Practical-8

AIM:- Write a python program to demonstrate the use of

i. if-else:- Python program to check whether the string is Symmetrical or Palindrome

ii. for() :- Program to multiply two matrices using nested loops.

iii. while() :- Program to reverse the number.

CODE:-

1.

```
def is_symmetrical(string):
    string = string.replace(" ", "").lower()
    return string == string[::-1]
def is_palindrome(string):
    return string == string[::-1]
input_string = input("Enter a string: ")
if is_symmetrical(input_string):
    print("The string is symmetrical.")
else:
    print("The string is not symmetrical.")
if is_palindrome(input_string):
    print("The string is a palindrome.")
else:
    print("The string is not a palindrome.")
```

2.

```
def matrix_multiplication(matrix1, matrix2):
 result = [[0 for _ in range(len(matrix2[0]))] for _ in range(len(matrix1))]
 for i in range(len(matrix1)):
```

```
  for j in range(len(matrix2[0])):
   for k in range(len(matrix2)):
     result[i][j] += matrix1[i][k] * matrix2[k][j]
 return result
matrix1 = [[1, 2, 3],
 [4, 5, 6],
 [7, 8, 9]]
matrix2 = [[9, 8, 7],
 [6, 5, 4],
 [3, 2, 1]]
result_matrix = matrix_multiplication(matrix1, matrix2)
print("Result of matrix multiplication:")
for row in result_matrix:
 print(row)
```

3.

```
def reverse_number(num):
 reversed_num = 0
 while num > 0:
    digit = num % 10
    reversed_num = reversed_num * 10 + digit
    num //= 10
 return reversed_num
num = int(input("Enter a number: "))
reversed_num = reverse_number(num)
print("Reversed number:", reversed_num)
```

OUTPUT:-

```
Enter a string: madam
The string is symmetrical.
The string is a palindrome.
Result of matrix multiplication:
[30, 24, 18]
[84, 69, 54]
[138, 114, 90]
Enter a number: 52
Reversed number: 25

=== Code Execution Successful ===
```

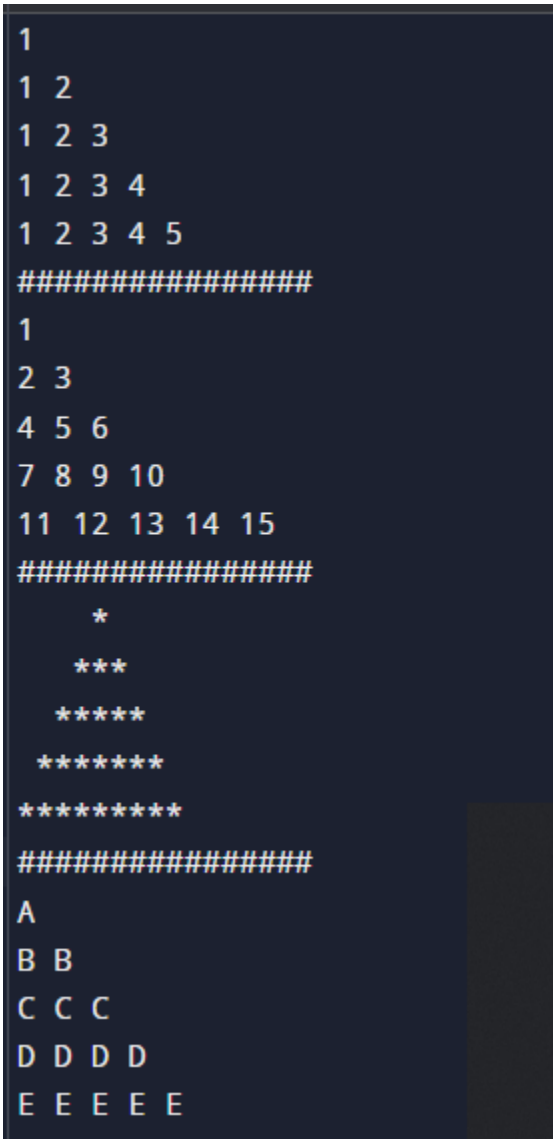
Practical-9

AIM:- Write a python program to display different patterns.

