

CONTENTS

S.No.	TITLE	PAGE NO.
1.	Checking Whether the Given Number is Positive, Zero or Negative	1
2.	Prime Number Checking	2
3.	Armstrong Number Checking	4
4.	Finding Roots of a Quadratic Equation	6
5.	Transposing a Matrix	9
6.	Multiplication Table Generation	11
7.	Finding ASCII Value of the Given Character	13
8.	Swapping Two Variables	14
9.	Generation of Fibonacci Sequence using Recursion	17
10.	Conversion of Decimal Number into Other Number Systems	19
11.	File Handling in Python	22
12.	Creation of Class and Objects	30
13.	Operator Overloading	32
14.	(a) Single Inheritance	35
	(b) Multiple Inheritance	37
15.	Exception Handling	39
16.	Client Server Programming using TCP Socket	41
17.	Client Server Programming using UDP Socket	45
18.	Student Information System using tkinter.	49
19.	Expenses Tracker using SQLite	54

Ex. No. : 1 Checking Whether the Given Number is Positive, Zero or Negative

Date:

Aim:

To check whether given number is positive or zero or negative.

Algorithm:

1. Start.
2. Read a number x from the user.
3. If x value is equal to zero, then print “the given number is zero”.
4. If x value is greater than zero, then print “the given number is positive”.
5. Otherwise print “the given number is negative”.

Program:

```
x=int(input("Enter a number : "))
if(x==0):
    print("Given Number is Zero")
elif(x>0):
    print("Given Number is Positive ")
else:
    print("Given Number is Negative")
```

Sample Input and Output:

Enter a number : 42

Given Number is Positive

Enter a number : -83

Given Number is Negative

Enter a number : 0

Given Number is Zero

Result:

Thus, the Python program has been written to check whether the given number is positive, negative or zero, and executed successfully.

Ex. No. : 2

Prime Number Checking

Date:

Aim:

To check whether given number is prime or not.

Algorithm:

1. Ask for input from the user. Read the input n and convert it into integer.
2. If $n > 1$, do the following:
 - i. Divide the number from 2 to $n/2$.
 - ii. When the remainder is 0 print “the number is not a prime number” and break the loop.
 - iii. When there is no remainder for the entire loop, then print “the number is a prime number”
3. Otherwise ask the user to enter a valid number.

Program:

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

Sample Input and Output:

Enter any number: 159

159 is not a prime number

Enter any number: 97

97 is a prime number

Enter any number:-27

-27 is not a valid number

Result:

Thus, the python program has been written to check whether the given number is prime or not and executed successfully.

Ex. No. : 3

Armstrong Number Checking

Date:

Aim:

To check whether a given number is an Armstrong number or not.

Algorithm:

1. Ask for input from the user. Read the input and convert it into integer.
2. Find the number of digits in the input using len().
3. Divide the number by 10 and separate the digits.
4. Raise each digit to the power of the number of digits in the input.
5. Find the sum of them.
6. If the sum is equal to the given number, then print that the number is an Armstrong number.
7. Otherwise print that the number is not an Armstrong number.

Program:

```
num=int (input ("Enter the number : "))
order=len(str(num))
temp=num
total=0
while temp>0:
    digit=temp%10
    total+=digit**order
    temp//=10
if num==total:
    print (num, "is an Armstrong number")
else:
    print (num, "is not an Armstrong number")
```

Sample Input and Output:

Hint: The first few Armstrong numbers are

1, 2, 3, 4, 5, 6, 7, 8, 9, 153, 370, 371, 407, 1634, 8208, 9474, 54748, ...

Enter the number : 370

370 is an Armstrong number

Enter the number : 218

218 is not an Armstrong number

Enter the number : 9474

9474 is an Armstrong number

Enter the number : 54748

54748 is an Armstrong number

Result:

Thus, the Python program to find the Armstrong numbers was developed and executed successfully.

Ex. No. : 4

Finding Roots of a Quadratic Equation

Date :

Aim:

To find the roots of a quadratic equation using python program.

Algorithm:

1. Start the program.
2. Import math and complex math modules.
3. Get the coefficients (a , b , and c) of the quadratic equation from the user.
4. Calculate the discriminant value d such that $d = b^2 - 4ac$
5. If $d > 0$ two distinct real roots exist. Find them using the formula $\frac{-b \pm \sqrt{d}}{2a}$ and print.
6. If $d = 0$ two equal real roots exist. Find them using the formula $\frac{-b}{2a}$ and print.
7. If $d < 0$ two complex roots exist. Find them using the formula $\frac{-b \pm \sqrt{d}}{2a}$ and print.

Program:

```
# -----Python Program to find roots of a Quadratic Equation-----  
  
import math  
import cmath  
  
print('Enter the coefficients one by one')  
a = float(input("a: "))  
b = float(input("b: "))  
c = float(input("c: "))  
  
d = (b * b) - (4 * a * c)  
if (d > 0):  
    root1 = (-b + math.sqrt(d)) / (2 * a)
```

```

root2 = (-b - math. sqrt(d)) / (2 * a)
print ('Two Distinct Real Roots Exist:')
print ('root1 = { } and root2 = { }'. format (root1, root2))
elif (d == 0):
    root1 = root2 = -b / (2 * a)
    print ("Two Equal and Real Roots Exist:")
    print ('root1 = { } and root2 = { }'. format (root1, root2))
elif (d < 0):
    root1 = (-b+cmath. sqrt(d))/(2*a)
    root2 = (-b+cmath. sqrt(d))/(2*a)
    print ('Two Distinct Complex Roots Exist:')
    print ('root1 = {:.2f} and root2 = {:.2f}'. format (root1, root2))

```

Sample Input and Output:

Enter the coefficients one by one

a: 1

b: -2

c: 1

Two Equal and Real Roots Exist:

root1 = 1.0 and root2 = 1.0

Enter the coefficients one by one

a: 1

b: 7

c: 12

Two Distinct Real Roots Exist:

root1 = -3.0 and root2 = -4.0

Enter the coefficients one by one

a: 1

b: 1

c: 1

Two Distinct Complex Roots Exist:

root1 = $(-0.50-0.87j)$ and root2 = $(-0.50+0.87j)$

Result:

Thus, the Python program for finding the roots of a quadratic equation was developed and executed successfully.

Ex. No. : 5

Transposing a Matrix

Date :

Aim:

To write a Python program to transpose a matrix using list/array and function.

Algorithm:

1. Start the program.
2. Get the size of the matrix and its elements in row major order.
3. Create the matrix while getting the elements using list.
4. Using a function transpose the given input matrix by creating new 2D array and exchanging the row and column elements and return the new array.
5. Print both input and transposed matrices.

Program

```
#-----Function to transpose the matrix-----  
  
def transpose(A):  
    tA = [[[A[j][i] for j in range(len(A))]] for i in range (len (A [0]))]  
    for row in tA:  
        return row  
  
#-----Main Program-----  
  
m=int(input("Enter number of rows of the matrix: "))  
n=int(input("Enter number of columns of the matrix: "))  
  
#-----Reading the matrix using List-----  
  
a=[]  
print("Enter the elements of the matrix (row-order): ")  
for i in range(m):  
    r=[]  
    for j in range(n):  
        r.append(int(input()))  
    a.append(r)
```

```
#-----Displaying the given matrix and its transpose-----  
print("The given matrix is:")  
print(a)  
print("The transpose of the matrix is:")  
print (transpose (a))
```

Sample Input and Output:

Enter number of rows of the matrix: 2

Enter number of columns of the matrix: 3

Enter the elements of the matrix (row-order):

3

8

1

9

4

6

The given matrix is:

[[3, 8, 1], [9, 4, 6]]

The transpose of the matrix is:

[[3, 9], [8, 4], [1, 6]]

Result:

Thus, the Python program has been developed to transpose a given matrix and executed successfully.

Ex. No. : 6

Multiplication Table Generation

Date:

Aim:

To write a python program to generate multiplication table by defining a user defined function for the given inputs and display the table.

