VEMU INSTITUTE OF TECHNOLOGY:: P.KOTHAKOTA

Chittoor-Tirupati National Highway, P.Kothakota, Near Pakala, Chittoor (Dt.), AP - 517112
(Approved by AICTE New Delhi, Permanently Affiliated to JNTUA, Ananthapuramu,
Accredited by NAAC, Recognized Under 2(F) &12(B) of UGC Act, An ISO 9001:2015 Certified Institute)

Department of CSE

II Year B.Tech R19 Regulations

19A05304P PYTHON PROGRAMMING LABORATORY

1. Install Python Interpreter and use it to perform different Mathematical Computations. Try to do all the operations present in a Scientific Calculator

Source code:

```
# This function adds two numbers
def add(x, y):
    return x + y

# This function subtracts two numbers
def subtract(x, y):
    return x - y

# This function multiplies two numbers
def multiply(x, y):
    return x * y

# This function divides two numbers
def divide(x, y):
    return x / y

print("Select operation.")
print("1.Add")
print("2.Subtract")
print("3.Multiply")
print("4.Divide")

while True:
# Take input from the user
    choice = input("Enter choice(1/2/3/4): ")

# Check if choice is one of the four options
    if choice in ('1', '2', '3', '4'):
        num1 = (input("Enter first number: "))
        num2 = (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))
```

```
            break
        else:
            print("Invalid Input")
```

Source code 2:

```
def calculate():
    operation = input('''
Please type in the math operation you would like to complete:
+ for addition
- for subtraction
* for multiplication
/ for division
''')

    number_1 = int(input('Please enter the first number: '))
    number_2 = int(input('Please enter the second number: '))

    if operation == '+':
        print('{} + {} = '.format(number_1, number_2))
        print(number_1 + number_2)

    elif operation == '-':
        print('{} - {} = '.format(number_1, number_2))
        print(number_1 - number_2)

    elif operation == '*':
        print('{} * {} = '.format(number_1, number_2))
        print(number_1 * number_2)

    elif operation == '/':
        print('{} / {} = '.format(number_1, number_2))
        print(number_1 / number_2)

    else:
        print('You have not typed a valid operator, please run the program again.')

# Call calculate() outside of the function
calculate()
```

Output:

Please type in the math operation you would like to complete:

+ for addition

- for subtraction

* for multiplication

/ for division

+

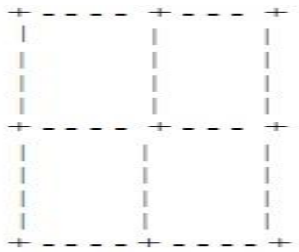
Please enter the first number: 55

Please enter the second number: 65

55 + 65 =

120

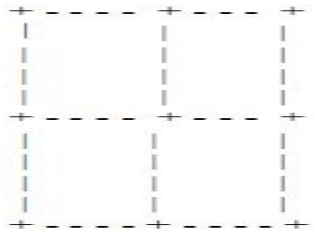
2. Write a function that draws a grid like the following:



Source code:

```
def grid(row, col):
   x = ('+----' * col + '+')
   y = ('\n' +'|    ' *  (col+1))
   return ((x + 3*y) +'\n')*row + x
print(grid(2,2))
```

Output:



3. Write a function that draws a Pyramid with # symbols

```
    #
   # # #
  # # # # #
# # # # # # #
```

Source code:

```
n = 0
r = 5
for m in range(1, r+1):
  for gap in range(1, (r-m)+1):
    print(end=" ")
  while n != (2*m-1):
    print("#", end="")
    n = n + 1
  n = 0
```

```
print()
```

Output:

```
    #
  # # #
# # # # #
# # # # # # #
```

4. Using turtle's concept draw a wheel of your choice

1. Import the turtle module
2. Create a turtle to control
3. Draw around using the turtle methods
4. Run turtle done ()

```
import turtle
bob = turtle.Turtle()
print(bob)
turtle.mainloop()
```

Drawing a square

```
import turtle
t = turtle.Turtle()
t.fd(100)
t.rt(90)
t.fd(100)
t.rt(90)
t.fd(100)
t.rt(90)
t.fd(100)
```

Drawing a circle

```
import turtle
t = turtle.Turtle()
t.circle(60)
```

Drawing a thick dot

```
import turtle
t = turtle.Turtle()
t.dot(20)
```

Changing the Screen Color

```
import turtle
turtle.bgcolor("blue")
```

Changing the Screen Title

```
import turtle
turtle.title("My Turtle Program")
```

Changing the Turtle Size

```
import turtle
t = turtle.Turtle()
t.shapesize(1,5,10)
t.shapesize(10,5,1)
t.shapesize(1,10,5)
t.shapesize(10,1,5)
```

Using for loops Draw a Square

```
# import turtle library
import turtle
my_pen = turtle.Turtle()
for i in range(4):
  my_pen.forward(50)
  my_pen.right(90)
turtle.done()
```

Draw a star

```
# import turtle library
import turtle
my_pen = turtle.Turtle()
for i in range(50):
  my_pen.forward(50)
  my_pen.right(144)
turtle.done()
```

Draw a Hexagon

```
# import turtle library
import turtle
polygon = turtle.Turtle()
my_num_sides = 6
my_side_length = 70
my_angle = 360.0 / my_num_sides
for i in range(my_num_sides):
  polygon.forward(my_side_length)
  polygon.right(my_angle)
turtle.done()
```

Draw a square inside another square box.

```
# import turtle library
import turtle
my_wn = turtle.Screen()
my_wn.bgcolor("light blue")
my_wn.title("Turtle")
my_pen = turtle.Turtle()
my_pen.color("black")
def my_sqrfunc(size):
  for i in range(4):
```

```
            my_pen.fd(size)
            my_pen.left(90)
            size = size - 5
        my_sqrfunc(146)
        my_sqrfunc(126)
        my_sqrfunc(106)
        my_sqrfunc(86)
        my_sqrfunc(66)
        my_sqrfunc(46)
        my_sqrfunc(26)
```

Drawing of another pattern

```
# import turtle library
import turtle
my_wn = turtle.Screen()
turtle.speed(2)
for i in range(30):
  turtle.circle(5*i)
  turtle.circle(-5*i)
  turtle.left(i)
turtle.exitonclick()
```

Drawing of another pattern

```
# import turtle library
import turtle
colors = [ "red","purple","blue","green","orange","yellow"]
my_pen = turtle.Pen()
turtle.bgcolor("black")
for x in range(360):
  my_pen.pencolor(colors[x % 6])
  my_pen.width(x/100 + 1)
  my_pen.forward(x)
  my_pen.left(59)
```

Turtle star using while loop

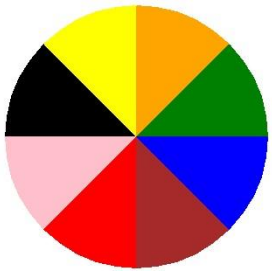
```
from turtle import *
color('red', 'yellow')
begin_fill()
while True:
   forward(200)
   left(170)
   if abs(pos()) < 1:
      break
end_fill()
done()
```

Source code for drawing a wheel:

```
import turtle
```

```
colors =['green','orange','yellow','black','pink','red','brown','blue']

def draw_color_wheel(colors, radius, center=(0, 0)):
    slice_angle = 360 / len(colors)
    heading, position = 90, (center[0] + radius, center[1])
    for color in colors:
        turtle.color(color, color)
        turtle.penup()
        turtle.goto(position)
        turtle.setheading(heading)
        turtle.pendown()
        turtle.begin_fill()
        turtle.circle(radius, extent=slice_angle)
        heading, position = turtle.heading(), turtle.position()
        turtle.penup()
        turtle.goto(center)
        turtle.end_fill()

draw_color_wheel(colors, 150, center=(25, 50))
turtle.hideturtle()
print('done - press any key to exit')
turtle.onkeypress(exit)
turtle.listen()
turtle.done()
```

Output:



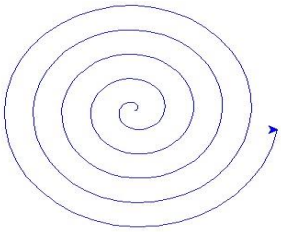
5. Write a program that draws Archimedean Spiral.

