fg='blue')
        # attach the label in the frame
        self.lbl.grid(row=2, column=0)

# create root window
root = Tk()

# create an object to MyButtons class
mb = MyButtons(root)

# the root window thw mouse click event
root.mainloop()
```

Message Widget :

A message is similar to a label. But messages are generally used to display multiple lines of the text where as a label is used to display a single line of text. All the text in the message will be displayed using the same font. To create a message, we need to create an object of message class as:

m = Message(f, text='This is a message that has more than one line of text.', width=200, font=('Roman', 20, 'bold italic'), fg='dark goldenrod')

Here, 'text' represents the text to be displayed in the message. The 'width' option specifies the message width in pixels. 'font' represents the font for the message. We can use option 'fg' for specifying foreground color and 'bg' for specifying background color for the message text.

Program : A python program to display a message in the frame.

```
from tkinter import *

class MyMessage:
    #constructor
    def __init__(self, root):
        # create a frame as child to root window
        self.f = Frame(root, height=350, width=500)

        # let the frame will not shrink
        self.f.propagate(0)

        # attach the frame to root window
        self.f.pack()
```

```
        # create a message widget with some text
        self.m = Message(self.f, text='This is a message that has more than one line of
text.', width=200, font=('Roman', 20, 'bold italic'), fg='dark goldenrod')

        # attach message to the frame
        self.m.pack(side=LEFT)

# create root window
root = Tk()

# create an object to MyMessage class
mb = MyMessage(root)

# the root window handles the mouse click event
root.mainloop()
```

Text Widget :

Text widget is same as label or message. But Text widget has several options and can display multiple lines of text in different colors and fonts. It is possible to insert text into a Text widget, modify it or delete it. We can also display images in the text widget. One can create a text widget by creating an object to Text class as:

t = Text (root, width=20, height=10, font= (‘Verdana’, 14, ‘bold’), fg=’blue’, bg=’yellow’, wrap=WORD)

Here, ‘t’ represents the object of Text class. ‘root’ represents an object of root window or frame. ‘width’ represents the width of the Text widget in characters. ‘height’ represents the height of the widget in lines. The option ‘wrap’ specifies where to cut the line. Wrap=CHAR represents that any line that is too long will be broken at any character. wrap=WORD will break the line in the widget after the last word that fits in the line. wrap=NONE will not wrap the lines. In this case, it is better to provide a horizontal scrollbar to view the lines properly in the text widget.

Program : A python program to create a text widget with a vertical scroll bar attached to it. Also, highlight the first line of the text and display an image in the text widget.

```
from tkinter import *

class MyText:
    # constructor
    def __init__(self, root):
        # create a text widget with 20 char width and 10 lines height
        self.t = Text(root, width=20, height=10, font=('Verdana', 14, 'bold'),
fg='blue', bg='yellow', wrap=WORD)

        # insert some text into the text widget
        self.t.insert(END, 'Text widget\nThis text is insertede into the text widget.\n
This is second line\n and this is third line\n')

        # attach text to the root
        self.t.pack(side=LEFT)

        # show image in the text widget
        #self.img = PhotoImage(file='moon.gif')
        #self.t.image_create(END, image=self.img)

        # create a tag with the name 'start'
        self.t.tag_add('start', '1.0', '1.11')

        # apply color to the tag
        self.t.tag_config('start', background='red', foreground='white', font=('Lucida
console', 20, 'bold italic'))

        # create a scrollbar widget to move the text vertically
        self.s = Scrollbar(root, orient=VERTICAL, command=self.t.yview)