Algorithm:

1. Start the program.
2. Read table number and the maximum limit as inputs from the user.
3. Define a function with for loop to iterate over maximum limit number of times to print the multiplication table for the given number and limit.
4. Call the function by passing table number and limit as arguments.
5. Stop the program.

Program:

```
# -----User Defined Function to generate the table -----
```

```
def mtable(num,limit):  
    for i in range(1, limit+1):  
        print(num, 'x', i, '=', num*i)
```

```
# -----Main Program -----
```

```
n = int(input('Which table do you want? '))  
maxi=int(input("\nWhat is the maximum limit? "))  
mtable(n,maxi)
```

Sample Input and Output:

Which table do you want? 8

What is the maximum limit? 16

$$8 \times 1 = 8$$

$$8 \times 2 = 16$$

$$8 \times 3 = 24$$

$$8 \times 4 = 32$$

$$8 \times 5 = 40$$

$$8 \times 6 = 48$$

$$8 \times 7 = 56$$

$$8 \times 8 = 64$$

$$8 \times 9 = 72$$

$$8 \times 10 = 80$$

$$8 \times 11 = 88$$

$$8 \times 12 = 96$$

$$8 \times 13 = 104$$

$$8 \times 14 = 112$$

$$8 \times 15 = 120$$

$$8 \times 16 = 128$$

Result:

Thus, a Python program has been written to generate multiplication table by defining a user defined function for the given inputs and executed successfully.

Ex. No. : 7

Finding ASCII Value of the Given Character

Date:

Aim:

To write a Python program to find ASCII value of the given character.

Algorithm:

1. Start the Program.
2. Get the character from the user.
3. Using ord() built in function obtain the ASCII value of the given character.
4. Display the character and its ASCII value.
5. Stop the Program

Program:

```
#----- Program to find the ASCII value of the given character-----  
cha = input('Enter any character: ')  
print("The ASCII value of {} is {}".format(cha,ord(cha)))
```

Sample Input and Output:

Enter any character: t

The ASCII value of t is 116

Enter any character: @

The ASCII value of @ is 64

Result:

Thus, a Python program has been written to find ASCII value of the given character using built in function and executed successfully.

Ex. No.: 8

Swapping Two Variables

Date:

Aim:

To write a Python program to swap two variables.

Algorithm:

1. Start the Program.
2. Get the input variables (X, Y) to be swapped.
3. (a) Method 1 (without using function):

The given variables can be swapped easily by using multiple assignment.

$X, Y = Y, X$

(OR)

3. (b) Method 2 (using function):
 - (i) Define a function to perform step 3(a) by getting X and Y as parameters.
 - (ii) Return the swapped variables when the function is called.
4. Print the values of X and Y after swapping.
5. Stop the Program.

Program:

Method 1:

```
# -----Python program to swap two variables without using function-----  
var1 = input('Enter First Variable: ')  
var2 = input('Enter Second Variable: ')  
print("Value of variable 1 before swapping: ", var1)  
print("Value of variable 2 before swapping: ", var2)
```

```
#----- swapping two numbers without using temporary variable-----  
var1, var2 = var2, var1  
print("Value of variable 1 after swapping: ", var1)  
print("Value of variable 2 after swapping: ", var2)
```

Sample Input and Output:

Enter First Variable: 58

Enter Second Variable: Tamilnadu

Value of variable 1 before swapping: 58

Value of variable 2 before swapping: Tamilnadu

Value of variable 1 after swapping: Tamilnadu

Value of variable 2 after swapping: 58

Method 2:

```
#----- Function to swap two variables -----  
def swap_var(a,b):  
    a,b=b,a  
    return (a,b)  
  
#----- Main Program -----  
var1 = input('Enter First Variable: ')  
var2 = input('Enter Second Variable: ')  
print('Before swapping: \n variable 1 = {0} \n variable 2 = {1}'.format(var1,var2))  
var1,var2=swap_var(var1,var2)  
print('After swapping: \n variable 1 = {0} \n variable 2 = {1}'.format(var1,var2))
```


Sample Input and Output:

Enter First Variable: Annamalai

Enter Second Variable: University

Before swapping:

variable 1=Annamalai

variable 2=University

After swapping:

variable 1=University

variable 2=Annamalai

Result:

Thus, a Python program has been written to swap two variables and executed successfully.

Ex. No.: 9

Generation of Fibonacci Sequence using Recursion.

Date:

Aim:

To write a Python program to display the Fibonacci sequence using recursion.

Algorithm:

1. Start the Program.
2. Get the number of terms as input.
3. Check if the number of terms is valid by checking if it is greater than or equal to zero. Otherwise give error message.
4. For valid input, print Fibonacci sequence from 0 to (number of terms -1) by printing 0 and 1 as it is, and by adding present and previous values.
5. Stop the Program.

Program:

#----- Recursive function to print Fibonacci sequence -----

```
def recur_fibo(n):
```

```
    if n <= 1:
```

```
        return n
```

```
    else:
```

```
        return (recur_fibo(n-1) + recur_fibo(n-2))
```

#-----Main Program-----

```
nterms = int(input("How many terms? "))
```

```
if nterms <0:
```

```
    print("Please enter a positive integer")
```

```
else:
```

```
    print("Fibonacci sequence:")
```

```
    for i in range(nterms):
```

```
        print(recur_fibo(i))
```

Sample Input and Output:

How many terms? 10

Fibonacci sequence:

0

1

1

2

3

5

8

13

21

34

Result:

Thus, a Python Program has been developed to generate Fibonacci sequence using recursion and executed successfully.

Ex. No. : 10 Conversion of Decimal Number into Other Number Systems

Date:

Aim:

To write a Python Program to convert decimal number into binary, octal and hexadecimal number system.

Algorithm:

1. Start the Program.
2. Get the input decimal value (dec) and choice of the number system.
3. Using switcher dictionary mapping if the match for the choice is found.
4. If the match is not found, display the default error message.
 - i. convert decimal to binary with built in function “bin(dec)”
 - ii. convert decimal to octal with built in function “oct(dec)”
 - iii. convert decimal to hexadecimal with built in function “hex(dec)”
5. Stop the Program.

Program:

#----- Function to convert decimal to other number systems using switcher -----

```
def decimal_to_bases(ch):
```

```
    switcher = {
```

```
        1: bin(dec),
```

```
        2: oct(dec),
```

```
        3: hex(dec)
```

```
    }
```

```
    return switcher.get(ch, "\n Invalid choice")
```

#----- Main Program -----

```
dec = int(input("Enter the decimal number: "))
```

```
op=['binary','octal','hexadecimal']  
  
print("Select the number system.")  
  
print("\n 1.Binary")  
  
print("\n 2.Octal")  
  
print("\n 3.Hexadecimal")  
  
ch = int(input("\n Enter choice(1/2/3): "))  
  
print(decimal_to_bases(ch))
```

Sample Input and Output:

Enter the decimal number: 16

Select the number system.

1.Binary

2.Octal

3.Hexadecimal

Enter choice(1/2/3): 1

0b10000

Enter the decimal number: 27

Select the number system.

1.Binary

2.Octal

3.Hexadecimal

Enter choice(1/2/3): 2

0o33

Enter the decimal number: 100

Select the number system.

1.Binary

2.Octal

3.Hexadecimal

Enter choice(1/2/3): 3

0x64

Enter the decimal number: 16

Select the number system.

1.Binary

2.Octal

3.Hexadecimal

Enter choice(1/2/3): 4

Invalid choice

Result:

Thus, a Python Program has been developed to convert the given decimal number into binary, octal and hexadecimal number system using switcher dictionary mapping, and executed successfully.

Ex. No. : 11

File Handling in Python

Date:

Aim:

To perform various file operations such as open, read, write, append, and close files.

Algorithm:

1. Create a file in the working directory or in any path. Write few lines in the file and save.
2. Open a file in the appropriate mode.
3. Perform the required operation.
4. Close the file.

The following file is considered as sample input file.

