Annamalai University

Department of Computer Science and Engineering

B.E(CSE) – VI SEMESTER – 'A' BATCH

1608PC602 – PYTHON PROGRAMMING

LECTURE NOTES

(UNITS IV & V)

Course Teacher:

Dr. A. GEETHA,

Professor.

PYTHON PROGRAMMING

Unit- IV

Files and Exception Handling

Files - used to store data permanently.

Exception Handling - make programs reliable and robust.

Text Input and Output :

A file is placed in a directory in a file system.

Absolute filename refers to directory path to file

eg C:\users\documents\python\program1.txt

Relative filename is relative to the current working directory

eg. program1.py

Files can be classified into
Text files
Binary files.

Text file : a file that can be read, created or modified

eg, python source programs stored in text files.

Binary file : a file that cannot be modified and that is stored in binary files

eg, Microsoft word files that are processed by Microsoft word program

Text file contains a sequence of characters
Binary file contains a sequence of bits.

Scanned with CamScanner

The syntax for all the file operations are given below.

<u>Opening a file:</u>

filevariable = open(filename, mode)

where open function returns a file object for filename.

Mode parameter specifies the file usage

eg., input = open("Program1.txt", "r")

The different file modes are:

"r" -> opens a file for reading
"w" -> opens a new file for writing.
"a" -> opens a file for appending data at end
"rb" -> opens a file for reading binary data
"wb" -> opens a file for writing binary data

<u>Writing data to a file:</u>

```
eg., outfile = open("Program1.txt", "w")
     outfile.write("Welcome\n")
     outfile.close()
```

A file object is created when a file is opened. This object is invoked with write method for writing into a file and close method for closing the file.

When the file is opened for writing, a file pointer is positioned at the beginning of file and the pointer moves forward when the file is read or written into.

<u>Reading Data from a file:</u>

read() -> to read a specified number of characters or all characters from the file and return as string.

Scanned with CamScanner

readline() -> to read the ~~read~~ line that ends with \n.
readlines() -> to read all lines into a list of strings.

Syntax:

```
fileobject.read([count]);
```

Example:

```
fo = open("program1.txt", "r")
str = fo.read(10);
print(str)
fo.close()
```

Output
First ten characters in the file Program1.txt

Appending Data.

- 'a' mode is used to append data to end of file.

Example

```
outfile = open("Program1.txt", "a")
outfile.write("CSE\n")
outfile.close()
```

Output:
CSE will be appended at the end of "Program1.txt" file.

Writing and Reading Numeric Data:

Example:

```
from random import randint
def main()
    outfile = open('e:\sample.txt', "w")
```

Scanned with CamScanner

```
    outfile.write(str(randint(0,7)))
    outfile.close()
    infile = open("e:\sample.doc", "r")
    s = infile.read()
    print(s)
    infile.close()
main()
```

File Dialogs

tkinter.filedialog module contains the function askopenfilename and asksaveasfilename for displaying the file Open and SaveAs dialog boxes

Example:

Syntax to display file dialog box to open a file:
filename := askopenfilename()
Syntax to display file dialog box for saving
filename = asksaveasfilename()

Coding.

```
from tkinter.filedialog import askopenfilename
from tkinter.filedialog import asksaveasfilename
filenameforReading = askopenfilename()
print("filenameforReading")
filenameforWriting = asksaveasfilename()
print("filenameforWriting")
```

Scanned with CamScanner

Case Study:

Counting Each Letter in a File

```
def main():
        filename = input("Enter a filename")
        infile = open(filename, "r")
        counts = 26 * [0]
        for line in infile:
            countletters(line.lower(), counts)
        for i in range(len(counts)):
            if counts[i] != 0:
                print(chr(ord('a')+i) + "appears"
                + str(counts[i]) +("time" if
                counts[i]== 1 else "times"))

        infile.close()

def countletters(line, counts):
        for ch in line:
            if ch.isalpha():
                counts[ord(ch) -
                        ord('a')] += 1

main()
```

Output:

```
Enter a filename : input.txt
a appears 15 times
   :
   :
x appears 8 times
```

Scanned with CamScanner

Retrieving ~~pri~~ Data from ~~q~~ the Web

To ~~a~~ read data from ~~pri~~ website

Syntax to open a URL

infile = urllib.request.urlopen("http://www.yahoo.com")

where urlopen function is defined in urllib.request

import urllib.request

Example: To count each letter in a file opened in website.

Coding:

```
import urllib.request
def main():
    url = input("Enter filename:")
    infile = urllib.request.urlopen(url)
    s = infile.read().decode()
    counts = countLetters(s.lower())
    for i in range(len(counts)):
        if counts[i] != 0:
            print(chr(ord('a') + i) +
            " appears" + str(counts[i]) +
            (" time" if counts[i] == 1 else " times"))
def CountLetters(s):
    Counts = 26 * [0]
    for ch in s:
        if ch.isalpha():
            counts[ord(ch) - ord('a')] += 1
    return counts

main()
```

Output:

```
Enter filename: ~~url~~ http://cs.armstrong.edu/liang/data/Lincoln.txt
a appears 102 times
x appears 18 times
```

Scanned with CamScanner

Exception Handling

Exception handling → enables a program to deal with exceptions and continue its normal execution.

* If the user enters a file or a URL that does not exist, program will be aborted and raise an error.
* If nonexistent filename is opened, an I/O error message is reported
* The lengthy error message is called traceback
* An error that occurs at runtime is also called an exception.
* This exception can be handled as follows.

```
try:
      <body>
  except <ExceptionType>:
      <handler>
```

- where <body> contains code that raises an exception.
- Wh- When an exception occurs, rest of code in <body> is skipped and the corresponding handler is executed.
- <handler> is the code that processes the exception.

Example

```
def main():
      while true:
        try
            filename = input("Enter filename").strip()
```

```
    infile = open(filename,"r")
    break
  except IOError:
        print('File does not exist')
main()
```

Output?

```
Enter filename : Sample.doc
File does not exist. Try again
```

The try / except block works as follows:

* At first, statements between try and except are executed.
* If no exception, except clause is skipped, and break statement is executed to exit while loop.
* If an exception occurs open function raises an exception, break statement is skipped.
* When an exception occurs, and if exception matches exception name, then except clause is executed.
* If exception name is not matched, exception is passed onto caller function.
(∴ If no handler is found, it is unhandled exception.

A try statement can have more than one except clause to handle different exceptions.

The syntax is given below:

```
try:
  <body>
except <Exceptiontype1>:
  <handler1>
      :
      :
except <ExceptiontypeN>:
    <handlerN>
except:
    <handlerExcept>
else:
    <process_else>
finally:
    <process_finally>
```

<u>Example:</u>

```
def main():
    try:
        number1, number2 = eval(input("Enter two
            numbers, seperated by comma"))
        result = number1 / number2
        print(result)
    except ZeroDivisionError:
        print("Division by zero")
    except SyntaxError:
        print("Comma may be missing")
    except:
        print("Input may be wrong")
    else:
        print("No exception")
    finally:
        print("Finally clause is executed")

main()
```

Scanned with CamScanner

Output:

```
Enter two numbers, seperated by comma: 4,5
0.8
No exceptions
Finally clause is executed.

Enter two numbers, seperated by comma: a, b
Input may be wrong
Finally clause is executed.
```

Raising Exceptions

* Exceptions are wrapped in objects which are created by classes.
* An exception is raised from a function.
* When a function detects an error, it creates an object from exception class and throws exception to the caller of the function with the syntax,

```
raise ExceptionClass("Wrong")
```

The program creates an instance of RuntimeError and raise exception as follows

```
ex = RuntimeError("Wrong")
raise ex
        or
raise RuntimeError("Wrong")
```

Example: Testprogram

```
from GeometricObject import GeometricObject
import math.
class Circle(GeometricObject)
```

Scanned with CamScanner

```
    def setRadius(self, radius):
        if radius < 0:
            raise RuntimeError("wrong radius")
        else:
            self.__radius = radius
    def getArea(self):
        return self.__radius * self.__radius * math.pi
```

Program : TestException.py

```
from Testprogram import Circle

try:
    c1 = Circle(5)
    print(c1, c1.getArea())
    :
    :
except RuntimeException:
    print("wrong radius")
```

<u>Processing Exceptions using Exception Objects</u>

- An exception object can be accessed in except clause
- The syntax to assign the exception object to a variable is:

```
try:
    <body>
except ExceptionType as ex:
    <handler>
```

<u>Example:</u>

```
try
    number = eval(input("Enter number"))
    print(number)
except NameError as ex:
    print("Exception", ex)
```

Scanned with CamScanner

Output:

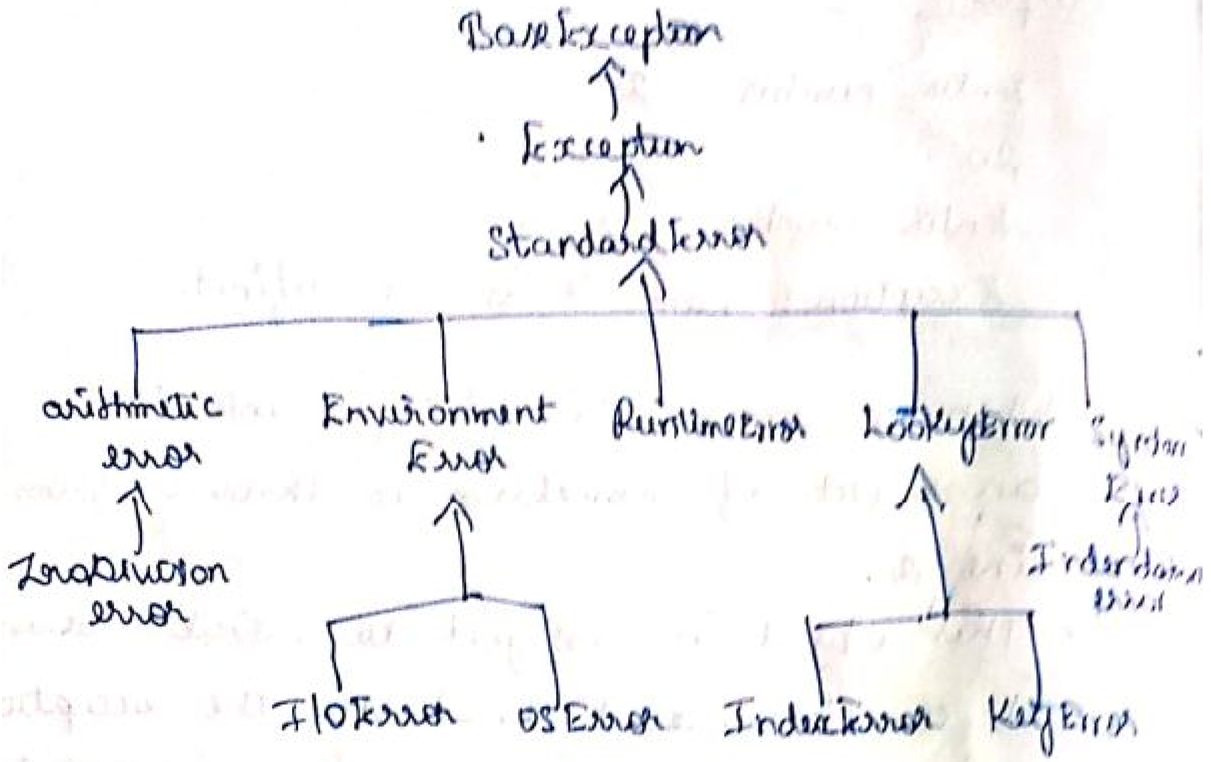
```
Enter number : 20
20
Enter number : s
Exception : name 's' is not defined.
```

* When a non-numeric value is entered, an object of NameError is thrown from line 2.
* This object is assigned to variable ex and it can be accessed to handle the exception.
* The _str_() method in ex is invoked to return a string that describes the exception.
* In the above example, 's' is not defined.

