Ex no: 01

Simple Arithmetic Calculator

Aim:

To write a Python program to create a simple calculator that can perform all arithmetic operations.

Procedure:

Step 1: Start the program.

Step 2: Open Python IDLE.

Step 3: Print the program title using a print statement.

Step 4: Declare the function calculate() by using the def statement.

Step 5: Get three inputs by using an input statement; two inputs are numeric, and another is an operator.

Step 6: Call calculate() method.

Step 7: Using conditional statements to check which operation to perform and return the output value.

Step 8: Save the program.

Step 9: Run the program by using a Python compiler or interpreter.

Step 10: Display the output.

Step 11: Stop the program.

Source Code:

```
print("ARITHMETIC CALCULATOR")
def calculate(n1,n2,op):
    if op == '+':
        result = n1+n2
    elif op == '-':
        result = n1-n2
    elif op == '*':
        result =  n1*n2
    elif op == '/':
        result = n1/n2
    elif op=='%':
        result =  n1%n2
    return result
number1 = float(input('Enter first number: '))
op = input('Enter operator (+,-,*,/,%): ')
number2 = float(input('Enter second number: '))
result=calculate(number1,number2,op)
print(number1,op,number2,'=',result)
```

Sample Input/Output:

```
ARITHMETIC CALCULATOR
Enter first number: 5
Enter operator (+,-,*,/,%): +
Enter second number: 8
5.0 + 8.0 = 13.0
```

Result:

Thus, the above-mentioned simple calculator program was created successfully, and the output was verified.

Ex no: 02

Control Flow Statements

Aim:

To write a Python program to find the greatest among three numbers, using control flow tools such as if statements.

Procedure:

Step 1: Start the program.

Step 2: Open Python IDLE.

Step 3: Print the program title using a print statement.

Step 4: Use input statements to obtain input a, b, and c from the user.

Step 5: Using an if statement to determine if 'A' is the most dominant among all.

Step 6: Using elif statements to check if 'B' is the greatest among all and to check if 'C' is the greatest among all.

Step 7: If there are no matching conditions, 'else' can be used to print 'All the values are equal'.

Step 8: Save the program.

Step 9: Use a Python compiler or interpreter to execute the program.

Step10: Display the output.

Step11: Stop the program.

Source Code:

```
print("CONTROL FLOW STATEMENTS")
print("Finding Greatest Number")
a=int(input("Enter the Value of A:"))
b=int(input("Enter the Value of B:"))
c=int(input("Enter the Value of C:"))
if a>b and a>c:          #A greater than B and Greater than C
    print("A is Greatest Among All")
elif b>a and b>c:        #B greater than A and Greater than C
    print("B is Greatest Among All")
elif c>a and c>b:        #C greater than A and Greater than B
    print("C is Greatest Among All")
```

```
else:          #A,B,C are Equal
    print("All the values are Equal")
```

Sample Input/Output:

```
CONTROL FLOW STATEMENTS
Finding Greatest Number
Enter the Value of A:50
Enter the Value of B:20
Enter the Value of C:10
A is Greatest Among All

CONTROL FLOW STATEMENTS
Finding Greatest Number
Enter the Value of A:100
Enter the Value of B:100
Enter the Value of C:100
All the values are Equal
```

Result:

Thus, the control flow statement program for finding the greatest number was created successfully and the output was verified.

Ex no: 03

Program Using For Loop

Aim:

To write a Python program that displays a multiplication table using a for loop.

Procedure:

Step1: Start the program.

Step 2: Open Python IDE.

Step 3: Print the program title 'Multiplication Table' using the 'Print' statement.

Step 4: Using the input function, prompt the user to input a variable num, n.

Step 5: Determine the value for i=1.

Step 6: Start a for loop to iterate through the numbers 1 to n using the range(1,n). Let the variable i represent the current iteration number.

Step 7: Calculate the product of number and i inside the loop and print a formatted string that shows the multiplication table.

Step 8: Save the program.

Step 9: Use a Python compiler or interpreter to execute the program.

Step 10: Display the output for the Multiplication table.

Step 11: Stop the program.