Data.txt

```
Priyalakshmi
Associate Professor
Department of Information Technology
Annamalai University
```

a) Reading a text file using read()

```
my_file=input('Enter your file name: ')
with open(my_file) as f1:
    print(f1.read())
```

Output

Enter your file name: Data.txt

Priyalakshmi

Associate Professor

Department of Information Technology

Annamalai University

b) Reading a text file using readline()

```
my_file=input('Enter your file name: ')
with open(my_file) as f1:
    print(f1.readline())
```

Output

Enter your file name: Data.txt
Priyalakshmi

c) Reading a text file using readlines()

```
my_file=input('Enter your file name: ')
with open(my_file) as f1:
    print(f1.readlines())
```

Output

Enter your file name: Data.txt
['Priyalakshmi\n', 'Associate Professor\n', 'Department of Information Technology\n', 'Annamalai University']

d) Reading N number of lines from a text file

Method 1

Efficient method, but does not work for files greater than N lines

```
my_file=input('Enter your file name: ')
N=int(input('How many lines do you want to read?: '))
with open(my_file) as reader:
    head=[next(reader) for x in range(N)]
print(head)
```


Output 1: (Give $N < \text{number of lines in your file}$)

Enter your file name: Data.txt

How many lines do you want to read?: 3

```
['Priyalakshmi\n', 'Associate Professor\n', 'Department of Information Technology\n']
```

Output 2: (Give $N > \text{number of lines in your file}$)**# When file has less than N number of lines, shows error**

Enter your file name: Data.txt

How many lines do you want to read?: 5

Traceback (most recent call last): File "main.py", line 6, in <module>

```
    head=[next(reader) for x in range(N)]
```

File "main.py", line 6, in <listcomp>

```
    head=[next(reader) for x in range(N)]
```

StopIteration

Method 2**# Works for files less than N lines.**

```
my_file=input('Enter your file name: ')
```

```
N=int(input('How many lines do you want to read?: '))
```

```
with open(my_file) as f1:
```

```
    head=f1.readlines()[0:N]
```

```
print(head)
```

Output:**# When file has less than N number of lines, shows no error**

Enter your file name: Data.txt

How many lines do you want to read?: 8

```
['Priyalakshmi\n', 'Associate Professor\n', 'Department of Information Technology\n', 'Annamalai University']
```

Method 3

```
from itertools import islice
```

```
my_file=input('Enter your file name: ')
```

```
N=int(input('How many lines do you want to read?: '))
```

```
with open(my_file) as f1:
```

```
    head=list(islice(f1,N))
```

```
print(head)
```

Output:

```
Enter your file name: Data.txt
```

```
How many lines do you want to read?: 8
```

```
['Priyalakshmi\n', 'Associate Professor\n', 'Department of Information Technology\n', 'Annamalai University']
```

Output:

```
Enter your file name: Data.txt
```

```
How many lines do you want to read?: 3
```

```
['Priyalakshmi\n', 'Associate Professor\n', 'Department of Information Technology\n']
```

e) Counting number of lines in a text file

```
fname = input("Enter file name: ")
```

```
num_lines = 0
```

```
with open(fname, 'r') as f:
    for line in f:
        num_lines += 1
print("Number of lines: ")
print(num_lines)
```

Output:

Enter file name: Data.txt

Number of lines:

4

f) Counting number of words in a text file in Python

```
fname = input("Enter file name: ")
num_words = 0
```

```
with open(fname, 'r') as f:
    for line in f:
        words = line.split()
        num_words += len(words)
```

```
print("Number of words:")
print(num_words)
```

Output:

Enter file name: Data.txt

Number of words:

9

g) Counting number of characters including and excluding spaces in a text file

```
fname = input("Enter file name: ")  
with open(fname, 'r') as file:  
    data = file.read()  
  
char_space = len(data)                                # counting characters including spaces  
# counting characters excluding spaces by replacing space with nothing  
char_no_space = len(data.replace(" ", ""))  
print('Number of characters with spaces:', char_space)  
print('\n Number of characters without spaces:', char_no_space)
```

Output:

Number of characters with spaces: 90

Number of characters without spaces: 85

h) Writing String to text file

```
fname = input("Enter file name: ")  
  
with open(fname, 'w') as text_file:  
    text_file.write('Welcome to python class')  
# Read the file and check the content  
print(open(fname).read())
```

Output:

Welcome to python class

i) Writing line by line to text file

```
fname = input("Enter file name: ")
with open(fname, 'w') as text_file:
    text_file.write('Welcome to python class \n')
    text_file.write('Learning python is fun \n')
# Read the file and check the content
print(open(fname).read())
```

Output

```
Welcome to python class
Learning python is fun
```

j) Writing multiple lines to text file

```
mylist=['python \n','C \n','Java \n']
with open('new_file.txt','w') as out:
    out.writelines(mylist)

with open('new_file.txt') as out1:
    print(out1.read())
```

Read the file and check the content.

Output:

```
python
C
Java
```

k) Appending a line in the existing file

```
mylist=['are programming languages \n', 'But learning python is fun' ]
with open('new_file.txt','a') as out:
```

```
        out.writelines(mylist)
with open('new_file.txt') as out1:
    print(out1.read())
```

Output:

python

C

Java

are programming languages

Result:

Thus, various file operations such as open, read, write, append, and close files have been performed using Python and the results are verified.

Ex. No. : 12

Creation of Class and Objects

Date:

Aim:

To define a class with constructor and create objects for finding distance between two points in two-dimensional space.

Algorithm:

1. Create a Class (for points in two-dimensional space) which consists of a constructor and two methods.
2. One method is to find the Euclidean distance between the origin (0,0) and the point in a two-dimensional space (x, y) as follows: $\sqrt{x^2 + y^2}$
3. Another function is to find the Euclidean distance between the two points (x_1, y_1) and (x_2, y_2) in a two-dimensional space as follows: $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$
4. Create a two-dimensional point as the object of the class and compute the distance between origin and the point.
5. Create two-dimensional points as the objects of the class and compute the distance between those two points.

Program:

```
From math import sqrt
```

```
# Create a class for two-dimensional space
```

```
class point():
```

```
    def __init__(self,a,b):          # Constructor to initialize a point
```

```
        self.x=a
```

```
        self.y=b
```

```
# Method to compute distance between origin and a point
```

```
def distancefromorigin(self):
```

```
    return (sqrt(self.x ** 2) + (self.y ** 2) )
```

Method to compute distance between two points

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

-----Main Program-----

```
x1,y1= (input("Enter the coordinates of a first point P1 (x1,y1): ")).split(',')  
x1,y1 =[int(x1),int(y1)]  
x2,y2= (input("Enter the coordinates of a second point P2 (x2,y2): ")).split(',')  
x2,y2 =[int(x2),int(y2)]  
  
p1 = point(x1,y1)  
p2 = point(x2,y2)  
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))
```

Sample Input and Output:

```
Enter the coordinates of a first point P1(x1,y1): 3,4  
Enter the coordinates of a second point P2(x2,y2): 5,8  
Distance from origin to P1: 19.0  
Distance from origin to P2: 69.0  
Distance from P1 to P2:4.47
```

Result:

Thus, a Python program has been developed to define a class with constructor and create objects for finding distance between two points in two-dimensional space.

Ex. No. : 13

Operator Overloading

Date:

Aim:

To overload the binary operators to perform operations on objects.

Algorithm:

1. Create a class for distance. Distance is measured in terms of feet and inches.
2. Write special function to overload binary operator ' > ' to compare two distances.
3. Write special function to overload binary operator ' < ' to compare two distances.
4. Write special function to overload binary operator ' + ' to add two distances.
5. Write a method to display the distance.
6. Get the inputs from the user in the form of feet and inches separated by space.
7. Compare them using '<' and '>' operators and display appropriate messages.
8. Add them and print the result using display method.