Defining Custom Exception classes:

* A custom exception class can be defined by extending BaseException or a subclass of BaseException.
* BaseException class is the root of exception classes.
* All Python exception classes inherit directly or indirectly from BaseException.
* Our own exception classes can be derived from BaseException or subclass of BaseException
* Exceptions raised are instances of classes as shown below:

Scanned with CamScanner



<u>Example:</u>

<u>Program</u> : <u>InvalidRadiusException.py</u>

```
class InvalidRadiusException (RuntimeError):
    def __init (self, radius):
        super().__init__()
        self.radius = radius
```

<u>Program : CircleWithCustomException.py</u>

```
from GeometricObject import GeometricObject
from InvalidRadiusException import
                            InvalidRadiusException
        :
        :
    def setradius (self, radius):
        if radius >= 0:
            self.radius = radius
        else:
            raise InvalidRadiusException (radius)
```

Scanned with CamScanner

Program 3 : TestCircle.py

```
try :
      c2 = Circle (-5)
      print (c2.getArea())
except InvalidRadiusException as ex :
      print ("invalid radius")
except Exception:
      print (" wrong")
```

Output:

```
    invalid radius
```

Binary I/O using Pickling

* Open a file using the mode rb or wb for reading binary or writing binary.
* Invoke pickle module's dump and load functions to write and read data.
* Binary IO in python is performed using dump and load functions in pickle module.
* pickle module implements serializing and deserializing objects.
* Serializing is to convert object into a stream of bytes.
* Deserializing is to extract an object from a stream of bytes.
* Serializing / Deserializing = pickling / unpickling = dumping / loading of objects.

Scanned with CamScanner

Dumping and Loading of Objects

- In python, all data are objects.
- Pickle module enables reading and writing of any data using dump and load functions

Example:

```
import pickle
def main():
        outfile = open("pickle.dat", "wb")
        pickle.dump(40, outfile)
        pickle.dump("Halo", outfile)
        outfile.close()
        infile = open("pickle.dat", "rb")
        print(pickle.load(infile))
        print(pickle.load(infile))
        infile.close()
main()
```

Output:

```
    40
    Halo
```

- In the above program, dump(object) serializes the object into a stream of bytes and stores in a file.
- load(object) reads a stream of bytes and deserializes them into an object.

Detection of end of file:

- If number of objects are unknown, then objects can be read repeatedly using load function until it throws an EOFError.

Scanned with CamScanner

Example:

```
import pickle
def main():
        outfile = open("pickle.dat", "wb")
        data = input("Enter numbers")
        while data != 0 :
                pickle.dump(data, outfile)
                        :
                        :
        outfile.close()
        infile = open("pickle.dat", "rb")
        end-of-file = False
        while not end-of-file :
        try:
            print(pickle.load(infile), end=" ")
        except EOFError :
            end-of-file = True
        infile.close()
        print("All objects are read")
main()
```

Output:

```
Enter numbers 6
Enter numbers 2
Enter numbers 0
6 2
All objects are read
```

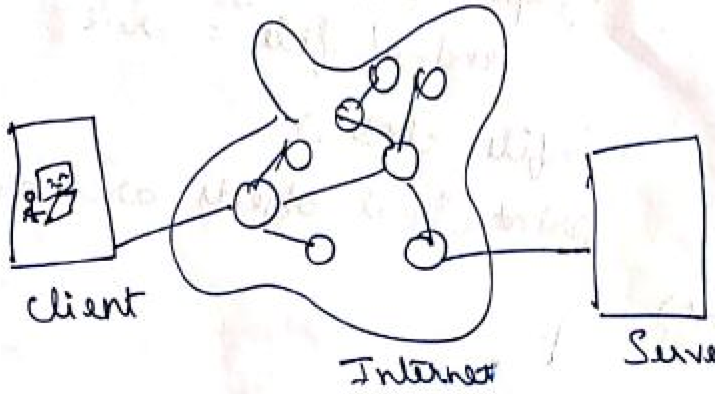
* The above program repeatedly reads an object using load function in a while loop until EOFError occurs.

Unit - IV

Network Programming

Client / Server Architecture:

- The server is a piece of hardware or softwar that provides a service which is needed by one or more users of the service called clients
- The server wait for client requests, responds to those clients and wait for more requests.
- The client contacts a server for a particular request, send necessary data, wait for server to reply, either completing request or indicating the cause of failure.
- The client / server architecture is given as



- Client / server architecture can be applied to computer hardware and also software.

Hardware Client / Server Architecture

- Example for hardware servers are Printers.
- Printers servers, process incoming print jobs and them to a printer or other printing device attached.

Scanned with CamScanner

* Another example is file server.
* Sun Microsystems' Network File System (NFS) supports file servers.

Software Client/Server Architecture:

* Example for software server is web server.
* The web server accepts client requests, send back web pages to clients by browsing and wait for next client request.
* Database servers and windows servers are other examples.

Client/Server Network Programming

* A communication endpoint is created that allows a server to listen for requests.
* Once a communication endpoint is setup, the listening server ends an infinite loop waiting for client to connect and respond to requests.
* The client also has to create a single communication endpoint and establishes a connection to the server.
* If the request has been processed and if the client has received the result, communication is terminated.

Sockets: Communication Endpoints

* There are various types of sockets that allow processes running on different or same computers to communicate with each other.

Scanned with CamScanner

* Sockets are computer networking data structures that embody the concept of Communication endpoint.
* Networked applications must create sockets before any type of communication Commences.
* There are two types of sockets

-> File Oriented: Both processes run on same computer supported by the file system
-> Unix sockets such as AF_UNIX are file-based sockets
-> Network Oriented : AF_INET6 is used for IPv6 addressing
-> AF_NETLINK sockets are used for IPC between user and kernel-level code
-> AF_TIPC allows clusters of computers to communicate without IP-based addressing.

* A socket address is comprised of a hostname and port number.
* Valid portnumbers range from 0-65535.
* A list of port numbers is obtained from
  http://www.iana.org/assignments/port-numbers

Connection-oriented vs Connectionless Sockets

* Connection-oriented Sockets
  -> a connection must be established before communication can occur.
  -> called virtual circuit or stream socket.
  -> offers sequenced, reliable and unduplicated delivery of data

Scanned with CamScanner

* Each message is broken up into multiple pieces and all are guaranteed to arrive at destination.
* All pieces are then put back together and delivered to the waiting application
* Transmission Control Protocol (TCP) implements connection oriented sockets
* SOCK_STREAM is the socket type used to create TCP sockets.
* Internet Protocol (IP) is used to find hosts in network.
* Thus the system is named as TCP/IP.

Connectionless Sockets

* No connection is necessary before any communication.
* Sequencing, reliability and non-duplication in process of data delivery are not guaranteed.
* Entire messages are sent and not broken into pieces.
* User Datagram Protocol (UDP) implements such connection types
* These sockets use IP to find hosts
* Hence the system is named as UDP/IP.
* SOCK_DGRAM is the socket type used.

Socket Object Methods

* A list of socket object methods is given below. TCP and UDP clients and servers are created using these methods.

Scanned with CamScanner

Socket Methods	Use.
s.bind() [server]	bind address to socket
s.listen() "	setup and start TCP listener
s.accept() "	Passively accept TCP
s.connect() [client]	Actively initiate TCP server connection.
s.recv() [general]	recive TCP message
s.send() "	send / Transmit TCP msg
s.getsockname()	address of current socket
s.shutdown()	shut down the connection
s.close()	close socket
s.makefile()	create a file object associated with socket.

Creating a TCP Server and TCP client

• A TCP server is created using:

```
ss = socket()              create server socket
ss.bind()                  bind socket to address
ss.listen()                listen for connections
inf_ loop:
    cs = ss.accept()       accept client connection
    comm_ loop:
        cs.recv()/cs.send()    dialog
    cs.close()                 close client socket
ss.close()                     close server socket
```

• A TCP client is created using:

```
cs = socket()                        creat client socket
cs.connect()                         attempt connection
comm_ loop:
    cs.send() / cs.recv()            dialog
cs.close()                           close client socket.
```

Scanned with CamScanner

Example : To accept messages from clients and returns with timestamp prefix.

Server Program:

```
from socket import *
from time import ctime
HOST = ''
PORT = 21567
BUFSIZ = 1024
ADDR = (HOST, PORT)
tcpSerSock = socket(AF_INET, SOCK_STREAM)
tcpSerSock. bind (ADDR)
tcpSerSock. listen(5)
while True :
    print (" waiting for connection')
    while True :
        data = tcpCliSock. recv(BUFSIZ)
        if not data :
            break
        tcpCliSock. send('[%s] %s' %
            (bytes(ctime (), 'utf-8'), data))
    tcpCliSock . close()
tcpSerSock. close()
```

Client Program:

```
from socket import *
HOST = '127.0.0.1'
PORT = 21567
BUFSIZ = 1024
ADDR = (HOST, PORT)
tcpCliSock = socket( AF_INET, SOCK_STREAM)
tcpCliSock. connect(ADDR)
while True :
    data = input('> ')
```

Scanned with CamScanner

```
    if not data:
        break
    tcpCliSock.send(data)
    data = tcpCliSock.recv(BUFSIZ)
    if not data:
        break
    print(data.decode('utf-8'))
tcpCliSock.close()
```

Execution of TCP client and Servers

* Start the server before any clients try to connect

Creation of UDP Server and UDP client

* A UDP server is created using:

```
ss = socket()
ss.bind()
inf_loop:
    cs = ss.recvfrom() / ss.sendto()
ss.close()
```

* A UDP client is created using

```
cs = socket()
comm_loop:
    cs.sendto() | cs.recvfrom()
cs.close()
```

Server Program :

```
from socket import *
from time import ctime
HOST = ' '
PORT = 21567
BUFSIZ = 1024
ADDR = (HOST, PORT)
```

Scanned with CamScanner

```
    udpSerSock = socket(AF_INET, SOCK_DGRAM)
    udpSerSock.bind(ADDR)
    while True:
        print (" waiting for message")
udpSerSock.close()
```

Client Program

```
from socket import *
HOST = 'localHost'
PORT = 21567
BUFSIZ = 1024
ADDR = (HOST, PORT)
udpCliSock = socket(AF_INET, SOCK_DGRAM)
while True:
        data = raw_input('>')
        if not data
            break
        udpclisock.sendto(data, ADDR)
        data, ADDR = udpCliSock.recvfrom(BUFSIZ)
        if not data:
            break
        print data
udpCliSock.close()
```

Twisted Framework:

* Twisted is an event-driven networking framework to develop asynchronous networked applications and protocols.
* It supports for network protocols, threading, security, authentication, chat, RDBMS database Integration, web/internet, e-mail, command-line arguments etc.

Scanned with CamScanner

Twisted TCP Server:

```
from twisted.internet import protocol, reactor
from time import ctime
PORT = 21567
class TSServProtocol(protocol.Protocol):
    def connectionMade(self):
        clnt = self.clnt = self.transport.
                getPeer().host
    def dataReceived(self, data):
        self.transport.write('%s', data)
factory = protocol.Factory()
factory.protocol = TSServProtocol
reactor.listenTCP(PORT, factory)
reactor.run()
```

Twisted TCP client:

```
from twisted.internet import protocol, reactor
HOST = 'localhost'
PORT = 21567
class TSClntProtocol(protocol.Protocol):
    def sendData(self):
        data = raw_input('> ')
        if data:
            self.transport.write(data)
        else:
            self.transport.loseConnection()
    def connectionMade(self):
        self.sendData()
    def dataReceived(self, data):
        print(data)
        self.sendData()
class TSClntFactory(protocol.ClientFactory):
    protocol = TSClntProtocol
```

Scanned with CamScanner

```
clientConnectionLost = clientConnectionFailed =
    \lambda self, connector, reason:
                         reactor.stop()

 reactor.connectTCP(HOST, PORT,
                      TSClntFactory())
 reactor.run()
```

Internet Client Programming

Internet Clients:

- Internet servers make the Internet possible.
- All machines on internet are either clients or servers.
- The machines providing services are servers
- The machins that are connected to those services are clients.

Transferring Files

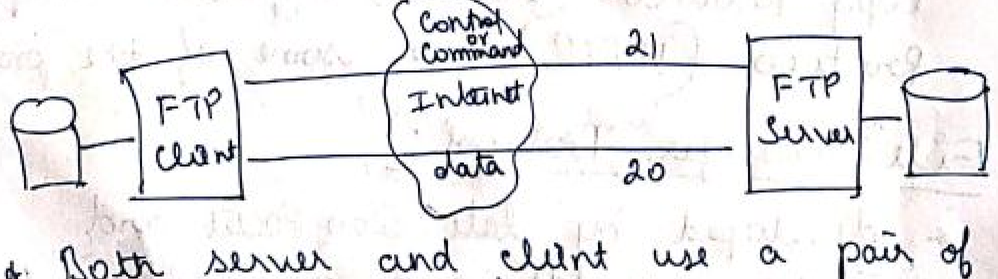
- File exchange is an internet activity.
- Many protocols exist to transfer files on internet.
- File Transfer Protocol (FTP), Unix-to-Unix Copy Protocol (UUCP), Hypertext Transfer Protocol (HTTP) are some of the protocols.

File Transfer Protocol:

- developed by late Jon Postel and Joyce Reynolds and published in 1985.
- It is used to download publicly accessible files in an anonymous fashion.

Scanned with CamScanner

- It is used to transfer files between two computers.
- A login/password is needed to access the remote host running FTP server.
- <u>Login</u> of an unregistered user is anonymous and <u>password</u> is e-mail address of client.
- FTP works as follows:
  1) client contacts FTP server on remote host
  2) client logins in with username & password
  3) client performs file transfers or information requests.
  4) client completes the transaction by logging out of remote host and FTP server.
- Sometimes, entire transaction is terminated, before it is completed.
- FTP will timeout after 15 minutes of inactivity.
- The communication between server and client using FTP is shown below:



- Both server and client use a pair of sockets for communication.
- Control port and data port

Scanned with CamScanner

* There are two types of FTP modes such as Active and Passive.
* Server's data port 20 is Active and server initiate connection to client's data port.
* For passive mode, client must initiate data connection.

Python and FTP:

* FTP client can be written using Python.
* When Python's FTP support is needed, ftplib module is imported and ftplib.FTP class is instantiated.
* Some of the ftplib.FTP class Methods are listed below:

*Method	Use
pwd()	current working directory
cwd(path)	change current working directory to path
retrlines(cmd[, cb])	Download text file
storlines (cmd, f)	upload text file
rename (old, new)	rename remote file from old to new
delete (path)	Delete remote file
mkd (directory)	Create remote directory
rmd (directory)	Remove remote directory
quit()	close connection and quit.

Scanned with CamScanner

Example: To download files from FTP Server

```
from ftplib import FTP
from datetime import datetime
start = datetime.now()
ftp = FTP('your-ftp-domain')
ftp.login ('your username', 'password')
files = ftp.nlst()
for file in files
        print (" Downloading")
        ftp.retrbinary (" RETR"+ file,
          . open (" download/to/your/directory)" +
                   file, 'wb').write)

ftp.close()
end = datetime.now()
diff = end - start
print (" All files downloaded")
```

Network News:

* Usenet → Usenet News System is a global archival bulletin board.
* Newsgroups → they are for any topic including poem, software, cooking, music etc.
* The entire system is a large global n/w of computers that participate in sharing Usenet postings.
* Once a user uploads a message to his local Usenet computer, it will be propagated

Scanned with CamScanner

to other adjoining Usenet Computers, and then to neighbours of those systems, until it reached the world around.

* These postings will be alive for a finite period of time.
* Each system has a list of newsgroups that it subscribes to and only postings of Interest.
* Older Usenet used UUCP for transporting.

<u>Network News Transfer Protocol (NNTP)</u>

* NNTP is a method by which newsgroup postings are downloaded.
* NNTP can be written using python.
* nntplib is imported and nntplib.NNTP is instantiated.

<u>Example</u>: To download a list of messages

```
import nntplib
import string
SERVER = "news.spam.egg"
GROUP : " comp.lang.python"
AUTHOR = " sam@pyware.com"
Server = nntplib.NNTP(SERVER)
resp, count, first, last, name =
                    server.group(GROUP)
print "count", "=>", count
print "range", '=>', first, last
resp, items = server.xover(first, last)
```

Scanned with CamScanner

```
authors = {}
subjects = {}
 for id, subject, author, date, message_id,
        references, size, lines in items:
        authors[author] = None
        if subject[:4] ="Re:":
             subject = subject [:4]
        subjects[subject] = None
        if string.find(author, AUTHOR) >=0:
             print id, subject
print "authors", "=>", len(authors)
print "subjects","=>", len(subjects)
```

Output:

```
    count = 907
    range : 57179 57971
    57474
      ...
      ...
     subjects => 200
```

E-Mails:

- e-mail message → it is a message consisting of header fields followed by a body.
- The Components of the e-mail system is explained as follows:
- Message Transport Agent (MTA) is a server process running on a mail exchange host which is responsible for routing, queuing and sending of e-mail.

Scanned with CamScanner

* MTA constitutes all hosts including source, hops in between and destination called 'agents' of "message transport'.
* MTAs need to know the 1) next MTA to forward and 2) how to talk to another MTA.

Sending E-Mail:

* A mail client must connect to a MTA to send e-mail.
* MTAs communicate by means of a protocol called Message Transport System
* This MTS protocol was not compatible with different computer types and networks.
* SMTP - Simple Mail Transfer Protocol came into existence in August 1982.
* SMTP was later extended ESMTP
* Another protocol called LMTP - Local Mail Transfer Protocol based on SMTP and EMSTP was defined in 1996.

* Python and SMTP

* smtplib must be imported and smtplib.SMTP class must be instantiated.

Example: To send e-mail using SMTP in python

```
import smtplib
sender = 'from@fromdomain.com'
receivers = ['to@todomain.com']
message = """From : From Person <from@fromdomain.com>
             To : To Person <to@todomain.com>
             Subject : SMTP e-mail test
```

Scanned with CamScanner

This is a test e-mail message
"""

```
try:
        smtpObj = smtplib.SMTP ('localhost')
        smtpObj . sendmail (sender, receivers,
                                        message)
      print ("Successfully sent")
except SMTPException:
      print ('Error')
```

<u>Receiving E-Mail:</u>

- <u>Post Office Protocol (POP)</u> was the first protocol published in 1984.
- The recent version is POP3.
- POP3 is used for downloading e-mails.
- <u>Internet</u> <u>Message</u> <u>Access</u> <u>Protocol</u> (IMAP) is another protocol published in 1988.

