Lab 2: Introduction to Object-Oriented Programming

1) Review: Working with tweets

Your first task is to download the Twitter starter code (tweet.py) the document into your **lab2** folder, and open it in PyCharm. Review the code from class, and if you have any questions, ask your TA.

Then, complete each of the TODOs found in that file in the following order (we've arranged these roughly by difficulty).

1. Implement the **User.verbosity** method.
2. Discuss with your partner which class the **retweet** function should go into, and then move it. (You'll have to make some minor modifications to the code as well.)
3. Implement the **User.hack** method.

2) Designing Classes

Create a new Python file in your lab2 folder and name it **registry.py**. Your task is to perform an object-oriented analysis of a problem specification, outlined in the steps below.

Note: design is often harder and more time-consuming than implementation! Work with a partner, and don't be afraid to take up the rest of the lab time to get a really solid design done. You'll find that if you spend your efforts today on the class design, actually implementing your design will be comparatively straight-forward.

Don't begin implementation until your design is complete.

1. *Read the problem description.*

   Download **specs.txt** the document into your **lab2** folder and read through the problem description.

2. *Decide what classes you need to design.*

   In a basic object-oriented design, each class usually models some noun in the problem description—but not every noun needs a class to model it. For example, an email address can be modelled by a simple str.

   The one class you need for sure is a class to represent the race registry as a whole. You can design a reasonable solution that has only that class. Decide whether you wish to have any other classes.

3. *Sample usage.*

   Next, picture how your class(es) will be used by writing some doctests in the class docstring of one of your new classes. Your doctests should illustrate the following behaviour:

   - Create a race registry.
   - Register the following runners:
     - Gerhard (with time under 40 minutes)
     - Tom (with time under 30 minutes)
     - Toni (with time under 20 minutes)
     - Margot (with time under 30 minutes)
     - Gerhard again (with time under 30 minutes—he's gotten faster)
     - Report the runners in the speed category of under 30 minutes.

   Remember, you haven't written the class(es) yet!

   Just like function doctests help illustrate the purpose of a function, your class docstring helps the user (and yourself!) understand how your class should be used.

4. *Designing the interface.*

   Use **Part 1** of the Class Design Recipe the document to create the public interface of each class. (You've already started doing this in the previous step.) Note that this involves a lot of documentation and writing of basic examples and tests, but still no implementation whatsoever. That will come later.

   This analysis will require a fair amount of thought. Don't worry about getting it completely right. If you find any ambiguities in the specifications, write them down, and try to come up with a solution you think is reasonable.

   A good skill to develop in this course is identifying ambiguities and proposing multiple solutions for such ambiguities in problem descriptions you receive. There's no "one right answer" to this task!

2) Designing Classes

If you still have time, take your design and implement all of the methods in each of the classes you defined.

We've provided code in the "main" block at the bottom of the file to run **doctest** to check your doctest examples. You should have already written a doctest example in your class docstring by following the above instructions; make sure you add some further examples to individual methods to help check the correctness of your code.