Source Code:

```
# Multiplication table Using for loop
print("\n\nMultiplication Table Using For loop ")

# Reading the input
num = int(input("\nEnter the value for Multiplication Table: "))
n = int(input("\nEnter the Limit: "))
i = 1

# using for loop to iterate multiplication table n times
print("\n\nMultiplication Table:")
for i in range(1, n+1):
   print(i,'x',num,'=',num*i)
```

<u>Output:</u>

```
Multiplication Table Using For loop
Enter the value for Multiplication Table: 5
Enter the Limit: 10
Multiplication Table:
1 x 5 = 5
2 x 5 = 10
3 x 5 = 15
4 x 5 = 20
5 x 5 = 25
6 x 5 = 30
7 x 5 = 35
8 x 5 = 40
9 x 5 = 45
10 x 5 = 50
```

<u>Result:</u>

Thus, the multiplication table program mentioned above was successfully displayed using a for loop and the output was verified.

Ex no: 04

Python Data Structures: Stack, Queue and Tuple Implementation

---

Aim:

To develop a Python program that implements the following data structures:

a. Use list as stack.

b. Use list as queue.

c. Tuple and sequence.

Procedure:

Step 1: Start the program.

Step 2: Open Python IDE.

Step 3: Print the program title 'Python Data Structures: Stack, Queue, and Tuple Implementation' using the print statement.

Step 4: Read the choice value from the user using an input statement.

Step 5: Define the function switch_case() and pass the choice value as an argument.

Step 6:

If choice==1,

a. Create an empty list to serve as a stack.
b. Push the element into the stack using the append() method.
c. Display Current Elements in a stack using a print statement.
d. Remove elements from a stack using the pop() method.
e. Display elements in a stack after using the pop() method.

If choice==2,

a. Import the necessary module and create an empty list to serve as a queue.
b. Enqueue an element into the queue using the append() method.
c. Display current elements in a queue using a print statement.
d. Remove elements from a queue using the popleft() method.
e. Display elements in a queue after they have been removed.

If choice==3,

a. Declare a tuple variable and assign different values.
b. Access tuple elements using subscript[] as an index.
c. Access a subset of elements using the slicing technique.

d. Add more elements to a tuple using the concatenation operator (+) by creating a new tuple.

If choice>3,

a. Display "Invalid Choice"

Step 7: Save the program.

Step 8: Use a Python compiler or interpreter to execute the program.

Step 9: Display the output of our choice.

Step 10: Stop the program.

Source Code:

```
from collections import deque
def switch_case(choice):
    if choice==1:
        print("Stack Implementation")
        # Using list as a stack
        stack = []
        # Push elements onto the stack
        stack.append(10)
        stack.append(20)
        stack.append(30)

        print("Before Removed Element in Stack:", stack)
        # Pop elements from the stack
        pop_element = stack.pop()
        print("Popped Element:", pop_element)

        # Current stack after popping
        print("After Removed Element in Stack:", stack)
    elif choice==2:
        print("Queue Implementation")
        # Using list as a queue
        queue = deque()
    # Enqueue elements into the queue
    queue.append(10)
    queue.append(20)
    queue.append(30)

    print("Before Removed Element in a Queue:", queue)

    # Dequeue elements from the queue
    dequeue_element = queue.popleft()
    print("Dequeued Element:", dequeue_element)

    # Current queue after dequeuing
    print("After Removed Element in a Queue:", queue)
elif choice==3:
    print("Tuple as a Sequence")
    # Defining a tuple
    my_tuple = (1, 2, 'a', 'b', 3.14)

    # Accessing elements
    first_element = my_tuple[0]
    third_element = my_tuple[2]

    # Slicing
    subset = my_tuple[1:4]

    # Concatenation
    new_tuple = my_tuple + ('x', 'y', 'z')

    # Printing results
    print("Original Tuple:", my_tuple)
    print("First Element:", first_element)
    print("Subset:", subset)
```

```
        print("New Tuple:", new_tuple)
    else:
        print("Invalid choice")
print("Python Data Structures: Stack, Queue and Tuple Implementation")
print("1.Using list as Stack")
print("2.Using list as Queue")
print("3.Tuple as Sequence")
choice=int(input("Enter your choice:"))
switch_case(choice)
```