Program:

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 compare two distances

```
def __lt__(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)
```

Method to display the distance

```
def show(self):
    print("Feet = ", self.feet, "Inches = ", self.inches)
```

#-----Main Program-----

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

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

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

Sample Input and Output:

Enter feet and inches of distance1 (separated by space): 8 4
Enter feet and inches of distance2 (separated by space): 6 9
Distance1 is greater than or equal to Distance2
Distance2 is less than Distance1
Sum of the two Distance is:
Feet = 15 Inches = 1

Result:

Thus, a Python program has been developed to overload the binary operators to perform operations on distance objects and executed successfully.

Ex. No. : 14 (a)

Single Inheritance

Date:

Aim:

To implement single inheritance by defining a new class from an existing class.

Algorithm:

1. Person is a base class constructed with name and age.
2. Employee is derived from Person class constructed with Designation and Salary.
3. Write a method under employee class to show employee details: name, age, designation, and salary.
4. Create objects of employee class and display employee details.

Program:

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)
```

Creating objects of employee class

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

Sample Output:

Employee Details:

Name: Arun

Age: 35

Designation: Data analyst

Salary: 50000

Result:

Thus, a Python program has been developed to implement single inheritance by defining a new class from an existing class and executed successfully.

Ex. No. : 14 (b)

Multiple Inheritance

Date:

Aim:

To implement multiple inheritance by defining a new class from one or more existing classes.

Algorithm:

1. Construct base class *person* with name and age.
2. Construct base class *student* with id and room number.
3. Derive resident class from *person* and *student* classes.
4. Write a method under resident class to show resident details: name, age, id and room number.
5. Create objects of resident class and display the resident details.

Program:

person is a base class

class person:

```
def __init__(self, n, a):  
    self.name = n  
    self.age = a
```

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 resident classes

```
r1 = resident("Priya", 30, 201900025,203)  
r1.show()
```

Sample Output:

Resident Details:

Name: Priya

Age: 30

Id: 201900025

Room no.: 203

Result:

Thus, a Python program has been developed to implement multiple inheritance by defining a new class from one or more existing classes and executed successfully.

Ex. No. : 15

Exception Handling

Date:

Aim:

To handle the exception that occurs at runtime and to throw (raise) an exception and handling it.

Algorithm:

1. Using try block get the inputs from user and perform an operation, e.g., division operation.
2. Raise an exception pertaining to the operation.
3. Handle exceptions that occur at runtime such as division by zero, syntax and run time errors using except, else and finally clauses.

Program:

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")
```


Sample Input and Output:

Enter two numbers separated by a comma: 4,0

Division by Zero

Finally Clause is Executed

Enter two numbers separated by a comma: 5 6

A comma may be Missing in the Input

Finally Clause is Executed

Enter two numbers separated by a comma: 0,9

Result is 0.0

May be Meaningless

Finally Clause is Executed

Enter two numbers separated by a comma: 6

Something Wrong in the Input

Finally Clause is Executed

Enter two numbers separated by a comma: 12,4

Result is 3.0

No Exceptions

Finally Clause is Executed

Result:

Thus, a Python program has been developed to demonstrate how Python is handling exceptions and executed successfully.

Ex. No. : 16

Client Server Programming using TCP Socket

Date:

Aim:

To create client server architecture using connection oriented (TCP) socket and send the current time string of the server to the client.

Algorithm:

1. Create server program as follows and save it (for example: server-tcp.py)
 - i. Create server socket using `socket()`
 - ii. Get local machine name using `gethostname()`
 - iii. Select a port from the range 1025 - 65535
 - iv. Bind socket using `socket` method `bind()` to address in the format(host name,port number)
 - v. Listen for client connections using `socket` method `listen()`
 - vi. Start server infinite loop and accept client connection using `socket` method `accept()`
 - vii. Get current time of the server and convert into string.
 - viii. Send the current time of the server to client
 - ix. Close client and server sockets
2. Create client program as follows and save it (for example: client-tcp.py)
 - i. Create client socket using `socket()`
 - ii. Get local machine name using `gethostname()`
 - iii. Select a port from the range 1025 - 65535
 - iv. Initiate TCP server connection using `socket` method `connect()` to address in the format (host name, port number)
 - v. Receive TCP message from the server using `socket` method `recv()`
 - vi. Close client socket
 - vii. Print the time information received from the server.
3. Open Command Prompt and run server program, and open another Command Prompt and run client program.

Server Program

```
from socket import *
import time

ss = socket(AF_INET, SOCK_STREAM)           # Create server socket
host = gethostname()                        # Get local machine name
port = 9999

ss.bind((host, port))                      # Bind socket to address
ss.listen(5)                               # Listen for connections and queue up to 5 requests
while True:                                # Server infinite loop
    cs, addr = ss.accept()                  # Accept client connection
    print("Got a connection from %s" % str(addr))
    currentTime = time.ctime(time.time()) + "\r\n" # Get current time of the server
    cs.send(currentTime.encode('ascii'))      # Send the current time of the server to client
    cs.close()                              # Close client socket
ss.close()                                 # Close server socket
```

Client Program

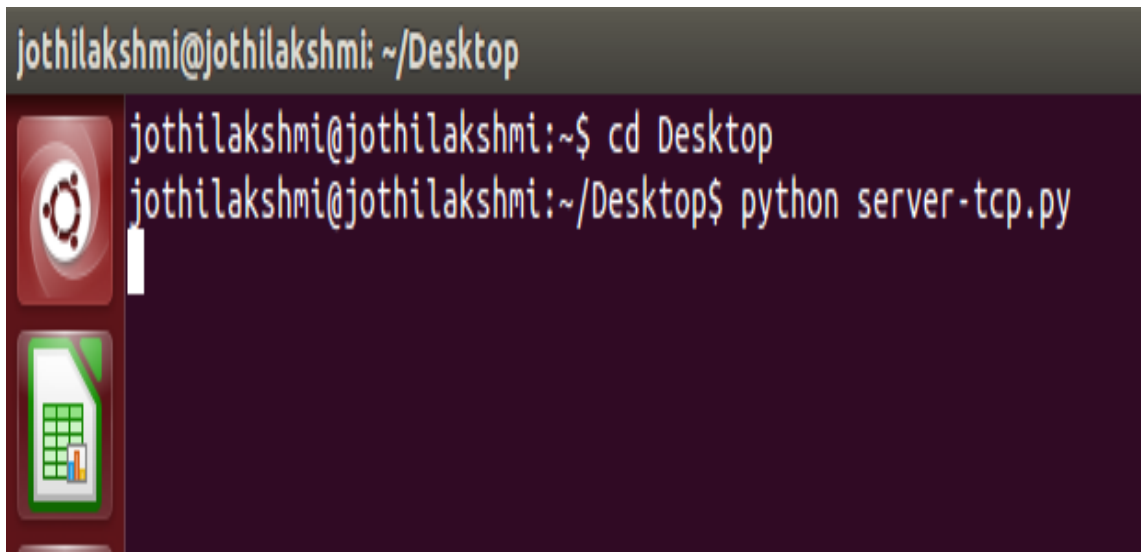
```
import socket

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # Create client socket
host = socket.gethostname()                           # Get local machine name
port = 9999

s.connect((host, port))                               # Initiate TCP server connection
tm = s.recv(1024)                                     # Receive TCP message no more than 1024 bytes
s.close()                                              # Close client socket
print("The time got from the server is %s" % tm.decode('ascii'))
```

Sample Input and Output:

