# CSCP409

# Python Programming Laboratory

## LIST of EXERCISES

**Write Python Program for the following:**

1. Generation of Prime Numbers
2. Solving Quadratic Equation
3. String Operations
4. Largest of n Numbers
5. Matrix Multiplication
6. Class and Objects
7. Single and Multiple Inheritance
8. Operator Overloading
9. File Handling
10. Exception Handling
11. Send and Receive Messages using UDP
12. File Transfer using File Transfer Protocol
13. Loan Calculator using Tkinter
14. Popup Menu based Arithmetic Operations using Tkinter
15. Display an Image and its Transformation using Tkinter and OpenCV-Python

**Additional Exercises**

1. Factorial of an Integer
2. Checking for Armstrong Number
3. Counting Number of Positive, Negative and Zero Elements in a List/Array
4. Swapping two integers using Function
5. Fibonacci Sequence using Recursion

**

# 1. Generation of Prime Numbers

**Aim:** To generate prime numbers within an interval using for and while statements.

**Algorithm:**

A prime number is an integer number greater than 1 whose only factors are 1 and itself. The first few prime numbers are 2, 3, 5, 7, 11, 13, 17, 19, 23 and 29.

To check whether a given number (num) is a prime number, first divide it by 2. If the result is an integer number (i.e., num is divisible by 2 or remainder of num/2 is zero or num % 2 is 0) then num is not a prime number. If not, try to divide it by prime numbers 3, 5, 7, 11, … , up to num-1 or divide it by 3, 4, 5, …. , up to num-1. If num is not divisible up to num-1 then num is a prime number.

**a) Coding: (using for statement)**

```
# To display all the prime numbers within a given interval [Lower, Upper]

lower = int(input("Enter Lower Limit: "))
upper = int(input("Enter Upper Limit: "))

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)
```

**Result:**

```
Enter Lower Limit: 20
Enter Upper Limit: 50
Prime numbers between 20 and 50 are:
23
29
```

```
31
37
41
43
47
>>>
```

**b) Coding: (using while statement)**

```
# Python program to display all the prime numbers within a given interval [Lower, Upper] using
while statement


lower = int(input("Enter Lower Limit: "))
upper = int(input("Enter Upper Limit: "))

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

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

**Result:**

```
Enter Lower Limit: 50
Enter Upper Limit: 100
Prime numbers between 50 and 100 are:
53
59
61
67
71
73
79
83
89
97
>>>
```

# 2. Solving Quadratic Equation

**Aim:** To find the roots of a quadratic equation given the coefficients a, b and c.

**Algorithm:**

In algebra, a quadratic equation is any equation of the form $ax^2 + bx + c = 0$, where $x$ is unknown and $a$, $b$, and $c$ are known numbers. The numbers $a$, $b$, and $c$ are the coefficients of the equation.

Examples: $x^2 + 5x + 6 = 0$, $4x^2 + 5x + 8 = 0$.

The values of x that satisfy the equation are called solutions or roots of the equation. A quadratic equation has always two roots. It can be found by computing $d = b^2 - 4ac$. The value of d is zero or +ve or -ve.

i) If d = 0 then then the roots are real and equal. They are root1 = -b/(2a) and root2 = -b/(2a).

ii) If d > 0 then the roots are real and different. They are root1=-b+√d/(2a) and root2=-b-√d/(2a).

iii) If d < 0 then the roots are complex or imaginary. Roots are of the form p + iq and p - iq, where p is the real part of the root and q is the imaginary part of the root. The real part of the root is p = -b/(2a) and the imaginary part is q = √-d/(2a).

**Coding:**

```
# Finding the roots of quadratic equation ax2 + bx + c = 0

# import math and complex math modules

import math,cmath

a,b,c= (input("Enter a, b and c: ")).split()
a,b,c =[int(a),int(b),int(c)]

d = (b**2) - (4*a*c)

if d==0:
   sol1 = -b/(2*a)
   sol2 = -b/(2*a)
   print("Roots are real and equal")

elif d>0:
   sol1 = (-b-math.sqrt(d))/(2*a)
   sol2 = (-b+math.sqrt(d))/(2*a)
```

```
    print("Roots are real and different")

elif d<0:
    sol1 = (-b-cmath.sqrt(d))/(2*a)
    sol2 = (-b+cmath.sqrt(d))/(2*a)
    print("Roots are imaginary")

print('{0} and {1}'.format(sol1,sol2))
```

**Result:**

```
Enter a, b and c: 1 4 4
Roots are real and equal
-2.0 and -2.0
>>>
Enter a, b and c: 1 5 6
Roots are real and different
-3.0 and -2.0
>>>

Enter a, b and c: 1 2 3
Roots are imaginary
(-1-1.4142135623730951j) and (-1+1.4142135623730951j)
>>>
```

# 3. String Operations

**Aim:** To perform the string operations using user defined functions.

**Algorithm:**

A string is a data type used in programming, such as an integer and floating point numbers, but it is used to represent text rather than numbers. It is comprised of a set of characters. Character array is used to store a string.
Examples: “Annamalai”, “2345”

The main operations on string are: length of a string (counting number of characters in the string), reversing a string, concatenating two strings together and comparing two strings. In Python, the arithmetic operator ‘+’ is used for string concatenation and the relational operators ‘<’, ‘<=’, ‘>’, ‘>=’ ‘==” are used for string comparison. In Python, a function is defined using the def keyword.

**Coding:**

```
# String Operations: String Length, String Reverse, String Concatenation and String Comparison

def strlen(str):
    counter = 0
    while str[counter:]:
        counter += 1
    return counter

def strrev(str):
     rstr=""
     l=strlen(str)
     while l>0:
          rstr = rstr + str[l-1]
          l=l-1
     return rstr

def strcat(st1,st2):
     return(st1+st2)

def strcmp(st1,st2):
     if(st1==st2):
         print(st1 + " and " + st2 + " are same")
     elif (st1>st2):
         print(st1 + " comes after " + st2 +" in the Dictionary")
     else:
         print(st1 + " comes before " + st2 +" in the Dictionary")
print("String Functions:\n    1. String Length \n    2. String Reverse \n    3. String
Concatenation\n    4. String Comparison\n")
n=int(input("Enter your Choice: "))
if(n==1):
    str = input("Enter a String: ")
    print("Length of the string is:",strlen(str))
elif (n==2):
    str = input("Enter a String: ")
    print("Reversed String is:", strrev(str))
elif (n==3):
    str1 = input("Enter the first String: ")
    str2 = input("Enter the second String: ")
    print("Concatenated string is:", strcat(str1,str2))
elif (n==4):
    str1 = input("Enter the first String: ")
    str2 = input("Enter the second String: ")
    strcmp(str1,str2)
else:
    print("Invalid Choice")
```

**Result:**

