**Project 2: Social Network Recommendation**

*Facebook suggests people you may (or should) be friends with. Netflix suggests movies you might like. Amazon suggests products to buy. How do they do that? In this project, you will learn one simple way to make such suggestions, called “collaborative filtering”. (The actual algorithms used by these companies are closely-guarded trade secrets.)*

*A computer system that makes suggestions is called a* ***recommender system****. Collaborative filtering is a recommendation technique based on the idea that, if your past behavior or preferences were similar to some other user's, then your future behavior may be as well. As a concrete example, suppose that you like John, Paul, and George, and other people like John, Paul, George, and Ringo. Then it stands to reason that you will like Ringo as well, even if you had never previously heard of him. The recommender system does not have to understand anything about what “John”, “Paul”, “George”, and “Ringo” are — they could even be brands of toilet paper, and the algorithm would work identically.*

*You will apply the skills you’ve learned this semester to implement a python program to visualize a small network of friends and make friend recommendations.* ***You may work with a partner****; make sure both students are clearly identified in the comments to the program.* ***You cannot work with the same partner as on project 1.***

**Suggested Steps**

Create a folder project2-uLogin1\_uLogin2, replacing the uLogins with the e-mail ids of you and your partner.

**Background Learning**

*First, let’s explore the new concepts we will need to successfully implement this project.*

1. We will use graphviz (<http://www.graphviz.org/Download.php>) to view our social networks. Download & install graphviz to view dot files such as the one below:  
   graph friends {  
    "Harry Potter" -- {"Hermione Granger" "Ron Weasley"}   
    "Harry Potter" -- {"Cho Chang" "Ginny Weasley" "Luna Lovegood"}  
    "Ron Weasley" -- {"Hermione Granger" "Ginny Weasley" "Lavender Brown"}  
    "Ginny Weasley" -- "Hermione Granger"  
    "Neville Longbottom" -- {"Luna Lovegood" "Ginny Weasley"}  
    "Severus Snape" -- "Draco Malfoy"  
    "Draco Malfoy" -- {"Crabbe" "Goyle" "Pansy Parkinson"}  
   }  
   Recommended downloads for [windows](http://www.graphviz.org/pub/graphviz/stable/windows/graphviz-2.38.msi) and [mac](http://www.graphviz.org/pub/graphviz/stable/macos/mountainlion/graphviz-2.36.0.pkg). On windows, go to the graphviz application in program files, go to bin, and use gvedit to open your dot files.
2. We will be reading in a [CSV file](https://support.bigcommerce.com/articles/Public/What-is-a-CSV-file-and-how-do-I-save-my-spreadsheet-as-one) of friend connections. See harry\_potter.csv for an example. We can use python to read our CSV file:  
     
   import csv  
     
   with open('harry\_potter.csv') as file:  
    reader = csv.reader(file)  
    for row in reader:  
    if(len(row) == 2):  
    print(row)  
    else:  
    print("Malformed line:", row)  
     
   Create a test program that reads in the sample csv file & prints out each row using the code above.
3. Modify your test program to print the friend connection information as a valid dot graph you can view with graphviz.  
     
   outfile = open('harry\_potter.dot', 'w') # open the file for writing  
   print("}", file=outfile) # print to the file instead of the screen  
   outfile.close() # close the file when finished  
     
   Each line of the dot file should look like:  
    "Harry Potter" -- "Hermione Granger"  
     
   Don’t forget to begin the dot file with “graph {“ and end with “}”. You’ll want to keep the double quotes around the names with spaces in them (i.e., with first & last names).
4. We will make use of sets to store friends, and play with randomly recommending a new friend not in the set. Consider the following code that randomly selects an element from a set:  
     
   import random  
     
   s = set(["A", "B", "C", “D”, “E”])  
   print(random.sample(s, 1))  
     
   Modify this example so that it returns a sample of 2 elements from the set s.  
     