Source code:

```
from turtle import *
from math import *
color("blue")
down()
for i in range(200):
```

```
    t = i / 20 * pi
    x = (1 + 5 * t) * cos(t)
    y = (1 + 5 * t) * sin(t)
    goto(x, y)
up()
done()
```

Output:

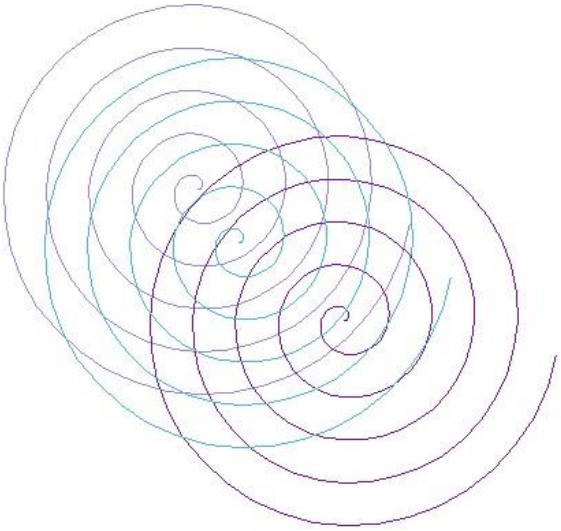


Source Code For multiple Archimedean Spiral

```
from turtle import Turtle, Screen
from math import pi, sin, cos
from random import randint, random

RADIUS = 180  # roughly the radius of a completed spiral

screen = Screen()

WIDTH, HEIGHT = screen.window_width(), screen.window_height()

turtle = Turtle(visible=False)
turtle.speed('fastest')  # because I have no patience

turtle.up()

for _ in range(3):
    x = randint(RADIUS - WIDTH//2, WIDTH//2 - RADIUS)
    y = randint(RADIUS - HEIGHT//2, HEIGHT//2 - RADIUS)
    turtle.goto(x, y)

    turtle.color(random(), random(), random())
    turtle.down()

    for i in range(200):
        t = i / 20 * pi
        dx = (1 + 5 * t) * cos(t)
        dy = (1 + 5 * t) * sin(t)

        turtle.goto(x + dx, y + dy)
```

```
    turtle.up()

screen.exitonclick()
```

Output



6. The letters of the alphabet can be constructed from a moderate number of basic elements,like vertical and horizontal lines and a few curves. Design an alphabet that can be drawnwith a minimal number of basic elements and then write functions that draw the letters.The alphabet can belong to any Natural language excluding English. You should considerat least ten letters of the alphabet.

SOURCE CODE:

```
# Python3 program for above implementation Function to print the string
def printString(str, ch, count):
    occ, i = 0, 0

    # If given count is 0
    # print the given string and return
    if (count == 0):
        print(str)

    # Start traversing the string
    for i in range(len(str)):

        # Increment occ if current char
        # is equal to given character
        if (str[i] == ch):
            occ += 1

        # Break the loop if given character has
        # been occurred given no. of times
        if (occ == count):
            break

    # Print the string after the occurrence
    # of given character given no. of times
    if (i < len(str)- 1):
        print(str[i + 1: len(str) - i + 2])
```

```
    # Otherwise string is empty
    else:
        print("Empty string")

# Driver code
if __name__ == '__main__':
    str = "Python learning"
    printString(str, 'e', 2)
```

OUTPUT

Empty string

7. The time module provides a function, also named time that returns the current Greenwich Mean Time in "the epoch", which is an arbitrary time used as a reference point. On UNIXsystems, the epoch is 1 January 1970.

>>> import time
>>> time.time()
1437746094.5735958

Write a script that reads the current time and converts it to a time of day in hours,minutes, and seconds, plus the number of days since the epoch.

SOURCE CODE:

```
import time
epoch=time.time()

#60*60*24=86400
total_sec = epoch % 86400
#60*60
hours = int(total_sec/3600)
total_minutes = int(total_sec/60)
mins = total_minutes % 60
sec = int(total_sec % 60)

days=int(epoch/86400)

print("The Current time is",hours,':',mins,':',sec)
print("Days since epoch:", days)
```

OUTPUT

The Current time is 16 : 15 : 21
Days since epoch: 18630

8. Given n+r+1 <= 2r . n is the input and r is to be determined. Write a program which computes minimum value of r that satisfies the above.

SOURCE CODE:

```
n = int(input("Enter n Value"))
for r in range(100):
    if((n+r+1)<=(2**r)):
        break
print("minimum value of r is",r)
```

OUTPUT

Enter n Value 5

minimum value of r is 4

9. Write a program that evaluates Ackermann function

SOURCE CODE

```
def A(m, n, s ="% s"):
    print(s % ("A(% d, % d)" % (m, n)))
    if m == 0:
        return n + 1
    if n == 0:
        return A(m - 1, 1, s)
    n2 = A(m, n - 1, s % ("A(% d, %% s)" % (m - 1)))
    return A(m - 1, n2, s)

print(A(1, 2))
```

OUTPUT

```
A( 1,  2)
A( 0, A( 1,  1))
A( 0, A( 0, A( 1,  0)))
A( 0, A( 0, A( 0,  1)))
A( 0, A( 0,  2))
A( 0,  3)
4
```

10. The mathematician Srinivasa Ramanujan found an infinite series that can be used to generate a numerical approximation of 1/ π :
Write a function called estimate_pi that uses this formula to compute and return an estimate of π.

$$\frac{1}{\pi} = \frac{2\sqrt{2}}{9801} \sum_{k=0}^{\infty} \frac{(4k)!(1103 + 26390k)}{(k!)^4 396^{4k}}$$

It should use a while loop to compute terms of the summation until the last term is smaller than 1e-15 (which is Python notation for 10 -15). You can check the result by comparing it to math.pi.

SOURCE CODE

```
import math

def factorial(n):
    if n == 0:
        return 1
    else:
        recurse = factorial(n-1)
        result = n * recurse
        return result

def estimate_pi():
    total = 0
    k = 0
    factor = 2 * math.sqrt(2) / 9801
    keep_going = True
    while keep_going == True:
        numerator = float(factorial(4*k) * (1103 + 26390 * k))
        denominator = float(factorial(k)**4) * 396**(4*k)
```

```
            term = factor * float(numerator/denominator)
            total += term
            k += 1
            if term < 1e-15:
                keep_going == False
                print ("Finished at iteration number", k)
                break
        return 1 / total

    print (estimate_pi())
```

OUTPUT

```
Finished at iteration number 3
3.141592653589793
```

11. Choose any five built-in string functions of C language. Implement them on your own in Python. You should not use string related Python built-in functions.