```
String Functions:
   1. String Length
   2. String Reverse
   3. String Concatenation
   4. String Comparison

Enter your Choice: 1
Enter a String: annamalai
Length of the string is: 9
>>>
String Functions:
   1. String Length
   2. String Reverse
   3. String Concatenation
   4. String Comparison

Enter your Choice: 2
Enter a String: university
Reversed String is: ytisrevinu

>>>
String Functions:
   1. String Length
   2. String Reverse
   3. String Concatenation
   4. String Comparison

Enter your Choice: 3
Enter the first String: annamalai
Enter the second String: university
Concatenated string is: annamalaiuniversity
>>>

String Functions:
   1. String Length
   2. String Reverse
   3. String Concatenation
   4. String Comparison

Enter your Choice: 4
Enter the first String: anna
Enter the second String: malai
anna comes before malai in the Dictionary
>>>
```

# 4. Largest of N Numbers

**Aim:** To find largest of n given numbers using list and function.

**Algorithm:**

A list is a data structure in Python. List contains ordered sequence of elements and it can be changeable. Python list is like an array in other languages. List need not have homogeneous (same data type) items or elements which makes it a most powerful data structure in Python. A single list may contain data types like integers, floating numbers, strings etc.,

**Creating List:** Initially, an empty list is created. The n elements are added to the list one by one using append() built-in function.

**Function:** In Python a function is defined using the def keyword. To find largest of n given numbers, the first number is assumed to be the largest. The second number is compared with the largest. If the second number is greater than the largest then the second number is made as the current largest number. This process is repeated for all other remaining numbers in the list to find the largest number. In Python, built-in function max() can also be used to find largest of n numbers (i.e., largest=max(lst))

**Coding:**

```
# Largest of n numbers using List and Function

def lar(lst,n):
    l=lst[0]
    for i in range(1,n):
        if (lst[i]>l):
            l=lst[i]
    return l


lst=[]
n=int(input("Enter total number of elements: "))
print("Enter the elements one by one: ")
for i in range(n):
    num=float(input())
    lst.append(num)
print("The given elements are:")
print(lst)
print("Largest number is",lar(lst,n))
```

**Result:**

```
Enter total number of elements: 5
Enter the elements one by one:
24
78
98
12
45
The given elements are:
[24.0, 78.0, 98.0, 12.0, 45.0]
Largest number is 98.0
>>>
```

# 5. Matrix Multiplication

**Aim:** To multiply two matrices of order m x n and n x p using list/array and function.

**Algorithm:**

An array is a data structure and it is used to store same type (data type) of elements/values. In python, list can contain elements of different data types.

**Creating List:** Initially, an empty list is created. The Matrix A elements (m x n elements) are added to the list one by one using append() built-in function.

**Creating Array:** The storage requirement for an array need to specified. The Matrix B elements ( n x p elements) are read one by one and stored into the array.

**Matrix Multiplication Algorithm:** Matrix A has m x n elements and matrix B has n x p elements. The result of A x B is a matrix C of order m x p. Each element of the matrix C is initialized to zero and it is computed as given below.

```
for i = 0 to m-1
for j = 0 to p-1
  begin
      c[i][j] = 0
      for k = 0 to n-1
          begin
               c[i][j] = c[i][j] + a[i][k] * b [k][j]
          end
    end
```

**Coding:**

```
# Matrix Multiplication: Matrix A is of order (m x n) and Matrix B is of order (n x p),
# Function for matrix multiplication

def matmul(a,b,m,n,p):
    c=[[0 for j in range  (0,p)]  for i in range (0,m)]
    for i in range(m):
        for j in range(p):
            c[i][j]=0
            for k in range(n):
                c[i][j]=c[i][j]+a[i][k]*b[k][j]
    return c
```

```
m=int(input("Enter number of rows of the first matrix: "))
n=int(input("Enter number of columns of the first matrix: "))
p=int(input("Enter number of columns of the second matrix: "))

# Reading matrix A using List

a=[]
print("Enter the elements of the first matrix (row-order): ")
for i in range(m):
   r=[]
   for j in range(n):
       r.append(int(input()))
   a.append(r)

# Initializing two dimensional array and reading matrix B

b=[[0 for j in range  (0,p)]  for i in range (0,n)]
print("Enter the elements of the second matrix (row-order): ")
for i in range(n):
   for j in range(p):
     b[i][j]=int(input())

print("Matrix A is:")
print(a)
print("Matrix B is:")
print(b)
print ("Resultant matrix is:")
print(matmul(a,b,m,n,p))
```
**Result:**
```
Enter number of rows of the first matrix: 2
Enter number of columns of the first matrix: 2
Enter number of columns of the second matrix: 2
Enter the elements of the first matrix (row-order):
1
2
3
4
Enter the elements of the second matrix (row-order):
1
1
1
1
Matrix A is:
[[1, 2], [3, 4]]
Matrix B is:
[[1, 1], [1, 1]]
```

```
Resultant matrix is:
[[3, 3], [7, 7]]
>>>
```

# 6. Class and Objects

**Aim:** To define a class with constructor and create objects.

**Class and Objects:**

A class in C++/Java/Python is the building block that leads to object-oriented programming. It is an user defined data-type which has data members and member functions. Data members are the data variables and member functions are the functions used to manipulate these variables. A class is defined in python using the keyword class followed by name of the class.

An object is an instance of a class. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) then the memory is allocated.

The __init__() function in the class is executed automatically every time the class is being used to create a new object. It is called as a constructor in object oriented terminology. This function is used to initialize the data members of the class.

In most of the object-oriented programming (OOP) languages access specifiers are used to limit the access to the variables and functions of a class. Most of the OOP languages use three types of access specifiers, they are: private, public and protected. In Python, all the variables and member functions of a class are public by default. Adding a prefix __ (double underscore) to the member variable or function makes them to be private.

**Coding:**

```
# To find the Euclidean distance between two pints in a three dimensional space using
#     class and objects.

