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.

PROBLEM:

1)

```
s = input("Enter a string: ")
# Check if the string is symmetrical
if s == s[::-1]:
    print("The string is symmetrical.")
else:
    print("The string is not symmetrical.")

# Check if the string is a palindrome
if s.lower() == s.lower()[::-1]:
    print("The string is a palindrome.")
else:
    print("The string is not a palindrome.")
```

2)

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

B = [[9, 8, 7],
     [6, 5, 4],
     [3, 2, 1]]

C = [[0, 0, 0],
     [0, 0, 0],
     [0, 0, 0]]

for i in range(len(A)):
    for j in range(len(B[0])):
        for k in range(len(B)):
            C[i][j] += A[i][k] * B[k][j]

for row in C:
    print(row)
```

3)

```
num = 1234
reversed_num = 0

while num > 0:
    digit = num % 10
    reversed_num = reversed_num * 10 + digit
    num //= 10

print("Reversed number:", reversed_num)
```

Output:

```
Enter a string: Ashu
The string is not symmetrical.
The string is not a palindrome.
[30, 24, 18]
[84, 69, 54]
[138, 114, 90]
Reversed number: 4321

[Process completed - press Enter]
```

PRACTICAL 9

AIM: Write a python program to display above patterns.

PROBLEM:

```
# Pattern 1
n = 5
for i in range(1, n+1):
    for j in range(1, i+1):
        print(j, end=" ")
    print()

print("--------------------")

# Pattern 2
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("--------------------")

# Pattern 3
n = 5
for i in range(1, n + 1):
    print(' ' * (n - i) + '*' * (2 * i - 1))

print("--------------------")

# Pattern 4
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

[Process completed - press Enter]
```

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

PROBLEM:

```
# Single Argument Function
def square(num):
    return num ** 2

# Multiple Arguments Function
def add(a, b):
    return a + b

# Arbitrary Arguments Function
def average(*args):
    total = sum(args)
    return total / len(args)

# Simple Calculator
def calculator(operation, num1, num2):
    if operation == '+':
        return num1 + num2
    elif operation == '-':
        return num1 - num2
    elif operation == '*':
        return num1 * num2
    elif operation == '/':
        if num2 != 0:
            return num1 / num2
        else:
            return "Error: Division by zero"
    else:
        return "Error: Invalid operation"

# Demo of Functions
print("Square of 5:", square(5))
print("Addition of 5 and 7:", add(5, 7))
print("Average of 3, 5, and 7:", average(3, 5, 7))

# Simple Calculator Input
```

```
operation = input("Enter operation (+, -, *, /): ")
num1 = float(input("Enter first number: "))
num2 = float(input("Enter second number: "))

# Perform Calculation
result = calculator(operation, num1, num2)
print("Result:", result)
```

Output:-

```
Square of 5: 25
Addition of 5 and 7: 12
Average of 3, 5, and 7: 5.0
Enter operation (+, -, *, /): -
Enter first number: 4
Enter second number: 2
Result: 2.0

[Process completed - press Enter]
```

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.

PROBLEM:

```
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: Ashu
Enter employee age: 18
Enter employee city: Surat
Employee Name: Ashu
Employee Age: 18
Employee City: Surat

[Process completed - press Enter]
```

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.

PROBLEM:

```
class Animal:
    def __init__(self, name, age):
        self._name = name
        self._age = age

    def display(self):
        print("Name:", self._name)
        print("Age:", self._age)


class Dog(Animal):
    def __init__(self, name, age, breed):
        super().__init__(name, age)
        self._breed = breed

    def display(self):
        super().display()
        print("Breed:", self._breed)


animal = Animal("Shubham", 5)
animal.display()
print()

donkey = Dog("Shubham", 5, "Donkey")
donkey.display()
```

Output:-

```
Name: Shubham
Age: 5

Name: Shubham
Age: 5
Breed: Donkey

[Process completed - press Enter]
```

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

PROBLEM: