Homework 5: Recursion and lists

This homework does not have a starting file. You should make a file called hw5.py and put all your work there. The descriptions of what should be done in that file are below.

You should test your code! Create objects, play around with them, and make sure they behave as they're supposed to. Test both what the various methods do and what they output. Try to test them in all sorts of weird situations to make sure that they work.

When you are done the assignment, upload the hw5.py file in a homework5 folder inside your submission folder. Also print the file and bring it to class.

Exercise 1: `Car`

The first class we will build is called `Car`, and it is meant to represent automobiles driving around some area. We will assume for this assignment that our world is a flat coordinate plane (measured in miles) with no obstacles, meaning all trips take the straight-line path between the start and end points.

The `Car` constructor should set up the basic starting values of things we care about regarding cars. For us, that will mean that it should take as input the x and y coordinates of the car's initial position, the amount of money the driver has, the miles per gallon that the car gets, and the size of the car's gas tank. (You should assume that the car begins with a full tank of gas.)

The class should have a `moveTo` method, which takes as input x and y coordinates of a new location and tries to move the car to that location. If the car has enough gas to make the trip, the car should be moved, the amount of gas remaining should be updated, and the method should return `True`. If the car does not have enough gas, it should not be moved or changed at all, and the method should return `False`.

Exercise 2: `GasStation`

We will now add gas stations to our world. The constructor for `GasStation` should create a new gas station, giving it a location (by taking x and y coordinates). It should also take a price, which is stored in the instance. (This is the price the station charges for gas, which we assume will never change.)

The class should also have a `refillCar` method, which takes as input a car object. This method checks to make sure the car is located at the gas station and that the driver has enough

money to fill the tank. If those checks fail, it should return `False`. Otherwise, it should fill up the car's tank, updating the amount of gas in the car and the amount of money the driver has accordingly, and then it should output `True`. To deal with the rounding errors involved in using floats, this method should consider a car to be at the gas station if the distance between them is less than .001 miles.

Exercise 3: `Taxi`

Finally, we add taxis to our simulation. Taxis are a type of car, and we'll have the `Taxi` class inherit from `Car`. Taxis have the additional behavior that they can pick up passengers, and the driving they do while carrying passengers they get paid for. To do this, you'll need to give taxi objects `pickup` and `dropoff` methods. The `pickup` method represents the taxi picking up a passenger. The `dropoff` method represents dropping off the passenger. The taxi driver should not get paid for the trip until the passenger has been dropped off. When that happens, the taxi should get paid at a rate of $2 to start plus $3 per mile driven since the passenger was picked up. The `pickup` and `dropoff` methods should, in addition to the above behavior, return `True` or `False` to signify if the method executed successfully. (In particular, `False` should be returned if `pickup` is run on a taxi that already has a passenger, or `dropoff` is run on a taxi that has no passenger.)

Reminder: Test everything

You should now have a fully functioning world. Make sure that's really true. Create some cars, some of which are taxis, and some gas stations and have things drive around. Make sure you look at both the behavior and the output of each method, and make sure you test on a variety of inputs that are supposed to fail (i.e., return `False`), as well as those that are supposed to work. Remember, though, that you should remove (or just comment out) your tests before you submit the file.