# Class called point consists of  a constructor (__init__())  and two functions
#     distancefromorgin() and distance()

import math
class point():
   def __init__(self,a,b,c):
      self.x=a
      self.y=b
      self.z=c


   def distancefromorigin(self):
      return ((self.x ** 2) + (self.y ** 2) +(self.z ** 2)) ** 0.5
```

```
    def distance(self, point2):
        xdiff = self.x-point2.x
        ydiff = self.y-point2.y
        zdiff = self.z-point2.z
        dist = math.sqrt(xdiff**2 + ydiff**2+ zdiff**2)
        return dist

x1,y1,z1= (input("Enter the coordinates of a first point P1(x1,y1,z1): ")).split()
x1,y1,z1 =[int(x1),int(y1),int(z1)]
x2,y2,z2= (input("Enter the coordinates of a second point P2(x2,y2,z2): ")).split()
x2,y2,z2 =[int(x2),int(y2),int(z2)]

# P1 and P2 are  objects  of point class

p1 = point(x1,y1,z1)
p2 = point(x2,y2,z2)
print('Distance from origin to P1:', p1.distancefromorigin())
print('Distance from origin to P2:', p2.distancefromorigin())
print('Distance from P1 to P2:',p1.distance(p2))
```

**Result:**

```
Enter the coordinates of a first point P1(x1,y1,z1): 1 2 3
1 2 3
Enter the coordinates of a second point P2(x2,y2,z2): 2 3 4
2 3 4
Distance from origin to P1: 3.7416573867739413
Distance from origin to P2: 5.385164807134504
Distance from P1 to P2: 1.7320508075688772
>>>
```

# 7. Single and Multiple Inheritance

**Aim:** To define a new class from one or more existing classes.

**Inheritance:**

Inheritance is a mechanism in which one class (derived class) acquires the property of another class (base class). With inheritance, we can reuse the variables and methods of the existing class. The existing class is called base class and the new class is called derived class. Hence, inheritance facilitates reusability and is an important concept of object oriented programming. Types of inheritance are: single inheritance, multiple inheritance, multi-level inheritance, hierarchical inheritance and hybrid inheritance.

Single inheritance enables a derived class to use the variables and functions defined in an existing class. In multiple inheritance, derived class inherits the characteristics and features from more than one existing classes.

In python, syntax for defining single inheritance is class **z(x)**, where x is the name of the base class and z is the name of the derived class. Similarly, multiple inheritance is defined using the syntax **class z(x, y)**, where x and y are the names of base classes and z is the name of the derived class.

**Coding:**

```
# person is a base class

class person:

    def __init__(self, n, a):

        self.name = n

        self.age = a



# employee is the class derived from person using single inheritance

class employee(person):

    def __init__(self, n,a, d,s):

        person.__init__(self,n,a)
        self.designation=d

        self.salary=s


     def show(self):

        print("Employee Details: ")

        print("  Name: ",self.name,"\n  Age:",self.age, "\n  Designation:",self.designation, "\n
Salary:",self.salary)


#  student is a  base class

class student:

    def __init__(self, id, rno):

        self.studentId = id

        self.roomno=rno


#  resident is a class derived from person and student using multiple inheritance

class resident(person, student):

    def __init__(self, n, a, id,rno):

        person.__init__(self, n, a)

        student.__init__(self, id,rno)


    def show(self):

        print("Resident Details:")

        print("  Name:", self.name,"\n  Age: ",self.age, "\n  Id:" ,self.studentId,"\n  Room no.:",self.roomno)


  # Creating objects of employee and resident classes

e1 =employee("Arun",35,"Data analyst",50000)

r1 = resident("John", 30, 201900025,203)
```

```
e1.show()
r1.show()
```

**Result:**

```
Employee Details:
  Name:  Arun
  Age: 35
  Designation: Data analyst
  Salary: 50000
Resident Details:
  Name: John
  Age:  30
  Id: 201900025
  Room no.: 203
>>>
```

# 8. Operator Overloading

**Aim:** To overload the binary operators to perform operations on objects.

**Operator Overloading:**

Operator overloading is an important concept in object oriented programming. It is a type of polymorphism in which a user defined meaning can be given to an operator in addition to the predefined meaning for the operator.

Operator overloading allow us to redefine the way operator works for user-defined types such as objects. It cannot be used for built-in types such as int, float, char etc., For example, '+' operator can be overloaded to perform addition of two objects of distance class.

Python provides some special function or magic function that is automatically invoked when it is associated with that particular operator. For example, when we use + operator on objects, the magic method __add__() is automatically invoked in which the meaning/operation for + operator is defined for user defined objects.

**Coding:**

```
# distance is a class. Distance is measured in terms of feet and inches

class distance:

    def __init__(self, f,i):

        self.feet=f

        self.inches=i



# overloading of binary operator > to compare two distances

    def __gt__(self,d):

        if(self.feet>d.feet):

            return(True)

        elif((self.feet==d.feet) and (self.inches>d.inches)):

            return(True)

        else:

            return(False)
```

```
 # overloading of binary operator + to add two distances
    def __add__(self, d):
        i=self.inches + d.inches
        f=self.feet + d.feet
        if(i>=12):
            i=i-12
            f=f+1
        return distance(f,i)

 # displaying the distance
    def show(self):
        print("Feet= ", self.feet, "Inches= ",self.inches)

a,b= (input("Enter feet and inches of distance1: ")).split()
a,b =[int(a),int(b)]
c,d= (input("Enter feet and inches of distance2: ")).split()
c,d =[int(c),int(d)]
d1 = distance(a,b)
d2 = distance(c,d)

if(d1>d2):
    print("Distance1 is greater than Distance2")
else:
    print("Distance2 is greater or equal to Distance1")
```

```
d3=d1+d2
print("Sum of the two Distance is:")
d3.show()
```

**Result:**

```
Enter feet and inches of distance1: 8 4
8 4
Enter feet and inches of distance2: 6 9
6 9
Distance1 is greater than Distance2
Sum of the two Distance is:
Feet=  15 Inches=  1
>>>
```

# 9. File Handling

**Aim:** To perform file operations such as open read, write and close on text and numeric data files.

**File Handling:**

A file is a collection of data stored in one unit, identified by a filename. It can be a text document, picture/image, audio, audio-video or other collection of data. The common format/extensions for text documents are .doc, .docx (Microsoft word documents), .odt (LibreOffice open document text), .pdf (Adobe portable document format) , rtf (Microsoft rich text format), .tex (LaTeX text), .txt (Microsoft Notepad text). The image file formats are: .jpg, .tiff, .gif, .png, bmp. The commonly used audio formats are: .wav and .mp3. The audio-video format includes .avi, .mp4, .mkv, .mov, .flv, .wmv etc.,

Python supports file handling and allow users to handle files i.e., to read and write files, along with many other file handling options, to operate on files. Python file functions open() and close() are used for opening and closing a file. The read() and write() functions are used for reading and writing text (or numeric or binary data) from/to the file, respectively. File opening modes in Python are: r (read), w (write), a (append), rb (reading binary data), wb(writing binary data).

**a) Coding: (Reading and Writing Text File)**

```
# File copy – content of a text file (input.txt) is copied to another text file (output.txt)

infile=open("/media/yughu/D/input.txt","r")

outfile=open("/media/yughu/D/output.txt","w")

lines = chars = 0

for line in infile:

   lines += 1

   chars += len(line)
```

```
    outfile.write(line)

print(lines, "lines copied,",chars, "characters copied")

infile.close()

outfile.close()
```

**Result:**

```
11 lines copied, 82 characters copied
```

## b) Coding: (Reding and Writing Numeric Data File)

```
# sum of  all the numbers in the input file (input.dat)  is computed and
# it is written to the output file (output.dat)

infile=open("/media/yughu/D/input.dat","r")
outfile=open("/media/yughu/D/output.dat","w")

sum=0
s = infile.read()
numbers = [int(x) for x in s.split()]
print("The numbers are:")
print(numbers)

for num in numbers:
   sum=sum+num

sum=str(sum)
outfile.write("Sum is ")
outfile.write(sum)

infile.close()
outfile.close()
```

**Result:**

```
The numbers are:

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

# 10. Exception Handling

**Aim:** To understand and handle different types of exceptions that occurs at runtime in the program and also the method of raising an exception.