<u>Python and POP3:</u>

- To write POP3 using python, import poplib and poplib.POP3 class must be instantiated.
- Some of the POP3 objects are:

```
POP3. user( )
POP3. stat()
POP3. retr()
POP3. dele( )
POP3. rset()
POP3. quit()
```

Scanned with CamScanner

Example: To open a mail box, retrieve and print all messages.

```
import getpass, poplib
M = poplib.POP3('localhost')
M.user(getpass.getuser())
M.pass_(getpass.getpass())
numMessages = len(M.list()[1])
for i in range(numMessages):
    for j in M.retr(i+1)[1]:
        print(j)
```

Example: SMTP and POP3 using Python

```
from smtplib import SMTP
from poplib import POP3
from time import sleep
SMTPSVR = "smtp.python"
POP3SVR = "pop.python"
origHdrs = ['From : from@python.com',
            'To : to@python.com,
            'Subject : test msg']
origBody = ['xxx', 'yyy', 'zzz']
origMsg = '\r\n\r\n'.join(['\r\n'.join(origHdrs). '\r\n'.
                join(origBody)])
sendSvr = SMTP( SMTPSVR)
errs = sendSvr.sendmail ('from@python.com, ('to@python.com',
                          origmsg)
sendSvr.quit()
assert len(errs) == 0, errs
sleep(10)
recvSvr = POP3(POP3SVR)
recvSvr.user('wesley')
recvSvr.pass_("no guns")
rsp, msg, siz = recSVR.retr(recSvr.stat()[0])
```

```
sep = msg.index('')
recBody = msg[sep+1:]
assert origBody ==
          recvBody
```

Scanned with CamScanner

UNIT V - DATABASE AND GUI

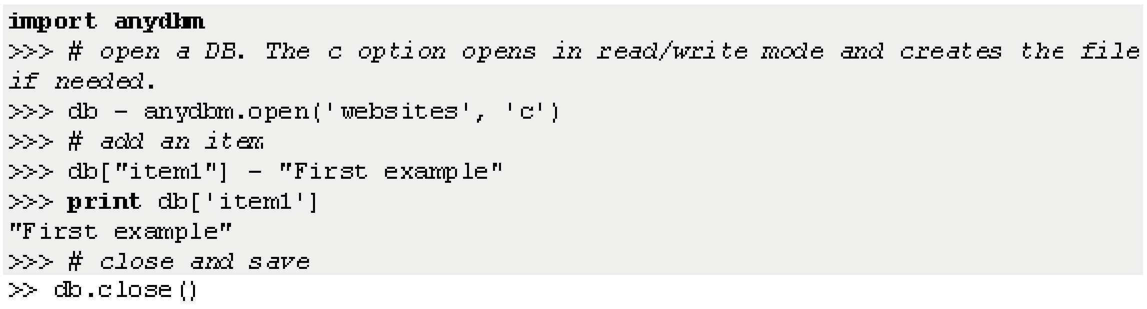
DBM DATABASES:

A database API is provided by Python that is very useful when needed to work with different type of databases. The data are stored within a DBM (database manager) persistent dictionaries that work like normal Python dictionaries except that the data is written to and read from disk. There are many DBM modules and the most common is the **anydbm** module.

The DBM modules work when the data needs to be stored as key/value pairs and can be used when :

- data needs are simple
- small amount of data
- use a relational database if support for transactions is required

Example: To store data into a DB using dictionary-like syntax.:



```
import anydbm
>>> # open a DB. The c option opens in read/write mode and creates the file
if needed.
>>> db - anydbm.open('websites', 'c')
>>> # add an item
>>> db["item1"] - "First example"
>>> print db['item1']
"First example"
>>> # close and save
>> db.close()
```

dbm — Interfaces to Unix “databases:

dbm is a generic interface to variants of the DBM database — dbm.gnu or dbm.ndbm.

Some of the functions are:

exception dbm.**error**

A tuple containing the exceptions that can be raised by each of the supported modules, with a unique exception also named dbm.error as the first item — the latter is used when dbm.error is raised.

dbm.**whichdb**(*filename*)

This function attempts to guess which of the several simple database modules available — dbm.gnu, dbm.ndbm or dbm.dumb — should be used to open a given file.

dbm.**open**(*file*, *flag='r'*, *mode=0o666*)

Open the database file *file* and return a corresponding object. If the database file already exists, the whichdb() function is used to determine its type and the appropriate module is used; if it does not exist, the first module listed above that can be imported is used.

Example : To records some hostnames and a corresponding title, and then print out the contents of the database:

```
import dbm

# Open database, creating it if necessary.
with dbm.open('cache', 'c') as db:

    # Record some values
    db[b'hello'] - b'there'
    db['www.python.org'] - 'Python Website'
    db['www.cnn.com'] - 'Cable News Network'

    # Note that the keys are considered bytes now.
    assert db[b'www.python.org'] -- b'Python Website'
    # Notice how the value is now in bytes.
    assert db['www.cnn.com'] -- b'Cable News Network'

    # Often-used methods of the dict interface work too.
    print(db.get('python.org', b'not present'))

    # Storing a non-string key or value will raise an exception
(most
    # likely a TypeError).
    db['www.yahoo.com'] - 4
```

SQL DATABASES

Python is used to connect the front-end of an application with the back-end database. SQLite can be connected with Python. Python has a native library for SQLite. It's working is given below:

1. To use SQLite, we must import sqlite3.
2. Then create a connection using connect() method and pass the name of the database you want to access if there is a file with that name, it will open that file. Otherwise, Python will create a file with the given name.
3. After this, a cursor object is called to be capable to send commands to the SQL. Cursor is a control structure used to traverse and fetch the records of the database. Cursor has a major role in working with Python. All the commands will be executed using cursor object only.
4. To create a table in the database, create an object and write the SQL command in it with being commented. Example:- sql_comm = "SQL statement"

5. And executing the command is very easy. Call the cursor method execute and pass the name of the sql command as a parameter in it. Save a number of commands as the sql_comm and execute them. After you perform all your activities, save the changes in the file by committing those changes and then lose the connection.

Example : To create table and show insertions into the table

```
# Python code to demonstrate table creation and
# insertions with SQL

# importing module
import sqlite3

# connecting to the database
connection = sqlite3.connect("myTable.db")

# cursor
crsr = connection.cursor()

# SQL command to create a table in the database
sql_command = """CREATE TABLE emp (
staff_number INTEGER PRIMARY KEY,
fname VARCHAR(20),
lname VARCHAR(30),
gender CHAR(1),
joining DATE);"""

# execute the statement
crsr.execute(sql_command)

# SQL command to insert the data in the table
sql_command = """INSERT INTO emp VALUES (23, "Rishabh", "Bansal", "M", "2014-03-28");"""
crsr.execute(sql command)

# another SQL command to insert the data in the table
sql_command = """INSERT INTO emp VALUES (1, "Bill", "Gates", "M", "1980-10-28");"""
crsr.execute(sql_command)

# To save the changes in the files. Never skip this.
# If we skip this, nothing will be saved in the database.
connection.commit()

# close the connection
connection.close()
```

Example : To fetch data from the table.

```
# Python code to demonstrate SQL to fetch data.
# importing the module
import sqlite3
# connect withe the myTable database
connection = sqlite3.connect("myTable.db")
# cursor object
crsr = connection.cursor()
# execute the command to fetch all the data from the table emp
crsr.execute("SELECT * FROM emp")
```

```
# store all the fetched data in the ans variable
ans = crsr.fetchall()
print(ans)
```

Example : To update records and display

```
import sqlite3

conn = sqlite3.connect('test.db')
print "Opened database successfully";

conn.execute("UPDATE COMPANY set SALARY = 25000.00 where ID = 1")
conn.commit
print "Total number of rows updated :", conn.total_changes

cursor = conn.execute("SELECT id, name, address, salary from
COMPANY")
for row in cursor:
   print "ID = ", row[0]
   print "NAME = ", row[1]
   print "ADDRESS = ", row[2]
   print "SALARY = ", row[3], "\n"

print "Operation done successfully";
conn.close()
```

Example : To delete records and display the remaining records.

```
#!/usr/bin/python

import sqlite3

conn = sqlite3.connect('test.db')
print "Opened database successfully";

conn.execute("DELETE from COMPANY where ID = 2;")
conn.commit()
print "Total number of rows deleted :", conn.total_changes

cursor = conn.execute("SELECT id, name, address, salary from
COMPANY")
for row in cursor:
   print "ID = ", row[0]
   print "NAME = ", row[1]
   print "ADDRESS = ", row[2]
   print "SALARY = ", row[3], "\n"

print "Operation done successfully";
conn.close()
```

GUI PROGRAMMING USING PYTHON

GUI – GRAPHICAL USER INTERFACE :

GUI is a desktop application which helps us to interact with the computers. They are used to perform different tasks in the desktops, laptops and other electronic devices. Some of the GUI apps are:

- Text-Editors to create, read, update and delete different types of files.
- Sudoku, Chess and Solitaire to play are games.
- Google Chrome, Firefox and Microsoft Edge to browse through the Internet.

Python Libraries :

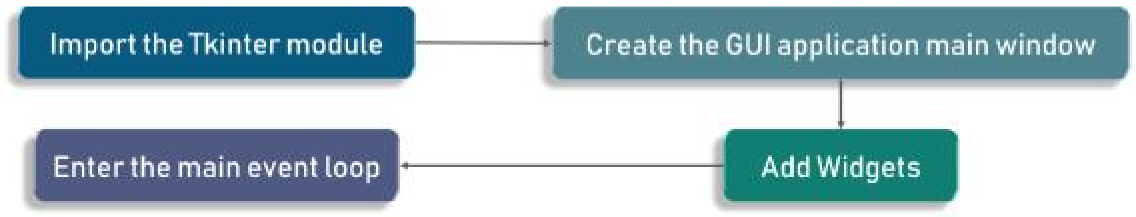
GUI can be created using the following libraries in Python:

- Kivy
- Python QT
- wxPython
- Tkinter

Tkinter:

Tkinter is actually an inbuilt **Python** module used to create simple **GUI** apps. It is the most commonly used module for **GUI** apps in the **Python**.

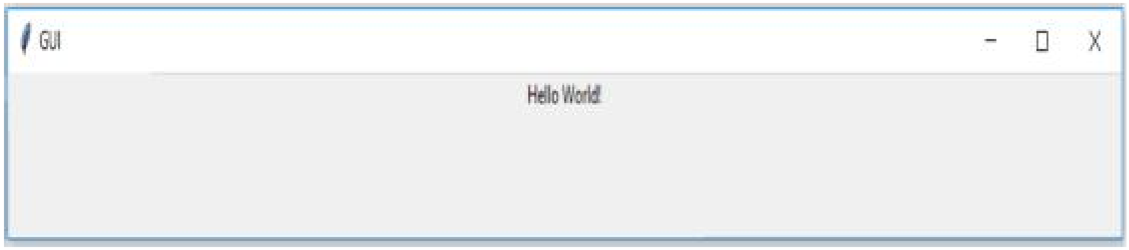
The following diagram shows how an application actually executes in Tkinter:


Import the Tkinter module
Create the GUI application main window
Add Widgets
Enter the main event loop

An event loop is basically telling the code to keep displaying the window until we manually close it. It runs in an infinite loop in the back-end.

```
import tkinter

window = tkinter.Tk()

# to rename the title of the window window.title("GUI")

# pack is used to show the object in the window

label = tkinter.Label(window, text = "Hello World!").pack()

window.mainloop()
```

Tkinter package is imported and window is defined. Also a window title GUI is shown on the title tab whenever you open an application. A label is output needs to be shown on the window. In this case it is hello world.



