As you continue working on Adventure, there are a myriad of potential pitfalls or sticking points. This week's questions are geared around helping you through these situations.

1. **Finding Common Mistakes**

   Adventure is a complex enough program that, as you make progress on the milestones, debugging or troubleshooting the errors that arise may begin to really test your debugging skills. After all, your code is now spread across 3-4 different files and compartmentalized across multiple classes. As such, it becomes imperative to systematically identify where an error is occurring, why that error is occurring, and then what steps you could take to remedy the situation. The strategy of changing something random hoping it will fix the issue will almost assuredly never work in a program of this complexity. As such, this first problem is geared at getting you some more practice debugging an Adventure-like program.

   In the template materials for this section is a folder names `BrokenAdvTM`, which contains an advanced version of the Teaching Machine program wherein someone was trying to add a points and rewards feature to the program. Their hope was to have each problem normally worth a single point, but to have several possible rewards randomly distributed to the problems that would boost or decrease their worth. To do so, they have added an entirely new class (`TMReward`) whose contents are read in from the `rewards.txt` file when a new course is created. Unfortunately, there are currently several errors in the code. Your task here is to track down and identify what the issues are, and to then fix them! There are not many errors here: only 3 real issues which require changing 5 lines of code.

2. **Adding messages to the teaching machine**

   As written, the `TeachingMachine.py` program provides no feedback when the user gives an incorrect answer. Here you will look into possible ways of implementing such feedback, which will also be directly applicable to the Adventure project.

   (a) Devise a strategy that would let the course designer specify an optional message along with each response. How would you need to adjust your data-file and corresponding internal representation?

   (b) While there are many strategies that you could have come up with, one that is easy to implement and mimics what Will Crowther did with Adventure is to allow questions in the Teaching Machine to have a "FORCED" answer. If a question has such an answer, then the user is **not** prompted for a response, and instead the user is taken immediately on to the new question after the text is displayed. An example of a snippet of the data file might thus look like:

```
DivQ1
What is the value of 3 / 2?
-----
1: DivMsg
1.5: DivQ4
*: DivQ3

DivMsg
The / operator always produces floats.
-----
FORCED: DivQ2

DivQ2
What is the value of 9 / 3?
-----
3: DivMsg
3.0: DivQ4
*: DivQ3
```

The `FORCED` entry in the question named `DivMsg` tells the Teaching Machine to continue automatically to `DivQ2` without asking the user for an answer, as could be seen in the following run transcript:

```
〉python TeachingMachine.py

Course name: Python
What is the value of 3 / 2?
> 1
The / operator produces floats.
What is the value of 9 / 3?
> 3
The / operator produces floats.
What is the value of 9 / 3?
>
```

Making the necessary changes in the `TMCourse` class is not difficult, you need only to check if a question has a `FORCED` answer before prompting the user for input, and set the next question accordingly. An example course is provided in the template file for you to test your program against.