CODE:-

```
n = 5
for i in range(1, n+1):
 for j in range(1, i+1):
    print(j, end=" ")
 print()
print("--------------------")
n = 5
start = 1
for i in range(1, n+1):
 for j in range(start, start+i):
    print(j, end=" ")
 print()
 start += i
print("--------------------")
n = 5
for i in range(1, n + 1):
 print(' ' * (n - i) + '*' * (2 * i - 1))
print("--------------------")
n = 5
start_char = ord('A')
for i in range(n):
 for j in range(i+1):
     print(chr(start_char + i), end=" ")
 print()
```

OUTPUT:-


1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
##################
1
2 3
4 5 6
7 8 9 10
11 12 13 14 15
##################
*
***
*****
****
****
##################
A
B B
C C C
D D D D
E E E E E

Practical-10

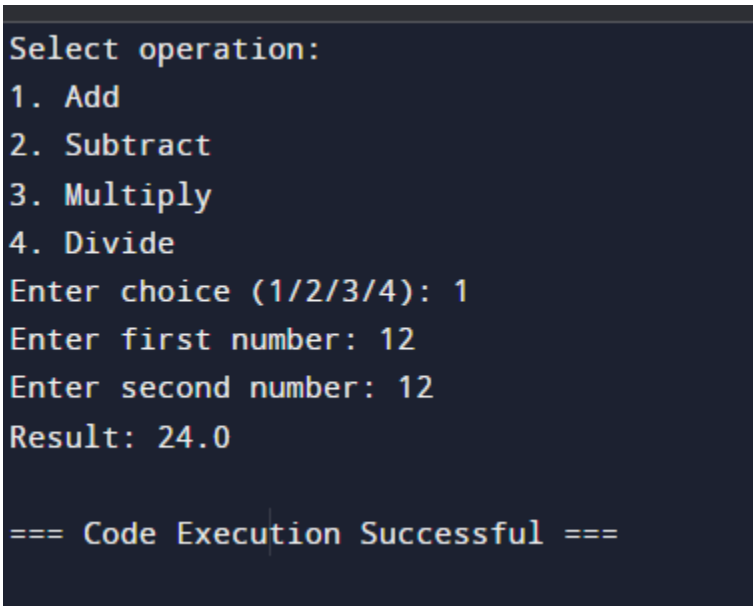
AIM:- Write a python program to demonstrate the use of user-defined functions with single, multiple and arbitrary arguments. WAP to design simple calculator.

CODE:-

```
def add(x, y):
 return x + y
def subtract(x, y):
 return x - y
def multiply(x, y):
 return x * y
def divide(x, y):
 if y == 0:
    return "Cannot divide by zero!"
 return x / y
print("Select operation:")
print("1. Add")
print("2. Subtract")
print("3. Multiply")
print("4. Divide")
choice = input("Enter choice (1/2/3/4): ")
num1 = float(input("Enter first number: "))
num2 = float(input("Enter second number: "))
if choice == '1':
    print("Result:", add(num1, num2))
elif choice == '2':
    print("Result:", subtract(num1, num2))
```

```
elif choice == '3':
    print("Result:", multiply(num1, num2))
elif choice == '4':
 print("Result:", divide(num1, num2))
else:
 print("Invalid input")
```

OUTPUT:-



Practical-11

AIM:- Create a class named employee having attributes - emp_name, emp_age, and emp_city. Create a method named get_data() in employee class that takes user input for these attributes. Derive a class named emp_derived() from the employee class, having an __init__() method that displays the attributes of the employee class upon instantiation.

CODE:-

```
class Employee:
 def __init__(self):
     self.emp_name = ""
     self.emp_age = 0
     self.emp_city = ""
 def get_data(self):
    self.emp_name = input("Enter employee name: ")
    self.emp_age = int(input("Enter employee age: "))
    self.emp_city = input("Enter employee city: ")
class EmpDerived(Employee):
 def __init__(self):
    super().__init__()
 def display_info(self):
     print("Employee Name:", self.emp_name)
     print("Employee Age:", self.emp_age)
     print("Employee City:", self.emp_city)
emp = EmpDerived()
emp.get_data()
emp.display_info()
```