PROCESSING EVENTS:

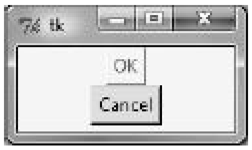
Tkinter GUI programming is event driven. After the user interface is displayed, the program waits for user interactions such as mouse clicks and key presses. This is specified in the following statement:

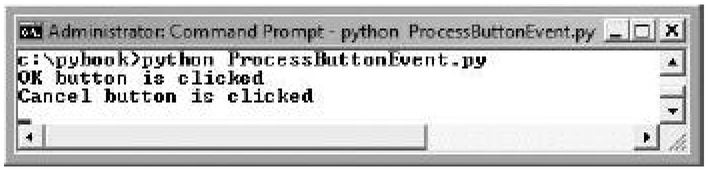
```
window.mainloop()
```

The statement creates an event loop. The event loop processes events continuously until you close the main window. A Tkinter widget can be bound to a function, which is called when an event occurs. When the user clicks a button, your program should process this event. You enable this action by defining a processing function and binding the function to the button, as shown below:

```
# Import all definitions from tkinter

def processOK():
    print("OK button is clicked")

def processCancel():
    print("Cancel button is clicked")

window = Tk() # Create a window
btOK = Button(window, text = "OK", fg = "red", command = processOK)
btCancel = Button(window, text = "Cancel", bg = "yellow",
                  command = processCancel)
btOK.pack() # Place the OK button in the window
btCancel.pack() # Place the Cancel button in the window

window.mainloop() # Create an event loop
```

Output:




Administrator: Command Prompt - python ProcessButtonEvent.py

```
c:\pybook>python ProcessButtonEvent.py
OK button is clicked
Cancel button is clicked
```

Tkinter Widgets

The basic component of a Tk-based application is called a widget. A component is also sometimes called a window, since, in Tk, "window" and "widget" are often used interchangeably. Tk is a package that provides a rich set of graphical components for creating graphical applications with Tcl.

Tk provides a range of widgets ranging from basic GUI widgets like buttons and menus to data display widgets.

Tk applications follow a widget hierarchy where any number of widgets may be placed within another widget, and those widgets within another widget. The main widget in a Tk program is referred to as the root widget and can be created by making a new instance of the TkRoot class.

Creating a Widget

The syntax for creating a widget is given below:

type variableName arguments options

where type here refers to the widget type like button, label, and so on

arguments can be optional and required based on individual syntax of each widget.

options range from size to formatting of each component.

WIDGET CLASSES:

Button A simple button, used to execute a command.

Canvas Structured graphics, used to draw graphs and plots, create graphics editors, and implement custom widgets.

Checkbutton Clicking a check button toggles between the values.

Entry A text entry field, also called a text field or a text box.

Frame A container widget for containing other widgets.

Label Displays text or an image.

Menu A menu pane, used to implement pull-down and popup menus.

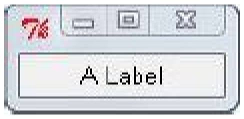
Menubutton A menu button, used to implement pull-down menus.

Message Displays a text. Similar to the label widget, but can automatically wrap text to a given width or aspect ratio.

Radiobutton Clicking a radio button sets the variable to that value, and clears all other radio buttons associated with the same
variable.

Text Formatted text display allows to display and edit text with various styles and attributes. Also supports embedded images and windows.

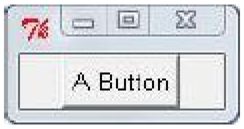
Label Widget



A Label widget shows text to the user

```
import Tkinter
parent_widget = Tkinter.Tk()
label_widget = Tkinter.Label(parent_widget, text="A Label")
label_widget.pack()
Tkinter.mainloop()
```

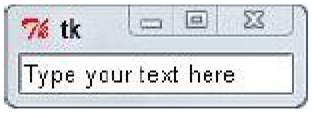
Button Widget



A Button can be on and off. When a user clicks it, the button emits an event. Images can be displayed on buttons.

```
import Tkinter
parent_widget = Tkinter.Tk()
button_widget = Tkinter.Button(parent_widget,
   text="A Button")
button_widget.pack()
Tkinter.mainloop()
```

Entry Widget

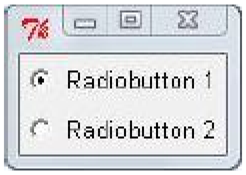


An Entry widget gets text input from the user.

```
import Tkinter
```

```
parent_widget = Tkinter.Tk()
entry_widget = Tkinter.Entry(parent_widget)
entry_widget.insert(0, "Type your text here")
entry_widget.pack()
Tkinter.mainloop()
```

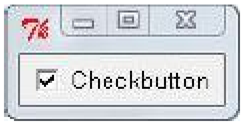
Radiobutton Widget



A Radiobutton lets to put buttons together, so that only one of them can be clicked. If one button is on and the user clicks another, the first is set to off.

```
import Tkinter
parent_widget = Tkinter.Tk()
v = Tkinter.IntVar()
v.set(1) # need to use v.set and v.get to
# set and get the value of this variable
radiobutton_widget1 = Tkinter.Radiobutton(parent_widget,
                            text="Radiobutton 1",
                            variable=v, value=1)
radiobutton_widget2 = Tkinter.Radiobutton(parent_widget,
                            text="Radiobutton 2",
                            variable=v, value=2)
radiobutton_widget1.pack()
radiobutton_widget2.pack()
Tkinter.mainloop()
```

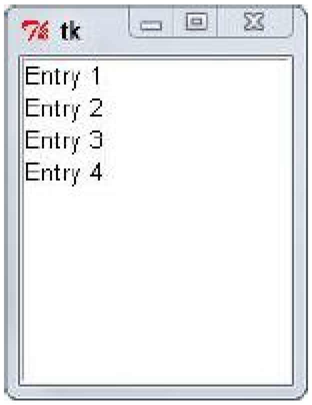
Checkbutton Widget



A Checkbutton records on/off or true/false status. Like a Radiobutton, a Checkbutton widget can be displayed without its check mark, and Tkinter variable is used to access its state.

```
import Tkinter
parent_widget = Tkinter.Tk()
checkbutton_widget = Tkinter.Checkbutton(parent_widget,
                                text="Checkbutton")
checkbutton_widget.select()
checkbutton_widget.pack()
Tkinter.mainloop()
```

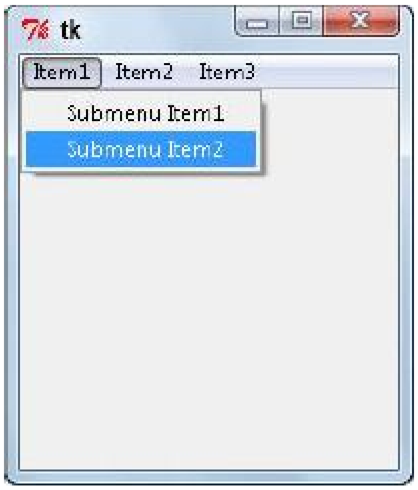
Listbox Widget



Listbox lets the user choose from one set of options or displays a list of items.

```
import Tkinter
parent_widget = Tkinter.Tk()
listbox_entries = ["Entry 1", "Entry 2",
                   "Entry 3", "Entry 4"]
listbox_widget = Tkinter.Listbox(parent_widget)
for entry in listbox_entries:
    listbox_widget.insert(Tkinter.END, entry)
listbox_widget.pack()
Tkinter.mainloop()
```

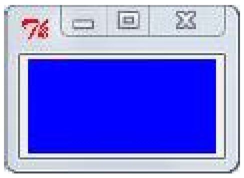
Menu Widget



The Menu widget can create a menu bar

```
import Tkinter
parent_widget = Tkinter.Tk()
def menu_callback():
    print("I'm in the menu callback!")
def submenu_callback():
    print("I'm in the submenu callback!")
menu_widget = Tkinter.Menu(parent_widget)
submenu_widget = Tkinter.Menu(menu_widget, tearoff=False)
submenu_widget.add_command(label="Submenu Item1",
                           command=submenu_callback)
submenu_widget.add_command(label="Submenu Item2",
                           command=submenu_callback)
menu_widget.add_cascade(label="Item1", menu=submenu_widget)
menu_widget.add_command(label="Item2",
                        command=menu_callback)
menu_widget.add_command(label="Item3",
                        command=menu_callback)
parent_widget.config(menu=menu_widget)
Tkinter.mainloop()
```

Canvas Widget



Canvas widget is used to to draw on. It supports different drawing methods.

```
import Tkinter
parent_widget = Tkinter.Tk()
canvas_widget = Tkinter.Canvas(parent_widget
                        bg="blue",
                        width=100,
                        height= 50)
canvas_widget.pack()
Tkinter.mainloop()
```

CANVAS WIDGET:

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

The syntax is given as:

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

where the parameters

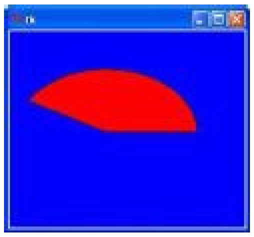
- master − This represents the parent window.
- options − Here is the list of most commonly used options for this widget. These options can be used as key-value pairs separated by commas.