Sample Input/Output:

```
Python Data Structures: Stack, Queue and Tuple Implementation
1.Using list as Stack
2.Using list as Queue
3.Tuple as Sequence
Enter your choice:1
Stack Implementation
Before Removed Element in Stack: [10, 20, 30]
Popped Element: 30
After Removed Element in Stack: [10, 20]

Python Data Structures: Stack, Queue and Tuple Implementation
1.Using list as Stack
2.Using list as Queue
3.Tuple as Sequence
Enter your choice:2

Queue Implementation
Before Removed Element in a Queue: deque([10, 20, 30])
Dequeued Element: 10
After Removed Element in a Queue: deque([20, 30])
```

```
Python Data Structures: Stack, Queue and Tuple Implementation
1.Using list as Stack
2.Using list as Queue
3.Tuple as Sequence
Enter your choice:3
Tuple as a Sequence
Original Tuple: (1, 2, 'a', 'b', 3.14)
First Element: 1
Subset: (2, 'a', 'b')
New Tuple: (1, 2, 'a', 'b', 3.14, 'x', 'y', 'z')

Python Data Structures: Stack, Queue and Tuple Implementation
1.Using list as Stack
2.Using list as Queue
3.Tuple as Sequence
Enter your choice:4
Invalid choice
```

Result:

Thus, the implementation of the above-mentioned data structures will be successfully completed.

Ex no: 05

Program using Modules

Aim:

To write a Python program to create a new module for mathematical operations and use it in another program.

Procedure:

Step 1: Start the program.

Step 2: Open Python IDE.

Step 3: Create a new module and create user-defined functions, namely sum, diff, product, and quo.

Step 4: Save the module program (maths.py).

Step 5: Using an import statement to include the module in our main program.

Step 6: Use a Print statement to print the program title.

Step 7: Declare two variables, namely num1 and num2, to read user inputs.

Step 8: Call the module functions using the dot operator (.).

Step 9: Save the main program (main.py).

Step 10: Use a Python compiler or interpreter to execute the program.

Step 11: Show the output for each module function.

Step 12: Stop the program.

Source Code:

Module Program (maths.py)

```
#Module name maths.py
#Creating functions in a module
def sum(a,b):
   return a+b;
def diff(a,b):
   return a-b;
def product(a,b):
   return a*b;

def quo(a,b):
   if(b==0):
       return "Zero cannot be divided"
   return a//b;
```

Main Program (main.py)

```
#importing module in our program
import maths
print("Mathematical Operations Using Module")
# Getting user input
num1=int(input("Enter the First Value:"))
num2=int(input("Enter the Second Value:"))
 #Calling functions from the module
print("Sum of two numbers is:",maths.sum(num1,num2))
print("Difference of two numbers is:",maths.diff(num1,num2))
print("Product of two numbers is:",maths.product(num1,num2))
print("Division of two numbers (Quotient) is:",maths.quo(num1,num2))
```

Sample Input/Output:

```
Mathematical Operations Using Module
Enter the First Value:10
Enter the Second Value:6
Sum of two numbers is: 16
Difference of two numbers is: 4
Product of two numbers is: 60
Division of two numbers (Quotient) is: 1
```

Result:

Thus, the above-mentioned Python module must be created successfully and perform mathematical operations successfully.

Ex no: 06

Files and Directories

Aim:

To develop a Python program which reads and writes files, as well as creates and deletes directories.

Procedure:

Step 1: Start the program.

Step 2: Open Python IDE.

Step 3: Import necessary modules.

Step 4: Use a Print statement to print the program title.

Step 5: Determine the directory path. Using an exception handling mechanism to handle errors during directory creation.

Step 6: Using the os.mkdir() method to create a new directory on the specified path.

Step 7: Using the open() method to open a new file with a specified file name and file mode.

Step 8: Retrieve the input string from the user and save it in a specified file.

Step 9: Close the file using the close() method.

Step 10: Open the file in read-only mode.

Step 11: Use a print statement to display the file's content.

Step 12: Close the file using the close() method.

Step 13: Read the user's choice value to delete the file and directory.

Step 14: Delete the file using the os.remove() method.

Step 15: Delete the directory using the os.rmdir() method.

Step 16: Save the program.

Step 17: Use a Python compiler or interpreter to execute the program.

Step 18: Display the output and check if the file and directory is created or not.

Step 19: Stop the program.

