CSC148, Lab #1, winter 2015

Introduction

The goals of this lab are:

- To give you practise **designing a class**. Object-oriented analysis was discussed in last week's lectures; also check out How to Think Like a Computer Scientist.
- To give you practise **implementing a class**. This will make you review lots of the material from CSC108: creating simple classes, writing methods, and basic data structures in Python. Don't hesitate to make use of the reference How to Think Like a Computer Scientist to read up on any topic that you've forgotten about.
- To give everyone practise **using the "function design recipe"**, especially students who have not used it in a previous course. The function design recipe was used in class last week. It is a strategy for writing functions that makes them easier to write correctly and leaves behind an excellent docstring. See the CSC108 Recipe for Designing Functions for details.
- Use standard Python style. You should consult CSC108 style guidelines and pep 8

In addition, this lab will make sure that:

- students who have not used the Wing IDE get familiar with it, and
- everyone can successfully submit files on MarkUs, the system where you'll later submit assignments.

If you are doing these exercises in a lab period, show your work to your TA frequently, so that they can make sure you are on the right track. Feel free to ask your TA questions — that's why they're here! We also encourage you to ask other students if you get stuck, and please be generous in helping others.

Getting started with Wing

1. Log in to your cdf account.
2. Start Wing by double-clicking on its icon.
3. Under your home directory, create a new directory named `csc148`.
4. Inside this directory, create another directory named `lab01` (you can make new directories using the `mkdir` command in the terminal, or through the File menu in Wing — create a new file, then right click the window that pops up and create a new folder).
5. Start the browser and navigate to the lab01 page on the CSC148H1 web site:

   `http://www.cdf.toronto.edu/~heap/148/W15/Labs/lab01`

   For each lab, you will find handouts and other resources linked from this page.
6. Download the file specs.txt from

   `http://www.cdf.toronto.edu/~heap/148/W15/Labs/lab01/`

   and save it in the `lab01` subdirectory you created above.
7. Open the file `specs.txt` in Wing, using the File/Open menu item. This is the specification for the code you have to write in the rest of this lab.

Before moving on to the next section, show your work to your TA.

Designing a class

1. Create a new file in Wing, using the File/New menu item.
2. Save your (currently empty) file so that you can name it `lab01.py`
3. Perform an object-oriented analysis of the specifications in specs.txt:
   - Write the header for an appropriate class to represent a race registry, along with a short docstring for the class. We've designed the problem so that you won't need any additional classes.
   - Identify all the other nouns in the specification, as well as the adjectives; each one may need to be represented by some instance variable(s). Decide what data does need to be represented in order to keep track of a race registry and choose appropriate data structure(s) for storing it — lists, strings, dictionaries, etc. There are lots of options; just pick something reasonable.
   - Record the decisions you just made: Write comments below your class docstring (and outside of any method) that describe suitable instance variables.
   - Identify every verb in the specification; each one is a candidate for a method. Decide which of these requires a method in order to satisfy the specification. For each, design a method following the CSC108 Recipe for Designing Functions (it works for methods too):
     - First, write an example of a call to the method. This is the start of a docstring.
     - Then add a type contract showing the types of the parameters and return value.
     - Then write an appropriate method header. Since this is a method not a function, don't forget that the first parameter must be `self`.
     - Finally, add to the docstring a description of what the method does.
     - (You will write the method body later.)

     If our problem specification doesn't cover all situations, make reasonable choices about what a method should do.

   This analysis will require a fair amount of thought. Don't worry about getting it completely right: you need to leave enough time to get to the next steps where you will implement your design.
4. Using your CDF username and password, log on to MarkUs at the following URL:

   `https://markus.cdf.toronto.edu/csc148-2015-01/`

   Find the Lab 1 submission page, and submit your file `lab01.py`. Get help from other students or your TA if you're not sure how to do this or if you run into any problems! You won't be graded on *what* you submit for this lab, only for the fact that you submitted.

Before moving on to the next section, show your work to your TA.

Implementing a class, part 1

1. Write the body of an initializer for your class. It will not need any arguments other than `self`. Your initializer must give an appropriate initial value to each instance variable that you described in your class comments earlier.
2. Save your file, then submit it on on MarkUs. If you are working with a partner, be sure that each of you submits at least one file during this lab.

Before moving on to the next section, show your work to your TA.

Implementing a class, part 2

1. Write the body of your method that registers a runner.
   Hint: Define a variable to store all the legal speed categories. This variable will never change value; the convention is to name such variables using all capitals. Define this variable outside your class so that all instances can share a single copy of it.
2. Save your file, then submit it on MarkUs.
   (Get in the habit of submitting your work multiple times to MarkUs — every time that you complete some feature in your code. New submissions simply replace old ones, and in addition, MarkUs keeps track of previous versions so that it is possible to go back to previous versions if needed.)
3. Write the body of your method that looks up the runners in a given speed category.
4. Again, save your file then submit it on MarkUs.

Before moving on to the next section, show your work to your TA.

Using your class

1. Create a new file named `lab01tester.py` where you will carry out the following tasks:
   - from `lab01` import `NameOfYourClass` (but replace `NameOfYourClass` with the actual name of the class you wrote).
   - 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), and Gerhard (with time under 30 minutes — he's gotten faster).
   - Report the runners in the speed category of under 30 minutes.
2. Submit your file `lab01tester.py` on MarkUs.
3. Run your file to make sure everything works... But don't panic if it doesn't!
4. If you have to, use the rest of your time to debug your code, with the help of other students, and your TA.

Show your work to your TA one last time. Congratulations: you're done with the first lab!

Additional practice

As well as the race registry, here are some additional exercises in designing and implementing classes. We set up each one so that one appropriate solution involves just a single class.

We certainly don't expect you to do this many exercises in the lab, but they are here for additional practice. We'll have a brief quiz at the end of the lab, which will involve an exercise similar to the race registry or one of the additional exercises.

How did you do?

Make sure you are off to a solid start in csc148 by identifying any concepts that caused you difficulty and mastering them before the course moves on. Use the resources linked to in this handout, as well as course office hours and the Computer Science Help Centre to get any help you need.