SOURCE CODE

len()

Looking at the first method, and the easiest one, len() directly returns the length of the passed string object where it is called. The len() method also works on other iterable objects like lists. The below-given code illustrates how we can use the function as well as how it works.

SOURCE CODE

```
#Given string whose length is to be found
str1="Python is great!"
print("The given String is:",str1)

#calling our len() method to calculate the length of str1
print("Length =",len(str1))
```

OUTPUT

```
The given String is: Python is great!
Length = 16
```

ascii()

SOURCE CODE

```
s = 5 #numbers
print(ascii(s))

s = True # boolean
print(ascii(s))

# strings
s = 'abc'
print(ascii(s))

s = 'èvõłvé'
print(ascii(s))
```

OUTPUT

```
5
```

```
True
'abc'
'\xe8v\xf5\u0142v\xe9'
```

ascii() example with list, tuple and dict

SOURCE CODE

```
l = ['æ', 'b', 'č']
print(ascii(l))

t = (1, 'æ', 'b', 'č', 5)
print(ascii(t))

d = {'â':'å', '2':2, 'ç':'ć'}
print(ascii(d))
```

OUTPUT

```
['\xe6', 'b', '\u010d']
(1, '\xe6', 'b', '\u010d', 5)
{'\xe2': '\xe5', '2': 2, '\xe7': '\u0107'}
```

BOOL() EXAMPLE

```
x = True
b = bool(x)

print(type(x))  # <class 'bool'>
print(type(b))  # <class 'bool'>
print(b)  # True

x = False
b = bool(x)
print(b)  # False

x = None
b = bool(x)

print(type(x))  # <class 'NoneType'>
print(type(b))  # <class 'bool'>
print(b)  # False
```

OUTPUT

```
<class 'bool'>
<class 'bool'>
True
False
<class 'NoneType'>
<class 'bool'>
False
```

BOOL() WITH STRINGS

```
# string examples
x = 'True'
b = bool(x)
print(type(x))  # <class 'str'>
print(type(b))  # <class 'bool'>
print(b)  # True
x = 'False'
b = bool(x)
print(b)  # True because len() is used
x = ''
print(bool(x))  # False, len() returns 0
```

OUTPUT

```
<class 'str'>
<class 'bool'>
True
True
False
```

BOOL() WITH NUMBERS

```
from fractions import Fraction
from decimal import Decimal

print(bool(10))  # True
print(bool(10.55))  # True
print(bool(0xF))  # True
print(bool(10 - 4j))  # True

print(bool(0))  # False
print(bool(0.0))  # False
print(bool(0j))  # False
print(bool(Decimal(0)))  # False
print(bool(Fraction(0, 2)))  # False
```

OUTPUT

```
True
True
True
True
False
False
False
False
False
```

BOOL() FUNCTION WITH COLLECTIONS AND SEQUENCES

```
tuple1 = ()
dict1 = {}
list1 = []
print(bool(tuple1))  # False
```

```
print(bool(dict1))  # False
print(bool(list1))  # False
```

OUTPUT

```
False
False
False
```

BYTEARRAY() WITH NO ARGUMENTS

```
b = bytearray()
print(b)
```

OUTPUT:

```
bytearray(b'')
```

BYTEARRAY() WITH STRING AND MUTABILITY

```
# string to bytearray
# encoding is mandatory, otherwise "TypeError: string argument without an encoding"
b = bytearray('abc', 'UTF-8')
print(b)
b[1] = 65  # mutable
print(b)
```

```
# string to bytearray
# encoding is mandatory, otherwise "TypeError: string argument without an encoding"
b = bytearray('abc', 'UTF-8')
print(b)
b[1] = 65  # mutable
print(b)
```

OUTPUT:

```
bytearray(b'abc')
bytearray(b'aAc')
```

BYTEARRAY() WITH INT ARGUMENT

```
b = bytearray(5)
print(b)
```

OUTPUT:

```
bytearray(b'\x00\x00\x00\x00\x00')
```

BYTES() WITH NO ARGUMENTS

```
b = bytes()
print(b)
```

Output: b''

BYTES() WITH STRING AND IMMUTABILITY

```
# string to bytes
# encoding is mandatory, otherwise "TypeError: string argument without an encoding"
b = bytes('abc', 'UTF-8')
print(b)

# Below code will throw error:
# TypeError: 'bytes' object does not support item assignment
# b[1] = 65  # immutable
```

Output: b'abc'

BYTES() WITH INT ARGUMENT

```
# bytes of given size, elements initialized to null
b = bytes(5)
print(b)
```

Output: b'\x00\x00\x00\x00\x00'

BYTES() WITH ITERABLE

```
# bytes from iterable
b = bytes([1, 2, 3])
print(b)
```

Output: b'\x01\x02\x03'

ORD()

```
x = ord('A')
print(x)

print(ord('ć'))
print(ord('ç'))
print(ord('$'))
```

OUTPUT

```
65
263
231
36
```

ENUMERATE()

ENUMERATE LIST

```
# initialize a list of list
data = ['Love', 'Hate', 'Death', 123, ['Alice', 'Bob', 'Trudy']]
```

```
# print the type of variable 'data'
print('The type of data is :', type(data))  # output is 'list'

data = enumerate(data)
# again, print the type of variable 'data'
print('The type of data is now :', type(data))  # output is 'enumerate'
```

OUTPUT

```
The type of data is : <class 'list'>
The type of data is now : <class 'enumerate'>
```

ACCESSING ENUMERATE OBJECT

```
# initialize a list of list
data = ['Love', 'Hate', 'Death', 123, ['Alice', 'Bob', 'Trudy']]
# make an enumerate object
enumObject = enumerate(data)

# access the enumerate object using loop
for element in enumObject:
   print(element)

print('\nStart index is changed to 100:')
# change the start index of the list to 100
enumObject = enumerate(data, 100)

# access the enumerate object using loop
for element in enumObject:
   print(element)
```

OUTPUT

```
(0, 'Love')
(1, 'Hate')
(2, 'Death')
(3, 123)
(4, ['Alice', 'Bob', 'Trudy'])

Start index is changed to 100:
(100, 'Love')
(101, 'Hate')
(102, 'Death')
(103, 123)
(104, ['Alice', 'Bob', 'Trudy'])
```

12. Given a text of characters, write a program which counts number of vowels, consonants And special characters.

SOURCE CODE

```
# Function to count number of vowels,consonant, digits and special character in a string.
```

```
def countCharacterType(str):

    # Declare the variable vowels, consonant, digit and special characters
    vowels = 0
    consonant = 0
    specialChar = 0
    digit = 0

    # str.length() function to count
    # number of character in given string.
    for i in range(0, len(str)):
        ch = str[i]
        if ( (ch >= 'a' and ch <= 'z') or (ch >= 'A' and ch <= 'Z') ):
            # To handle upper case letters
            ch = ch.lower()
            if (ch == 'a' or ch == 'e' or ch == 'i'
                    or ch == 'o' or ch == 'u'):
                vowels += 1
            else:
                consonant += 1

        elif (ch >= '0' and ch <= '9'):
            digit += 1
        else:
            specialChar += 1

    print("Vowels:", vowels)
    print("Consonant:", consonant)
    print("Digit:", digit)
    print("Special Character:", specialChar)

# Driver function.
str = "PYTHON IS GREAT12345"
countCharacterType(str)
```

OUTPUT

```
Vowels: 4
Consonant: 9
Digit: 5
Special Character: 2
```

13. Given a word which is a string of characters. Given an integer say ‘n’, rotate each character by ‘n’ positions and print it. Note that ‘n’ can be positive or negative.