        # attach the scroll bar to the text widget
        self.t.configure(yscrollcommand=self.s.set)

        # attach the scroll bar to the root window
        self.s.pack(side=RIGHT, fill=Y)

# create root window
root = Tk()

# create an object to MyText class
mt = MyText(root)

# the root window handles the mouse click event
root.mainloop()
```

Scrollbar Widget :

A scroll bar is a widget that is useful to scroll the text in another widget. For example, the text in the Text, Canvas, Frame or Listbox can be scrolled from top to bottom or left to right scroll bars. There are two types of scroll bars. They are horizontal and vertical. The horizontal scroll bar is useful to view the text from left to right. The vertical scroll bar is useful to scroll the text from top to bottom. To create a scroll bar, we have to create Scrollbar class object as:

h = Scrollbar(root, orient=HORIZONTAL, bg='green', command=t.xview)

Here, 'h' represents the Scrollbar object which is created as a child to 'root' window. The option 'orient' indicates HORIZONTAL for horizontal scroll bars and VERTICAL indicates vertical scroll bars. 'bg' represents back ground color for the scroll bar. This option may not work in windows since window operating system may force some default back ground color for the scroll bar which will not be disturbed by the *tkinter*. The option 'command' represents the method that is to be executed. The method 'xview' is executed on the object 't'. Here, 't' may represents a widget like Text widget or Listbox.

Program : A python program to create a horizontal scroll bar and attach it to a Text widget to view the text from left to right.

```
from tkinter import *

class MyScrollbar:
    # constructor
    def __init__(self, root):
        # create a Text widget with 70 chars width and 15 lines height
        self.t = Text(root, width=70, height=15, wrap=NONE)

        # insert some text into the text widget
        for i in range(50):
            self.t.insert(END, "This is some text")

        # attach text widget to root window at the top
        self.t.pack(side=TOP, fill=X)

        # create a horizontal scroll bar and attach it to text widget
        self.h = Scrollbar(root, orient=HORIZONTAL, command=self.t.xview)

        # attach text widget to the horizontal scroll bar
        self.t.configure(xscrollcommand=self.h.set)

        # attach Scrollbar to root window at the bottom
        self.h.pack(side=BOTTOM, fill=X)

# create root window
root = Tk()

# create an object ro MyScrollbar class
ms = MyScrollbar(root)

# the root window handles the mouse click event
root.mainloop()
```

Checkbutton Widget :

Check buttons, also known as check boxes are useful for the user to select one or more options from available group of options. Check buttons are displayed in the form of square shaped boxes. When a check button is selected, a tick mark is displayed on the button. We can create check buttons using the Checkbutton class as:

c1 = Checkbutton(f, bg='yellow', fg='green', font=('Georgia', 20, 'underline'), text='java', variable=var1, command=display)

Here, 'c1' is the object of Checkbutton class that represents a check button. 'f' indicates frame for which the check button becomes a child. 'bg' and 'fg' represents the back ground and fore ground color used for check button. 'font' represents the font name, size and style. 'text' represents the text to be displayed after the check button. The option 'variable' represents an object of IntVar() class. 'command' represents the method to be called when the user clicks the check button.

The class 'IntVar' is useful to know the state of the check button, whether it is clicked or not. The IntVar class object can be created as:

var1 = IntVar()

When the check button is clicked or selected, the value of 'var1' will be 1, otherwise its value will be 0. To retrieve the value from 'var1', we should use the get() method as:

x = var1.get() # x value can be 1 or 0

Program : A python program to create 3 check buttons and know which option are selected by the user.

```
from tkinter import *

class Mycheck:
    # constructor
    def __init__(self, root):
        # create a frame as child to root window
        self.f = Frame(root, height=350, width=500)

        #let the frame will not shrink
        self.f.propagate()

        # attach the frame to root window
```

```
        self.f.pack()

        # create IntVar class variables
        self.var1 = IntVar()
        self.var2 = IntVar()
        self.var3 = IntVar()

        # create check boxes and bind them to display method
        self.c1 = Checkbutton(self.f, bg='yellow', fg='green', font=('Georgia', 20,
'underline'), text='Java', variable=self.var1, command=self.display)
        self.c2 = Checkbutton(self.f, text='Python', variable=self.var2,
command=self.display)
        self.c3 = Checkbutton(self.f, text='.NET', variable=self.var3,
command=self.display)

        # attach check boxes to the frame
        self.c1.place(x=50, y=100)
        self.c2.place(x=200, y=100)
        self.c3.place(x=350, y=100)

    def display(self):
        # retrieve the control variable values
        x = self.var1.get()
        y = self.var2.get()
        z = self.var3.get()

        # string is empty initially
        str = ''

        # catch user choice
        if x == 1:
            str += 'Java'
        if y == 1:
            str += 'Python'
        if z == 1:
            str += '.NET'

