CSC148 Summer 2018: Lab 1

Introduction

The goals of this lab are:

- To get you familiar with PyCharm IDE.
    - Other IDEs are okay, as long as you have PythonTA installed. If you want, you can download PyCharm to use it to install other libraries for your computer, and then switch to your other IDE (e.g. Wing, IDLE)
- To get you familiar with PythonTA's style requirements.
- To give you practice designing a class.
- To give you practice implementing a class.

Don't hesitate to make use of other resources for this lab, including the course notes, your TAs, instructor, or other students.

Getting started

1. Follow the instructions to configure Pycharm and create a **csc148** directory to set up your work. If you're working on the lab computers, you do **not** need to install PyCharm, but you should still make sure you have PythonTA installed.
2. Download lab_pyta.txt and place it in the directory (or directories) where you'll be working.
   This file will be used by python_ta for all of your labs to check their style.

Using PythonTA

At the bottom of all of the files you write, include the following code (if you already have a main block, just add the body to the end of it):

```
if __name__ == '__main__':
    import python_ta
    python_ta.check_all(config="lab_pyta.txt")
```

Your lab_pyta.txt should be in the same folder as the .py files you're running. PythonTA will raise errors regarding style, specifying the lines you need to fix. You should get familiar with what the errors mean, and how to fix them: this will be important for your exercises and assignments.

Designing classes

On the next page are specifications for various classes that you should design. You may organize your files however you'd like: I recommend having a separate file for each class, and a folder called **lab1** that holds all of these.

While there are multiple specifications below, you are not expected to implement all of them. At the very least, try to design and implement 1. The others are just for more practice.

Bookshelf

Context: A bookshelf keeps track of multiple books.

When we add a book to a bookshelf, we need its name and author. When we look at the contents of a bookshelf, we should be able to get all of its books in alphabetical order. We can take books off the shelf based off their name.

Bookshelves have a limit to how many books they can hold, and we can't add to a bookshelf that's already full. We should also be able to find all books on the shelf written by a particular author.

App List

Context: An app list should keep track of various apps, organized by category/genre.

Given a name and genre, an app list should keep track of that app. We need to be able to get a list of all apps within a certain genre, as well as getting all of the apps in the app list in alphabetical order.

We should also be able to print our the the app list, such that it gives us all of the genres and the names of apps in each genre.

Shop Catalogue

Below is some client code for a shop catalogue. You should write a class that can run the code without raising any AssertionErrors:

```
s = ShopCatalogue("UofT Bookstore")
s.add_item("Chips", 0.99, 3)
assert str(s) == "UofT Bookstore has: Chips (x3) for 0.99 each"

s.add_item("Chips", 0.99, 10)
assert str(s) == "UofT Bookstore has: Chips (x13) for 0.99 each"

s.add_item("Pencil", 2.50, 3)
assert str(s) == "UofT Bookstore has: Chips (x13) for 0.99 each, Pencil (x3)
for 2.50 each"

s.remove_item("Pencil", 2)
assert str(s) == "UofT Bookstore has: Chips (x13) for 0.99 each, Pencil (x1)
for 2.50 each"

s.remove_item("Pencil", 1)
assert str(s) == "UofT Bookstore has: Chips (x13) for 0.99 each"

s.add_item("Pop", 1.95, 3)
s.add_item("Pencil", 2.50, 2)
assert s.get_items_below(2.00) == ['Chips', 'Pop']
```