SOURCE CODE

```
# Python program for Left Rotation and Right Rotation of a String
```

```
# In-place rotates s towards left by d
def leftrotate(s, d):
    tmp = s[d : ] + s[0 : d]
    return tmp

# In-place rotates s  # towards right by d
def rightrotate(s, d):

   return leftrotate(s, len(s) - d)

# Driver code
if __name__=="__main__":

    str1 = "Python is great"
    print(leftrotate(str1, 2))

    str2 = "Python is great"
    print(rightrotate(str2, 2))
```

OUTPUT

```
thon is greatPy
atPython is gre
```

14. Given rows of text, write it in the form of columns.

SOURCE CODE

```
# Import pandas package
import pandas as pd

# Define a dictionary containing employee data
data = {'Name':['Jai', 'Princi', 'Gaurav', 'Anuj'],
        'Age':[27, 24, 22, 32],
        'Address':['Delhi', 'Kanpur', 'Allahabad', 'Kannauj'],
        'Qualification':['Msc', 'MA', 'MCA', 'Phd']}

# Convert the dictionary into DataFrame
df = pd.DataFrame(data)

# converting and overwriting values in column
df["Name"]= df["Name"].str.lower()

print(df)
```

OUTPUT

15. Given a page of text. Count the number of occurrences of each latter (Assume case insensitivity and don't consider special characters). Draw a histogram to represent the same

SOURCE CODE

```
# Python3 code to demonstrate   each occurrence frequency using  naive method
```

```
# initializing string
test_str = "Python programming"

# using naive method to get count  of each element in string
all_freq = {}

for i in test_str:
    if i in all_freq:
        all_freq[i] += 1
    else:
        all_freq[i] = 1

# printing result
print ("Count of all characters is :\n "+  str(all_freq))
```

OUTPUT

```
Count of all characters is :
 {'P': 1, 'y': 1, 't': 1, 'h': 1, 'o': 2, 'n': 2, ' ': 1, 'p': 1, 'r': 2, 'g': 2, 'a': 1, 'm': 2, 'i': 1}
```

16. Write program which performs the following operations on list's. Don't use built-in functions

a) Updating elements of a list
b) Concatenation of list's
c) Check for member in the list
d) Insert into the list
e) Sum the elements of the list
f) Push and pop element of list
g) Sorting of list
h) Finding biggest and smallest elements in the list
i) Finding common elements in the list

a) UPDATING ELEMENTS OF A LIST

SOURCE CODE

```
list = ['physics', 'chemistry', 1997, 2000];
print ("Value available at index 2 : ")
print (list[2])
list[2] = 2001;
print ("New value available at index 2 : ")
print (list[2])
```

OUTPUT

```
Value available at index 2 :
1997
New value available at index 2 :
2001
```

b) CONCATENATION OF LIST'S

SOURCE CODE

```
# Initializing lists
test_list1 = [1, 4, 5, 6, 5]
test_list2 = [3, 5, 7, 2, 5]

# using naive method to concat
for i in test_list2 :
```

```
        test_list1.append(i)

    # Printing concatenated list
    print ("Concatenated list using naive method : "+ str(test_list1))
```

OUTPUT

```
Concatenated list using naive method : [1, 4, 5, 6, 5, 3, 5, 7, 2, 5]
```

C) CHECK FOR MEMBER IN THE LIST

SOURCE CODE

```
# Initializing list
test_list = [ 1, 6, 3, 5, 3, 4 ]

print("Checking if 4 exists in list ( using loop ) : ")

# Checking if 4 exists in list using loop
for i in test_list:
    if(i == 4) :
        print ("Element Exists")

print("Checking if 4 exists in list ( using in ) : ")

# Checking if 4 exists in list using in
if (4 in test_list):
    print ("Element Exists")
```

OUTPUT

```
Checking if 4 exists in list ( using loop ) :
Element Exists
Checking if 4 exists in list ( using in ) :
Element Exists
```

d) INSERT INTO THE LIST

SOURCE CODE

```
# Python3 program for use of insert() method

list1 = [ 1, 2, 3, 4, 5, 6, 7 ]

# insert 10 at 4th index
list1.insert(4, 10)
print(list1)

list2 = ['a', 'b', 'c', 'd', 'e']

# insert z at the front of the list
list2.insert(0, 'z')
print(list2)
```

OUTPUT

```
[1, 2, 3, 4, 10, 5, 6, 7]
['z', 'a', 'b', 'c', 'd', 'e']
```

E) SUM THE ELEMENTS OF THE LIST

SOURCE CODE

```
# Python program to find sum of elements in list
total = 0

# creating a list
list1 = [11, 5, 17, 18, 23]

# Iterate each element in list and add them in variale total
for ele in range(0, len(list1)):
    total = total + list1[ele]

# printing total value
print("Sum of all elements in given list: ", total)
```

OUTPUT

```
Sum of all elements in given list:  74
```

f) PUSH AND POP ELEMENT OF LIST

SOURCE CODE

```
# Python code to demonstrate Implementing stack using list
stack = ["Amar", "Akbar", "Anthony"]
stack.append("Ram")
stack.append("Iqbal")
print(stack)

# Removes the last item
print(stack.pop())

print(stack)

# Removes the last item
print(stack.pop())

print(stack)
```

OUTPUT

```
['Amar', 'Akbar', 'Anthony', 'Ram', 'Iqbal']
Iqbal
['Amar', 'Akbar', 'Anthony', 'Ram']
Ram
['Amar', 'Akbar', 'Anthony']
```

g) SORTING OF LIST

SOURCE CODE

```
numbers = [1, 3, 4, 2]

# Sorting list of Integers in ascending
numbers.sort()

print(numbers)
```

OUTPUT

```
[1, 2, 3, 4]
```

SOURCE CODE 2

```
strs = ["zebra", "code", "ide", "practice"]

# Sorting list of Integers in ascending
strs.sort()

print(strs)
```

OUTPUT

```
['code', 'ide', 'practice', 'zebra']
```

h) FINDING BIGGEST AND SMALLEST ELEMENTS IN THE LIST

SOURCE CODE

```
# Python prog to illustrate the following in a list
def find_len(list1):
    length = len(list1)
    list1.sort()
    print("Largest element is:", list1[length-1])
    print("Smallest element is:", list1[0])
    print("Second Largest element is:", list1[length-2])
    print("Second Smallest element is:", list1[1])

# Driver Code
list1=[12, 45, 2, 41, 31, 10, 8, 6, 4]
Largest = find_len(list1)
```

OUTPUT

```
Largest element is: 45
Smallest element is: 2
Second Largest element is: 41
Second Smallest element is: 4
```

i) FINDING COMMON ELEMENTS IN THE LIST

SOURCE CODE

```
# Python program to find the common elements in two lists
def common_member(a, b):
    a_set = set(a)
    b_set = set(b)

    if (a_set & b_set):
        print(a_set & b_set)
    else:
        print("No common elements")


a = [1, 2, 3, 4, 5]
```

```
b = [5, 6, 7, 8, 9]
common_member(a, b)

a = [1, 2, 3, 4, 5]
b = [6, 7, 8, 9]
common_member(a, b)
```

OUTPUT

```
{5}
No common elements
```

SOURCE CODE 2

```
# Python program to find common elements in Both sets using intersection function in
sets function
def common_member(a, b):
    a_set = set(a)
    b_set = set(b)

    # check length
    if len(a_set.intersection(b_set)) > 0:
        return(a_set.intersection(b_set))
    else:
        return("no common elements")


a = [1, 2, 3, 4, 5]
b = [5, 6, 7, 8, 9]
print(common_member(a, b))

a =[1, 2, 3, 4, 5]
b =[6, 7, 8, 9]
print(common_member(a, b))
```

OUTPUT

```
{5}
No common elements
```

17. Write a program that reads a file, breaks each line into words, strips whitespace and punctuation from the words, and converts them to lowercase.