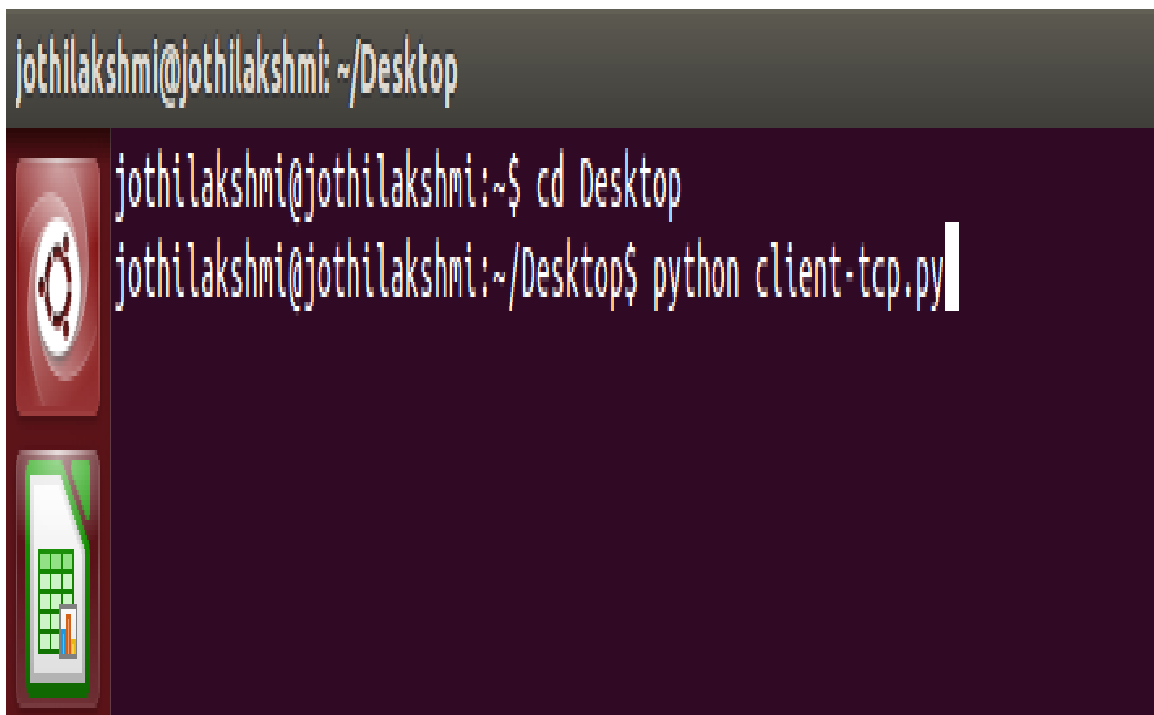
Executing server program

A terminal window with a dark purple background and a grey title bar. The title bar contains the text 'jothilakshmi@jothilakshmi: ~/Desktop'. The terminal shows the following commands and their outputs:

```
jothilakshmi@jothilakshmi:~/Desktop
jothilakshmi@jothilakshmi:~$ cd Desktop
jothilakshmi@jothilakshmi:~/Desktop$ python server-tcp.py
```

On the left side of the terminal, there is a vertical dock with two icons: a red square icon with a white circular arrow and a green square icon with a white grid pattern.

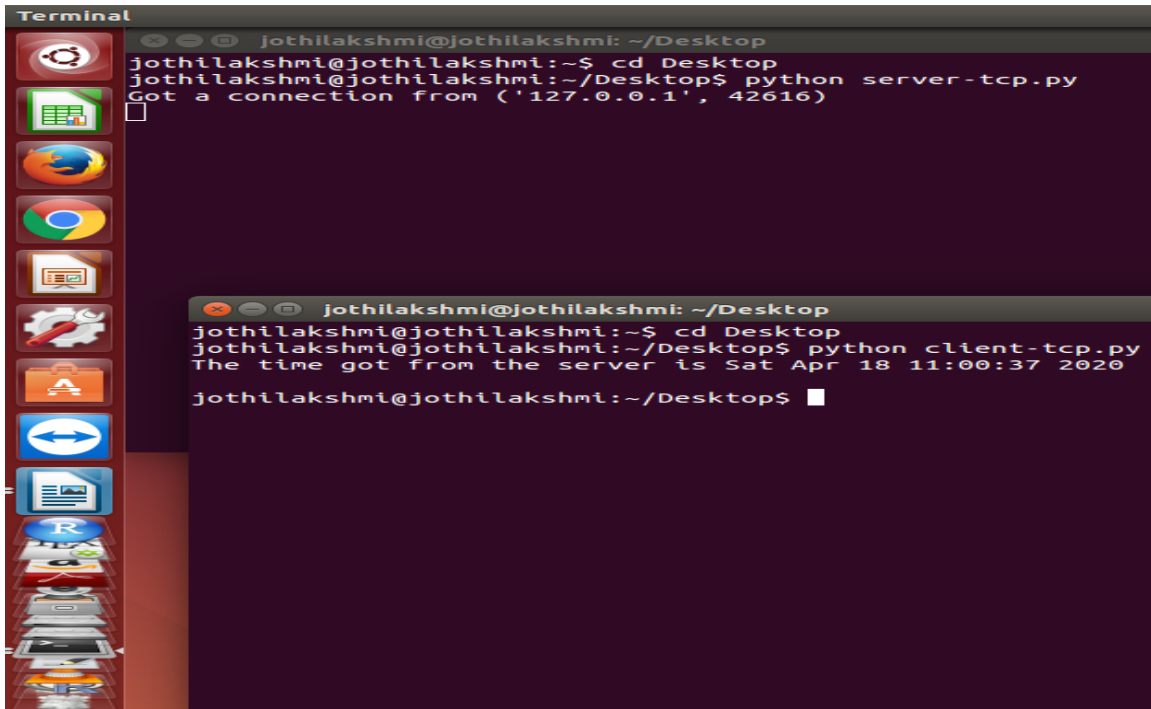
Executing client program

A terminal window with a dark purple background and a grey title bar. The title bar contains the text 'jothilakshmi@jothilakshmi: ~/Desktop'. The terminal shows the following commands and their outputs:

```
jothilakshmi@jothilakshmi:~/Desktop
jothilakshmi@jothilakshmi:~$ cd Desktop
jothilakshmi@jothilakshmi:~/Desktop$ python client-tcp.py
```

On the left side of the terminal, there is a vertical dock with two icons: a red square icon with a white circular arrow and a green square icon with a white grid pattern.

Screenshot of Client and Server



```
Terminal
jothilakshmi@jothilakshmi: ~/Desktop
jothilakshmi@jothilakshmi:~$ cd Desktop
jothilakshmi@jothilakshmi:~/Desktop$ python server-tcp.py
Got a connection from ('127.0.0.1', 42616)

jothilakshmi@jothilakshmi: ~/Desktop
jothilakshmi@jothilakshmi:~$ cd Desktop
jothilakshmi@jothilakshmi:~/Desktop$ python client-tcp.py
The time got from the server is Sat Apr 18 11:00:37 2020
jothilakshmi@jothilakshmi:~/Desktop$
```

Result:

Thus, a Python program has been developed to create client server architecture using connection oriented (TCP) socket and send the current time string of the server to the client. The client and server scripts were executed successfully, and the output was verified.

Ex. No. : 17

Client Server Programming using UDP Socket

Date:

Aim:

To create client server architecture using connectionless (User Datagram Protocol (UDP)) socket and send a message from the client to server.

Algorithm:

1. Create server program as follows and save it (for example: server-udp.py)
 - i. Create server socket using `socket()`
 - ii. Get local machine name using `gethostname()`
 - iii. Select a port from the range 1025 - 65535
 - iv. Bind socket using socket method `bind()` to address in the format(host name, port number)
 - v. Wait for a message from a client
 - vi. Receive message from client using socket method `recvfrom()`
 - vii. Print the message received from client
2. Create client program as follows and save it (for example: client-udp.py)
 - i. Create client socket using `socket()`
 - ii. Get local machine name using `gethostname()`
 - iii. Select a port from the range 1025 - 65535
 - iv. Send message to UDP server using socket method `sendto()`
3. Open Command Prompt and run server program, and open another Command Prompt and run client program.