        # display the user selection as a lable
        lbl = Label(text=str, fg='blue').place(x=50, y=150, width=200, height=20)

# create root  window
root = Tk()

# create an object to Mybutton class
mb = Mycheck(root)

# the root window handles the mouse click event
root.mainloop()
```

Radiobutton Widget :

A radio button is similar to a check button, but it is useful to select only one option from a group of available options. A radi button is displayed in the form of round shaped button. The user cannot select more than one option in case of radio buttons. When a radio button is selected, there appears a dot in the radio button. We can create a radio button as an object of the Radiobutton class as:

r1 = Radiobutton(f, bg=’yellow’, fg=’green’, font=(‘Georgia’, 20, ‘underline’), text=’Male’, variable=var, value=1, command=display)

The option ‘text’ represents the string to be displayed after the radio button. ‘variable’ represents the object of IntVar class. ‘value’ represents a value that is set to this object when the radio button is clicked. The object of IntVar class can be created as:

var = IntVar()

When the user clicks the radio button, the value of this ‘var’ is set to the value given in ‘value’ option, i.e. 1. It means ‘var’ will become 1 if the radio button ‘r1’ is clicked by the user. In this way, it is possible to know which button is clicked by the user.

Program : A python program to create radio buttons and know which button is selected by the user.

```
from tkinter import *

class Myradio:
    # constructor
    def __init__(self, root):
        # create a frame as child to root window
        self.f = Frame(root, height=350, width=500)

        #let the frame will not shrink
        self.f.propagate()

        # attach the frame to root window
        self.f.pack()

        # create IntVar class variables
        self.var = IntVar()


        # create check boxes and bind them to display method
        self.r1 = Radiobutton(self.f, bg='yellow', fg='green', font=('Georgia', 20,
'underline'), text='Male', variable=self.var, value=1, command=self.display)
        self.r2 = Radiobutton(self.f, text='Female', variable=self.var, value=2,
command=self.display)


        # attach check boxes to the frame
        self.r1.place(x=50, y=100)
        self.r2.place(x=200, y=100)


    def display(self):
        # retrieve the control variable values
        x = self.var.get()
```

```
        # string is empty initially
        str = ''

        # catch user choice
        if x == 1:
            str += 'You selected: Male'
        if x == 2:
            str += 'You selected: Female'


        # display the user selection as a lable
        lbl = Label(text=str, fg='blue').place(x=50, y=150, width=200, height=20)

# create root  window
root = Tk()

# create an object to Mybutton class
mb = Myradio(root)

# the root window handles the mouse click event
root.mainloop()
```

Entry Widget :

Entry widget is useful to create a rectangular box that can be used to enter or display one line of text. For example, we can display names, passwords or credit card numbers using Entry widgets. An Entry widget can be created as an object of Entry class as:

e1 = Entry(f, width=25, fg='blue', bg='yellow', font=('Arial', 14), show='*')

Here, 'e1' is the Entry class object. 'f' indicates the frame which is the parent component for the Entry widget. 'width' represents the size of the widget in number of characters. 'fg' indicates the fore ground color in which the text in the widget is displayed. 'bg' represents the back ground color in the widget. 'font' a tuple that contains font family name, size and style. 'show' represents a character that replace the originally typed characters in the Entry widget. For example, show='*' is useful when the user wants to hide his password by displaying stars in the place of characters.

After typing text in the Entry widget, the user presses the Enter button. Such an event should be linked with the Entry widget using bind() method as:

e1.bind("<Return>", self.display)

When the user presses Enter (or Return) button, the event is passed to display() method. Hence, we are supposed to catch the event in the display method, using the following statement:

```
def display(self, event):
```

As seen in the preceding code, we are catching the event through an argument 'event' in the display() method. This argument is never used inside the method. The method consist of the code that is to be executed when the user pressed Enter button.

Program : A python program to create Entry widget for entering user name and password any display the entered text.

