1. CLOSE A FILE

```
fo = open("foo.txt", "wb")
print "Name of the file: ", fo.name
Close opened file
fo.close()
```

2. OPEN A FILE

```
Fileobject=open(filename[,access_mode][,buffering])
```

3. READ A FILE

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

```
fo = open("foo.txt", "r+")
str = fo.read(10);
print "Read String is : ", str
fo.close()
```

4. WRITE A FILE

```
fileObject.write(string);
fo = open("foo.txt", "wb")
fo.write( "Python is a great language.\nYeah its great!!\n");
fo.close()
```

5. THE READLINE() METHOD

```
infile = open("C://P.txt", "r")
print("(1) Using read(): ")
line1=infile.readline() print(line1)
line1=infile.readline() print(line1)
line1=infile.readline() print(line1)
infile.close()
```

6. PYTHON FILE AND DIRECTORY

1. rename():

Syntax: os.rename(existing_file_name, new_file_name)

eg:

```
import os
os.rename('mno.txt','pqr.txt')
```

2. remove():

Syntax:os.remove(file_name)

```
eg: import os
os.remove('mno.txt')
```

3. mkdir()

Syntax: os.mkdir("file_name")

eg: import os

os.mkdir("new")

4. chdir()

Syntax: os.chdir("file_name")

eg: import os

os.chdir("new")

5. getcwd()

Syntax: os.getcwd()

eg: import os

print os.getcwd()

6. Following program illustrates a program that copies data from a source file to a target file and counts the number of lines and characters in the file.

```
import os.path
import sys
```

```
def main():
        f1 = input("Enter a source file: ").strip()
        f2 = input("Enter a target file: ").strip()
    if os.path.isfile(f2) :
        print(f2 + " already exists")
        sys.exit()
    infile = open(f1, "r")
    outfile = open(f2, "w")
    countLines = countChars = 0 20
    for line in infile:
        countLines += 1
        countChars += len(line)
        outfile.write(line)
     print(countLines, "lines and", countChars, "chars copied")
    infile.close()
    outfile.close()
    main()
```

7. # Program to check Armstrong numbers in certain interval

```
lower = int(input("Enter lower range: "))
```

```
upper = int(input("Enter upper range: "))
for num in range(lower, upper + 1):
   order = len(str(num))
   sum = 0
   temp = num
   while temp > 0:
      digit = temp % 10
      sum += digit ** order
      temp //= 10
   if num == sum:
      print(num)
```

8. BINARY SEARCH

```
def binarySearch(alist, item):
    first = 0
    last = len(alist)-1
    found = False
    while first<=last and not found:
        midpoint = (first + last)//2
        if alist[midpoint] == item:
            found = True
            print("{0} found at position {1}".format(item,midpoint+1))
        else:
            if item < alist[midpoint]:
                last = midpoint-1
            else:
                first = midpoint+1

    return found

testlist = [0, 1, 2, 8, 13, 17, 19, 32, 42,]
```

```
print(binarySearch(testlist, 13))
```

9. ''' Program make a simple CALCULATOR that can add, subtract, multiply and divide using functions '''

```
def add(x, y):
  return x + y
def subtract(x, y):
    return x - y
def multiply(x, y):
    return x * y
def divide(x, y):
  return x / y
print("Select operation.")
print("1.Add")
print("2.Subtract")
print("3.Multiply")
print("4.Divide")
choice = input("Enter choice(1/2/3/4):")
num1 = int(input("Enter first number: "))
num2 = int(input("Enter second number: "))
if choice == '1':
  print(num1,"+",num2,"=", add(num1,num2))
elif choice == '2':
  print(num1,"-",num2,"=", subtract(num1,num2))
elif choice == '3':
  print(num1,"*",num2,"=", multiply(num1,num2))
elif choice == '4':
  print(num1,"/",num2,"=", divide(num1,num2))
else:
  print("Invalid input")
```

10.# Python program to display all the prime numbers within an interval

```
lower = int(input("Enter lower range: "))
upper = int(input("Enter upper range: "))
```

```
print("Prime numbers between",lower,"and",upper,"are:")

for num in range(lower,upper + 1):
   if num > 1:
      for i in range(2,num):
         if (num % i) == 0:
            break
      else:
         print(num)
```

