Question 1: Matrix Multiplication

A matrix is a rectangle of numbers in rows and columns. An NxM matrix has N rows and M columns.

The definition of "multiplying" two matrices together is somewhat complex, in fact. The details can be found at matrix multiplication. The main idea is this: If we want to multiply a matrix, A., by another matrix, B, we do so by multiplying the row elements of A by the column elements of B, and adding these terms together. So, for example, if A is a 2 x 3 matrix, and B is a 3 x 1 matrix, then the product C = A x B will be a 2 x 1 matrix, as shown below.

$$\underset{2\times 3}{\begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \end{bmatrix}} \times \underset{3\times 1}{\begin{bmatrix} t_{11} \\ t_{21} \\ t_{31} \end{bmatrix}} = \underset{2\times 1}{\begin{bmatrix} M_{11} \\ M_{21} \end{bmatrix}}$$

The elements of C are given by:

$$M_{11} = r_{11} * t_{11} + r_{12}*t_{21} + r_{13}*t_{31}$$

$$M_{21} = r_{21}*t_{11} + r_{22}*t_{21} + r_{23}*t_{31}$$

In this question, we will consider a simple case of matrix multiplication, which is as follows:

Write a program that reads a 1xN matrix A and an NxN matrix B from input and outputs the 1xN matrix product, C. N can be of any size >= 1.

- A is represented as a list of the integers found on the first line of input.
- B is represented as a list of N rows, each of which is a list of N integers.
- Each of the next N input lines contains the integers for one row of B.
- Note: Input is one row at a time but multiplication uses columns of B.

So, for example, if the input is:

```
2 3
1 2
3 4
```

Then the output should be:

```
11 16
```

If the input is:

```
2 3 4 5
1 2 3 4
3 4 5 6
7 8 4 2
4 3 2 1
```

Then the output should be:

```
59 63 47 39
```

Your program should check to make sure that the input data is properly formatted (in terms of the sizes of the matrices). For example, if the input is:

```
2 3 4 5
1 2 3 4
3 4 5 6
7 8 4 2
```

Then the output should be:

```
The input matrices cannot be multiplied! Rows of A not
equal to columns of B!
```

Question 2: Fancy Car

Program Specifications Write a FancyCar class to support basic operations such as drive, add gas, honk horn and start engine. fancy_car.py is provided with function stubs. Follow each step to gradually complete all instance methods.

FancyCar objects should contain five internal variables – a string to hold the model of the car, a float to hold the miles-per-gallon fuel economy, an integer to hold the odometer reading, a Boolean to say whether the engine is on or off, and a float to hold the number of gallons of fuel remaining in the gas tank.

Step 0: Complete the constructor to initialize the model and miles per gallon (MPG) with the values of the parameters. By default, the model is initialized to "Old Clunker", and MPG is initialized to 24.0. Initialize the odometer to 5 miles, the engine on/off Boolean to False, and the amount of fuel in the gas tank to be the value of the global variable FULL_TANK.

Step 1. Complete the first four instance methods to check the odometer, check the gas gauge, get the model, and get the miles per gallon.

Step 2. Complete the honk_horn() instance method to output the following statement with the car model (e.g. if the model of the car is “Honda Civic”):

```
The Honda Civic says beep beep!
```

Step 3 . Complete the drive() instance method. The odometer should increase by miles_to_drive, and the amount of gas in the tank should decrease by miles_to_drive/ MPG. You should only update the appropriate internal variables if miles_to_drive is positive.

Step 4 . Complete the add_gas() instance method. Increase the amount of gas in the tank by the amount parameter only if amount is positive, up to a maximum of FULL_TANK.

Step 5. Update drive() to determine if the car runs out of gas. If so, the parameter miles_to_drive will not be achieved and the gas tank will have 0.0 gallons. Ex: drive(100) will not be possible with only three gallons of gas and MPG of 20.0. The maximum driving distance is 60 miles with three gallons of gas and MPG of 20.0. Therefore, the odometer will only increase by 60 instead of the requested 100 and the gas tank will have 0.0 gallons (not a negative amount).

Step 6. Complete the the start_engine() instance method to set the boolean attribute to True. Complete the stop_engine() instance method to set the boolean attribute to false. Update drive() to only update attributes if the engine is on, and the engine turns off if the car runs out of gas. Update add_gas() to only add gas if the engine is off.

I am NOT going to use your main program for anything … you will want to write a main program in order to test your class definitions to make sure that they work as expected. I am going to use my own main program to test your class definitions!