```
from tkinter import *

class MyEntry:
    # constructor
    def __init__(self, root):
        # create a frame as child to root window
        self.f = Frame(root, height=350, width=500)

        # let the frame will not shrink
        self.f.propagate(0)

        # attach the frame to root window
        self.f.pack()

        # labels
        self.l1 = Label(text='Enter Username : ')
        self.l2 = Label(text='Enter Password : ')

        # create Entry widget for username
        self.e1 = Entry(self.f, width=25, fg='blue', bg='yellow', font=('Arial', 14))

        # create Entry widget for pass word. the text in the widget is replaced by
stars (*)
        self.e2 = Entry(self.f, width=25, fg='blue', bg='yellow', show='*')

        # when user presses Enter, bind that event to display method
        self.e2.bind("<Return>", self.display)

        # place labels and entry widgets in the frame
        self.l1.place(x=50, y=100)
        self.e1.place(x=200, y=100)
        self.l2.place(x=50, y=150)
        self.e2.place(x=200, y=150)

    def display(self, event):
        # retrieve the values from the entry widget
        str1 = self.e1.get()
        str2 = self.e2.get()
```

```
        # display the values using labels
        lbl1 = Label(text='Your name is  : '+str1).place(x=50, y=200)
        lbl2 = Label(text='Your Password is : '+str2).place(x=50, y=220)

# create root window
root = Tk()

# create an object to MyButtons class
mb = MyEntry(root)

# the root window handles the mouse click event
root.mainloop()
```

Spinbox Widget :

A Spin box widget allows the user to select values from a given set of value. The values may be a range of numbers or a fixed set of strings.

The spin box appears as a long rectangle attached with arrowheads pointing towards up and down. The user can click on the arrowhead to see the next value or previous value. The user can also edit the value being displayed in the spin box just like he can do in case of Entry widget.

Program : A python program to create two spin boxes and retrieve the values displayed in the spin boxes when the user clicks on a push button.

```
from tkinter import *

class MySpinbox:
    # constructor
    def __init__(self, root):
        # create a frame as child to root window
        self.f = Frame(root, height=350, width=500)

        # let the frame will not shrink
        self.f.propagate(0)

        # attach the frame to root window
        self.f.pack()

        # these ae control variables for Spinboxes
        self.val1 = IntVar()
        self.val2 = StringVar()

        # create spinbox with numbers from 5 to 15
        self.s1 = Spinbox(self.f, from_=5, to=15, textvariable=self.val1, width=15,
fg='blue', bg='yellow', font=('Arial', 14, 'bold'))

        # create spinbox with a tuple of strings
        self.s2 = Spinbox(self.f, values=('Hyderabad', 'Delhi', 'Kolkata',
'Bangalore'), textvariable=self.val2, width=15, fg='black', bg='LightGreen',
font=('Arial', 14, 'bold italic'))

        # create a button and bind it with displa() method
```

```
        self.b = Button(self.f, text='Get values from spinboxes', command=self.display)

        # place spinboxes and button widgets in the frame
        self.s1.place(x=50, y=50)
        self.s2.place(x=50, y=100)
        self.b.place(x=50, y=150)

    def display(self):
        # retrieve the values form the spinbox widgets
        a = self.val1.get()
        s = self.val2.get()

        # display the values using labels
        lbl1 = Label(text='Selected Value is : ' + str(a)).place(x=50, y=200)
        lbl2 = Label(text='Selected City is :' +s).place(x=50, y=220)

# create root window
root = Tk()

# create an object to MySpinbox class
mb = MySpinbox(root)

# the root window handles the mouse click event
root.mainloop()
```

Listbox Widget :

A list box is useful to display a list of items in a box so that the user can select 1 or more items. To create a list box, we have to create an object of Listbox class, as:

lb = Listbox(f, font="Arial 12 bold", fg='blue', bg='yellow', height=8, width=24, activestyle='underline', selectmode=MULTIPLE)

Here, 'lb' is the list box object. The option 'height' represents the numbers of lines shown in the list box. 'width' represents the width of the list box in terms of number of characters and the default is 20 characters. The option 'activestyle' indicates the appearance of the selected item. It may be 'underline', 'dotbox', or 'none'. The default value is 'underline'. The option 'selectemode' may take any of the following values:

- **BROWSE :** Normally, we can select one item (or line) out of a list box. If we click on an item and then drag to a different item, the selection will follow the mouse. This is the default value of 'selectmode' option.

- **SINGLE :** This represents that we can select only one item (or line) from all available list of items.

- **MULTIPLE :** We can select 1 or more number of items at once by clicking on the items. If an item is already selected, clicking second time on the item will un-select it.

- **EXTENDED :** We can select any adjacent group of items at once by clicking on the first item and dragging to the last item.

Program : A python program to create a list box with universities names and display the selected Universities names in a text box.

