2. Class Practice questions

Q.1 Create a class MaxFinder that identifies the largest number in a list.
Input:[45,2,49,3]
Output:49

```
class MaxFinder:
    def __init__(self, numbers):
        self.numbers = numbers

    def find_max(self):
        if not self.numbers:
            return None   # handle empty list
        return max(self.numbers)

nums = [45, 2, 49, 3]
finder = MaxFinder(nums)
print("Input:", nums)
print("Output:", finder.find_max())
```

Q.2 Last Digit in Words: Write a class with a method that takes an integer and prints the last digit of that number in words.
Input:[123]
Output:Three

```
class LastDigitInWords:
    digit_words = ["Zero", "One", "Two", "Three", "Four",
                   "Five", "Six", "Seven", "Eight", "Nine"]
    def __init__(self, number):
        self.number = number
    def print_last_digit_word(self):
        last_digit = abs(self.number) % 10  # get last digit
        print(self.digit_words[last_digit])

num = 123
obj = LastDigitInWords(num)
print("Input:", num)
print("Output:", end=" ")
obj.print_last_digit_word()
```

Q.3 Student Grade Calculator: Implement a Student class with attributes for name and a list of marks(for 5 subjects). Include a method to calculate the average and determine the grade.

Input:[85,78,90,88,91]

Output:Joy grade=A

```python
class Student:
    def __init__(self, name, marks):
        self.name = name
        self.marks = marks  # list of marks for 5 subjects

    def calculate_average(self):
        return sum(self.marks) / len(self.marks)

    def determine_grade(self):
        avg = self.calculate_average()
        if avg >= 90:
            return 'A+'
        elif avg >= 80:
            return 'A'
        elif avg >= 70:
            return 'B'
        elif avg >= 60:
            return 'C'
        elif avg >= 50:
            return 'D'
        else:
            return 'F'

    def display_result(self):
        grade = self.determine_grade()
        print(f"{self.name} grade={grade}")

marks = [85, 78, 90, 88, 91]
student = Student("Joy", marks)
student.display_result()
```

Q.4 Define an abstract base class Polygon with an abstract method area. Implement this in derived classes Rectangle and Triangle.

Output:Rectangle Area: 200

Triangle Area: 50.0

```python
from abc import ABC, abstractmethod
```

```
class Polygon(ABC):
    @abstractmethod
    def area(self):
        pass

class Rectangle(Polygon):
    def __init__(self, length, width):
        self.length = length
        self.width = width

    def area(self):
        return self.length * self.width

class Triangle(Polygon):
    def __init__(self, base, height):
        self.base = base
        self.height = height

    def area(self):
        return 0.5 * self.base * self.height

rect = Rectangle(20, 10)
tri = Triangle(10, 10)
print("Rectangle Area:", rect.area())
print("Triangle Area:", tri.area())
```

Q.5 Design a class that tracks how many objects have been created from it and has a method to display this count.

```
class ObjectCounter:
    count = 0   # class variable to track number of objects

    def __init__(self):
        ObjectCounter.count += 1

    @classmethod
    def display_count(cls):
        print("Number of objects -", cls.count)

obj1 = ObjectCounter()
obj2 = ObjectCounter()
obj3 = ObjectCounter()

ObjectCounter.display_count()
```

Q.6 Implement a class Account with a private attribute balance and provide methods to deposit and withdraw safely, checking for sufficient funds.

Ouput: Deposited: 50, New Balance: 150
Withdrew: 100, Remaining Balance: 50
Insufficient funds

```
class Account:
    def __init__(self, initial_balance=0):
        self.__balance = initial_balance   # private attribute

    def deposit(self, amount):
        if amount > 0:
            self.__balance += amount
            print(f"Deposited: {amount}, New Balance: {self.__balance}")
        else:
            print("Deposit amount must be positive")

    def withdraw(self, amount):
        if amount <= self.__balance:
            self.__balance -= amount
            print(f"Withdrew: {amount}, Remaining Balance: {self.__balance}")
        else:
            print("Insufficient funds")

    def get_balance(self):
        return self.__balance

acc = Account(100)
acc.deposit(50)
acc.withdraw(100)
acc.withdraw(200)
```

Q.7 Static and Class Methods Demonstrate the use of static and class methods in a class Calculator with methods to add and multiply numbers.

Output: Addition:add(5,3) =8
Multiplication: multi(5,4)=20

```
class Calculator:
    @staticmethod
    def add(a, b):
        return a + b

    @classmethod
    def multiply(cls, a, b):
        return a * b
print("Addition: add(5,3) =", Calculator.add(5, 3))
print("Multiplication: multi(5,4) =", Calculator.multiply(5, 4))
```

Q.8 Write a Python program to create a person class. Include attributes like name, country and date of birth. Implement a method to determine the person's age.

Output: **Person:1**
Name:Rahul Verma
Country:India
DOB:16/09/2000
Age:25

```
from datetime import datetime

class Person:
    def __init__(self, name, country, dob):
        self.name = name
        self.country = country
        self.dob = datetime.strptime(dob, "%Y-%m-%d")   # format: YYYY-MM-DD

    def get_age(self):
        today = datetime.today()
        age = today.year - self.dob.year
        if (today.month, today.day) < (self.dob.month, self.dob.day):
            age -= 1
        return age

p1 = Person("Neeraj", "India", "2000-05-15")
print("Name:", p1.name)
print("Country:", p1.country)
print("Date of Birth:", p1.dob.date())
print("Age:", p1.get_age())
```

Q.9 Write a Python program that overloads the operator + and > for a custom class.

```
class Box:
    def __init__(self, length, width, height):
        self.length = length
        self.width = width
        self.height = height
    def __add__(self, other):
        return Box(self.length + other.length, self.width + other.width,.height +
other.height)
```

```
    def __gt__(self, other):
        return self.volume() > other.volume()

    def volume(self):
        return self.length * self.width * self.height

def __str__(self):
        return f"Box({self.length}, {self.width}, {self.height})"

obj1 = Box(2, 3, 4)
obj2 = Box(1, 2, 5)

obj3 = obj1 + obj2
print("After addition:", obj3)

if obj1 > obj2:
    print("obj1 is bigger than obj2")
else:
    print("obj2 is bigger than or equal to obj1")
```

Q.10 Write a Python program that checks if one class is a subclass of another.

Output: 1 Staff
2 Teacher(staff)

```
class Staff:
    def __init__(self):
        print("1 Staff")

class Teacher(Staff):
    def __init__(self):
        super().__init__()
        print("2 Teacher(Staff)")
obj = Teacher()
print("Is Teacher a subclass of Staff:", issubclass(Teacher, Staff))
print("Is Staff a subclass of Teacher:", issubclass(Staff, Teacher))
```