SOURCE CODE

```
file1 = open("myfile.txt","w")
L = ["This is Delhi \n","This is Paris \n","This is London \n"]

# \n is placed to indicate EOL (End of Line)
file1.write("Hello \n")
file1.writelines(L)
file1.close() #to change file access modes

file1 = open("myfile.txt","r+")

print ("Output of Read function is ")
```

```
print (file1.read())
print

# seek(n) takes the file handle to the nth
# bite from the beginning.
file1.seek(0)

print ("Output of Readline function is ")
print (file1.readline())
print

file1.seek(0)

# To show difference between read and readline
print ("Output of Read(9) function is ")
print (file1.read(9))
print

file1.seek(0)

print ("Output of Readline(9) function is ")
print (file1.readline(9))

file1.seek(0)
# readlines function
print ("Output of Readlines function is ")
print (file1.readlines())
print (file1.close())
```

OUTPUT

```
Output of Read function is
Hello
This is Delhi
This is Paris
This is London

Output of Readline function is
Hello

Output of Read(9) function is
Hello
Th
Output of Readline(9) function is
Hello

Output of Readlines function is
['Hello \n', 'This is Delhi \n', 'This is Paris \n', 'This is London \n']
None
```

SOURCE CODE

```
# Python3 code to demonstrate working of removing punctuations in string Using
loop + punctuation string
```

```
# initializing string
test_str = "Gfg, is best : for ! Python ;"

# printing original string
print("The original string is : " + test_str)

# initializing punctuations string
punc = '''!()-[]{};:'"\, <>./?@#$%^&*_~'''

# Removing punctuations in string Using loop + punctuation string
for ele in test_str:
   if ele in punc:
      test_str = test_str.replace(ele, "")

# printing result
print("The string after punctuation filter : " + test_str)
```

OUTPUT

```
The original string is : Gfg, is best : for ! Python ;
The string after punctuation filter : GfgisbestforPython
```

18. Go to Project Gutenberg (http://gutenberg.org) and download your favorite out-of-copyrightbook in plain text format. Read the book you downloaded, skip over the header information atthe beginning of the file, and process the rest of the words as before. Then modify the program tocount the total number of words in the book, and the number of times each word is used. Printthe number of different words used in the book. Compare different books by different authors,written in different eras.

SOURCE CODE

```
f = open('test.txt', 'w')
f.write('apple\n')
f.write('orange\n')
f.write('pear\n')
f.close()    # Always close the file
# Check the contents of the file created

# Open the file created for reading and read line(s) using readline() and readlines()
f = open('test.txt', 'r')
f.readline()        # Read next line into a string
'apple\n'
f.readlines()       # Read all (next) lines into a list of strings
['orange\n', 'pear\n']
f.readline()        # Return an empty string after EOF
''
f.close()

# Open the file for reading and read the entire file via read()
f = open('test.txt', 'r')
f.read()            # Read entire file into a string
'apple\norange\npear\n'
f.close()

# Read line-by-line using readline() in a while-loop
f = open('test.txt')
```

```
line = f.readline()   # include newline
while line:
        line = line.rstrip()  # strip trailing spaces and newline
        # process the line
        print(line)
        line = f.readline()
apple
orange
pear
f.close()
```

OUTPUT

```
apple
orange
pear
Traceback (most recent call last):
  File "C:/Users/BHANU/Documents/Bhanu python/2.py", line 32, in <module>
    apple
NameError: name 'apple' is not defined
```

19. Go to Project Gutenberg (http://gutenberg.org) and download your favorite out-of-copyrightbook in plain text format. Write a program that allows you to replace words, insert words anddelete words from the file.

SOURCE CODE

```
import string, time
def del_punctuation(item):
    '''
        This function deletes punctuation from a word.
    '''
    punctuation = string.punctuation
    for c in item:
        if c in punctuation:
            item = item.replace(c, '')
    return item

def break_into_words():
    '''
        This function reads file, breaks it into
        a list of used words in lower case.
    '''
    book = open('tsawyer.txt')
    words_list = []
    for line in book:
        for item in line.split():
            item = del_punctuation(item)
            item = item.lower()
            #print(item)
            words_list.append(item)
    return words_list

def create_dict():
    '''
        This function calculates words frequency and
```

```
    returns it as a dictionary.
  '''
  words_list = break_into_words()
  dictionary = {}
  for word in words_list:
    if word not in dictionary:
      dictionary[word] = 1
    else:
      dictionary[word] += 1

  return dictionary

dictionary = create_dict()
dictionary.pop('', None)  # accidentally 5 empty strings appeared in the dictionary. why?


print('The total number of words in the book is {}'.format(len(break_into_words())))
print('The number of different words used in the book {}'.format(len(dictionary)))

start_time = time.time()
print('The total number of words in the book is {}'.format(len(break_into_words())))
print('The number of different words used in the book {}'.format(len(dictionary)))
function_time = time.time() - start_time

print('Running time is {0:.4f} s'.format(function_time))
```

OUTPUT

20. Consider all the files on your PC. Write a program which checks for duplicate files in your PC and displays their location. Hint: If two files have the same checksum, they probably have thesame contents.

SOURCE CODE

```
def create(size):
        t1= datetime.now()
        list2 = []   # empty list is created
        list1 = get_drives()
        print ("Drives are \n")
        for d in list1:
                print d," " ,
        print "\nCreating Index..."
        for each in list1:
                process1 = Thread(target=search1, args=(each,size))
                process1.start()
                list2.append(process1)

        for t in list2:
                t.join() # Terminate the threads

        print len(dict1)
        pickle_file = open("mohit.dup1","w")
        cPickle.dump(dict1,pickle_file)
```

```
        pickle_file.close()
        t2= datetime.now()
        total =t2-t1
        print "Time taken to create " , total
def file_open():
        pickle_file  = open("mohit.dup1", "r")
        file_dict = cPickle.load(pickle_file)
        pickle_file.close()
        return file_dict
```

OUTPUT

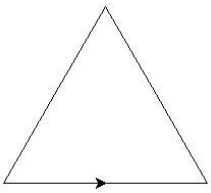
21. Consider turtle object. Write functions to draw triangle, rectangle, polygon, circle and sphere. Use object oriented approach.