Server Program

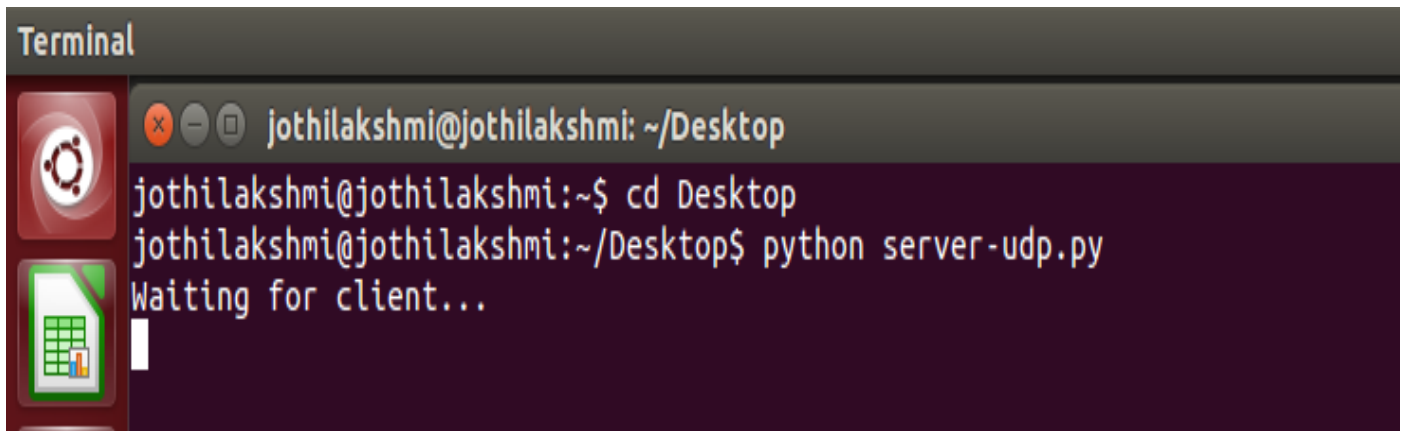
```
import socket                                # Import socket module
sock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM) # Create UDP server socket
udp_host = gethostname()                    # Get local machine name
udp_port = 12345                            # Specify port to connect
sock.bind((udp_host,udp_port))              # Bind socket to address
while True:
    print "Waiting for client..."
    data,addr = sock.recvfrom(1024)          # Receive message from client
    print "Received Messages:",data," from",addr
```

Client Program

```
import socket                                # Import socket module
sock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM) # Create client UDPsocket
udp_host = socket.gethostname()              # Get local machine name
udp_port = 12345                            # Specify port to connect
msg = "Hello Python!"
print "UDP target IP:", udp_host
print "UDP target Port:", udp_port
sock.sendto(msg,(udp_host,udp_port))         # Send message to UDP server
```

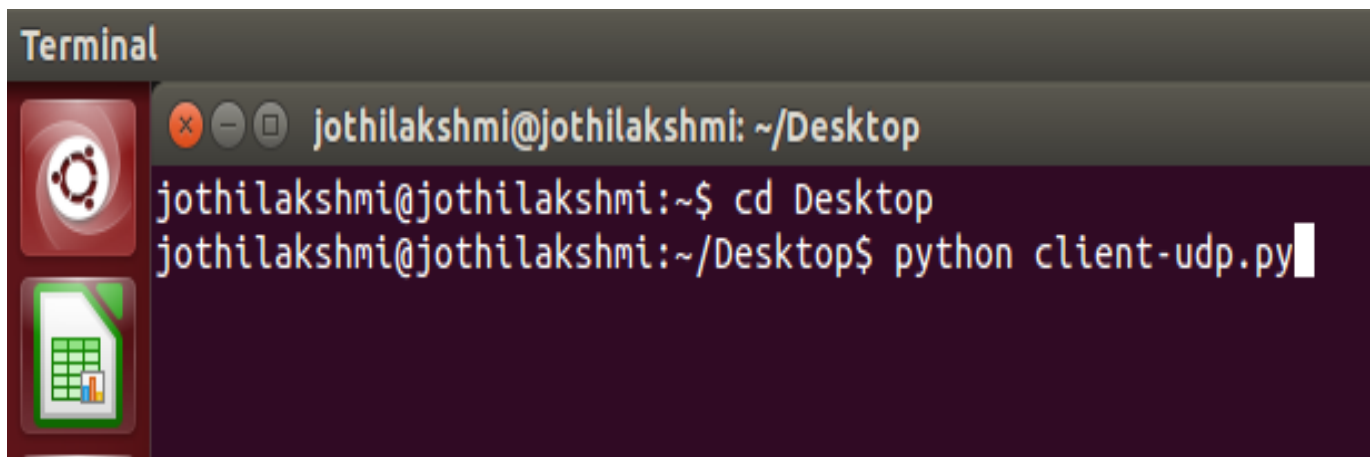
Sample Input and Output:

Executing server program

A terminal window titled "Terminal" with a dark background. The prompt is "jothilakshmi@jothilakshmi: ~/Desktop". The user enters "cd Desktop" and then "python server-udp.py". The output is "Waiting for client...".

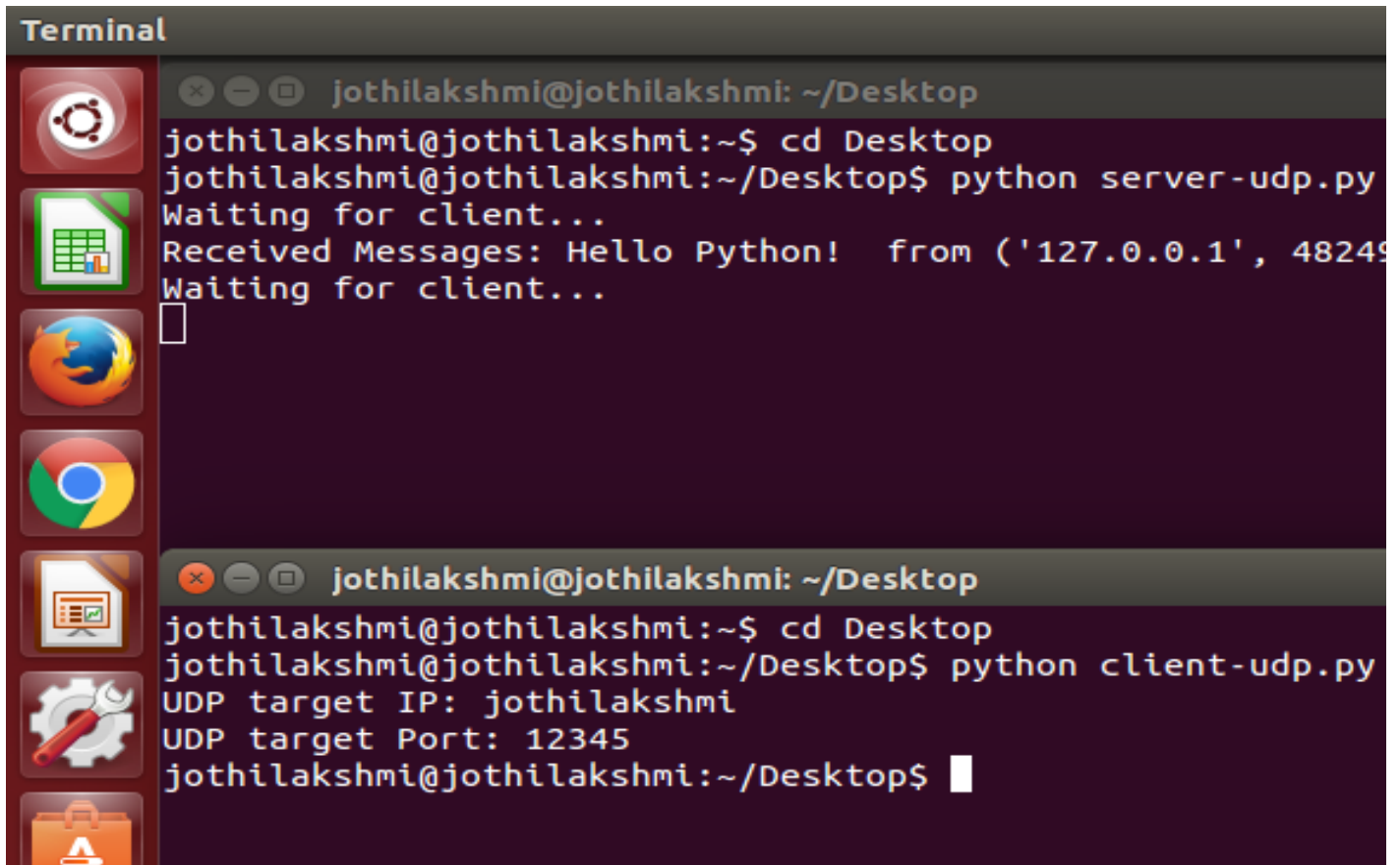
```
Terminal
jothilakshmi@jothilakshmi: ~/Desktop
jothilakshmi@jothilakshmi:~$ cd Desktop
jothilakshmi@jothilakshmi:~/Desktop$ python server-udp.py
Waiting for client...
```