11.# Python program to check if the input number is prime or not

```
num = int(input("Enter a number: "))
if num > 1:
   for i in range(2,num):
      if (num % i) == 0:
         print(num,"is not a prime number")
         print(i,"times",num//i,"is",num)
         break
   else:
      print(num,"is a prime number")
      else:
   print(num,"is not a prime number")
```

UNIT 3

1. CLASS COMPLEX (SELF,COSTRUCTOR)

```
class ComplexCompute:
        def __init__(self, realPart, imagPart):
```

```
            self.realPart = realPart
            self.imagPart = imagPart
        def add(self, other):
            resultR = self.realPart+other.realPart
            resultI = self.imagPart+other.imagPart
            result = complex(resultR, resultI)
            return result
        def sub(self, other):
            resultR = self.realPart-other.realPart
            resultI = self.imagPart-other.imagPart
            result = complex(resultR, resultI)
            return result
c1 = ComplexCompute(2,2)
c2 = ComplexCompute(1,1)
print "sum is:", c1.add(c2)
print "Difference is:",c1.sub(c2)
```

2. DESTROCTOR

```
class Point:
    def __init( self, x=0, y=0):
        self.x = x
        self.y = y
    def __del__(self):
        class_name = self.__class__.__name__
        print class_name, "destroyed"
pt1 = Point()
pt2 = pt1
pt3 = pt1
```

```
print id(pt1), id(pt2), id(pt3) # prints the ids of the obejcts
del pt1
del pt2
del pt3
```

3. CLASS INHERITANCE

```
class Parent: # define parent class
parentAttr = 100
def __init__(self):
print "Calling parent constructor"
def parentMethod(self):
print 'Calling parent method'
def setAttr(self, attr):
Parent.parentAttr = attr
def getAttr(self):
print "Parent attribute :", Parent.parentAttr
class Child(Parent): # define child class
def __init__(self):
print "Calling child constructor"
def childMethod(self):
print 'Calling child method'
c = Child() # instance of child
c.childMethod() # child calls its method
c.parentMethod() # calls parent's method
c.setAttr(200) # again call parent's method
c.getAttr() # again call parent's metho
```

4. CALLING PARENT CLASS CONSTRUCTOR THROUGH CHILD CONSTRUCTOR

```
class Parent: # define parent class
parentAttr = 100
def __init__(self,a):
print "Calling parent constructor"
Parent.parentAttr=a
def parentMethod(self):
print 'Calling parent method'
def setAttr(self, attr):
Parent.parentAttr = attr
def getAttr(self):
print "Parent attribute :", Parent.parentAttr
class Child(Parent): # define child class
def __init__(self,parent):
print "Calling child constructor"
Parent.__init__(self,parent)
def childMethod(self):
print 'Calling child method'
c = Child(300) # instance of child
c.childMethod()
c.parentMethod()
c.getAttr()
c.setAttr(200) # again call parent's method
c.getAttr() # again call parent's method
```

5. DATA HIDING

```
class JustCounter:
    __secretCount = 0
    def count(self):
        self.__secretCount += 1
        print self.__secretCount
    def __hidden(self):
        print "Am hidden..."
```

```
        def accesshidden(self):
                self.__hidden()
counter = JustCounter()
counter.count()
counter.count()
print counter._JustCounter__secretCount

counter.accesshidden()
counter._JustCounter__hidden()
counter.__hidden()
print counter.__secretCount
```

Output

```
122
Am hidden...
Am hidden...
Traceback (most recent call last):
File "opoverload.py", line 77, in <module>
counter.__hidden()
AttributeError: JustCounter instance has no attribute '__hidden'
```

6. SINGLE LEVEL INHERITANCE

```
class Instrument:
        def __init__(self, name):
                self.name = name
        def has_strings(self):
                return True
class PercussionInstrument(Instrument):
        def has_strings(self):
        return False
 guitar = Instrument('guitar')
drums = PercussionInstrument('drums')
print ('Guitar has strings: {0}'.format(guitar.has_strings()))
```

```
print ('Guitar name: {0}'.format(guitar.name))
print ('Drums have strings: {0}'.format(drums.has_strings()))
print ('Drums name: {0}'.format(drums.name))
```