   Create another set, r: r = set(["A", "C", “E”, “G”])  
   Print the results of the following [set](https://docs.python.org/3.4/library/stdtypes.html#set) operations: union, intersection, and difference.
5. We will be creating a dictionary of friends where the friend name is the key and the set of all friend connections is the value. Create a simple dictionary with two keys, even & odd, that stores the set of even & odd numbers from 1 to 10. Print it to test your code.

**Implement the project**

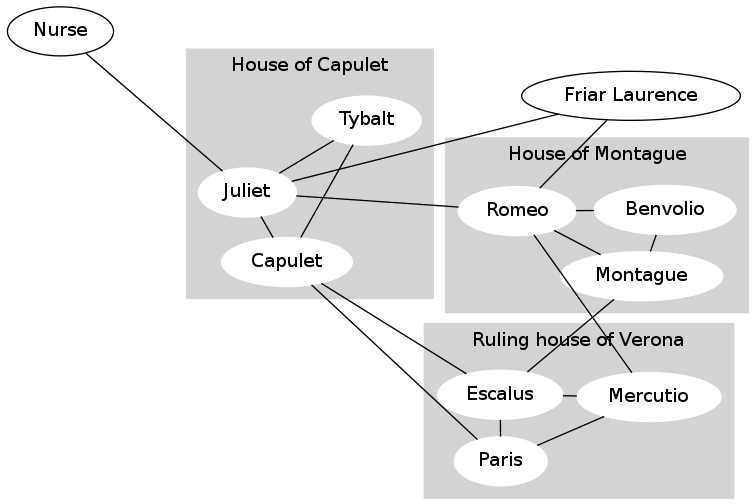
*Now you’re ready to tackle the assignment. Make sure you define your functions at the top of the file with good comments & doc strings. Test your functions at the bottom of the file, after your functions.*

Create a program **social\_network.py** that reads in a csv file of friend connections and implements the following functions:

1. **get\_friend\_dictionary(filename):** takes a csv filename as a parameter and returns a dictionary of friend connections where the friend name is the key and the set of all friend connections is the value.
   * Hint: to add an element to a set, use the **add** function.
   * Recommended helper function: **add\_friend(user1, user2, friends)** that adds a connection from user1 to user2 in friends.
   * *Test by printing your dictionary of friends.*
2. **print\_friend\_network(friends, filename)**: takes a friend dictionary and dot filename as parameters and prints out a graph of the friends.
   * Hint: you can use the **join** function to concatenate all the items in a list into a string with a separator in between each one
   * Hint: don’t forget the double quotes around each name
   * Hint: it’s ok to have two edges connecting each friend
   * *Test by printing & viewing the graph.*
3. **not\_friends(user, friends):** this helper function takes a user and a friend dictionary as parameters and returns a set of people who are not friends with user.
   * Hint: python sets support set operations such as intersection and difference  
     <https://docs.python.org/3.4/library/stdtypes.html#set>
   * *Test on Harry Potter. Harry Potter should* ***not*** *be in Harry Potter’s list*
   * In our example, Harry Potter is not currently friends with: Neville Longbottom, Lavender Brown, Severus Snape, Draco Malfoy, Crabbe, Goyle, or Pansy Parkinson.
4. **random\_recommender(user, friends):** takes a user and a friend dictionary (called friends) as parameters and recommends a list of 5 random people who are not currently friends with the user who should be. *Test by recommending friends for Harry Potter and Draco Malfoy.*
5. **collaborative\_filtering\_recommender(user, friends):** takes a user and a friend dictionary (called friends) as parameters and recommends a list of the top 5 people who are not currently friends with the user who should be based on the maximum number of common friends.
   * Hint: See detailed explanation below.
   * *Test by recommending friends for Mercuitio*
   * *Test by recommending friends for Harry Potter, Cho Chang & Severus Snape.*

Intuitively, if non-friend Y is your friend's friend, then maybe Y should be your friend too. If person Y is the friend of many of your friends, then Y is an even better recommendation. The best friend recommendation is the person with whom you have the largest number of mutual friends.