Executing client program

A terminal window titled "Terminal" with a dark background. The prompt is "jothilakshmi@jothilakshmi: ~/Desktop". The user enters "cd Desktop" and then "python client-udp.py".

```
Terminal
jothilakshmi@jothilakshmi: ~/Desktop
jothilakshmi@jothilakshmi:~$ cd Desktop
jothilakshmi@jothilakshmi:~/Desktop$ python client-udp.py
```


Screenshot of Client and Server



The screenshot shows a Linux terminal window titled "Terminal" with a dark background. The terminal displays the execution of a Python UDP server and client. The server script, `server-udp.py`, is run in the `~/Desktop` directory. It outputs "Waiting for client...", then "Received Messages: Hello Python! from ('127.0.0.1', 48249)", and "Waiting for client..." again. The client script, `client-udp.py`, is also run in the `~/Desktop` directory. It outputs "UDP target IP: jothilakshmi" and "UDP target Port: 12345". The terminal window has a sidebar on the left with icons for various applications, including a terminal, a spreadsheet, a web browser, and a presentation viewer.

```
Terminal
jothilakshmi@jothilakshmi: ~/Desktop
jothilakshmi@jothilakshmi:~$ cd Desktop
jothilakshmi@jothilakshmi:~/Desktop$ python server-udp.py
Waiting for client...
Received Messages: Hello Python! from ('127.0.0.1', 48249)
Waiting for client...
jothilakshmi@jothilakshmi:~/Desktop$

jothilakshmi@jothilakshmi: ~/Desktop
jothilakshmi@jothilakshmi:~$ cd Desktop
jothilakshmi@jothilakshmi:~/Desktop$ python client-udp.py
UDP target IP: jothilakshmi
UDP target Port: 12345
jothilakshmi@jothilakshmi:~/Desktop$
```

Result:

Thus, a Python program has been developed to create client server architecture using connectionless (UDP) socket and send a message from the client to server. The client and server scripts were executed successfully, and the output was verified.

Ex. No. : 18

Student Information System using tkinter

Date:

Aim:

To create a student information system using tkinter modules and stored information in separate file.

Algorithm:

Create a class called studentInformaton as given below:

- 1) In the `__init__()` method, initialize the following attributes:
 - a) `self.name`: A string variable to store the student's name.
 - b) `self.roll`: An integer variable to store the student's roll number.
 - c) `self.reg`: An integer variable to store the student's registration number.

- 2) Create a method called `appui()`
 - a) In this method, create three frames and pack them.
 - b) In the first frame, create a label and an entry widget to enter the student's name.
 - c) In the second frame, create a label and an entry widget to enter the student's roll number.
 - d) In the third frame, create a label and an entry widget to enter the student's registration number.

- 3) Create a method called `onsubmit()`
 - a) In this method, open a file called `student.txt` in append mode.
 - b) Write the student's name, roll number, and registration number to the file.
 - c) Close the file.
 - d) Display a messagebox to indicate that the student has been successfully added.

Program

```
from tkinter import *
from tkinter import messagebox

class studentInformaton(Tk):
    def __init__(self):
        super().__init__()
        self.geometry('500x500')
        self.resizable(width=False , height=False)
        self.title("Student Information System")
        self.name = StringVar()
        self.roll = IntVar()
        self.reg = IntVar()

    def appui(self):
        Label(self,text="B.E. Information Technology",font="Bold").pack()
        myframe1 = Frame(self,width=100,height=100,bg="pink")
        myframe1.pack()
        Label(myframe1 , text="Enter Student Name ",font=("Bold")).pack()
        Entry(myframe1,font=("Bold"),textvariable=self.name).pack(ipadx=20)
        myframe2 = Frame(self,width=100,height=100,bg="pink")
        myframe2.pack()
        Label(myframe2 , text="Enter Student Roll Number",font=("Bold")).pack()
        Entry(myframe2,font=("Bold"),textvariable=self.roll).pack(ipadx=20)
        myframe3 = Frame(self,width=100,height=100,bg="pink")
```

```

myframe3.pack()
Label(myframe3 , text="Enter student Register Number",font=("Bold")).pack()
Entry(myframe3,font=("Bold"),textvariable=self.reg).pack(ipadx=20)
myframe4 = Frame(self,width=100,height=100)
myframe4.pack()
Button(myframe4 , text="Submit" , command=self.onsubmit,bg="pink",width=20,
height=2).pack()

```

```

def onsubmit(self):
    self.myfile = open("student.txt",'a+')
    self.myfile.writelines(f"""
        Name : {self.name.get()}
        roll number : {self.roll.get()}
        reg number : {self.reg.get()}
    """)
    self.myfile.close()
    messagebox.showinfo("Status","Successfully student added! ")

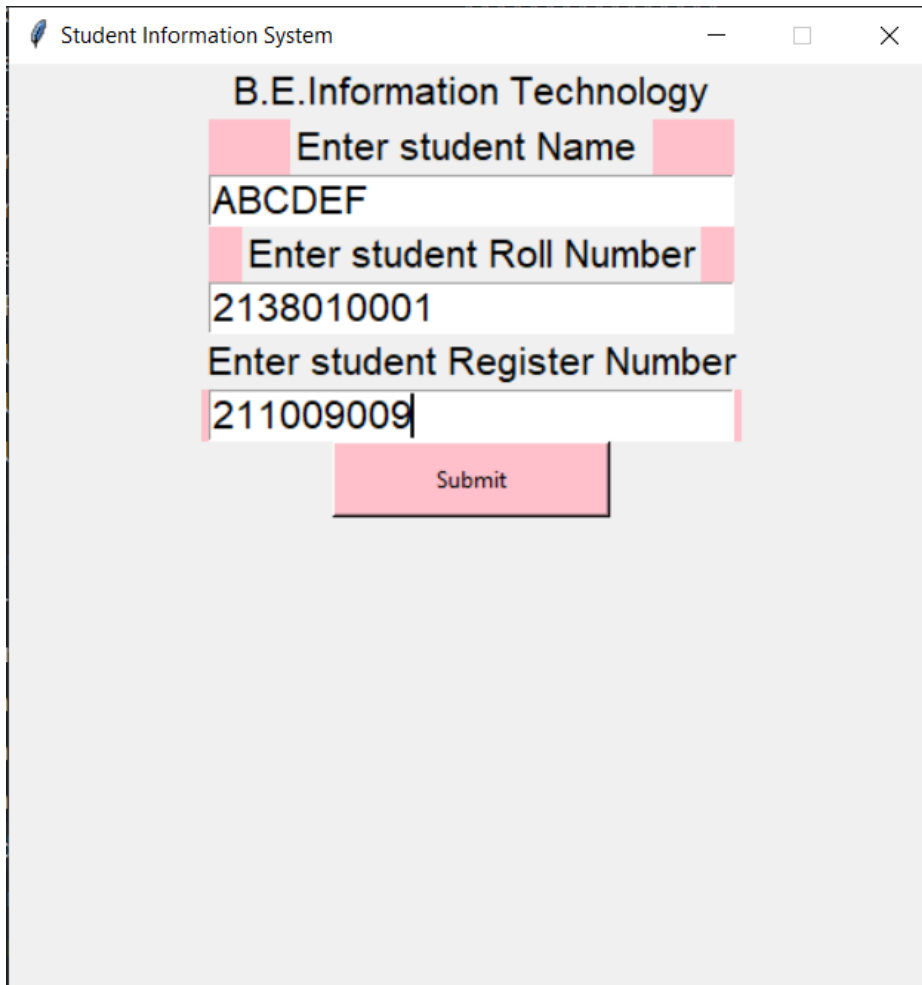
```

```

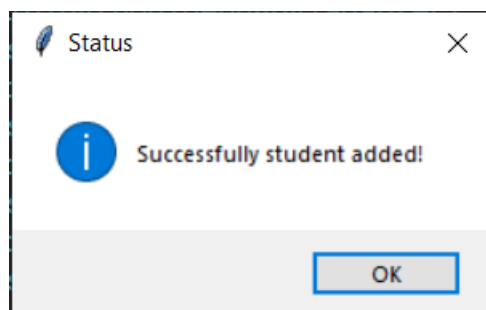
student1 = studentInformaton()
student1.appui()
student1.mainloop()