Example:

```
import Tkinter

top - Tkinter.Tk()

C - Tkinter.Canvas(top, bg-"blue", height-250, width-300)

coord - 10, 50, 240, 210
arc - C.create_arc(coord, start-0, extent-150, fill-"red")

C.pack()
```

```
top.mainloop()
```

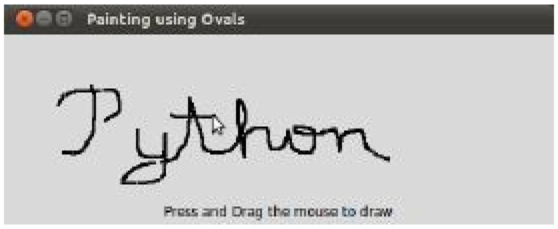
Result:



Example: To paint into a canvas using a small oval

```
from tkinter import *

canvas_width = 500
canvas_height = 150

def paint( event ):
   python_green = "#476042"
   x1, y1 = ( event.x - 1 ), ( event.y - 1 )
   x2, y2 = ( event.x + 1 ), ( event.y + 1 )
   w.create_oval( x1, y1, x2, y2, fill = python_green )

master = Tk()
master.title( "Painting using Ovals" )
w = Canvas(master,
           width=canvas_width,
           height=canvas_height)
w.pack(expand = YES, fill = BOTH)
w.bind( "<B1-Motion>", paint )

message = Label( master, text = "Press and Drag the mouse to draw" )
message.pack( side = BOTTOM )

mainloop()
```

Result:



GEOMETRY MANAGERS:

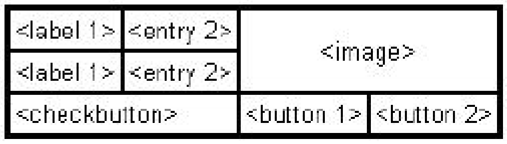
Tkinter uses a geometry manager to place widgets inside a container. Tkinter supports three geometry managers such as the grid manager, the pack manager, and the place manager as explained below.

1. **pack() method:** It organizes the widgets in blocks before placing in the parent widget.
2. **grid() method:** It organizes the widgets in grid (table-like structure) before placing in the parent widget.
3. **place() method:** It organizes the widgets by placing them on specific positions directed by the programmer.

Grid Manager:

The **Grid** geometry manager puts the widgets in a 2-dimensional table. The master widget is split into a number of rows and columns, and each "cell" in the resulting table can hold a widget. The **grid** manager is the most flexible of the geometry managers in Tkinter.

Example : To create the following layout using grid manager

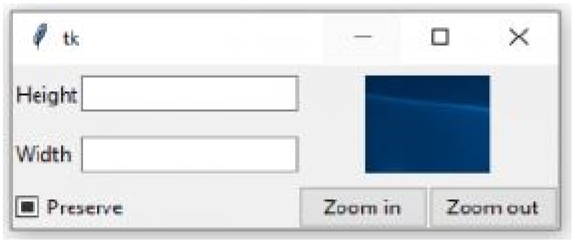


<label 1>	<entry 2>	<image>	
<label 1>	<entry 2>		
<checkbutton>		<button 1>	<button 2>

Program:

```
# import tkinter module
from tkinter import * from tkinter.ttk import *

# creating main tkinter window/toplevel
master = Tk()

# this will create a label widget
l1 = Label(master, text = "Height")
l2 = Label(master, text = "Width")
```

```
# grid method to arrange labels in respective
# rows and columns as specified
l1.grid(row = 0, column = 0, sticky = W, pady = 2)
l2.grid(row = 1, column = 0, sticky = W, pady = 2)

# entry widgets, used to take entry from user
e1 = Entry(master)
e2 = Entry(master)

# this will arrange entry widgets
e1.grid(row = 0, column = 1, pady = 2)
e2.grid(row = 1, column = 1, pady = 2)

# checkbutton widget
c1 = Checkbutton(master, text = "Preserve")
c1.grid(row = 2, column = 0, sticky = W, columnspan = 2)

# adding image (remember image should be PNG and not JPG)
img = PhotoImage(file - r"C:\Users\Admin\Pictures\capture1.png")
img1 = img.subsample(2, 2)

# setting image with the help of label
Label(master, image = img1).grid(row = 0, column = 2,
       columnspan = 2, rowspan = 2, padx = 5, pady = 5)

# button widget
b1 = Button(master, text = "Zoom in")
b2 = Button(master, text = "Zoom out")

# arranging button widgets
b1.grid(row = 2, column = 2, sticky = E)
b2.grid(row = 2, column = 3, sticky = E)

# infinite loop which can be terminated
# by keyboard or mouse interrupt
mainloop()
```
Output:



Pack Manager:

The Pack geometry manager packs widgets in rows or columns. The options like **fill**, **expand**, and **side** can be used to control pack manager. It is created as follows:

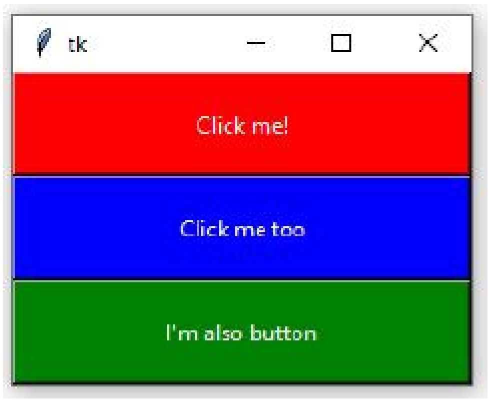
- Put a widget inside a frame (or any other container widget), and have it fill the entire frame
- Place a number of widgets on top of each other

- Place a number of widgets side by side

Example : Putting a widget inside frame and filling entire frame
using **expand** and **fill** options and placing widgets on top of each other.

```
# Importing tkinter module
from tkinter import *
# from tkinter.ttk import *

# creating Tk window
master = Tk()

# cretaing a Fra, e which can expand according
# to the size of the window
pane = Frame(master)
pane.pack(fill = BOTH, expand = True)

# button widgets which can also expand and fill
# in the parent widget entirely
b1 = Button(pane, text = "Click me !",
               background = "red", fg = "white")
b1.pack(side = TOP, expand = True, fill = BOTH)

b2 = Button(pane, text = "Click me too",
               background = "blue", fg = "white")
b2.pack(side = TOP, expand = True, fill = BOTH)

b3 = Button(pane, text = "I'm also button",
               background = "green", fg = "white")
b3.pack(side = TOP, expand = True, fill = BOTH)

mainloop()
```

Output:



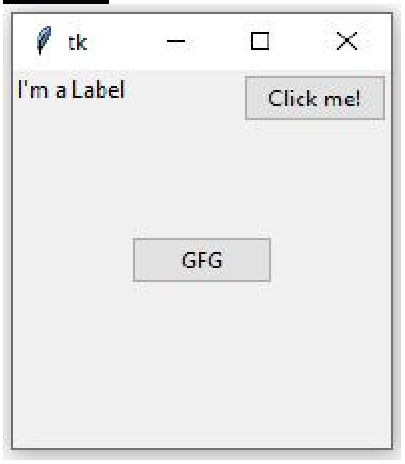
Place Manager:

The **Place** geometry manager is the simplest of the three general geometry managers provided in Tkinter. It allows to explicitly set the position and size of a window, either in absolute terms, or relative to another window.

Example:

```
# Importing tkinter module
from tkinter import * from tkinter.ttk import *

# creating Tk window
master = Tk()

# setting geometry of tk window
master.geometry("200x200")

# button widget
b1 = Button(master, text = "Click me !")
b1.place(relx = 1, x =-2, y = 2, anchor = NE)

# label widget
l = Label(master, text = "I'm a Label")
l.place(anchor = NW)

# button widget
b2 = Button(master, text = "GFG")
b2.place(relx = 0.5, rely = 0.5, anchor = CENTER)

# infinite loop which is required to
# run tkinter program infinitely
# until an interrupt occurs
mainloop()
```

Output:



DISPLAYING IMAGES:

An image can be added to a label, button, check button, or radio button. To create an image, the **PhotoImage** class as follows can be used.

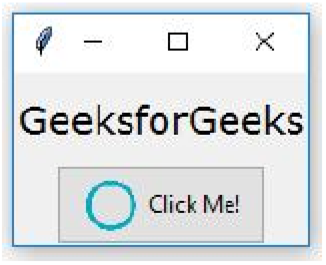
photo = PhotoImage(file = imagefilename)

The image file must be in GIF format. You can use a conversion utility to convert image files in other formats into GIF format.

Example: To show both image and text on Button.

```
# importing only those functions
# which are needed
from tkinter import *
from tkinter.ttk import *

# creating tkinter window
root = Tk()

# Adding widgets to the root window
Label(root, text = 'GeeksforGeeks', font =(
  'Verdana', 15)).pack(side = TOP, pady = 10)

# Creating a photoimage object to use image
photo = PhotoImage(file = r"C:\Gfg\circle.png")

# Resizing image to fit on button
photoimage = photo.subsample(3, 3)

# here, image option is used to
# set image on button
# compound option is used to align
# image on LEFT side of button
Button(root, text = 'Click Me !', image = photoimage,
                    compound = LEFT).pack(side = TOP)

mainloop()
```

Output:

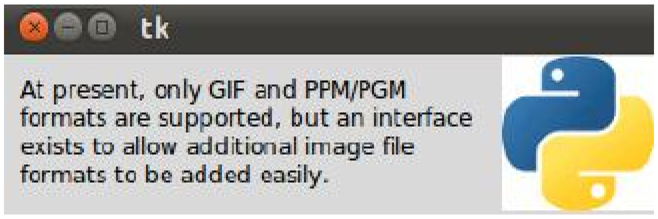


Example: To add images and text to a label

```
import tkinter as tk


root = tk.Tk()

logo = tk.PhotoImage(file="python_logo_small.gif")


w1 = tk.Label(root, image=logo).pack(side="right")


explanation = """At present, only GIF and PPM/PGM
```

```
formats are supported, but an interface
exists to allow additional image file
formats to be added easily."""

w2 = tk.Label(root,
              justify=tk.LEFT,
              padx = 10,
              text=explanation).pack(side="left")
root.mainloop()
```

Output:



At present, only GIF and PPM/PGM
formats are supported, but an interface
exists to allow additional image file
formats to be added easily.

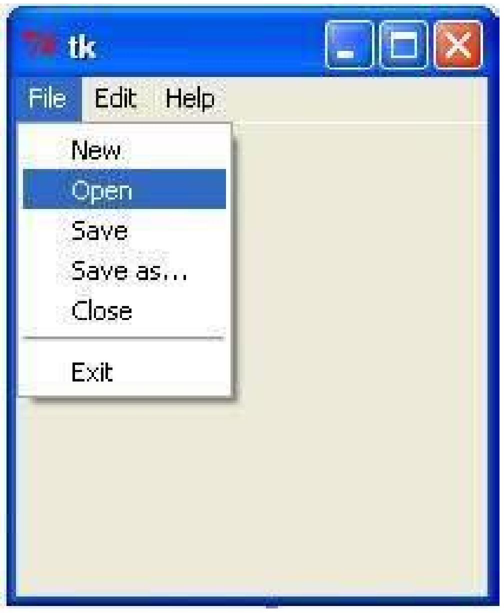
MENUS:

Tkinter is used to create menus, popup menus, and toolbars. Tkinter provides a comprehensive solution for building graphical user interfaces. Menus make selection easier and are widely used in windows. **Menu** class is used to create a menu bar and a menu, and **add_command** method to add items to the menu.