OUTPUT:-

```
Enter employee name: shubham
Enter employee age: 19
Enter employee city: surat
Employee Name: shubham
Employee Age: 19
Employee City: surat

=== Code Execution Successful ===
```

Practical-12

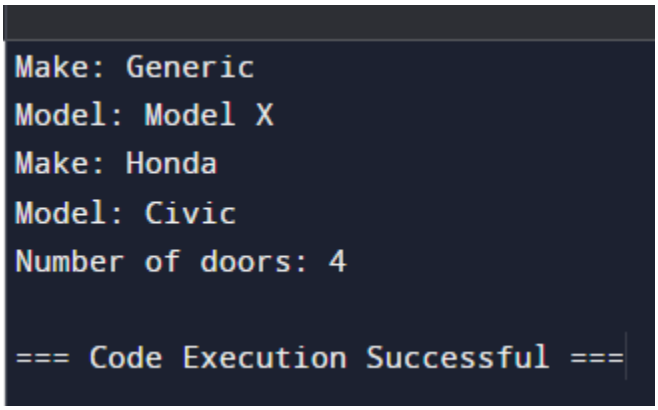
AIM:- Write a python program to show the need of inheritance and encapsulation. The display() method that prints class attribute values along with attributes of its super class.

CODE:-

```
class Vehicle:
  """  Base class for vehicles.  """
  def __init__(self, make, model):
    self.__make = make  # Encapsulated attribute
    self.model = model  # Public attribute
  def display(self):
    """Prints the vehicle's make and model."""
    print(f"Make: {self.__make}")
    print(f"Model: {self.model}")
class Car(Vehicle):
  """  Derived class representing cars.  """
  def __init__(self, make, model, num_doors):
    super().__init__(make, model)  # Inheriting from Vehicle class
    self.num_doors = num_doors
  def display(self):
    """    Overrides the display() method from the base class.
   Prints specific information about the car.
    """
    super().display()  # Calling the base class display method
    print(f"Number of doors: {self.num_doors}")


# Create a Vehicle object (limited functionality due to encapsulation)
```

```
vehicle = Vehicle("Generic", "Model X")

# Create a Car object

car = Car("Honda", "Civic", 4)

# Calling display methods

vehicle.display()

car.display()
```

OUTPUT:-



Practical-13

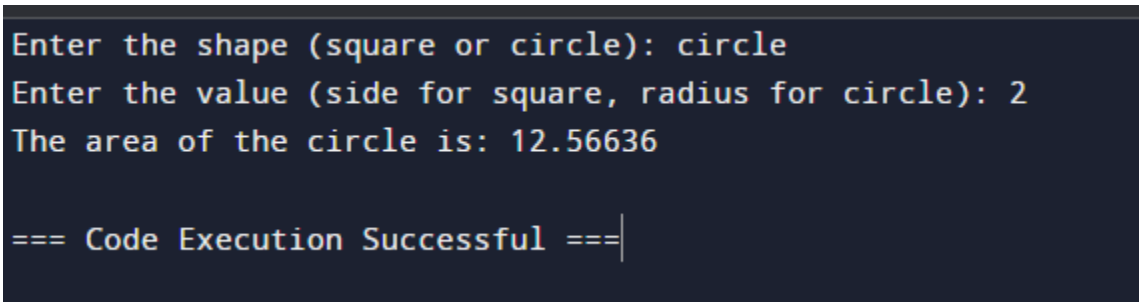
AIM:- Write a python program to create a class named area. Define a class method find_area() that can find areas of different shapes whose value is given by the user. Invoke the class method by instantiation and prove method overloading.

CODE:-

```
class Area:
  @classmethod
  def find_area(cls, shape, *args):

    if shape.lower() == "square":
      if len(args) != 1:
        raise ValueError("Square requires one side length")
      side = args[0]
      return side * side
    elif shape.lower() == "circle":
      if len(args) != 1:
        raise ValueError("Circle requires one radius")
      radius = args[0]
      return 3.14159 * radius * radius
    else:
      raise ValueError("Unsupported shape")

shape = input("Enter the shape (square or circle): ")
try:
  value = float(input("Enter the value (side for square, radius for circle): "))
except ValueError:
```

```
  print("Invalid input. Please enter a number.")
  exit()
area = Area.find_area(shape, value)
print(f"The area of the {shape} is: {area}")
```

OUTPUT:-



Practical-14

AIM:- Write a python program to demonstrate the use of method overriding.