Source Code:

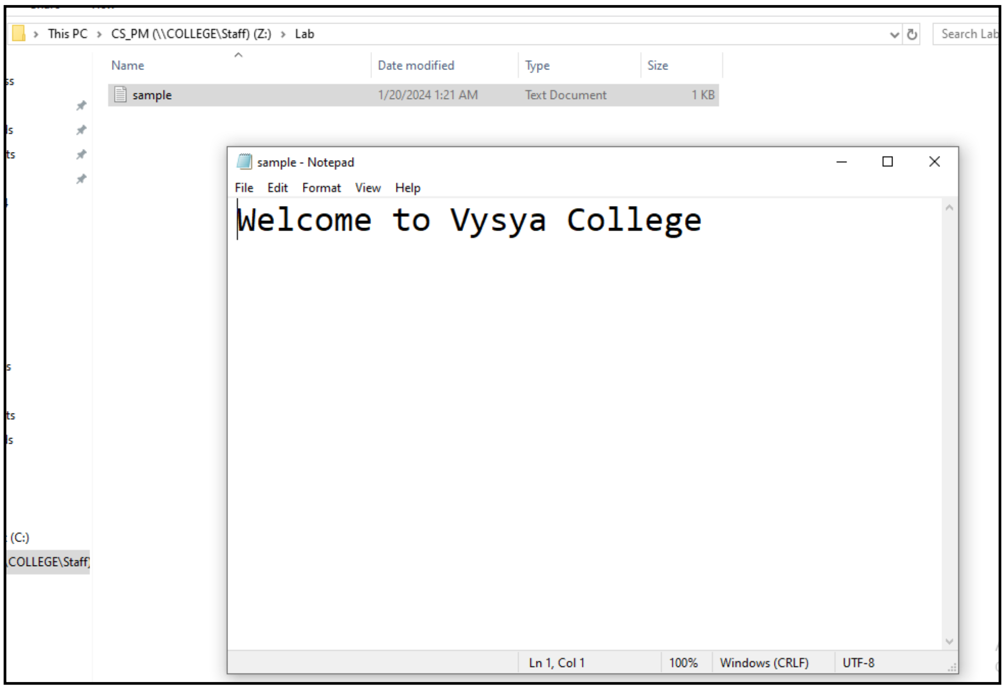
```
import os
print("FILE HANDLING PROGRAM IN PYTHON")
```

```
# Create a Directory
path = "Z:\Lab"
try:
    os.mkdir(path)
except OSError:
    print ("Creation of the directory %s failed" % path)
else:
    print ("Successfully created the directory %s " % path)

#Writing the content
file=open("Z:\Lab\sample.txt","w")
str1=input("Enter your String:")
file.write(str1)
file.close()

#Reading the content
file=open("Z:\Lab\sample.txt","r")
print("File Content:\n",file.read())
file.close()

# Remove the Directory
choice=int(input("Do you want to remove the directory press 1:"))
if(choice==1):
    os.remove("Z:\Lab\sample.txt")
    os.rmdir(path)
    print ("Successfully Deleted the directory %s " % path)
else:
    print ("Deletion of the directory %s failed" % path)
```

Sample Input/Output:

```
FILE HANDLING PROGRAM IN PYTHON
Successfully created the directory Z:\Lab
```

```
Enter your String: Welcome to Vysya College
File Content:
 Welcome to Vysya College
```


This PC › CS_PM (\\COLLEGE\Staff) (Z:) › Lab
Search Lab
(C:)
COLLEGE\Staff
sample - Notepad
File Edit Format View Help

```
Name      Date modified        Type            Size
sample    1/20/2024 1:21 AM    Text Document   1 KB
```

```
Welcome to Vysya College
```

```
Ln 1, Col 1    100%    Windows (CRLF)    UTF-8
```

```
Do you want to remove the directory press 1:1
Successfully Deleted the directory Z:\Lab
```

Result:

Thus, the above-mentioned file and directory were created and also deleted successfully.

Ex no: 07

Exception Handling

Aim:

To create a python program that can handle the exception.

Procedure:

Step 1: Start the program.

Step 2: Open Python IDE.

Step 3: Print the program title using a **Print** statement.

Step 4: Using a try block to handle an exception.