**Exception Handling:**

Exceptions are run-time anomalies or abnormal conditions that a program encounters during its execution such as division by zero (ZeroDivisionError), opening a file for reading that does not exist (IOError), indentation is not specified properly (IndentationError) etc., In general, an exception breaks the normal flow of execution. Exception handling enables a program to deal with exceptions and continue its normal execution.

A try statement in Python can have more than one except clause to handle different exceptions. The statement can also have an optional else and/or finally statement. The try-except syntax is:

```
try:
        <body>
except <ExceptionType1>:
        <handler1>
            ...
            ...
except <ExceptionTypeN>:
        <handlerN>
except:
        <handlerExcept>
else:
        <process_else>
finally:
        <process_finally>
```

The multiple excepts are similar to elifs in python. When an exception occurs, it is checked to match an exception in an except clause after the try clause one by one (sequentially). If a match is found, the handler for the matching case is executed and the rest of the except clauses are skipped. Note that the <ExceptionType> in the last except clause may be omitted. If the exception does not match any of the exception types before the last except clause, the <handlerExcept> for the last except clause is executed.

A try statement may have an optional else clause, which is executed if no exception is raised in the try body.

A try statement may have an optional finally clause, which is intended to define cleanup actions that must be performed under all circumstances.

**Coding:**

# Hadling exceptions that occurs at runtime such as division by zero, syntax error and

# raising and handling the exception.

```
try:
   number1, number2 = eval(input("Enter two numbers separated by a comma: "))
   result = number1 / number2
   print("Result is", result)
   if(number1==0):
      raise RuntimeError()


except ZeroDivisionError:
   print("Division by Zero")
except SyntaxError:
   print("A comma may be Missing in the Input")
except RuntimeError:
   print("May be Meaningless")
except:
   print("Something Wrong in the Input")
else:
   print("No Exceptions")
finally:
   print("Finally Clause is Executed")
```

**Result:**

```
Enter two numbers separated by a comma: 4,0

4,0

Division by Zero

Finally Clause is Executed

>>>

Enter two numbers separated by a comma: 5 6

5 6

A comma may be Missing in the Input

Finally Clause is Executed

>>>

Enter two numbers separated by a comma: 0,9

0,9

Result is 0.0

May be Meaningless

Finally Clause is Executed

>>>

Enter two numbers separated by a comma: 6

6

Something Wrong in the Input

Finally Clause is Executed

>>>

Enter two numbers separated by a comma: 12,4
12,4
Result is 3.0
No Exceptions
```

```
Finally Clause is Eexecuted
```

# 11. Send and Receive Messages using UDP

**Aim:** To send and receive messages between client and server using UDP.

**Procedure:**

UDP (User Datagram Protocol) is connection-less protocol which is suitable for applications that require efficient communication that doesn't have to worry about packet loss.

**Implementing the client:**

To begin we will need to import the socket python module. Once we’ve got this we need to declare the IP address that we will be trying to send our UDP messages to as well as the port number. This port number is arbitrary but ensures that you aren’t using a socket that has already been taken. Now that we've declared these few variables it’s time to create the socket through which we will be sending our UDP message to the server. And finally, once we've constructed our new socket it's time to write the code that will send our UDP messages.

**Implementing the server:**

Once we have coded our client we then need to move on to creating our server program which will be continuously listening on our defined IP address and port number for any UDP messages. It is essential that this server has to be run prior to the execution of the client python script or the client script will fail. Once we’ve imported the socket module and declared our IP address and port number we can create another socket which will look exactly like the socket we constructed in our client program. And finally, once we've created our server socket, we need to write the code that will keep our script continuously listening to this socket until its termination.

**Client Coding:** (client.py)

```
import socket            #importing the socket library
host = '127.0.0.1'         #host IP address
port = 5001                #assign random port value above 1024(up to 1024 ports are reserved)
obj = socket.socket()    #next create a socket object
obj.connect((host,port))           #connecting to the server
message = input("Type message: ")
while message != 'bye':
    obj.send(message.encode())              # Send the encoded message to Server
    data = obj.recv(1024).decode()        # Receive the message from Server, packet size 1024 bytes
    print ('Received from server: ' + data)
    message = input("Type message: ")
obj.send(message.encode())
obj.close()              #close the connection
```

**Server Coding:** (server.py)

```
import socket
host = "127.0.0.1"
port = 5001
server = socket.socket()
server.bind((host,port))           #bind the socket to assigned host IP and port
server.listen()                      #listen for incoming connections
conn, addr = server.accept()      #Establish connection with client
print ("Connection from: " + str(addr))
```

```
while True:
    data = conn.recv(1024).decode()      # Receive the message from Client
    if  str(data)== 'bye':
        print ("Received from Client: " + str(data))
        print ("Connection Terminated ")
        conn.close()
        break
    print ("Received from Client: " + str(data))
    data = input("Type message: ")
    conn.send(data.encode())                # Send the message to Client
```

**Result:**

**Window 1:** (client.py)

```
Type message: Hai
Received from server: What is your favorite programming language?
Type message: Python
Received from server: OK. Number of Open Source Python Packages is available
in the Web.
Type message: Where it is available?
Received from server: Type "PyPI" in Google.
Type message: bye
```

**Window 2:** (server.py)

```
Connection from: ('127.0.0.1', 54363)
Received from Client: Hai
Type message: What is your favorite programming language?
Received from Client: Python
Type message: OK. Number of Open Source Python Packages is available in the
Web.
Received from Client: Where it is available?
Type message: Type "PyPI" in Google.
Received from Client: bye
Connection Terminated.
```

# 12. File Transfer using File Transfer Protocol

**Aim:** To send files from local host to server using FTP.

**Procedure (Server):** The server will host the network using the assigned host and port using which the client will connect to the server. The server will act as the sender and the user will have to input the filename of the file that he/she would like to transmit. The user must make sure that the file that needs to be sent is in the same directory as the "server.py" program.

**Procedure (Client):** The client program will prompt the user to enter the host address of the server while the port will already be assigned to the port variable. Once the client program has connected to the server it will ask the user for a filename to be used for the file that will be received from the server. Lastly the client program will receive the file and leave it in the same directory under the same filename set as the user.