7. MULTILEVEL INHERITANCE

```
class Instrument:
        def __init__(self, name):
                self.name = name
        def has_strings(self):
                return True
class StringInstrument(Instrument):
        def __init__(self, name, count):
                super(StringInstrument, self).__init__(name)
                self.count = count
class Guitar(StringInstrument):
        def __init__(self):
                super(Guitar, self).__init__('guitar', 6)
guitar = Guitar()
print ('Guitar name: {0}'.format(guitar.name))
print ('Guitar count: {0}'.format(guitar.count))
```

8. Overriding Methods

```
class Parent:
   def myMethod(self):
           print ('Calling parent method')
class Child(Parent):
   def myMethod(self):
```

```
        print ('Calling child method')
c = Child() # instance of child
c.myMethod()
```

9. ABSTRACT CLASS

```
from abc import ABCMeta, abstractmethod

import sys
import traceback
class Instrument(object):
        __metaclass__ = ABCMeta
        def __init__(self, name):
                self.name = name
        def has_strings(self):
                pass
class StringInstrument(Instrument):
        def has_strings(self):
 return True
guitar = StringInstrument('guitar')
print ('Guitar has strings: {0}'.format(guitar.has_strings()))
try:
        guitar = Instrument('guitar')
except:
        traceback.print_exc(file=sys.stdout)
```

10. PERSISTENT STORAGE OF OBJECTS

```
import pickle
a = 'test value’
fileObject = open("sample",'wb')

pickle.dump(a,fileObject)

fileObject.close()

fileObject = open("sample",'rb')
while True:
   try:
         b1 = pickle.load(fileObject)
         print(b1)
   except EOFError:
         fileObject.close()
         break
```

11. COMPLEX (OPERATOR OVERLOADING)

```
    class Complex1:
         def __init__(self, realPart=0, imagPart=0):
               self.realPart = realPart
               self.imagPart = imagPart
```

```
        def __add__(self, other):
                resultR = self.realPart+other.realPart
                resultI = self.imagPart+other.imagPart
                result = Complex1(resultR, resultI)
                return result
        def __sub__(self, other):
                resultR = self.realPart-other.realPart
                resultI = self.imagPart-other.imagPart
                result = Complex1(resultR, resultI)
                return result
c1 = Complex1(2,3)
c2 = Complex1(1,4)
c3 = Complex1()
c4 = Complex1()
c3 = c1+c2
print "sum is:",c3
c4 = c1 - c2
print "Difference is:",c4
```

12. COMPRESSION (OPERATOR OVERLOADING)

```
class test:
        def __init__(self,a):
                self.a=a
        def __gt__(self,other):
```

```
        if self.a > other.a:
            return True
        else:
            return False
t1=test(15)
t2=test(70)
print t1>t2
print t2>t1
```

UNIT 4

1. Finding sum of two numbers by taking input from text boxes.

```
from tkinter import *
root = Tk()
e1 = Entry(root)
e2 = Entry(root)
l = Label(root)
def callback():

        a= int(e1.get())
        b= int(e2.get())
        total=a+b
        l.config(text="answer = %s" % total)
        b = Button(root, text="add them", command=callback)
for widget in (e1, e2, l, b):
        widget.pack()
b.mainloop()
```

2. RADIOBUTTON

```
from tkinter import *
def sel():
        selection = "You selected the option " + str(var.get())
        label.config(text = selection)
root = Tk()
var = IntVar()
R1 = Radiobutton(root, text="Option 1", variable=var, value=1, command=sel)
R1.pack()
R2 = Radiobutton(root, text="Option 2", variable=var, value=2, command=sel)
R2.pack()
R3 = Radiobutton(root, text="Option 3", variable=var, value=3, command=sel)
R3.pack()
label = Label(root)
label.pack()
root.mainloop()
```

3. CHECKBOX

```
from tkinter import *
master = Tk()
def var_states():
        print("male: %d,\nfemale: %d" % (var1.get(), var2.get()))
Label(master, text="Your sex:").grid(row=0, sticky=W)
var1 = IntVar()
Checkbutton(master, text="male", variable=var1).grid(row=1, sticky=W)
var2 = IntVar()
Checkbutton(master, text="female", variable=var2).grid(row=2, sticky=W)
Button(master, text='Quit', command=master.quit).grid(row=3, sticky=W,
pady=4)
Button(master, text='Show', command=var_states).grid(row=4, sticky=W,
pady=4)
mainloop()
```

