CS3240: Homework 2

Deadline and submission rules: See Collab for the due date and time.
Collaboration rules:
You must **write the code on your own**. You may only talk to other students about questions related to understanding the problem, i.e. what we want you to do. But you may not talk about how to solve it, either about design or about coding. You may not talk or get help from anyone outside the class other that the TAs and instructor.

Write Python methods that correctly implement the following functions. Put all of these in the file that is listed for each part below. Your file may contain other methods than the ones specified here. We will run-test only the methods listed here. Follow the specifications carefully. Name each method **exactly** as listed here.

Part 1: List and Dictionary Manipulation

(The first two functions below were written as part of a previous lab. Please go ahead and re-submit the code for those with this HW. They will be run-tested but only for some simple cases.)

Put all the functions for Part 1 in a file called hw2_p1.py. (We might use this as a module, and module names must have underscores not hyphens in their names. Also, use the technique for making this a module by which some main() or test methods are only executed `if __name__ == "__main__"`.)

- `maxmin(list)`
  Assume the parameter is a any valid list containing comparable values of the same type. Return a *tuple* where the first value is the maximum value found in the list (based on the > operator) and the second value in the tuple is the minimum value found in the list (based on the < operator). If the list is empty, return the special Python value `None`. Examples:
  For the list `[1, 3, 3]`, return `(3, 1)`
  For the list `[3, 1, -2]`, return `(3, -2)`
  For the list `['Q', 'Z', 'C', 'A']`, return `('Z', 'A')`
  Note: Constraint for this homework: do not use the built-in Python methods max() or min() for this. You cannot sort the list either.

- `common_items(list1, list2)`
  Assume that `list1` and `list2` are valid lists. Return a list that contains items that are found in both lists. If there are no such common items, return an empty list. The list returned will not contain any duplicate items.
  Note: Constraint for this homework: do not use the built-in Python support for sets for this.

- `notcommon_items(list1, list2)`
  Assume that `list1` and `list2` are valid lists. Return a list that contains items that only occur in one of `list1` or `list2` (i.e. not in both lists). The list returned will not contain any duplicate items.
  Note: Constraint for this homework: do not use the built-in Python support for sets for this.

- `count_list_items(list)`
  Assume the parameter is a valid list that may contain duplicate items. Return a dictionary that stores counts of how often each item occurs, i.e. each key in the dictionary will be a unique item from the list, and that key's value will be how often it occurs in the list. Example:
  For the list `[1, 3, 2, 2, 3, 1, 1, 2]`, the dictionary returned would contain `{1: 3, 2: 3, 3: 2}`

Part 2: A Simple Class

You'll create a class called OurSet that implements a simple set ADT. Python has support for sets in another way, but don't use that. Instead, use a Python list as the underlying data store in your class. Put your class and any supporting functions you want in a file called hw2_set.py. (We might use this as a module, and module names must have underscores not hyphens in their names.)
Define these functions inside your OurSet class. Unless a return value for a method is indicated below, it does not matter what you return.
Note: Constraint for this homework: do not use the built-in Python support for sets for this.

- `__init__(self)`
  This constructor is used to initialize an empty set.

- `add(self, item)`
  If the parameter to this method is not already in the set object, add it to the set. Return `True` or `False` to indicate if the item was added to the set.

- `add_list(self, list)`
  Add each item in the list to the set, unless it already is in the set. Return `True` if any item was added the set, otherwise `False`.

- `__str__(self)`
  Return a string representation of the set, in a format like this: `<2, 5, 7, 11>`
  where the contents of the set are surrounded by "angle-brackets" and each item is separated by a comma and exactly one space.

- `__len__(self)`
  Returns the number of items in the set object.

- `__iter__(self)`
  To allow you to use `in` and `not in` to process items in an OurSet object, you need to define this method. In our case, you just need to know that an iterator is defined for the list inside your set object, so you can just do something with that. It's simple, just one line of code!

- `union(self, set2)`
  Carry out a set-union operation between the current set object and the parameter. Return the union as the return value. Do not modify the current set object or the parameter. (Note: you don't have to do this for this homework, but the nice way to do this would be to define the `__add__(self, other)` method so you could just use the + operator to do a union!)

- `intersection(self, set2)`
  Carry out a set-intersection operation between the current set object and the parameter. Return the intersection as the return value. Do not modify the current set object or the parameter.

Constraints:

- Be sure to name the files exactly as listed above. Submit the two files individually, i.e. not in a Zip file etc.
- For Part 1, comments are not required other than a one-line Python docstring for each function.
- For Part 2, comments are not required other than a one-line Python docstring for each function and a multi-line docstring for the class itself.
- You are not required to turn in unit-testing code, but you are expected to carry out good unit testing as part of your normal development work. **We will test your functions thoroughly.**

- In the lab that occurs after the deadline for this homework, you may be asked to demonstrate you know the basics of how to use a debugger to set breakpoints, step through lines of code, and see the values of variables. So learn the debugger in PyCharm (or your other favorite debugger/IDE) while doing this assignment.
- In each file, we will want to be able to import these as modules. Therefore, any code that is outside of class definitions or function definitions that would execute if you ran the file directly must be controlled using the **`if __name__ == "__main__"`** technique.