TRIANGLE
SOURCE CODE

```
import turtle

# Screen() method to get screen
wn=turtle.Screen()

# creating tess object
tess=turtle.Turtle()

def triangle(x,y):

   # it is used to draw out the pen
   tess.penup()

   # it is used to move curson at x
   # and y position
   tess.goto(x,y)

   # it is used to draw in the pen
   tess.pendown()
   for i in range(3):

      # move cursor 100 unit
      # digit forward
      tess.forward(100)

      # turn cursor 120 degree left
      tess.left(120)

      # Again,move cursor 100 unit
      # digit forward
      tess.forward(100)

# special built in function to send current
# position of cursor to traingle
turtle.onscreenclick(triangle,1)
```

```
turtle.listen()

# hold the screen
turtle.done()
```

OUTPUT



RECTANGLE

SOURCE CODE

```
# draw Rectangle in Python Turtle
import turtle

t = turtle.Turtle()

l = int(input("Enter the length of the Rectangle: "))
w = int(input("Enter the width of the Rectangle: "))

for _ in range(4):

  # drawing length
  if _% 2 == 0:
    t.forward(l) # Forward turtle by l units
    t.left(90) # Turn turtle by 90 degree

  # drawing width
  else:
    t.forward(w) # Forward turtle by w units
    t.left(90) # Turn turtle by 90 degree
```

OUTPUT

Enter the length of the Rectangle: 300
Enter the width of the Rectangle: 50

POLYGON

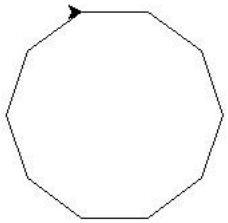
SOURCE CODE

```
# draw any polygon in turtle

import turtle

# creating turtle pen
t = turtle.Turtle()

# taking input for the no of the sides of the polygon
n = int(input("Enter the no of the sides of the polygon : "))
```

```
# taking input for the length of the sides of the polygon
l = int(input("Enter the length of the sides of the polygon : "))

for _ in range(n):
    turtle.forward(l)
    turtle.right(360 / n)
```

OUTPUT

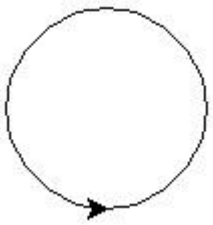
```
Enter the no of the sides of the polygon: 10
Enter the length of the sides of the polygon: 50
```



CIRCLE SOURCE CODE

```
import turtle

# Initializing the turtle
t = turtle.Turtle()

 r = 50
t.circle(r)
```

OUTPUT

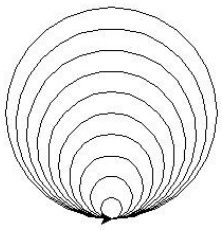


TANGENT CIRCLES

SORCE CODE

```
# Python program to demonstrate tangent circle drawing
  import turtle
 t = turtle.Turtle()
  # radius for smallest circle
r = 10
 # number of circles
n = 10
 # loop for printing tangent circles
for i in range(1, n + 1, 1):
    t.circle(r * i)
```

OUTPUT

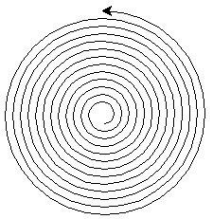


SPIRAL CIRCLE

SOURCE CODE

```
# Python program to demonstrate spiral circle drawing
import turtle
t = turtle.Turtle()
# taking radius of initial radius
r = 10
# Loop for printing spiral circle
for i in range(100):
    t.circle(r + i, 45)
```

OUTPUT



COCENTRIC CIRCLES

SOURCE CODE

```
# Python program to demonstrate concentric circle drawing

import turtle
t = turtle.Turtle()
# radius of the circle
r = 10
# Loop for printing concentric circles
for i in range(50):
    t.circle(r * i)
    t.up()
    t.sety((r * i)*(-1))
    t.down()
```

SPHARE SOURCE CODE

```
# import the turtle modules
import turtle
```

```
# define the function
# for triangle
def form_tri(side):
    for i in range(3):
        my_pen.fd(side)
        my_pen.left(120)
        side -= 10
# Forming the window screen
tut = turtle.Screen()
tut.bgcolor("green")
tut.title("Turtle")

my_pen = turtle.Turtle()
my_pen.color("orange")

tut = turtle.Screen()

# for different shapes
side = 300
for i in range(10):
    form_tri(side)
    side -= 30
```

SOURCE CODE 2

```
# import the turtle modules
import turtle

# define the function
# for square
def form_sq(side):
    for i in range(4):
        my_pen.fd(side)
        my_pen.left(90)
        side -= 5


# Forming the window screen
tut = turtle.Screen()
tut.bgcolor("green")
tut.title("Turtle")

my_pen = turtle.Turtle()
my_pen.color("orange")

tut = turtle.Screen()

# for different shapes
side = 200

for i in range(10):
    form_sq(side)
```

```
side-= 20
```

SOURCE CODE 3

```
# import the turtle modules
import turtle
# define the function
# for hexagon
def form_hex(side):
    for i in range(6):
        my_pen.fd(side)
        my_pen.left(300)
        side -= 2

# Forming the window screen
tut = turtle.Screen()
tut.bgcolor("green")
tut.title("Turtle")

my_pen = turtle.Turtle()
my_pen.color("orange")

tut = turtle.Screen()
# for different sizes
side = 120
for i in range(5):
    form_hex(side)
    side -= 12
```

22. Write a program illustrating the object oriented features supported by Python.

Class is defined under a “Class” Keyword.

class class1(): // class 1 is the name of the class

Objects:
Objects are an instance of a class. It is an entity that has state and behavior. In a nutshell, it is an instance of a class that can access the data.

Syntax: obj = class1()

Here obj is the “object “ of class1.

Creating an Object and Class in python:
Example:

```
class employee():
    def __init__(self,name,age,id,salary):   #creating a function
        self.name = name # self is an instance of a class
        self.age = age
        self.salary = salary
        self.id = id
```

```
emp1 = employee("harshit",22,1000,1234) #creating objects
emp2 = employee("arjun",23,2000,2234)
print(emp1.__dict__)#Prints dictionary
```

Output: {'name': 'harshit', 'age': 22, 'salary': 1234, 'id': 1000}

INHERITANCE:

Inheritance is defined as the capability of one class to derive or inherit the properties from some other class and use it whenever needed. Inheritance provides the following properties:

- It represents real-world relationships well.
- It provides reusability of code. We don't have to write the same code again and again. Also, it allows us to add more features to a class without modifying it.
- It is transitive in nature, which means that if class B inherits from another class A, then all the subclasses of B would automatically inherit from class A.

```
# Base class or Parent class
class Child:
    # Constructor
    def __init__(self, name):
        self.name = name
    # To get name
    def getName(self):
        return self.name
    # To check if this person is student
    def isStudent(self):
        return False
# Derived class or Child class
class Student(Child):
    # True is returned
    def isStudent(self):
        return True
 # Driver code
# An Object of Child
std = Child("Ram")
print(std.getName(), std.isStudent())
# An Object of Student
std = Student("Shivam")
print(std.getName(), std.isStudent())
```

Output:

```
Ram False
Shivam True
```

Types of Inheritance depends upon the number of child and parent classes involved. There are four types of inheritance in Python:

Single Inheritance: Single inheritance enables a derived class to inherit properties from a single parent class, thus enabling code reusability and the addition of new features to existing code.

SOURCE CODE

```
# Base class
class Parent:
    def func1(self):
        print("This function is in parent class.")

# Derived class
class Child(Parent):
    def func2(self):
        print("This function is in child class.")

# Driver's code
object = Child()
object.func1()
object.func2()
```

OUTPUT:

```
This function is in parent class.
This function is in child class.
```

Multiple Inheritance: When a class can be derived from more than one base class this type of inheritance is called multiple inheritance. In multiple inheritance, all the features of the base classes are inherited into the derived class.