```
from tkinter import *

class ListBoxDemo:
    # constructor
    def __init__(self, root):
        self.f = Frame(root, width=700, height=400)
        # let the frame will not shrink
        self.f.propagate(0)

        # attach the frame to root window
        self.f.pack()

        # create a label
        self.lbl = Label(self.f, text="Click one or more of the universities below :",
font=('Calibri', 14))
        self.lbl.place(x=50, y=50)

        # create list box with universities names
        self.lb = Listbox(self.f, font='Arial 12 bold', fg='blue', bg='yellow',
height=8, selectmode=MULTIPLE)
        self.lb.place(x=50, y=100)

        # using for loop insert items into list box
        for i in ["Standford Universities", "Oxford Universities", "Texas
A&MUniversities", "Cambridge Universities", "Universities of California"]:
            self.lb.insert(END, i)

        # bind the ListboxSelect event to on_selected() items
        self.lb.bind('<<ListboxSelect>>', self.on_select)

        # create text box to display selected items
        self.t = Text(self.f, width=40, height=6, wrap=WORD)
        self.t.place(x=300, y=100)

    def on_select(self, event):
        # create an empty list box
        self.lst = []

        # know the indexes of the selected items
        indexes = self.lb.curselection()

        # retrieve the items names depending on indexes append the items names to the
list box
        for i in indexes:
            self.lst.append(self.lb.get(i))
```

```
        # delete the previous content of the text box
        self.t.delete(0.0, END)

        # insert the new contents contents into the text box
        self.t.insert(0.0, self.lst)

# create root window
root = Tk()

# title for the root window
root.title("List box demonstration")

# create object to our class
obj = ListBoxDemo(root)

#  handle any events
root.mainloop()
```

Menu Widget :

A menu represents a group of items or options for the user to select form. For example, when we click on 'File' menu, it may display options like 'New', 'Open', 'Save', etc. We can select any option depending on our requirements. 'New', 'Open', 'Save' – these options are called menu items. Thus, a menu is composed of several menu items. Similarly, 'Edit' is a menu with menu items like 'Cut', 'Paste', etc. Generally, we see menus displayed in a bar, called menu bar.

Program : A python program to create a menu bar and adding File and Edit menus with some menu items.

```
from tkinter import *

class MyMenuDemo:
    def __init__(self, root):
        # create a member
        self.menubar = Menu(root)

        # attach the menubar to the root window
        root.config(menu=self.menubar)

        # create file menu
        self.filemenu = Menu(root, tearoff=0)

        # create menu items in file menu
        self.filemenu.add_command(label="New", command=self.donothing)
        self.filemenu.add_command(label="Open", command=self.donothing)
        self.filemenu.add_command(label="Save", command=self.donothing)

        # add a horizontal line as separator
        self.filemenu.add_separator()

        # create another menu item below the separator
```

```
        self.filemenu.add_command(label="Exit", command=root.destroy)

        # add the file menu with a name 'File' to the menubar
        self.menubar.add_cascade(label="File", menu=self.filemenu)

        # create edit menu
        self.editmenu = Menu(root, tearoff=0)

        # create menu items in edit menu
        self.editmenu.add_command(label="Cut", command=self.donothing)
        self.editmenu.add_command(label="Copy", command=self.donothing)
        self.editmenu.add_command(label="Paste", command=self.donothing)

        # add the edit menu with a name 'Edit' to the menubar
        self.menubar.add_cascade(label="Edit", menu=self.editmenu)

    def donothing(self):
        pass

# create root window
root = Tk()

# title for the root window
root.title("A menu example.")

# create object to our class
obj = MyMenuDemo(root)

# define the size of the root window
root.geometry('600x350')

# handle any events
root.mainloop()
```

***