Step 5: Get user input for number1 and number2 within the eval() method.

Step 6: Find the quotient and store it in the result.

Step 7: If number1 value is equal to 0, it will display the runtime error message.

Step 8: If the comma is omitted when reading the input, it will display the syntax error.

Step 9: If number2 value is equal to 0, it will display the zero division error.

Step 10: If number1 and number2 are read by comma, it will find the result and display the result value; otherwise, it will display the corresponding exception message.

Step 11: Save the program.

Step 12: Use a Python compiler or interpreter to execute the program.

Step 13: Display the output and check if the exceptions are handled correctly.

Step 14: Stop the program.

Source Code:

```
print("EXCEPTION HANDLING")
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("\t******")
```

__**Sample Input/Output:**__

Output 1:

```
EXCEPTION HANDLING
Enter two numbers separated by a comma: 8,3
Result is 2.6666666666666665
No Exceptions
        ******
```

Output 2:

```
EXCEPTION HANDLING
Enter two numbers separated by a comma: 8,0
Division by Zero
        ******
```

Output 3:

```
EXCEPTION HANDLING
Enter two numbers separated by a comma: 8 6
A comma may be Missing in the Input
        ******
```

Output 4:

```
EXCEPTION HANDLING
Enter two numbers separated by a comma: 0,7
```

```
Result is 0.0
May be Meaningless
    ******
```

Output 5:

```
EXCEPTION HANDLING
Enter two numbers separated by a comma: 4
Something Wrong in the Input
    ******
```

Result:

Thus, the above-mentioned exception handling program was created successfully, and the output was verified.

Ex no: 08

Program using Class

Aim:

To create a Python program that utilizes classes to determine the area of a square, rectangle, and circle.

Procedure:

Step 1: Start the program.

Step 2: Open Python IDE.

Step 3: Creating a new class by using the keyword class followed by the class name Area.

Step 4: Add three different methods to the class: circle, square, and rectangle.

Step 5: Display the program title using the Print statement.

Step 6: Create an object for a class named obj.

Step 7: Use input statements to obtain necessary inputs from the user.

Step 8: Call the class methods using object(obj.) followed by the method name.

Step 9: Save the program.

Step 10: Use a Python compiler or interpreter to execute the program.

Step 11: Display the output for square, rectangle, and circle areas.

Step 12: Stop the program.

Source Code:

```
class Area:
    def circle(self,r):
        print("Area of the Circle is :",3.14*r*r)
    def square(self,a):
        print("Area of the Square is :",(a*a))
    def rectangle(self,l,b):
        print("Area of the Rectangle is :",(l*b))
print("Program Using Class")
obj=Area()
radius=float(input("Enter the radius :"))
obj.circle(radius)
```

```
side=int(input("\nEnter the side of a Square:"))
obj.square(side)

length=int(input("\nEnter the length of a rectangle:"))
width=int(input("Enter the width of a rectangle:"))
obj.rectangle(length,width)
```

Sample Input/Output:

```
Program Using Class
Enter the radius :6
Area of the Circle is : 113.03999999999999

Enter the side of a Square:8
Area of the Square is : 64

Enter the length of a rectangle:4
Enter the width of a rectangle:8
Area of the Rectangle is : 32
```

Result:

Thus, the Python program mentioned above for finding the area of square, rectangle, and circle has been executed successfully.

Ex no: 09

MYSQL Address Book

Aim:

To write a Python program to connect with MySQL and create an address book.

Procedure:

Step 1: Start the program.

Step 2: Open Python IDE.

Step 3: Import mysql.connector module.

Step 4: Connect to MySQL Database:

a. Specify the host, username, password, and database name.

b. Establish a connection to the MySQL database.

Step 5: Create Table If Not Exists:

a. Create a cursor.

b. Execute query to create "contacts" table if not exists.

c. Commit changes.

Step 6: Define Functions:

a. contact_details(): Insert contact details into "contacts" table.

b. display_details(): Retrieve and display all contact details.

Step 7: Print the program title.

Step 8: Get user input for the number of contact details to be inserted.

Step 9: Call the contact_details() function to insert contact details into the database.

Step 10: Call the display_details() to retrieve and print contact details**.**

Step 11: Close the cursor and the database connection.

Step 12: Save the program.