```

Sample Input and Output:



The screenshot shows a window titled "Student Information System". Inside the window, the text "B.E.Information Technology" is displayed at the top. Below this, there are three input fields with corresponding labels: "Enter student Name" (containing "ABCDEF"), "Enter student Roll Number" (containing "2138010001"), and "Enter student Register Number" (containing "211009009"). A pink "Submit" button is located at the bottom of the input section.



The file students.txt has been opened and the contents are verified.



A screenshot of a Notepad window titled "students.txt - Notepad". The window has a menu bar with "File", "Edit", "Format", "View", and "Help". The text content is as follows:

```
Name : ABCDEF  
roll number : 2138010001  
reg number : 211009009
```

The status bar at the bottom shows "Ln 1, Col 1", "100%", "Windows (CRLF)", and "UTF-8".

Result:

Thus, a Python program has been developed to create a student information system using tkinter in python, executed successfully, and the output was verified.

Ex. No. : 19

Expenses Tracker using SQLite

Date:

Aim:

The aim of this program is to create an expenses tracker using SQLite. It allows users to record expenses, view expense history, and calculate total expenses.

Algorithm :

1. Import the sqlite3 module to work with the SQLite database.
2. Define a function create_connection to create a connection to the SQLite database specified by the database_name. If the connection is successful, return the connection; otherwise, print any errors and return None.
3. Define a function create_table that creates a table named "Products" if it doesn't already exist in the database. The table should have three columns: "id," "product_name," and "product_cost."
4. Define a function add_product to insert a new product into the "Products" table. It takes the conn, product_name, and product_cost as parameters.
5. Define a function view_products to retrieve all products from the "Products" table and return them as a list of tuples. If there are no products, return None.
6. Define a function calculate_total_cost to calculate the sum of all product costs in the "Products" table and return the total cost. If there is an error, return None.
7. Define a function delete_product to delete a product with a specific p_id from the "Products" table.
8. Create a database connection using create_connection with the name "mydb" and create the "Products" table using create_table.
9. Start an infinite loop to display the menu and handle user choices.

10. Inside the loop display the menu with options to:
 - a) Add a product
 - b) View products
 - c) Calculate total expenses
 - d) Delete a product
 - e) Exit
11. Based on the user's choice, perform the corresponding action using the functions defined earlier.
12. When the user selects "5" (Exit), break out of the loop to end the program.

Program :

```
import sqlite3
```

```
def create_connection(database_name):
```

```
    try:
```

```
        conn = sqlite3.connect(database_name)
```

```
        return conn
```

```
    except sqlite3.Error as e:
```

```
        print(e)
```

```
        return None
```

```
def create_table(conn):
```

```
    try:
```

```
        cursor = conn.cursor()
```

```
        cursor.execute("""CREATE TABLE IF NOT EXISTS Products (
```

```
            id INTEGER PRIMARY KEY AUTOINCREMENT,
```

```
            product_name TEXT NOT NULL,
```

```
            product_cost INTEGER NOT NULL
```



```

)"""
conn.commit()
except sqlite3.Error as e:
    print(e)

def add_product(conn, product_name, product_cost):
    try:
        cursor = conn.cursor()
        cursor.execute("INSERT INTO Products(product_name, product_cost) VALUES (?, ?)",
            (product_name, product_cost))
        conn.commit()
    except sqlite3.Error as e:
        print(e)

def view_products(conn):
    try:
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM Products")
        my_products = cursor.fetchall()
        return my_products
    except sqlite3.Error as e:
        print(e)
        return None

```

```

def calculate_total_cost(conn):
    try:
        cursor = conn.cursor()
        cursor.execute("SELECT SUM(product_cost) FROM Products")
        total_cost = cursor.fetchone()[0]
        return total_cost
    except sqlite3.Error as e:
        print(e)
        return None

def delete_product(conn, p_id):
    try:
        cursor = conn.cursor()
        cursor.execute("DELETE FROM Products WHERE id = ?", (p_id,))
        conn.commit()
    except sqlite3.Error as e:
        print(e)

database = create_connection("mydb")

if database is not None:
    create_table(database)
    while True:
        print("\nExpenses Tracker Menu:")
        print("1. Add Product")
        print("2. View Products")
        print("3. Calculate Total Expenses")
        print("4. Delete Product")

```

```

print("5. Exit")

choice = input("Enter your choice: ")

if choice == "1":
    product_name = input("Enter product name: ")
    product_cost = int(input("Enter product cost: "))
    add_product(database, product_name, product_cost)

elif choice == "2":
    products = view_products(database)
    if products:
        print("ID\tName\tCost\t ")
        for product in products:
            print(f"{product[0]}\t{product[1]}\t{product[2]}")
    else:
        print("No products found.")

elif choice == "3":
    total = calculate_total_cost(database)
    if total is not None:
        print(f"Total expenses: {total}")
    else:
        print("Error calculating total expenses.")

elif choice == "4":
    p_id = int(input("Enter product id: "))

```

```
delete_product(database, p_id)
```

```
elif choice == "5":
```

```
break
```

INPUT AND OUTPUT:

Expenses Tracker Menu:

1. Add Product
2. View Products
3. Calculate Total Expenses
4. Delete Product
5. Exit

Enter your choice: 1

Enter product name: Laptop

Enter product cost: 80000

Expenses Tracker Menu:

1. Add Product
2. View Products
3. Calculate Total Expenses
4. Delete Product
5. Exit

Enter your choice: 1

Enter product name: Smartphone

Enter product cost: 15000

Expenses Tracker Menu:

1. Add Product
2. View Products
3. Calculate Total Expenses
4. Delete Product
5. Exit

Enter your choice: 2

ID	Name	Cost
----	------	------

1	Laptop	80000
---	--------	-------

2	Smartphone	15000
---	------------	-------

Expenses Tracker Menu:

1. Add Product
2. View Products
3. Calculate Total Expenses
4. Delete Product
5. Exit

Enter your choice: 3

Total expenses: 95000

Expenses Tracker Menu:

1. Add Product
2. View Products
3. Calculate Total Expenses
4. Delete Product
5. Exit

Enter your choice: 4

Enter product id: 1

Expenses Tracker Menu:

1. Add Product
2. View Products
3. Calculate Total Expenses
4. Delete Product
5. Exit

Enter your choice: 2

ID	Name	Cost
2	Smartphone	15000

Expenses Tracker Menu:

1. Add Product
2. View Products
3. Calculate Total Expenses
4. Delete Product
5. Exit

Enter your choice: 5

Result :

Thus, a program which allows the user to track expenses by adding expenses, viewing expense history, and calculating total expenses has been implemented and tested. The program demonstrates basic CRUD operations using SQLite for expense tracking.