CIS 480 - Python - Fall 2005
WEEK 13 LAB EXERCISE and Homework #9

week 13 lab exercise due: Thursday, November 17th, END of lab
HW #9 due: Thursday, December 1st, 12:00 noon

FOR THE MODULES for the Week 13 Lab and HW #9:
...start each module with a **docstring** (as described in lecture) that includes the file name of the module, your name, the date the module was last modified, and any other documentation/description you'd like to provide.

For each class in a module, include at least a one-line **docstring** giving a brief desccription of that class.
For each function in a module, include at least a one-line **docstring** giving a brief description of that module. (Do this for methods --- functions inside of a class --- as well as for functions not within a class.)

Make sure that if you run the function **help** for your module, for a class within your module, or for a function within your module (or within a class within your module), that you indeed see your docstrings incorporated.

WEEK 13 LAB EXERCISE

1. Make sure that you can **ssh** to cs-server from the NHW 244 lab. Carefully note the signs on the lab door --- they give some tips you might need to log into cs-server remotely.

2. Let us play with **PYTHONPATH** on cs-server a little bit.

Create a new directory **python_bin** under your home directory:

```
cd                      # get to your home directory
mkdir python_bin        # make this new directory
chmod 700 python_bin    # protect it from prying eyes
```

Now, let's make a **PYTHONPATH** environment variable that include this directory:

First, find out the **name** of your home directory. Type the **pwd** command at the cs-server prompt; when I do this, I get: **/home/st10** You will likely get **/home/** followed by your cs-server username.

Now, open up the file **.bashrc** in your home directory; you can use **`pico .bashrc`** or **`vi .bashrc`** or **`emacs .bashrc`** for this. Don't REMOVE anything that is there --- just use the down-arrow key to go all the way down to the bottom of this file, and add the following lines, except replacing YOUR username for **st10** below:

```
PYTHONPATH=$PYTHONPATH:/home/st10/python_bin     # replace st10 with
export PYTHONPATH                                #    YOUR username!!
```

Save your modified **.bashrc** file, and exit. Either log off and log on again, **or** type the command

```
source .bashrc
```

...to re-run **.bashrc**'s commands and thus create your PYTHONPATH.

Note that, with this done, your **PYTHONPATH** will be set up each time you log into cs-server via a bash shell.

3. SO --- let's TRY OUT that PYTHONPATH, and see.

FIRST: Copy the file **lab13.py** from my account to your new **python_bin** account as follows:

```
cp ~st10/480lab13_public/lab13.py ~/python_bin
```

NEXT: Create a NEW, SEPARATE directory **480lab13**:

```
cd
mkdir 480lab13
chmod 700 480lab13
cd 480lab13
```

...so that you are now in your new directory.

Start up the python interpreter, **while in the 480lab13 directory.**

Try to **import lab13** within the **python** interpreter you started up while in **480lab13**.

If it works --- then so did your **PYTHONPATH**! Now any module you put in your **python_bin** directory can be imported in any of your cs-server directories.

VERIFY that it worked by trying **help(lab13)** --- you'll see **lab13.py**'s docstrings in action.

I'll have you show me that **lab13.py** is in your **python_bin** directory, have you go to your **480lab13** directory, start up a **python** session there and **import lab13**, and then type **help(lab13)**.

I'll also ask you what we need to do to look at **sys.path** at this point, to verify that your **python_bin** directory is part of it. You'll need to show me that you know what to do.

4. When you read the **help(lab13)** results, you'll see that a Python class resides within.

(a) Write a Python statement here that will **create an instance of this class**.

_____

(b) Write a Python statement here that will execute a **method of your choice** on the **instance** from part **(a)**.

_____

(c) According to typical OOP practice, the class within **lab13.py** is missing a method --- when it is reasonable for a user to modify a data attribute, it is considered to be a good idea to provide a **modifier** method for that attribute. (**set_price** is an example of such a modifier method within **lab13.py**).

Since the quantity attribute **item_quent** will likely need to be changed by the user, add a method **set_quant** to the class within **lab13.py**

Write its body here, and then add it to **lab13.py**. I'll check that it is there as part of your lab exercise check. When done, write your name on the 'Next:' list on the board to get your work checked. All of the above must be completed before the end of your lab time.

HOMEWORK #9

1. Create a new module **hw09.py** --- in a directory OTHER THAN **python_bin** --- and write an import or a from statement within it that will make **lab13.py**'s class available in that module **hw09.py**. (Note that I will have a copy of **lab13.py**, as modified by the lab exercise, in my **python_bin** directory in case I need to run your hw09.py, and my PYTHONPATH is the same as that set up in the lab exercise.)