Example : To create menu using Menu class and add command method.

```
from Tkinter import *

def donothing():
   filewin = Toplevel(root)
   button = Button(filewin, text="Do nothing button")
   button.pack()

root = Tk()
menubar = Menu(root)
filemenu = Menu(menubar, tearoff=0)
filemenu.add_command(label="New", command=donothing)
filemenu.add_command(label="Open", command=donothing)
filemenu.add_command(label="Save", command=donothing)
filemenu.add_command(label="Save as...", command=donothing)
filemenu.add_command(label="Close", command=donothing)

filemenu.add_separator()

filemenu.add_command(label="Exit", command=root.quit)
menubar.add_cascade(label="File", menu=filemenu)
editmenu = Menu(menubar, tearoff=0)
editmenu.add_command(label="Undo", command=donothing)

editmenu.add_separator()

editmenu.add_command(label="Cut", command=donothing)
editmenu.add_command(label="Copy", command=donothing)
editmenu.add_command(label="Paste", command=donothing)
editmenu.add_command(label="Delete", command=donothing)
editmenu.add_command(label="Select All", command=donothing)

menubar.add_cascade(label="Edit", menu=editmenu)
helpmenu = Menu(menubar, tearoff=0)
helpmenu.add_command(label="Help Index", command=donothing)
helpmenu.add_command(label="About...", command=donothing)
menubar.add_cascade(label="Help", menu=helpmenu)

root.config(menu=menubar)
root.mainloop()
```

Output:



POPUP MENUS:

A popup menu, also known as a context menu, is like a regular menu, but it does not have a menu bar and it can float anywhere on the screen. Creating a popup menu is similar to creating a regular menu. First, an instance of Menu is created , and then items are added to it. Finally, widget is bound with an event to pop up the menu.

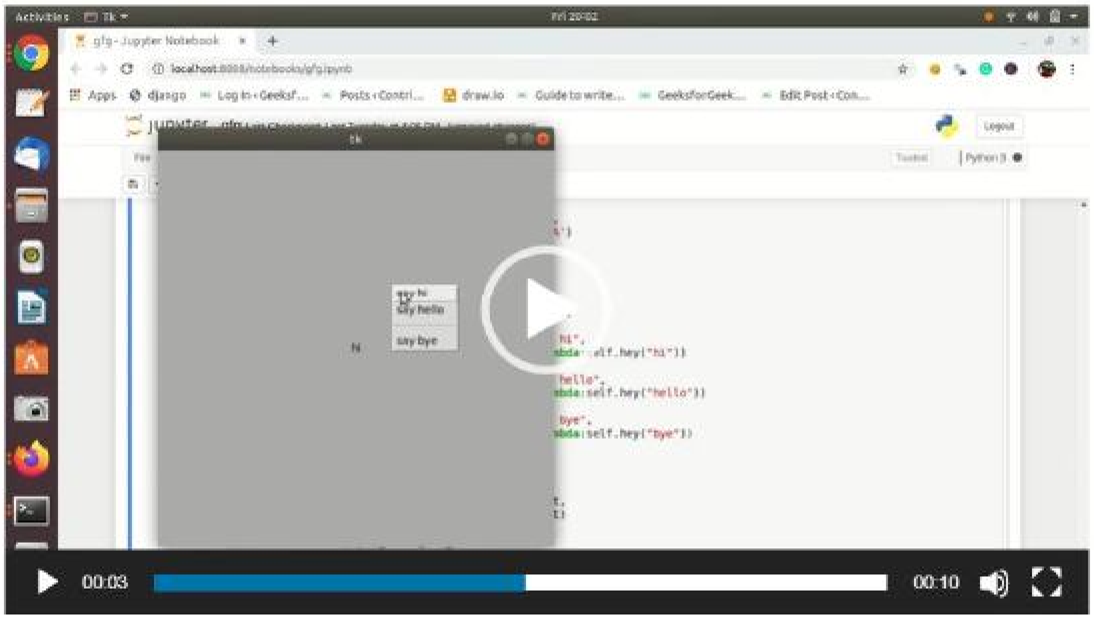
Example : To create a popup menu

```
#creating popup menu in tkinter
import tkinter

class A:
    #creates parent window
    def __init__(self):

        self.root = tkinter.Tk()
        self.root.geometry('500x500')

        self.frame1 = tkinter.Label(self.root,
                                    width = 400,
                                    height = 400,
                                    bg = '#AAAAAA')
        self.frame1.pack()

    #create menu
    def popup(self):
        self.popup_menu = tkinter.Menu(self.root,
                                       tearoff = 0)

        self.popup_menu.add_command(label = "say hi",
                                    command = lambda:self.hey("hi"))

        self.popup_menu.add_command(label = "say hello",
                                    command = lambda:self.hey("hello"))
        self.popup_menu.add_separator()
        self.popup_menu.add_command(label = "say bye",
                                    command = lambda:self.hey("bye"))

    #display menu on right click
    def do_popup(self,event):
        try:
            self.popup_menu.tk_popup(event.x_root,
                                     event.y_root)
        finally:
            self.popup_menu.grab_release()

    def hey(self,s):
        self.frame1.configure(text = s)

    def run(self):
        self.popup()
        self.root.bind("<Button-3>",self.do_popup)
        tkinter.mainloop()
```

```
a = A()
a.run()
```

<u>Output:</u>

A popup menu appears on right click.



<u>MOUSE, KEY EVENTS, AND BINDINGS:</u>

The **bind** method is used to bind mouse and key events to a widget. The event is a standard Tkinter object, which is automatically created when an event occurs. Every handler has an event as its argument. The following example defines the handler using the event as the argument:

menu.post(event.x_root, event.y_root)

The **event** object has a number of properties describing the event pertaining to the event. For example, for a mouse event, the **event** object uses the **x**, **y** properties to capture the current mouse location in pixels.

The events and their properties are listed below:

<ButtonReleased-i> An event occurs when a mouse button is released.

<Double-Button-i> An event occurs when a mouse button is double-clicked.

<Enter> An event occurs when a mouse pointer enters the widget.

<Key> An event occurs when a key is pressed.

<Leave> An event occurs when a mouse pointer leaves the widget.

<Return> An event occurs when the *Enter* key is pressed. You can bind any key such as *A*,

B, *Up*, *Down*, *Left*, *Right*

in the keyboard with an event.

<Shift+A> An event occurs when the *Shift+A* keys are pressed. You can combine *Alt*, *Shift*, and *Control* with other keys.

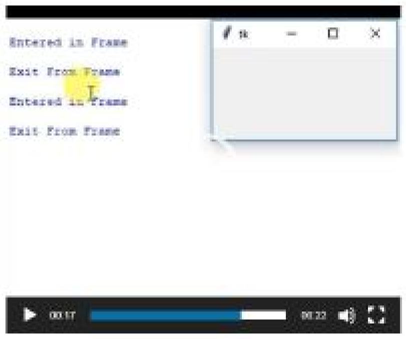
<Triple-Button-i> An event occurs when a mouse button is triple-clicked.

Binding function is used to deal with the events. We can bind **Python's functions** and methods to an event as well as we can bind these functions to any particular widget.

Example: Binding mouse movement with tkinter Frame.

```
from tkinter import * from tkinter.ttk import *
  # creates tkinter window or root window
root = Tk()
root.geometry('200x100')
  # function to be called when mouse enters in a frame
def enter(event):
    print('Button-2 pressed at x = % d, y = % d'%(event.x, event.y))
  # function to be called when when mouse exits the frame
def exit_(event):
    print('Button-3 pressed at x = % d, y = % d'%(event.x, event.y))
  # frame with fixed geomerty
frame1 = Frame(root, height = 100, width = 200)
  # these lines are showing the
# working of bind function
# it is universal widget method
frame1.bind('<Enter>', enter)
frame1.bind('<Leave>', exit_)
frame1.pack()
mainloop()
```

Output:



```
Entered in Frame
Exit From Frame
Entered in Frame
Exit From Frame
```

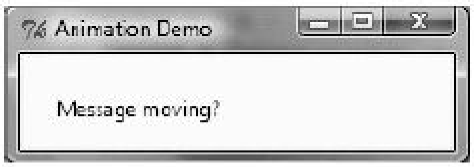
ANIMATIONS:

Animations can be created by displaying a sequence of drawings. The Canvas class can be used to develop animations. Graphics and text can be displayed on the canvas using the move(tags, dx, dy) method to move the graphic with the specified tags.

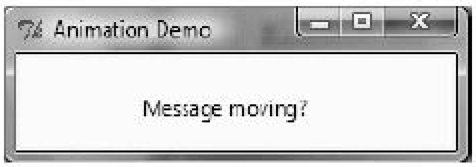
Example: To create an animation

```
 1  from tkinter import * # Import all definitions from tkinter
 2
 3  class AnimationDemo:
 4      def __init__(self):
 5          window = Tk() # Create a window
 6          window.title("Animation Demo") # Set a title
 7
 8          width = 250 # Width of the canvas
 9          canvas = Canvas(window, bg = "white",
10              width = 250, height = 50)
11          canvas.pack()
12
13          x = 0 # Starting x position
14          canvas.create_text(x, 30,
15              text = "Message moving?", tags = "text")
16
17          dx = 3
18          while True:
19              canvas.move("text", dx, 0) # Move text dx unit
20              canvas.after(100) # Sleep for 100 milliseconds
21              canvas.update() # Update canvas
22              if x < width:
23                  x += dx  # Get the current position for string
24              else:
25                  x = 0 # Reset string position to the beginning
26                  canvas.delete("text")
27                  # Redraw text at the beginning
28                  canvas.create_text(x, 30, text = "Message moving?",
29                      tags = "text")
30
31          window.mainloop() # Create an event loop
32
33  AnimationDemo() # Create GUI
```

Output:



Message moving?



Message moving?