**Server Coding:**

```
import socket

s = socket.socket()
host = socket.gethostname()   #Get localhost IP address
port = 8080                   #assign port for session
s.bind((host,port))           #bind the socket to assigned host IP and port
s.listen(1)                   #put the socket into listening mode for 1 client
print(host)
print("Waiting for any incoming connection... ")
conn, addr = s.accept()
print(addr, "Has connected to the server")

filename = input(str("Enter the name of the file to be transmitted: "))
file = open(filename , 'rb')      # Opens a file for reading only in binary format
file_data = file.read(1024)
conn.send(file_data)
print("File has been transmitted successfully")
```

**Client Coding:**

```
import socket
s = socket.socket()
host = input(str("Please enter the host address of the sender: "))
port = 8080
s.connect((host,port))
print("Connected ... ")

filename = input(str("Please enter a filename for the incoming file: "))
file = open(filename, 'wb')        # Opens a file for writing only in binary format
file_data = s.recv(1024)
file.write(file_data)
file.close()
print("File has been received successfully.")
```

**Result:**

**Windows 1: (**server.py**)**

```
Dell     (# localhost name)
Waiting for any incoming connection...
('192.168.43.193', 49395) Has connected to the server
Enter the name of the file to be transmitted: ex1.py
File has been transmitted successfully
```

**Windows 2: (**client.py**)**

```
Please enter the host address of the sender: Dell
Connected ...
Please enter a name for the incoming file: exercise1.py
File has been received successfully.
```

# 13. Loan Calculator using Tkinter

**Aim:** To implement a loan calculator using Tkinter.

**Tkinter:**

Python offers multiple options for developing GUI (Graphical User Interface). Out of all the GUI methods, Tkinter is the most commonly used method. It is a standard Python interface to the Tk GUI toolkit shipped with Python. Python with Tkinter is the fastest and easiest way to create the GUI applications. Creating a GUI using Tkinter is an easy task.

A GUI application is created using Tkinter as follows:

1. Import the Tkinter module using **import Tkinter**.
2. Create the GUI application main window using window =Tk( ).
3. Add one or more widgets to the GUI application.
4. Enter the main event loop to take action against each event triggered by the user.

Tkinter provides various controls, such as buttons, labels and text boxes used in a GUI application. These controls are commonly called widgets. There are currently 15 types of widgets in Tkinter.

**Loan Calculation using Tkinter:**

In this coding, a GUI application is developed for computing loan payments. The code consists of the following steps:

1. Design the user interface consisting of labels using Label( ), text entry boxes using Entry( ), and a button using Button( ).

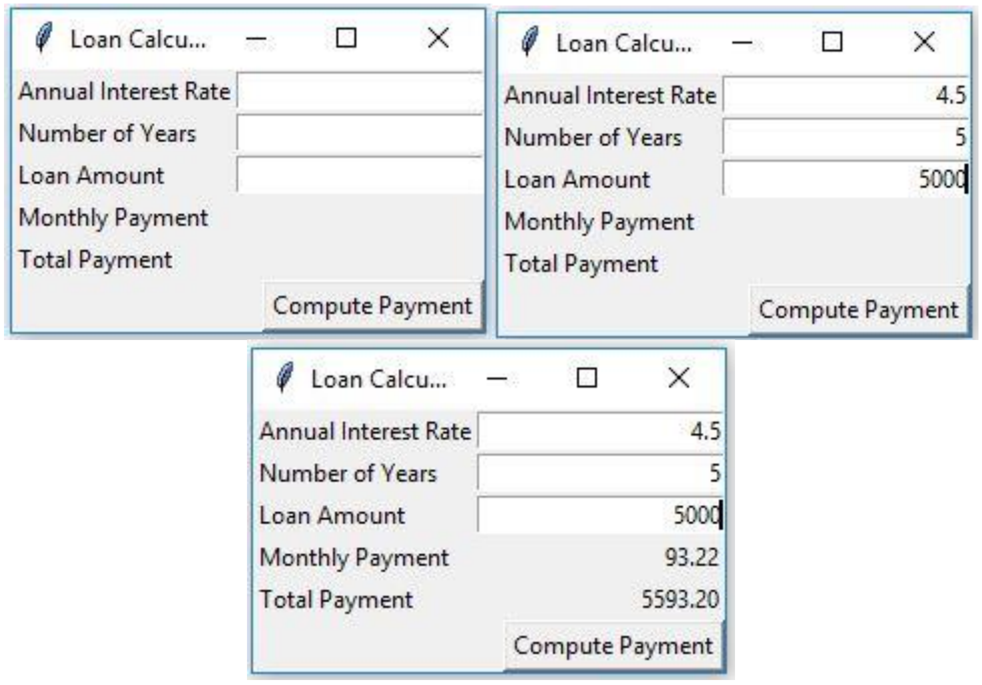
2. Process the event. When the button is clicked, the program invokes a callback functions, using getMonthlyPayment( ) to obtain the user input for the interest rate, number of years, and loan amount from the text entries. The monthly and total payments are computed using

computePayment( ) and displayed in the labels.

## Coding:

```
from tkinter import * # Import all definitions from tkinter
class LoanCalculator:
  def __init__(self):
    window = Tk()    # Create a window
    window.title("Loan Calculator")    # Set title
    Label(window, text = "Annual Interest Rate").grid(row = 1,column = 1, sticky = W)
    Label(window, text = "Number of Years").grid(row = 2,column = 1, sticky = W)
    Label(window, text = "Loan Amount").grid(row = 3,column = 1, sticky = W)
    Label(window, text = "Monthly Payment").grid(row = 4,column = 1, sticky = W)
    Label(window, text = "Total Payment").grid(row = 5,column = 1, sticky = W)
    self.annualInterestRateVar = StringVar()
    Entry(window, textvariable = self.annualInterestRateVar,justify = RIGHT).grid(row = 1, column = 2)
    self.numberOfYearsVar = StringVar()
    Entry(window, textvariable = self.numberOfYearsVar,justify = RIGHT).grid(row = 2, column = 2)
    self.loanAmountVar = StringVar()
    Entry(window, textvariable = self.loanAmountVar,justify = RIGHT).grid(row = 3, column = 2)
    self.monthlyPaymentVar = StringVar()
    lblMonthlyPayment = Label(window, textvariable =self.monthlyPaymentVar).grid(row = 4, column = 2,sticky
                                                                                                  = E)
    self.totalPaymentVar = StringVar()
    lblTotalPayment = Label(window, textvariable =self.totalPaymentVar).grid(row = 5,column = 2, sticky = E)
    btComputePayment = Button(window, text = "Compute Payment",command =
    self.computePayment).grid(row = 6, column = 2, sticky = E)
    window.mainloop() # Create an event loop
  def computePayment(self):
        monthlyPayment =
        self.getMonthlyPayment(float(self.loanAmountVar.get()),float(self.annualInterestRateVar.get()) /
        1200,int(self.numberOfYearsVar.get()))
        self.monthlyPaymentVar.set(format(monthlyPayment, "10.2f"))
        totalPayment = float(self.monthlyPaymentVar.get()) * 12  * int(self.numberOfYearsVar.get())
        self.totalPaymentVar.set(format(totalPayment, "10.2f"))
    def getMonthlyPayment(self,loanAmount, monthlyInterestRate, numberOfYears):
        monthlyPayment = loanAmount * monthlyInterestRate / (1- 1 / (1 + monthlyInterestRate) **
(numberOfYears
                                                                                                    * 12))
        return monthlyPayment;
LoanCalculator()    # Create GUI
```