Then, add to **hw09.py** a class **Tea** that is a subclass of the class in **lab13.py**.

Tea should have a constructor that expects 7 arguments (the five that are the same as those in **lab13.py**'s class's constructor, PLUS an argument giving the form of this tea (loose, bags, etc.) PLUS an argument giving the tea variety (darjeeling, oolong, green, etc.)

Tea should also have a method **get_form** that returns the form of a tea instance, and a method **get_variety** that returns the variety of a tea instances. (These kinds of methods that return the values of data attributes are sometimes called **accessors** or **accessor methods**.)

Finally, give **Tea** the method that will give it a string representation that includes all 6 of its data attributes at this point.

2. To do some semi-automated testing of **Tea**, add the code to the bottom of **hw09.py** that allows statements to be run only when **hw09** is being executed as a top-level program (not imported within the python interpreter).

Write statements within this part of **hw09.py** that:

* create an instance of the class copied from lab13.py, and an instance of the class **Tea**.

* prints a message to the screen noting that "**True's == passed, False == failed**"

* Now --- call EACH method of the class copied from lab13,py (except for __init__ and __repr__) for both your **Tea** instance **and** for your other instance, except instead of printing the result, print the result of comparing that call to the expected value of that call --- for example,

```
print myTea.get_size() == 24
```

* Then, somehow show that __repr__ works for each of your instances; there is more than one way to do this. Somehow printing the string representation of each to the screen **would** be accepted for this part.

Create an output file as so (at the cs-server command line):

```
python hw09.py > hw09.out
```

3. Now, add a function **add_to_inventory** to module **hw09.py**.

Do you see that one could build a **dictionary** of instances of the classes created in **lab13.py** and **hw09.py**? We'll make the keys an inventory id number, and the values **ItemForSale** or **Tea** instances.

So, **add_to_inventory** will expect a dictionary that is either empty, or structured as described above. It will prompt the user to enter new inventory items, asking if there is another to add and, if so, for each new item, asking the user for the needed information to be able to then create a new instance, generate what the next inventory id should be, and then add that new key and new instance to the passed dictionary. (Note that your questioning will need to include if the item is a Tea or not --- if so, it will need to ask for the additional information needed for a Tea!) It should keep asking for items until the user indicates that he/she has no more to add.

When done, **add_to_inventory** should return the resulting dictionary.

4. Now that we have a way to create an inventory, let's do some things with it.

Add function **inventory_worth** to **hw09.py** --- it will take an inventory dictionary as parameter (structured as described in #3) and simply return the current retail worth of all of the items in inventory currently.

5. For the next thing I want to do --- we might need to be able to tell if an inventory item is a Tea or not.

There is probably a slick way to do this --- since I cannot locate it, we'll write a kluge as some **dir** practice.

Consider: built-in function **dir** returns a **list** of the names of the attributes of the argument object passed to it. Thus, one could see if an item **myThing** was a Tea instance by seeing if **dir(myThing)** contained the string 'get_form' or 'get_variety' within it.

Add function **is_tea** to **hw09.py** --- it simply returns boolean True if its argument is a Tea instance, and returns boolean False otherwise.

6. Now let's find a way to make a simple report of our inventory.

Add function **list_inventory** to **hw09.py** --- it takes an inventory dictionary as parameter (structured as described in #3) and should print to the screen a heading (column headings) for inventory number, brand name, price, quantity in stock, and tea variety ONLY, and then after that print THAT information for each item in inventory, each on its own line lined up under the headings. You need to **nicely format** the result --- dollars and cents for the prices, everything nicely lined up, etc. (For non-Tea instances, you should leave the tea variety column blank; for Tea instances, you should list the tea's variety. Can you see how **is_tea** from problem #5 might help with this?)

7. Finally, add another subclass of **ItemForSale** to **hw09.py**. It needs to add at least one additional data attribute, it needs to have appropriate **__init__** and **__repr__** methods, it needs to add an accessor method for each additional data attribute, and it needs to add one additional other method of your choice.

(Ideas? Produce that might need whether it is organic or not; Wine that might need a vintage; etc.)

Make sure that instances of your new subclass can be included as instances in inventory dictionaries in the functions in problems #3, #4, #5, #6.

By the due date and time given at the beginning of this handout, use **~st10/480submit** to submit your final versions of **hw09.py** and **hw09.out** (And remember --- you can submit more than one version of these before the deadline, if inspiration strikes after a submission. As the syllabus notes, I'll simply grade the latest version that was submitted before the deadline.)