Question no 1:

**Problem Statement**

Aria is working with a pair of numbers X and Y for her math project. She needs a program to calculate the addition and subtraction of the negated values of X and Y.

Help her create a program that takes X and Y as input, negates them using the - operator overloading, and then calculates and prints the addition and subtraction of the negated values.

**For Example:**

Assume that X = 10 and Y = 20. The negated values are X = -10 & Y = -20.

* Addition: (-10) + (-20) = -30
* Subtraction: (-10) - (-20) = 10

**Input format :**

The input consists of two space-separated integers: X and Y.

**Output format :**

The output displays two space-separated integers:

* Addition of the negated values X and Y
* Subtraction of the negated values X and Y

**Refer to the sample output for formatting specifications.**

**Code constraints :**

-1000 ≤ X, Y ≤ 1000

**Sample test cases :**

**Input 1 :**

10 20

**Output 1 :**

-30 10

**Input 2 :**

-5 -6

**Output 2 :**

11 -1

**Input 3 :**

498 967

**Output 3 :**

-1465 469

Question no 2

**Problem Statement**

Ethan is driving his car with an initial velocity (in m/s) and suddenly accelerates at a constant rate (in m/s²) for a certain duration (in seconds). He wants to write a program that calculates and displays the final velocity of the car.

Help Ethan calculate the final velocity by overloading the \* operator in the **Acceleration** class.

**Formula**: Final velocity = Initial velocity + (Acceleration \* Time)

**Input format :**

The input consists of three space-separated double values:

1. Initial velocity (in m/s)
2. Acceleration (in m/s²)
3. Time (in s)

**Output format :**

The output displays a double value representing the final velocity followed by " m/s", rounded to one decimal place.

**Refer to the sample output for the formatting specifications.**

**Code constraints :**

In this scenario, the test cases fall under the following constraints:

1.0 ≤ Initial velocity ≤ 500.0

1.0 ≤ Acceleration ≤ 50.0

1.0 ≤ Time ≤ 50.0

**Sample test cases :**

**Input 1 :**

10.3 5.0 2.1

**Output 1 :**

20.8 m/s

**Input 2 :**

410.2 9.8 4.5

**Output 2 :**

454.3 m/s

Question no 3

**Problem Statement**

Arjun is passionate about creating unique cylindrical art pieces. Develop a program that takes input for the radius and height of his sculptures and calculates the volume.

**Base class Circle:** Contains a private attribute called radius, representing the radius of a circle.

**Privately inherited derived class Cylinder:** It is inherited from the Circle class and has a private attribute named height, representing the height of the cylinder.

**volume()** method inside the **Cylinder** class: To calculate the volume of the cylinder.

**Formula:** Volume = 3.14 \* radius \* radius \* height

**Input format :**

The input consists of two space-separated double values, which represent the radius and height of the cylinder.

**Output format :**

The output prints a double value, representing the volume of the cylinder, rounded to two decimal places.

**Refer to the sample outputs for the formatting specifications.**

**Code constraints :**

In this scenario, the test cases fall under the following constraints:

1.0 ≤ radius, height ≤ 50.0

**Sample test cases :**

**Input 1 :**

5.1 3.6

**Output 1 :**

294.02

**Input 2 :**

3.4 6.2

**Output 2 :**

225.05

Question no 4

**Problem Statement**

Liam, a recent graduate, is excited about purchasing his first car and is considering a loan. To calculate the total interest paid, he wants to design a class structure using multiple inheritance.

Create a class named **CarLoan** inheriting from the classes: **CarPrice** and **LoanInterestRate**. This program calculates the total interest paid over the loan period, helping Liam understand the financial implications.

* CarPrice class - stores the price as a protected attribute.
* LoanInterestRate class - stores the interest rate as a protected attribute.
* CarLoan class - calculates the total interest.

**Note**: Total Interest = price \* interest rate \* years.

**Input format :**

The first line of input consists of a double value representing the car price.

The second line consists of a double value representing the interest rate.

The third line consists of an integer representing the loan duration in years.

**Output format :**

The output prints "Total interest paid: Rs.X" where X is the total interest paid over the years, rounded off to two decimal places.

**Refer to the sample output for formatting specifications.**

**Code constraints :**

0.0 ≤ Price ≤ 1,000,000.0

0.0 ≤ Rate ≤ 100.0

0 < years ≤ 10

**Sample test cases :**

**Input 1 :**

120575.50

0.05

5

**Output 1 :**

Total interest paid: Rs.30143.88

**Input 2 :**

224564.89

3.12

2

**Output 2 :**

Total interest paid: Rs.1401284.91

**Input 3 :**

165050.52

5.52

2

**Output 3 :**

Total interest paid: Rs.182215

Question no 5

**Problem Statement**

Diego is developing a program to evaluate the efficiency of hybrid vehicles based on fuel consumption and horsepower. The program will establish a class hierarchy to represent various vehicle types and compute their efficiency ratios.

**Vehicle**: A base class with attributes for fuel consumption (liters per 100 km) and horsepower, featuring constructors to initialize these values and a method to calculate the efficiency ratio defined as horsepower/fuel consumption.

**Electric**: A subclass representing electric vehicles, inheriting attributes and methods from Vehicle.

**Gasoline**: A subclass for gasoline vehicles, also inheriting from Vehicle.

**Hybrid**: A subclass inheriting from both Electric and Gasoline, using a Virtual Base Class and Vehicle attributes for initialization. Implement the **labelEfficiency()** method in the Hybrid class to categorize the efficiency ratio as follows:

* If the efficiency ratio is less than 5.0, return "Low Efficiency".
* If the efficiency ratio is between 5.0 and 10.0 (inclusive), return "Moderate Efficiency".
* If the efficiency ratio is greater than 10.0, return "High Efficiency".

**Input format :**

The first line of input consists of an integer representing the fuel consumption of the hybrid vehicle (in liters per 100 km).

The second line of input consists of an integer representing the horsepower of the hybrid vehicle.

**Output format :**

The first line of output displays a double with two decimal places, representing the efficiency ratio of the hybrid vehicle.

The second line of output displays a string describing the efficiency based on the following conditions:

1. If the ratio < 5.0, return "Low Efficiency".
2. If 5.0 <= ratio <= 10.0, return "Moderate Efficiency".
3. If the ratio > 10.0, return "High Efficiency".

**Refer to the sample output for the formatting specifications.**

**Code constraints :**

0 ≤ fuelConsumption ≤ 100

0 ≤ horsepower ≤ 1000

**Sample test cases :**

**Input 1 :**

20

20

**Output 1 :**

1.00

Low Efficiency

**Input 2 :**

8

50

**Output 2 :**

6.25

Moderate Efficiency

**Input 3 :**

9

120

**Output 3 :**

13.33

High Efficiency