**Result:**



```
Loan Calcu...
Annual Interest Rate
Number of Years
Loan Amount
Monthly Payment
Total Payment
                    Compute Payment
```

```
Loan Calcu...
Annual Interest Rate        4.5
Number of Years               5
Loan Amount                5000
Monthly Payment
Total Payment
                    Compute Payment
```

```
Loan Calcu...
Annual Interest Rate        4.5
Number of Years               5
Loan Amount                5000
Monthly Payment           93.22
Total Payment           5593.20
                    Compute Payment
```

The loan calculator has been implemented using Tkinter.

# 14. Popup Menu based Arithmetic Operations using Tkinter

**Aim:** To create a popup menu for arithmetic operations using Tkinter.

**Popup menu using Tkinter:**

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, you create an instance of Popup menu using Menu( ), and then you can add items to it using menu.add_command( ). Finally, you bind a widget with an event to pop up the menu. Popup menu commands are used to perform actions to be displayed in a **canvas** which is created using canvas( ).

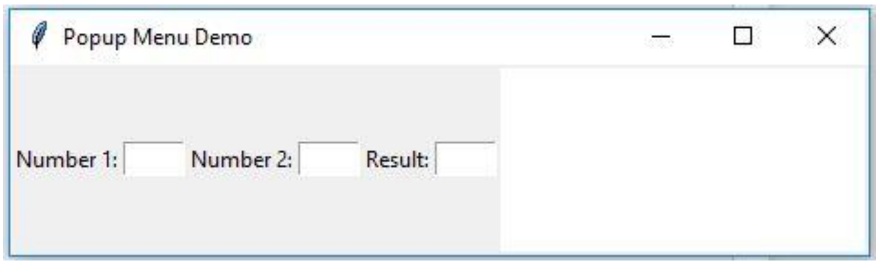
The canvas widget is used to add the structured graphics to the python application. It is used to display text, images, graph and plots to the python application. The menu items use callback functions to instruct the canvas to perform actions.

In this coding, a popup menu is displayed by clicking the right mouse button followed by binding it with the canvas using canvas.bind( ). Each item in the popup menu is selected to perform the corresponding arithmetic operations. The inputs are obtained in two textboxes and the output is displayed in one textbox.

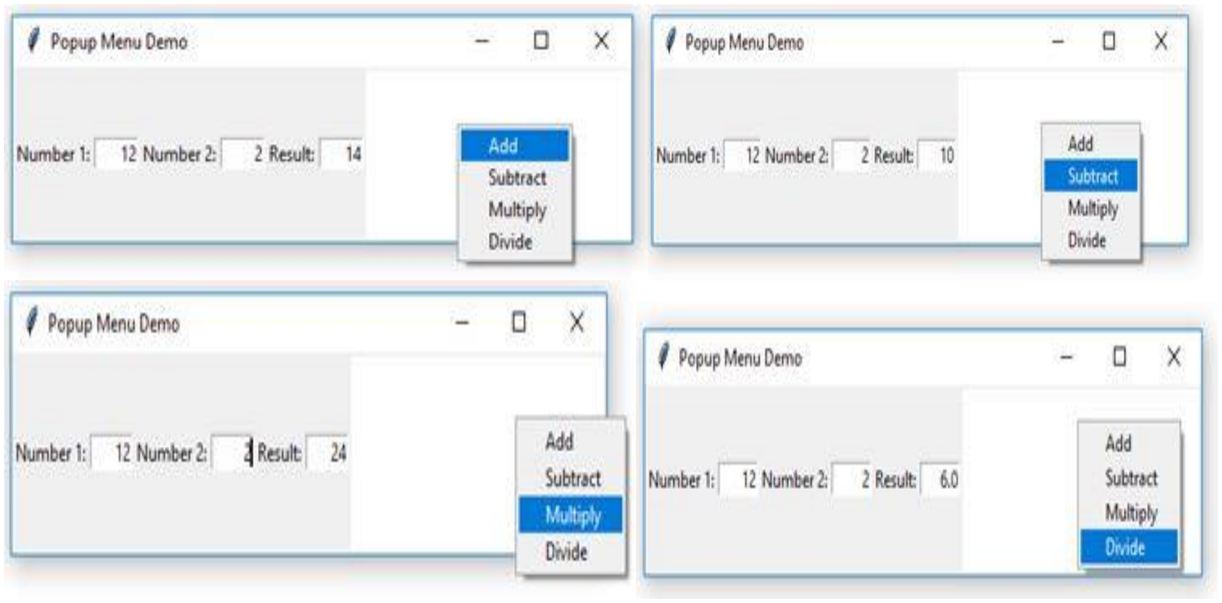
**Coding:**

```
from tkinter import * # Import all definitions from tkinter
class PopupMenuDemo:
  def __init__(self):
    window = Tk() # Create a window
    window.title("Popup Menu Demo") # Set title
    # Create a popup menu
    self.menu = Menu(window, tearoff = 0)
    self.menu.add_command(label = "Add",command = self.add)
    self.menu.add_command(label = "Subtract",command = self.subtract)
    self.menu.add_command(label = "Multiply",command = self.multiply)
    self.menu.add_command(label = "Divide",command = self.divide)
    Label(window, text = "Number 1:").pack(side = LEFT)
    self.v1 = StringVar()
    Entry(window, width = 5, textvariable = self.v1,justify = RIGHT).pack(side = LEFT)
    Label(window, text = "Number 2:").pack(side = LEFT)
    self.v2 = StringVar()
    Entry(window, width = 5, textvariable = self.v2,justify = RIGHT).pack(side = LEFT)
    Label(window, text = "Result:").pack(side = LEFT)
    self.v3 = StringVar()
    Entry(window, width = 5, textvariable = self.v3,justify = RIGHT).pack(side = LEFT)
                  # Place canvas in window
    self.canvas = Canvas(window, width = 200,height = 100, bg = "white")
    self.canvas.pack()
     # Bind popup to canvas
    self.canvas.bind("<Button-3>", self.popup)
    window.mainloop() # Create an event loop
def add(self):
    self.v3.set(eval(self.v1.get()) + eval(self.v2.get()))
def subtract(self):
    self.v3.set(eval(self.v1.get()) - eval(self.v2.get()))
def multiply(self):
    self.v3.set(eval(self.v1.get()) * eval(self.v2.get()))
def divide(self):
    self.v3.set(eval(self.v1.get()) / eval(self.v2.get()))
def popup(self, event):
        self.menu.post(event.x_root, event.y_root)

PopupMenuDemo() # Create GUI
```

**Result:**



Number 1: Number 2: Result:



Number 1: 12 Number 2: 2 Result: 14

Number 1: 12 Number 2: 2 Result: 10

Number 1: 12 Number 2: 2 Result: 24

Number 1: 12 Number 2: 2 Result: 6.0

The popup menu for arithmetic operations has been created using Tkinter and the results are verified.