Consider the example friend graph below for the characters in Shakespeare’s *Romeo & Juliet*:



An edge between person *A* and person *B* means that *A* considers *B* a friend, and also *B* considers *A* a friend. In the image above, ignore the gray background and the labels for the families ("houses"); those are there just to help you interpret the graph but are not part of the social network itself.

In the collaborative\_filtering\_recommender function, you will implement a way to answer the question, “For user X, who is the best person to recommend as a friend?” More specifically, you will answer the more comprehensive question: “For user X, list some *non-friends* in order, starting with the best friend recommendation and ending with the worst.” A non-friend is a user who is not X and is not a friend of X.

Your task will be to write code that, given a user U in the social network, produces friend recommendations for U, in order from best to worst. You will do this by assigning each potential friend a number called a score, where higher scores indicate a better match. Then you can sort your list according to the score.

As a concrete example, consider Mercutio in the Romeo and Juliet graph above.

Mercutio has two friends in common with Capulet (Escalus and Paris).

Mercutio has two friends in common with Montague (Escalus and Romeo).

Mercutio has one friend in common with Benvolio (Romeo).

Mercutio has one friend in common with Friar Laurence (Romeo).

Mercutio has one friend in common with Juliet (Romeo).

Mercutio has no friends in common with the Nurse.

Mercutio has no friends in common with Tybalt.

Therefore, Capulet and Montague are the best friend recommendations for Mercutio, and the Nurse and Tybalt are the worst friend recommendations. (In fact, the Nurse and Tybalt are such poor friend recommendations that your program will not even suggest them.)

**Calculating the score**

For each person in the social network who is not already a friend of the user, calculate the number of friends in common. This is your score. I recommend creating a dictionary of potential new friends and their scores (the number of friends in common) so you can use the built-in [sorted](https://docs.python.org/3.1/library/functions.html#sorted) function to sort by values and find the top-5 scores (<https://wiki.python.org/moin/HowTo/Sorting/>). The operator.itemgetter function makes a good key (<https://docs.python.org/3/library/operator.html#operator.itemgetter>). The easiest way to get the top 5 elements is to use list comprehensions, but you can also use a loop.

**Extra Credit**

* Can you recommend the top 5 non-zero scores? In other words, recommend up to 5 friends, and don’t recommend suggested friends with 0 scores.
* Can you print just 1 edge between friends in the dot graph, rather than 2?

**Grading**

You will be graded on the following:

* **Correctness:** Your code should perform as specified. Correctness will count for the largest portion of your grade.
* **Docstrings:** For each function that you design from scratch, write a good docstring. Do not change the docstrings that we have already written for you, except to add another example or two.
* **Internal comments:** Within functions, the more complicated parts of your code should also be described using "internal" comments. For this project, internal comments will be more important than on the labs.
* **Programming style:** Your variable names should be meaningful and your code as simple and clear as possible.
* **Good use of helper functions:** If you find yourself repeating a task, you should add a helper function and call that function instead of duplicating the code. And if a function body is more than about 20 lines long, consider introducing helper functions to do some of the work -- even if they will only be called once.
* **Formatting style:** Each line must be less than **80 characters** long *including spaces*. You should break up long lines using \.

**Submitting**

When submitting your program, make sure that the following tests are printed:

* print("Harry Potter: ", random\_recommender("Harry Potter", friends))
* print("Draco Malfoy: ", random\_recommender("Draco Malfoy", friends))
* print("Mercutio: ", collaborative\_filtering\_recommender("Mercutio", friends))
* print("Harry Potter: ", collaborative\_filtering\_recommender("Harry Potter", friends))
* print("Cho Chang: ", collaborative\_filtering\_recommender("Cho Chang", friends))
* print("Severus Snape: ", collaborative\_filtering\_recommender("Severus Snape", friends))

Zip your project using the same steps as for the WODs, giving the zip file the same name as your project2-uLogin1\_uLogin2 folder name, and submit to moodle.