1. **Lists** (30 pts.)

   Put the following functions in a file called `list_intro.py`. Put your unit tests of the functions in `list_test.py` and it should be possible to run your tests automatically by typing `python list_test.py` into the command line.

   (a) Write a function called `flatten(inputList)` that takes `inputList`, a list with some elements that are lists (for example, `[[2,1], 4.5, [3,’a’,True]]`) and returns a single list with elements that are the elements of the sublists in order (example above should return `[2, 1, 4.5, 3, ’a’, True]`).

   (b) Write a function called `viewHalf(inputList)` that returns a list containing every other element (beginning with index 1) of `inputList`. The script below that’s imported the function should output `[2, 4]` followed by `[1, 2, 3, 4, 5]`:

   ```python
   b = [1, 2, 3, 4, 5]
   a = viewHalf(b)
   print(a)
   print(b)
   ```

   (c) Write a function called `removeHalf(inputList)` that removes the every other element (beginning with the 0th element) of `inputList`. The function should not have a return statement. For example, the script below should output `[2, 4]`:

   ```python
   b = [1, 2, 3, 4, 5]
   removeHalf(b)
   print(b)
   ```

2. **Tuples** (20 pts.)

   Put the function for (b) in `tuple_intro.py`. Put your unit tests of any functions you write in `tuple_test.py` so they can be automatically run.

   (a) Suppose you have two variables `a` and `b` and you wish to swap the values of `a` and `b`. In many programming languages you would need to define an intermediate variable like this:

   ```python
   t = a
   a = b
   b = t
   ```

   but in Python this can be done in one line. What’s the statement you would use to do this in one line? (doesn’t need to be stored in a script)

   (b) Write a function `triangle(pt0, pt1, pt2)` that takes in three tuples which each contain an x and y coordinate of the vertex of a triangle, then returns a tuple containing the area and perimeter of the triangle. It should be possible to call it as `area, perim = triangle(pt0, pt1, pt2)` once the coordinates have been assigned values.

3. **Inventory analysis** (20 pts.)

   In the file `inventory.py` write a function `singleSupplier(warehouses)` that takes as input `warehouses`, which is a list of lists that store each warehouse’s inventory, and outputs a set of items that are only available at one warehouse. Assume items are input in
a standard way, so don’t worry about capitalizations, hyphens vs. spaces, etc... Put your unit tests in inventory_test.py. You should be able to call it in this way (with output in no particular order):

```python
>>> from inventory import singleSupplier
>>> Seattle = ['roller-skates', 'piano', 'piano', 'purse', 'markers', 'laptop', 'deck of cards', 'piano', 'roller-skates']
>>> Tucson = ['purse', 'purse', 'magnet', 'notebook', 'markers', 'laptop', 'markers']
>>> Austin = ['deck of cards', 'magnet', 'deck of cards', 'notebook', 'laptop', 'spatula']
>>> locations = [Seattle, Tucson, Austin]
>>> items = singleSupplier(locations)
>>> print(items)
set(['roller-skates', 'piano', 'spatula'])
since roller-skates and pianos are only in Seattle, and spatulas are only available in Austin.
```

4. **Product ratings** (30 pts.)

Write these functions in the file `product_ratings.py`, and your corresponding unit tests in `rating_test.py`.

(a) Say there’s a dictionary `ratings` of unique product names and their ratings (integers 1 to 5), write a function called `customerAvg(products, ratings)` that takes a set of a few product names called `products` and the `ratings` dictionary as inputs and returns the average rating (decimal number) of each of the group of products. If a product is listed but isn’t in ratings, don’t include it in the average. A script containing these lines should output 1.6666666666666667:

```python
myRatings= {'deck of cards' : 4, 'laptop': 1, 'spatula': 3, 'notebook': 2, 'markers': 2}
officeSupplies = set(['laptop', 'notebook', 'markers', 'pencils'])
print(customerAvg(officeSupplies, myRatings))
```

(b) Now say you’re given `customers`, a list of dictionaries, each of which corresponds to the ratings dictionary of one customer (assume they have no more than one chance to rate any product). Some customers may have rated some of the same products, but not necessarily all the same products. Write a function called `productAvg(products, customers)` that takes the `customers` list and a set of a few `products` and returns a dictionary containing the product names as keys and the average rating by customers as the values. As a toy example, a script like this:

```python
Anna = { 'roller-skates': 5, 'magnet': 2 }
Barry = { 'spatula': 1, 'magnet': 1, 'piano': 5, 'purse': 3 }
Carla = { 'markers': 4, 'roller-skates': 3, 'magnet': 2 }
reviewers = [Anna, Barry, Carla]
funThings = set(['magnet', 'roller-skates', 'piano'])
print(productAvg(funThings, reviewers))
```

should output something like this:
```
{ 'magnet': 1.6666666666666667, 'roller-skates': 4.0, 'piano': 5.0 }
```

**Instructions for submitting the homework**
Before submitting, make sure your code runs on the corn machines. You should submit a compressed/zippered folder named FirstLast_HW3 (where First is your first name and Last is your last name) to your Drop Box on CourseWork containing the following files:

- writeup.pdf
- list_intro.py
- list_test.py
- tuple_intro.py
- tuple_test.py
- inventory.py
- inventory_test.py
- product_ratings.py
- rating_test.py

**Function interfaces:** Most problems will require you to write a function with a particular interface (i.e. a specified filename, a specified function name, and specified input/output variable(s)). Points will be deducted if the requested interface is not used. You may create helper functions, smaller functions that your function calls to organize and break up larger problems into smaller ones. Unless specifically prohibited, you may use any modules in the standard python library. You may reuse code snippets from previous assignments.

**Unit tests required:** All functions must have a set of unit tests which are totally automated and give meaningful output (you’re strongly encouraged to use the unittest module for this). You’ll be graded more on your code itself than the unit tests, but you’ll get a better grade if you include a meaningful test that fails and shows some flaw in your code than to have the grader’s tests fail.