# 15. Display an Image and its Transformation using Tkinter and OpenCV-Python

**Aim:** To read and display an RGB color image and convert it into grayscale, negative and edge images.

**Image:**

An image is defined as a two-dimensional function, f(x, y), where x and y are spatial coordinates, and the amplitude of F at any pair of coordinates (x, y) is called the intensity of that image at that point. When x, y, and amplitude values of F are finite, it is called a digital image. In other words, an image can be defined by a two-dimensional array specifically arranged in rows and columns. Digital Image is composed of a finite number of elements, each of which has a particular value at a particular location. These elements are referred to as pixels or picture elements.

Images are represented in rows and columns as below:

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & f(0,2) & \cdots & f(0,N-1) \\ f(1,0) & f(1,1) & f(1,2) & \cdots & f(1,N-1) \\ \vdots & \vdots & \vdots & & \vdots \\ f(M-1,0) & f(M-1,1) & f(M-1,2) & \cdots & f(M-1,N-1) \end{bmatrix}$$

where each element in the matrix is called pixel or picture element.

**Types of Images:**

In this coding the select image button is used to select an image. The file dialog is used to select the desired input image from the hard disk. After selecting an image four kinds of operations are performed by clicking one button for each operation. The RGB button is used to display the

RGB color image, the gray button is used to display the gray image of the selected image, the edge button is used to display the edge of the selected image, and the negative button is used to show the negative of the selected image.

**RGB Image:**

An RGB image is an image in which each pixel is specified by three values, one each for the red, blue, and green components of the pixel's color. RGB images are stored as an m-by-n-by-3 data array that defines red, green, and blue color components for each individual pixel. The color of each pixel is determined by the combination of the red, green, and blue intensities stored in each color plane at the pixel's location. Graphics file formats store RGB images as 24-bit images, where the red, green, and blue components are 8 bits each. This yields a potential of 16 million ($2^{24}$) colors. A pixel whose color components are (0, 0, 0) is displayed as black, and a pixel whose color components are (1, 1, 1) i.e., (255, 255, 255) is displayed as white.

**Grayscale Image:**

Grayscale is a range of monochromatic shades from black to white. Therefore, a grayscale image contains only shades of gray and no color. The main characteristic of grayscale images is the equality of the red, green, and blue color levels. The function cv2.cvtColor ( ) in OpenCV-Python is used to convert an RGB color image into gray image.

An RGB image is converted to gray images using

$$Y = 0.299R + 0.587G + 0.114B$$

**Negative Image:**

Negative image is a kind of image that can be formed by subtracting the RGB value from 255. The cv2.bitwise_not ( ) is used to convert the RGB color image into negative image in color.

An RGB color image (i.e., positive image) is converted into a negative image using

$$R_n = 255 – R_p$$

$$G_n = 255 - G_p$$

$B_n = 255 – B_p$

where p and n are positive and negative, respectively.

**Edge Image:**

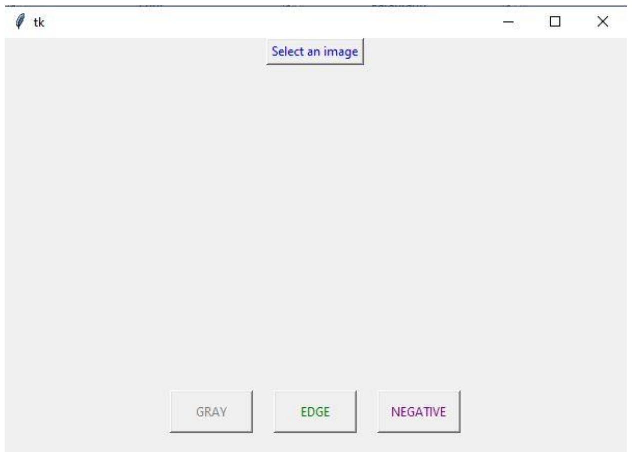
Edge detection includes a variety of mathematical methods that aim at identifying points in a digital image at which the image brightness changes sharply or has discontinuities. The points at which image brightness changes sharply are typically organized into a set of curved line segments termed edges. Edges are often associated with the boundaries of objects in an image. In this coding, Canny edge detection method is used to convert RGB image into edge image using cv2.canny ( ).