CODE:-

```
class Shape:
  def calculate_area(self):
    print("Area calculation not implemented in base class.")
class Square(Shape):
  def __init__(self, side_length):
    self.side_length = side_length
  def calculate_area(self):
    area = self.side_length * self.side_length
    print(f"Area of square: {area}")
class Circle(Shape):
  def __init__(self, radius):
    self.radius = radius
  def calculate_area(self):
    area = 3.14159 * self.radius * self.radius
    print(f"Area of circle: {area}")
square = Square(5)
circle = Circle(3)
square.calculate_area()
circle.calculate_area()
```

OUTPUT:-

```
Area of square: 25
Area of circle: 28.274309999999996

=== Code Execution Successful ===
```

Practical-15

AIM:- Write a python program to perform basic matrix operations on user entered matrices.

CODE:-

```
def matrix_input(rows, cols):
    matrix = []
    print("Enter the elements row-wise:")
    for i in range(rows):
        row = []
        for j in range(cols):
            element = float(input(f"Enter element [{i+1}][{j+1}]: "))
            row.append(element)
        matrix.append(row)
    return matrix


def matrix_addition(matrix1, matrix2):
    if len(matrix1) != len(matrix2) or len(matrix1[0]) != len(matrix2[0]):
        print("Matrices should have the same dimensions for addition.")
        return None
    result = [[matrix1[i][j] + matrix2[i][j] for j in range(len(matrix1[0]))] for i in
range(len(matrix1))]
    return result


def matrix_subtraction(matrix1, matrix2):
    if len(matrix1) != len(matrix2) or len(matrix1[0]) != len(matrix2[0]):
        print("Matrices should have the same dimensions for subtraction.")
```

```
        return None
    result = [[matrix1[i][j] - matrix2[i][j] for j in range(len(matrix1[0]))] for i in range(len(matrix1))]
    return result

def matrix_multiplication(matrix1, matrix2):
    if len(matrix1[0]) != len(matrix2):
        print("Number of columns in the first matrix should be equal to the number of rows in the second matrix for multiplication.")
        return None
    result = [[sum(matrix1[i][k] * matrix2[k][j] for k in range(len(matrix2))) for j in range(len(matrix2[0]))] for i in range(len(matrix1))]
    return result

def print_matrix(matrix):
    for row in matrix:
        print(row)

def main():
    print("Matrix Operations Program")
    choice = input("Choose operation:\n1. Addition\n2. Subtraction\n3. Multiplication\nEnter choice (1/2/3): ")

    rows1 = int(input("Enter number of rows for matrix 1: "))
    cols1 = int(input("Enter number of columns for matrix 1: "))
    matrix1 = matrix_input(rows1, cols1)

    rows2 = int(input("Enter number of rows for matrix 2: "))
    cols2 = int(input("Enter number of columns for matrix 2: "))
```

```
    matrix2 = matrix_input(rows2, cols2)

    if choice == '1':
        result = matrix_addition(matrix1, matrix2)
    elif choice == '2':
        result = matrix_subtraction(matrix1, matrix2)
    elif choice == '3':
        result = matrix_multiplication(matrix1, matrix2)
    else:
        print("Invalid choice.")
        return

    if result:
        print("Resultant matrix:")
        print_matrix(result)

if __name__ == "__main__":
    main()
```

OUTPUT:-

```
Matrix Operations Program
Choose operation:
1. Addition
2. Subtraction
3. Multiplication
Enter choice (1/2/3): 1
Enter number of rows for matrix 1: 2
Enter number of columns for matrix 1: 2
Enter the elements row-wise:
Enter element [1][1]: 2
Enter element [1][2]: 2
Enter element [2][1]: 2
Enter element [2][2]: 2
Enter number of rows for matrix 2: 2
Enter number of columns for matrix 2: 2
Enter the elements row-wise:
Enter element [1][1]: 3
Enter element [1][2]: 3
Enter element [2][1]: 3
Enter element [2][2]: 3
Resultant matrix:
[5.0, 5.0]
[5.0, 5.0]

=== Code Execution Successful ===
```