CSC148 exercise #2, winter 2017
due: Sunday January 29th, before 10 p.m.

In this exercise, you'll work on applying what you've learned about inheritance, a fundamental type of relationship between classes, to extend your work from last week. Specifically, you will see how to use inheritance to design classes for different types of vehicles, all of which support the same basic operations, but do so in their own unique ways.

starter code

Save these files to your ex2 folder (inside exercises).

- ex2.py [submit this]
- obfuscated_stack.py
- ex2-test.py
- pylint.txt

task 1: Cars and other Vehicles

Recall that last week, you helped design and implement the Super Duper ridesharing system, which tracks and moves cars across a 2-D grid.

To build on your work from last week, you will now allow Super Duper to support not just cars, but other vehicles as well.

To do this, we have defined a Vehicle interface — please read its documentation carefully. We have also modified the SuperDuperManager class (minus dispatch) to make use of this interface (our code likely looks very similar to your solution from last week).

You have two responsibilities here:

- Update the Car class you implemented last week to now implement the Vehicle interface. All characteristics of Cars from last week are applied to this week.
- Create two new classes Helicopter and UnreliableMagicCarpet, both of which implement Vehicle, according to the following descriptions:

  Helicopter: Helicopter: starts at position (3, 5), the Super Duper launchpad, with a specified amount of fuel. It uses 1 unit of fuel per unit of distance, but it can travel diagonally, and always goes in a straight line between its current position and target destination. Like Car, it does not move and uses no fuel if it does not have enough fuel to complete the journey. After each move, round the amount of fuel down to the nearest integer.

  UnreliableMagicCarpet: starts at a random position (x, y) where both x and y are random integers between 0 and 10, inclusive. Unlike the other two vehicles, it does not use any fuel when it moves (so its fuel attribute never changes from its starting value). However, when told to move to position (x, y), it instead moves to a random position (x + dx, y + dy), where dx and dy are random integers between -2 and 2, inclusive. (So some of the time it might move to the correct position, but it often doesn't.)

Warning: Please do **not** modify the SuperDuperManager class.

Keep the following points in mind when implementing your three classes:

- A constructor for the superclass Vehicle is provided - you should use it inside all of the subclass constructors.
- We're expecting a certain public interface for the constructor of each class. See the SuperDuperManager's "add_vehicle" method for details.
- The fuel_needed method is documented by left unimplemented in Vehicle: all of its subclasses are required to implement it.
- A default implementation of move is given in Vehicle. For each subclass, you should look carefully at whether this implementation makes sense, or whether to override.

using Stack

The point of this exercise is to get you familiar with the Stack data type: what is represents, and how to use one in a program. You will act as clients of the Stack, and so will not care about the implementation details at all, but rather look at the method docstrings to understand how to use them.

Read Week 3 material from the course web site on Stack and be sure you understand methods **_init_**, **add**, **remove**, and **is_empty**.

Complete the function reverse_top_two according to the specifications in the docstring.

Note: to test your code, you should use the provided stack module obfuscated_stack.py. We've deliberately run the source code through a Python obfuscator to discourage you from trying to understand the implementation of the Stack. Doing so defeats the purpose of the exercise, which is getting you to use a class based on its public interface, without knowing how it's implemented.

auto-testing

We run the unittests from **ex2-test.py** on your code, plus 4 more tests. Together these provide 90% of your grade for this exercise. The remaining 10% of your grade is provided by running:
python_ta.check_all(config="pylint.txt") on your **ex2.py**