SOURCE CODE

```
# Base class1
class Mother:
    mothername = ""
    def mother(self):
        print(self.mothername)

# Base class2
class Father:
    fathername = ""
    def father(self):
        print(self.fathername)

# Derived class
class Son(Mother, Father):
    def parents(self):
        print("Father :", self.fathername)
```

```
        print("Mother :", self.mothername)

# Driver's code
s1 = Son()
s1.fathername = "RAM"
s1.mothername = "SITA"
s1.parents()
```

OUTPUT:

```
Father : RAM
Mother : SITA
```

Multilevel Inheritance

In multilevel inheritance, features of the base class and the derived class are further inherited into the new derived class. This is similar to a relationship representing a child and grandfather.

SOURCE CODE

```
# Base class
class Grandfather:

    def __init__(self, grandfathername):
        self.grandfathername = grandfathername

# Intermediate class
class Father(Grandfather):
    def __init__(self, fathername, grandfathername):
        self.fathername = fathername

        # invoking constructor of Grandfather class
        Grandfather.__init__(self, grandfathername)

# Derived class
class Son(Father):
    def __init__(self,sonname, fathername, grandfathername):
        self.sonname = sonname

        # invoking constructor of Father class
        Father.__init__(self, fathername, grandfathername)

    def print_name(self):
        print('Grandfather name :', self.grandfathername)
        print("Father name :", self.fathername)
        print("Son name :", self.sonname)

#  Driver code
s1 = Son('Prince', 'Rampal', 'Lal mani')
```

```
print(s1.grandfathername)
s1.print_name()
```

OUTPUT:

```
Lal mani
Grandfather name : Lal mani
Father name : Rampal
Son name : Prince
```

Hierarchical Inheritance: When more than one derived classes are created from a single base this type of inheritance is called hierarchical inheritance. In this program, we have a parent (base) class and two child (derived) classes.

SOURCE CODE

```
# Base class
class Parent:
    def func1(self):
        print("This function is in parent class.")

# Derived class1
class Child1(Parent):
    def func2(self):
        print("This function is in child 1.")

# Derivied class2
class Child2(Parent):
    def func3(self):
        print("This function is in child 2.")

# Driver's code
object1 = Child1()
object2 = Child2()
object1.func1()
object1.func2()
object2.func1()
object2.func3()
```

OUTPUT:

```
This function is in parent class.
This function is in child 1.
This function is in parent class.
This function is in child 2.
```

Hybrid Inheritance: Inheritance consisting of multiple types of inheritance is called hybrid inheritance.

SOURCE CODE

```
class School:
    def func1(self):
        print("This function is in school.")

class Student1(School):
    def func2(self):
        print("This function is in student 1. ")

class Student2(School):
    def func3(self):
        print("This function is in student 2.")

class Student3(Student1, School):
    def func4(self):
        print("This function is in student 3.")

# Driver's code
object = Student3()
object.func1()
object.func2()
```

OUTPUT:

```
This function is in school.
This function is in student 1.
```

POLYMORPHISM

What is Polymorphism: The word polymorphism means having many forms. In programming, polymorphism means same function name (but different signatures) being uses for different types.

SOURCE CODE

```
# Python program to demonstrate in-built polymorphic functions

# len() being used for a string
print(len("python"))

# len() being used for a list
print(len([10, 20, 30]))
```

OUTPUT

```
6
3
```

Examples of used defined polymorphic functions:

A simple Python function to demonstrate Polymorphism

SOURCE CODE

```
def add(x, y, z = 0):
    return x + y+z

# Driver code
print(add(2, 3))
print(add(2, 3, 4))
```

OUTPUT

```
5
9
```

POLYMORPHISM WITH CLASS METHODS:

Below code shows how python can use two different class types, in the same way. We create a for loop that iterates through a tuple of objects. Then call the methods without being concerned about which class type each object is. We assume that these methods actually exist in each class.

SOURCE CODE

```
class India():
    def capital(self):
        print("New Delhi is the capital of India.")

    def language(self):
        print("Hindi is the most widely spoken language of India.")

    def type(self):
        print("India is a developing country.")

class USA():
    def capital(self):
        print("Washington, D.C. is the capital of USA.")

    def language(self):
        print("English is the primary language of USA.")

    def type(self):
        print("USA is a developed country.")

obj_ind = India()
obj_usa = USA()
for country in (obj_ind, obj_usa):
    country.capital()
    country.language()
```

```
country.type()
```

OUTPUT

```
New Delhi is the capital of India.
Hindi is the most widely spoken language of India.
India is a developing country.
Washington, D.C. is the capital of USA.
English is the primary language of USA.
USA is a developed country.
```

POLYMORPHISM WITH INHERITANCE:

In Python, Polymorphism lets us define methods in the child class that have the same name as the methods in the parent class. In inheritance, the child class inherits the methods from the parent class. However, it is possible to modify a method in a child class that it has inherited from the parent class. This is particularly useful in cases where the method inherited from the parent class doesn't quite fit the child class. In such cases, we re-implement the method in the child class. This process of re-implementing a method in the child class is known as Method Overriding.

SOURCE CODE:

```
class Bird:
  def intro(self):
    print("There are many types of birds.")

  def flight(self):
    print("Most of the birds can fly but some cannot.")

class sparrow(Bird):
  def flight(self):
    print("Sparrows can fly.")

class ostrich(Bird):
  def flight(self):
    print("Ostriches cannot fly.")

obj_bird = Bird()
obj_spr = sparrow()
obj_ost = ostrich()

obj_bird.intro()
obj_bird.flight()

obj_spr.intro()
obj_spr.flight()

obj_ost.intro()
```

```
obj_ost.flight()
```

OUTPUT:

```
There are many types of birds.
Most of the birds can fly but some cannot.
There are many types of birds.
Sparrows can fly.
There are many types of birds.
Ostriches cannot fly.
```

POLYMORPHISM WITH A FUNCTION AND OBJECTS:

It is also possible to create a function that can take any object, allowing for polymorphism. In this example, let's create a function called "func()" which will take an object which we will name "obj". Though we are using the name 'obj', any instantiated object will be able to be called into this function. Next, lets give the function something to do that uses the 'obj' object we passed to it. In this case lets call the three methods, viz., capital(), language() and type(), each of which is defined in the two classes 'India' and 'USA'. Next, let's create instantiations of both the 'India' and 'USA' classes if we don't have them already. With those, we can call their action using the same func() function:

SOURCE CODE:

```
class India():
    def capital(self):
        print("New Delhi is the capital of India.")

    def language(self):
        print("Hindi is the most widely spoken language of India.")

    def type(self):
        print("India is a developing country.")

class USA():
    def capital(self):
        print("Washington, D.C. is the capital of USA.")

    def language(self):
        print("English is the primary language of USA.")

    def type(self):
        print("USA is a developed country.")

def func(obj):
    obj.capital()
    obj.language()
    obj.type()

obj_ind = India()
obj_usa = USA()
```

```
func(obj_ind)
func(obj_usa)
```

OUTPUT:

```
New Delhi is the capital of India.
Hindi is the most widely spoken language of India.
India is a developing country.
Washington, D.C. is the capital of USA.
English is the primary language of USA.
USA is a developed country.
```

ENCAPSULATION
SOURCE CODE

```
# Creating a Base class
class Base:
    def __init__(self):
        self.a = "python programming"
        self.__c = "python programming"

# Creating a derived class
class Derived(Base):
    def __init__(self):

        # Calling constructor of
        # Base class
        Base.__init__(self)
        print("Calling private member of base class: ")
        print(self.__c)
# Driver code
obj1 = Base()
print(obj1.a)
```