4. ASKFLOATFUNCTION

```
import tkinter.messagebox
import tkinter.simpledialog
import tkinter.colorchooser
tkinter.messagebox.showwarning("showwarning", "This is a warning")
tkinter.messagebox.showerror("showerror", "This is an error")
isYes = tkinter.messagebox.askyesno("askyesno", "Continue?")
print(isYes)
isOK = tkinter.messagebox.askokcancel("askokcancel", "OK?")
print(isOK)
isYesNoCancel = tkinter.messagebox.askyesnocancel( "askyesnocancel", "Yes, No, Cancel?")
print(isYesNoCancel)
name = tkinter.simpledialog.askstring( "askstring", "Enter your name")
print(name)
```

```
age = tkinter.simpledialog.askinteger( "askinteger", "Enter your age")
print(age)
weight = tkinter.simpledialog.askfloat( "askfloat", "Enter your weight")
print(weight)
from tkinter import *
def donothing():
filewin = Toplevel(root)
button = Button(filewin, text="Do nothing button")
button.pack()
```

5. MENU

```
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()
```

6. THE PROGRAM DISPLAYS A LINE, AND A RECTANGLE.(CANVAS)

```
from tkinter import *
top = Tk()
C = Canvas(top, bg="blue", height=250, width=300)
line = C.create_line(10,10,200,200,fill='white')
rectangle = C.create_rectangle(20,20,190,90,fill='blue')
C.pack()
top.mainloop()
```

7. THREAD

```
import _thread
import time
```

```
# Define a function for the thread
def print_time( threadName, delay):
        count = 0
        while count < 5:
                time.sleep(delay)
                count += 1
                print ("%s: %s" % ( threadName, time.ctime(time.time()) ))
# Create two threads as follows
try:
        _thread.start_new_thread( print_time, ("Thread-1", 2, ) )
        _thread.start_new_thread( print_time, ("Thread-2", 4, ) )
except:
        print ("Error: unable to start thread")
```

8. Creating Thread Using *Threading* Module

```
import threading
import time
exitFlag = 0
class myThread (threading.Thread):
        def __init__(self, threadID, name, counter):
                threading.Thread.__init__(self)
                self.threadID = threadID
                self.name = name
                self.counter = counter
        def run(self):
                print ("Starting " + self.name)
                print_time(self.name, self.counter, 5)
                print ("Exiting " + self.name)
 def print_time(threadName, delay, counter):
        while counter:
```

```
                if exitFlag:
                        threadName.exit()
                time.sleep(delay)
                print ("%s: %s" % (threadName, time.ctime(time.time())))
                counter -= 1
# Create new threads
thread1 = myThread(1, "Thread-1", 1)
thread2 = myThread(2, "Thread-2", 2)
# Start new Threads
thread1.start()
thread2.start()
thread1.join()
thread2.join()
print ("Exiting Main Thread")
```

9. SYNCHRONIZING THREADS

```
import threading
import time
class myThread (threading.Thread):
def __init__(self, threadID, name, counter):
threading.Thread.__init__(self)
self.threadID = threadID
self.name = name
self.counter = counter
def run(self):
print ("Starting " + self.name)
# Get lock to synchronize threads
threadLock.acquire()
print_time(self.name, self.counter, 3)
```

```
# Free lock to release next thread
threadLock.release()
def print_time(threadName, delay, counter):
while counter:
time.sleep(delay)
print ("%s: %s" % (threadName, time.ctime(time.time())))
counter -= 1
threadLock = threading.Lock()
threads = []
```

10. SERVER

```
import socket
# create a socket object
serversocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# get local machine name
host = socket.gethostname()
port = 9999
# bind to the port
serversocket.bind((host, port))
# queue up to 5 requests
serversocket.listen(5)
 while True:
        # establish a connection
        clientsocket,addr = serversocket.accept()
        print("Got a connection from %s" % str(addr))
        msg='Thank you for connecting'+ "\r\n"
        clientsocket.send(msg.encode('ascii'))
        clientsocket.close()
```

11. CLIENT

```
import socket
# create a socket object
socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# get local machine name
host = socket.gethostname()
port = 9999
# connection to hostname on the port.
s.connect((host, port))
# Receive no more than 1024 bytes
msg = s.recv(1024)
s.close()
print (msg.decode('ascii'))
```