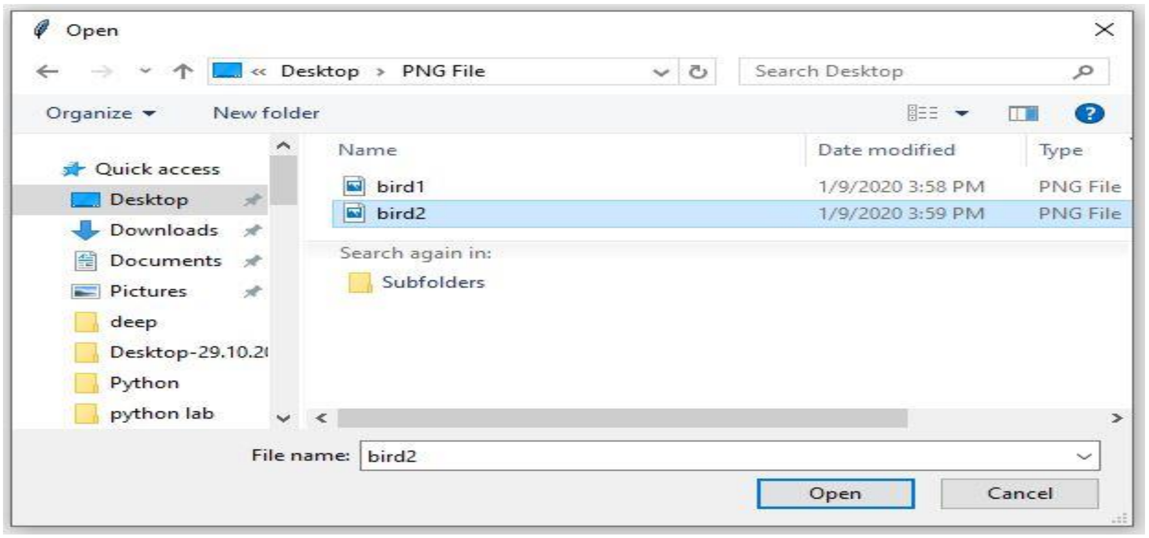
**Coding:**

```
# import the necessary packages
from tkinter import *
from PIL import Image
from PIL import ImageTk
from tkinter import filedialog
import cv2
from tkinter import filedialog
from tkinter.filedialog import askopenfilename
def start():
    global image1,path
    global panel
    path = filedialog.askopenfilename()
    image1 = cv2.imread(path)
    image1 = cv2.resize(image1,(200,100))
    image1 = cv2.cvtColor(image1, cv2.COLOR_BGR2RGB)
    image = Image.fromarray(image1)
    image = ImageTk.PhotoImage(image)
    if panel is None:
        panel = Label(image=image)
        panel.image = image
        panel.pack(side="left", padx=30, pady=60)
    else:
        panel.configure(image=image)
        panel.image = image

def gray_image():
    global panelB
    gray = cv2.cvtColor(image1, cv2.COLOR_BGR2GRAY)
    gray = Image.fromarray(gray)
    gray = ImageTk.PhotoImage(gray)
    if panelB is None:
        panelB = Label(image=gray)
        panelB.image = gray
        panelB.pack(side="left", padx=30, pady=60)
    else:
        panelB.configure(image=gray)
        panelB.image = gray

def edge_image():
    global panelB
    gray = cv2.cvtColor(image1, cv2.COLOR_BGR2GRAY)
    edge = cv2.Canny(gray, 50, 100)
    edge = Image.fromarray(edge)
    edge = ImageTk.PhotoImage(edge)
    if panelB is None:
        panelB = Label(image=edge)
        panelB.image = edge
        panelB.pack(side="left", padx=30, pady=60)
    else:
        panelB.configure(image=edge)
        panelB.image = edge

def negative_image():
    global panelB
    image = cv2.cvtColor(image1, cv2.COLOR_BGR2RGB)
    negative = cv2.bitwise_not(image)
    negative = Image.fromarray(negative)
    negative = ImageTk.PhotoImage(negative)

    if panelB is None:
        panelB = Label(image=negative)
        panelB.image = negative
        panelB.pack(side="left", padx=30, pady=60)
    else:
        panelB.configure(image=negative)
        panelB.image = negative
root = Tk()
root.geometry('600x400')
fm = Frame(root, width=300, height=200)
fm.pack(side=BOTTOM, expand=NO, fill=NONE)
```

```
panelB = None

panel = None

btn = Button(root, text="Select an image", command=start,activebackground='red',fg="blue")

btn.pack(side="top")

btn2 = Button(fm, text="GRAY", command=gray_image,activebackground='red',height=2, width=10,fg="gray")

btn2.pack(side=LEFT, padx="10", pady="20")

btn3 = Button(fm, text="EDGE", command=edge_image,activebackground='red',height=2, width=10,fg="green")

btn3.pack(side=LEFT, padx="10", pady="20")

btn4 = Button(fm, text="NEGATIVE", command=negative_image,activebackground='red',height=2, width=10,fg="purple")

btn4.pack(side=LEFT, padx="10", pady="20")

root.mainloop()
```

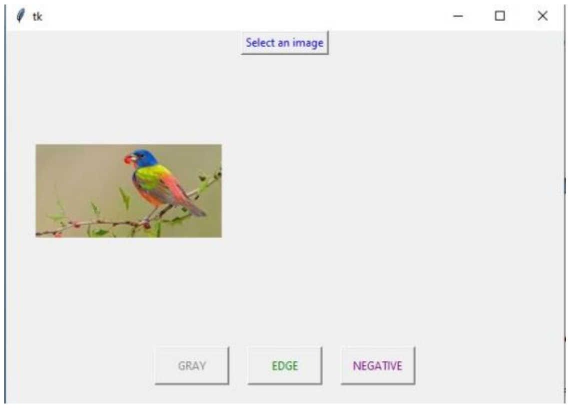
**Result:**

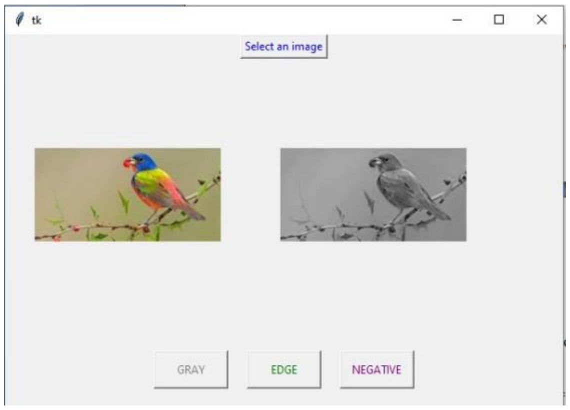


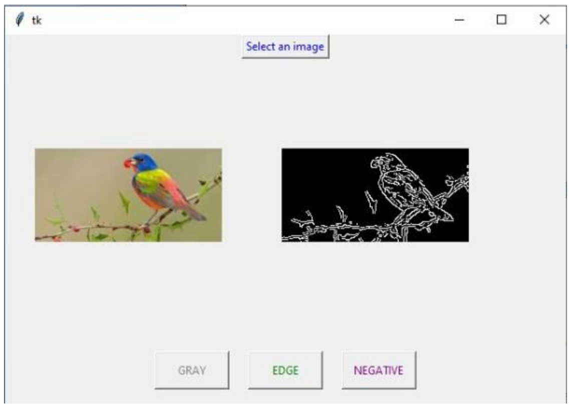


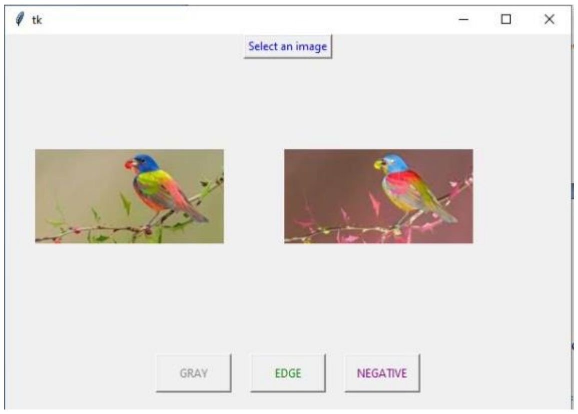
| Name | Date modified | Type |
|---|---|---|
| bird1 | 1/9/2020 3:58 PM | PNG File |
| bird2 | 1/9/2020 3:59 PM | PNG File |

File name: bird2









The given RGB color image is converted into grayscale, negative and edge images using Tkinter.

***