OUTPUT

```
python programming
```

ABSTRACT CLASSES OR ABSTRACTION
SOURCE CODE 1:

```
# Python program showing abstract base class work

from abc import ABC, abstractmethod

class Polygon(ABC):
```

```
    # abstract method
    def noofsides(self):
        pass

class Triangle(Polygon):

    # overriding abstract method
    def noofsides(self):
        print("I have 3 sides")

class Pentagon(Polygon):

    # overriding abstract method
    def noofsides(self):
        print("I have 5 sides")

class Hexagon(Polygon):

    # overriding abstract method
    def noofsides(self):
        print("I have 6 sides")

class Quadrilateral(Polygon):

    # overriding abstract method
    def noofsides(self):
        print("I have 4 sides")

# Driver code
R = Triangle()
R.noofsides()

K = Quadrilateral()
K.noofsides()

R = Pentagon()
R.noofsides()

K = Hexagon()
K.noofsides()
```

OUTPUT:

```
I have 3 sides
I have 4 sides
I have 5 sides
I have 6 sides
```

SOURCE CODE 2:

```
# Python program showing  abstract base class work

from abc import ABC, abstractmethod
class Animal(ABC):

    def move(self):
        pass

class Human(Animal):

    def move(self):
        print("I can walk and run")

class Snake(Animal):

    def move(self):
        print("I can crawl")

class Dog(Animal):

    def move(self):
        print("I can bark")

class Lion(Animal):

    def move(self):
        print("I can roar")

# Driver code
R = Human()
R.move()

K = Snake()
K.move()

R = Dog()
R.move()

K = Lion()
K.move()
```

OUTPUT:

```
I can walk and run
```

I can crawl
I can bark
I can roar

IMPLEMENTATION THROUGH SUBCLASSING:

SOURCE CODE

```
# Python program showing implementation of abstract class through sub classing

import abc

class parent:
   def geeks(self):
      pass

class child(parent):
   def geeks(self):
      print("child class")

# Driver code
print( issubclass(child, parent))
print( isinstance(child(), parent))
```

OUTPUT:

True
True

CONCRETE METHODS IN ABSTRACT BASE CLASSES:

SOURCE CODE

```
# Python program invoking a   method using super()

import abc
from abc import ABC, abstractmethod

class R(ABC):
   def rk(self):
      print("Abstract Base Class")

class K(R):
   def rk(self):
      super().rk()
      print("subclass ")

# Driver code
r = K()
r.rk()
```

OUTPUT:

Abstract Base Class
Subclass

ABSTRACT PROPERTIES:

SOURCE CODE

```
# Python program showing  abstract properties

import abc
from abc import ABC, abstractmethod

class parent(ABC):
    @abc.abstractproperty
    def python(self):
        return "parent class"
class child(parent):

    @property
    def python(self):
        return "child class"

try:
    r =parent()
    print( r.geeks)
except Exception as err:
    print (err)

r = child()
print (r.python)
```

OUTPUT:

Can't instantiate abstract class parent with abstract methods python
child class

ABSTRACT CLASS INSTANTIATION:

SOURCE CODE

```
# Python program showing abstract class cannot be an instantiation
from abc import ABC,abstractmethod

class Animal(ABC):
    @abstractmethod
    def move(self):
        pass
class Human(Animal):
    def move(self):
        print("I can walk and run")

class Snake(Animal):
```

```
    def move(self):
        print("I can crawl")

class Dog(Animal):
    def move(self):
        print("I can bark")

class Lion(Animal):
    def move(self):
        print("I can roar")

c=Animal()
```

OUTPUT:

```
Traceback (most recent call last):
  File "/home/ffe4267d930f204512b7f501bb1bc489.py", line 19, in
    c=Animal()
TypeError: Can't instantiate abstract class Animal with abstract methods move
```

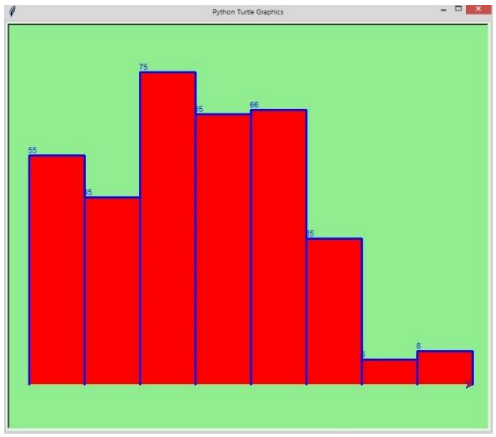
23. Design a Python script using the Turtle graphics library to construct a turtle bar chart representing the grades obtained by N students read from a file categorising them into distinction, first class, second class, third class and failed.

SOURCE CODE:

```
import turtle
def drawBar(t, height):
    """ Get turtle t to draw one bar, of height. """
    t.begin_fill()              # start filling this shape
    t.left(90)
    t.forward(height)
    t.write(str(height))
    t.right(90)
    t.forward(40)
    t.right(90)
    t.forward(height)
    t.left(90)
    t.end_fill()              # stop filling this shape

xs = [55,45,75,65,66,35,6,8]  # here is the data
maxheight = max(xs)
numbars = len(xs)
border = 10

wn = turtle.Screen()            # Set up the window and its attributes
wn.setworldcoordinates(0-border, 0-border, 40*numbars+border, maxheight+border)
wn.bgcolor("lightgreen")

tess = turtle.Turtle()          # create tess and set some attributes
tess.color("blue")
tess.fillcolor("red")
tess.pensize(3)
```

```
for a in xs:
    drawBar(tess, a)

wn.exitonclick()
```

OUTPUT



24. Design a Python script to determine the difference in date for given two dates in YYYY:MM:DD format(0 <= YYYY <= 9999, 1 <= MM <= 12, 1 <= DD <= 31) following the leap year rules.

AIM:

SOURCE CODE:

```
import datetime
def date_validation(year,month,day ):

    isValidDate = True

    try :
        datetime.datetime(int(year),
                          int(month), int(day))

    except ValueError :
        isValidDate = False

    if(isValidDate) :
        print ("Yes")
    else :
        print ("No")

date_validation(2000,12,12)
date_validation(2000,11,31)
```

OUTPUT

```
Yes
No
```

25. Design a Python Script to determine the time difference between two given times in HH:MM:SS format.(0 <= HH <= 23, 0 <= MM <= 59, 0 <= SS <= 59)

Aim: To find the difference between two given times using python programming.

Source Code:

```
def removeColon(s):

    if (len(s) == 4):
        s = s[:1] + s[2:]

    if (len(s) == 5):
        s = s[:2] + s[3:]

    return int(s)

# Main function which finds difference
def diff(s1, s2):

    # Change string
    # (eg. 2:21 --> 221, 00:23 --> 23)
    time1 = removeColon(s1)
    time2 = removeColon(s2)

    # Difference between hours
    hourDiff = time2 // 100 - time1 // 100 - 1;

    # Difference between minutes
    minDiff = time2 % 100 + (60 - time1 % 100)

    if (minDiff >= 60):
        hourDiff += 1
        minDiff = minDiff - 60

    # Convert answer again in string with ':'
    res = str(hourDiff) + ':' + str(minDiff)

    return res

# Driver code
s1 = "14:00"
s2 = "18:45"

print(diff(s1, s2))
```

OUTPUT

```
4:45
```