Step 13: Use a Python compiler or interpreter to execute the program.

Step 14: Display the output.

Step 15: Stop the program.

<u>Source Code:</u>

```
import mysql.connector
#Connection to Mysql Database
connection=mysql.connector.connect(
    host='localhost',
    username='root',
    password='vysya',
    database='addressbook'
    )

#Create Table if not Already Exists
cursor=connection.cursor()
cursor.execute(" Create Table If not Exists contacts(ID int AUTO_INCREMENT PRIMARY KEY,Name varchar(255) not null,Phno varchar(20),Email varchar(255))")

#Save Changes
connection.commit()

#Add Contact Details
def contact_details(name,phno,email):
    qry="insert into contacts(NAME,PHNO,EMAIL) values (%s,%s,%s)"
    cursor.execute(qry,(name,phno,email))
    connection.commit()

#Display Contact Details
def display_details():
    qry="Select *from contacts"
    cursor.execute(qry)
    details=cursor.fetchall()
```

```
 if not details:
      print("No Contact Details Found")
   else:
      print("Contact Details in a Database are:")
      for detail in details:
         print(detail)

#main program
print("\t\tMYSQL ADDRESSBOOK")
n=int(input("Enter no of Contact details: "))
for i in range (0,n):
   print("\t\tADDRESSBOOK DETAILS: ",i+1)
   name=input("Enter your Name: ")
   phno=input("Enter your Phone Number: ")
   email=input("Enter your Email: ")
   contact_details(name,phno,email)

display_details()

#Close the Connection
cursor.close()
connection.close()
print("Connection Disconnected...")
```

Sample Input/Output:

```
MYSQL ADDRESSBOOK
Enter no of Contact details: 2
                ADDRESSBOOK DETAILS: 1
Enter your Name: Hari
Enter your Phone Number: 458652
Enter your Email: hari@gmail.com
```

```
            ADDRESSBOOK DETAILS: 2
Enter your Name: Ravi
Enter your Phone Number: 478236
Enter your Email: ravi@yahoo.co.in
Contact Details in a Database are:
(1, 'Hari', '458652', 'hari@gmail.com')
(2, 'Ravi', '478236', 'ravi@yahoo.co.in')
Connection Disconnected...
```

Result:

Thus, the Python program mentioned above for creating a MySQL address book has been executed successfully.

Ex no: 10

String Handling and Regular Expressions

Aim:

To write a Python program using string handling and regular expressions.

Algorithm:

Step 1: Start the program.

Step 2: Open Python IDLE.

Step 3: Print the program title using a print statement.

Step 4: Import re module.

Step 5: Declare the variable and allow the user to provide input.

Step 6: Create a pattern and set it aside in a variable.

Step 7: Using the findall() function, determine if the supplied string contains the proper email address.

Step 8: Using an if statement to display the correct email addresses or to display the "No email addresses found" message.

Step 9: Define the class py_reverse.

Step 10: Get the input string from the user and display it in a reversed manner by using string methods like split(), reverse(), and join().

Step 11: Save the program.

Step 12: Run the program by using a Python compiler or interpreter.

Step 13: Display the output.

Step 14: Stop the program.

Source Code:

```
import re
print("\t\t Program for String Handling and Regular Expressions”)
input_text = """
Some example Emails are:
Valid:
professor123@gmail.com
alan2004@gmail.com
Invalid:
```

```
unknown@gmail
googlegmail.com
"""

pattern = r'\S+@\S+\.\S+'
emails = re.findall(pattern, input_text)

if emails:
    print("Found email addresses:")
    for email in emails:
        print(email)
else:
    print("No email addresses found.")

class py_reverse:
   def revr(self, strs):
       sp=strs.split()
       sp.reverse()
       res=" ".join(sp)
       return res

str1=input("Enter a string with 2 or more words: ")
print("Reverse of string word by word: \n",py_reverse().revr(str1));
```

Sample Input/Output:

```
               Program for String Handling and Regular Expressions
Found email addresses:
professor123@gmail.com
alan2004@gmail.com
Enter a string with 2 or more words: Welcome to BCA
Reverse of string word by word:
BCA to Welcome
```

Result:

Thus, the Python program mentioned above for string handling and regular expression has been successfully executed.