The animation is done essentially in the following three statements in a loop (lines 19–21):

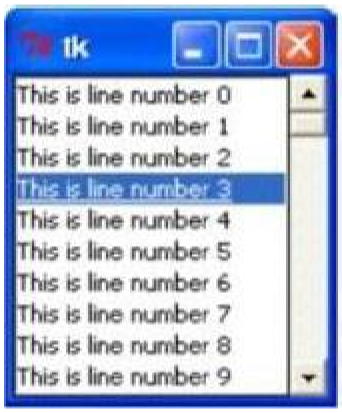
canvas.move(**"text"**, dx, **0**) # Move text dx unit
canvas.after(**100**) # Sleep for 100 milliseconds
canvas.update() # Update canvas

SCROLLBARS:

A Scrollbar widget can be used to scroll the contents in a Text, Canvas, or Listbox widget vertically or horizontally.

```
from Tkinter import *

root = Tk()
scrollbar = Scrollbar(root)
scrollbar.pack( side = RIGHT, fill = Y )

mylist = Listbox(root, yscrollcommand = scrollbar.set )
for line in range(100):
   mylist.insert(END, "This is line number " + str(line))

mylist.pack( side = LEFT, fill = BOTH )
scrollbar.config( command = mylist.yview )

mainloop()
```

Output:



```
This is line number 0
This is line number 1
This is line number 2
This is line number 3
This is line number 4
This is line number 5
This is line number 6
This is line number 7
This is line number 8
This is line number 9
```

STANDARD DIALOG BOXES:

Standard dialog boxes can be used to display message boxes or to prompt the user to enter numbers and strings.

Message Dialogues:

The message dialogues are provided by the 'messagebox' submodule of tkinter. 'messagebox' consists of the following functions, which correspond to dialog windows:

- askokcancel(title=None, message=None, **options)
  Ask if operation should proceed; return true if the answer is ok
- askquestion(title=None, message=None, **options)
  Ask a question

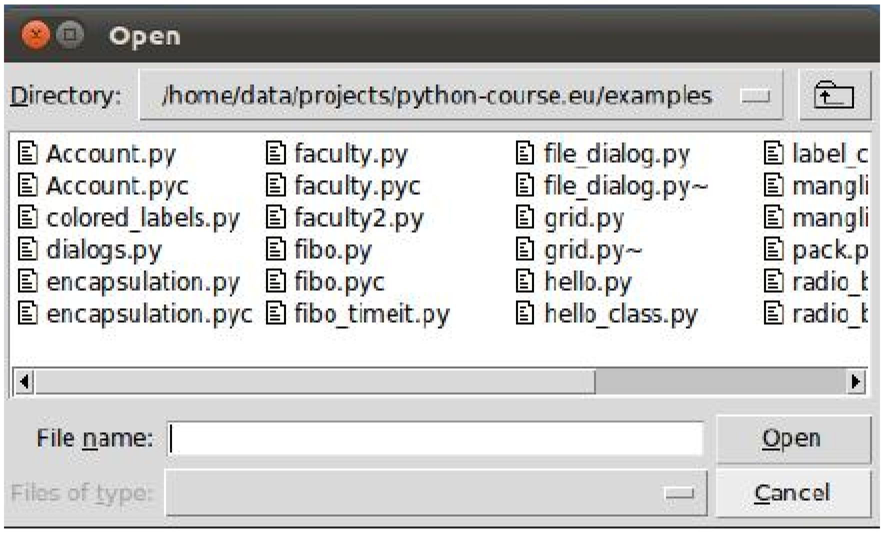
- askretrycancel(title=None, message=None, **options)
  Ask if operation should be retried; return true if the answer is yes
- askyesno(title=None, message=None, **options)
  Ask a question; return true if the answer is yes
- askyesnocancel(title=None, message=None, **options)
  Ask a question; return true if the answer is yes, None if cancelled.
- showerror(title=None, message=None, **options)
  Show an error message
- showinfo(title=None, message=None, **options)
  Show an info message
- showwarning(title=None, message=None, **options)
  Show a warning message

Open File Dialogue:

```
import tkinter as tk
from tkinter import filedialog as fd

def callback():
    name= fd.askopenfilename()
    print(name)

errmsg = 'Error!'
tk.Button(text='File Open',
       command=callback).pack(fill=tk.X)
tk.mainloop()
```

Output:

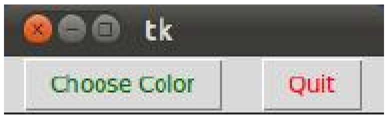
The above code creates a window with a single button with the text "File Open". If the button is pushed, the following window appears:

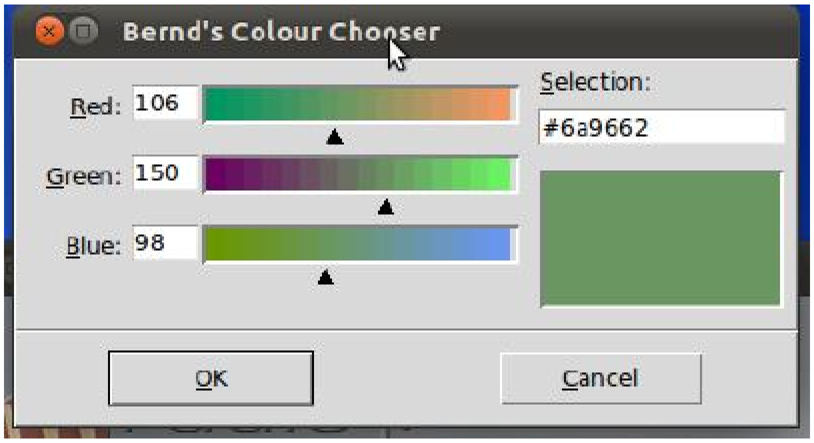


Colour Dialogue:

```
import tkinter as tk
from tkinter.colorchooser import askcolor
def callback():
    result = askcolor(color="#6A9662",
                      title = "Bernd's Colour Chooser")
    print(result)
    root = tk.Tk()
tk.Button(root,
          text='Choose Color',
          fg="darkgreen",
          command=callback).pack(side=tk.LEFT, padx=10)
tk.Button(text='Quit',
          command=root.quit,
          fg="red").pack(side=tk.LEFT, padx=10)
tk.mainloop()
```

Output:





LIST BOXES:

A Listbox widget is used to display a list of items from which a user can select a number of items.

The syntax for a listbox creation is :

```
listbox = Listbox(root, bg, fg, bd, height, width, font, ..)
```

where optional parameters are:

- root – root window.
- bg – background colour
- fg – foreground colour
- bd – border
- height – height of the widget.
- width – width of the widget.
- font – Font type of the text.
- highlightcolor – The colour of the list items when focused.
- yscrollcommand – for scrolling vertically.
- xscrollcommand – for scrolling horizontally.
- cursor – The cursor on the widget which can be an arrow, a dot etc.

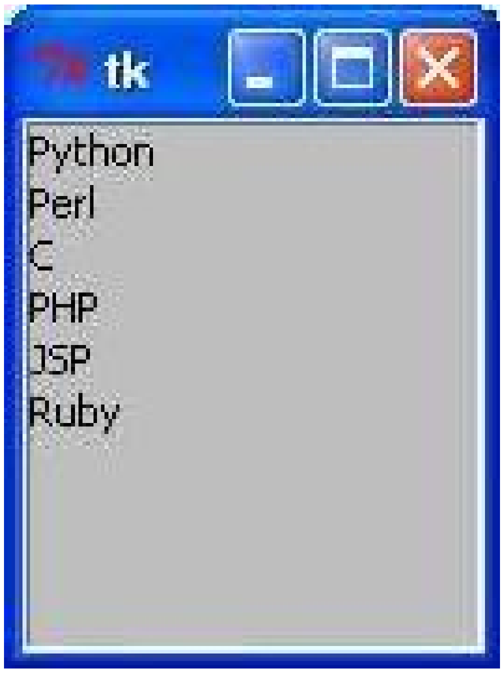
Common methods are:

- yview – allows the widget to be vertically scrollable.
- xview – allows the widget to be horizontally scrollable.
- get() – to get the list items in a given range.
- activate(index) – to select the lines with a specified index.
- size() – return the number of lines present.
- delete(start, last) – delete lines in the specified range.
- nearest(y) – returns the index of the nearest line.

Example 1: To create a Listbox

```
from Tkinter import *
import tkMessageBox
import Tkinter

top = Tk()

Lb1 = Listbox(top)
Lb1.insert(1, "Python")
Lb1.insert(2, "Perl")
Lb1.insert(3, "C")
Lb1.insert(4, "PHP")
Lb1.insert(5, "JSP")
Lb1.insert(6, "Ruby")

Lb1.pack()
top.mainloop()
```

Output:

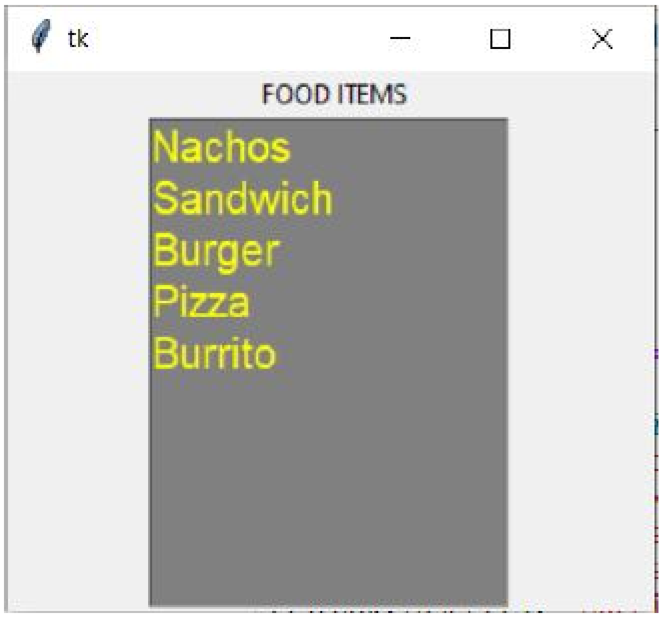


```
Python
Perl
C
PHP
JSP
Ruby
```

Example 2: To create a listbox

```
from tkinter import *


# create a root window.
top = Tk()

# create listbox object
listbox = Listbox(top, height = 10,
                  width = 15,
                  bg = "grey",
                  activestyle = 'dotbox',
                  font = "Helvetica",
                  fg = "yellow")

# Define the size of the window.
top.geometry("300x250")

# Define a label for the list.
label = Label(top, text = " FOOD ITEMS")

# insert elements by their
# index and names.
listbox.insert(1, "Nachos")
listbox.insert(2, "Sandwich")
listbox.insert(3, "Burger")
listbox.insert(4, "Pizza")
listbox.insert(5, "Burrito")
 # pack the widgets
label.pack()
listbox.pack()
#Display untill User
# exits themselves.
top.mainloop()
```

Output:



```
FOOD ITEMS
Nachos
Sandwich
Burger
Pizza
Burrito
```