Assignment 1

Logistics

Each group must create one IPython notebook for this assignment. Please name the notebook *"HW1_Lastnames_of_group_members"*.

The assignments are due on **February 14th, before the beginning of class**.

Deliverables. You must write up your solution for each problem in an IPython Notebook. This should contain the following parts:

1. A description of your approach: how you will split up the problem into distinct functions, and what will each function do?
2. Each function, along with the docstring just below the function name that precisely states the input and output of the function.
3. Finally, show one sample run of the code.

1 Rock, Paper, Scissors (30 points)

This is a popular game (see here), and you must create a computer player.

Game setup. Each game round consists two turns, the first by the computer and the second by a human. The computer continues playing rounds until the human chooses to quit.

- The computer chooses one of Rock, Paper, and Scissors, but keeps its choice secret.
- The computer asks for the human's input.
- The human chooses one of Rock, Paper, and Scissors, or Quit.

- Unless the human quits, the computer figures out the result of the game, as follows:
  - Rock smashes Scissors, so Rock beats Scissors.
  - Scissors can cut up paper, so Scissors beat Paper.
  - Paper covers Rock, so Paper beats Rock.

  If both players chose the same, it is a draw. The computer reports the result of this round.
- If the human chooses to quit, the computer reports:
  - the number of games played, and
  - the number of times the human won.

Computer's brains. We will give the computer some smarts. It must be able to exploit any human biases. For example, if the human seems to prefer Rock, the computer should play Paper (which beats Rock).

Hence, **your program should remember how many Rock, Paper, or Scissors were played by the human.** Note that we don't need to remember the order in which the human chooses these; the total counts so far for each choice will be enough.

Gotchas.

- *User input:* How you want to receive the user's input is up to you, but you *must* check the user's input to make sure it is valid. If it isn't, request the user for input again.

2 Movies (30 points)

The file "movies.csv" contains information about actors and actresses, and some of the movies that have acted in. We want to find actors who have been co-stars in two movies.

Data format. The data is a comma-separated-value (CSV) file, with the following fields:

1. Actor Name
2. Comma-separated list of movie names

Problem details. We want to create the following user interface:

- Ask the user for movie name 1
- Ask for movie name 2
- **Check the inputs!**
- Print the following information:
  1. All the actors who were in both movies. This should be empty if there are no such actors.
  2. All the actors who were in Movie 1 but *not* in Movie 2.
  3. Everyone in Movie 2 but *not* in Movie 1.

3 Song-writing (40 points)

We want to automatically write some rap. (Or is it hip hop?) Consider the following poetry ("Young, Wild, & Free"):

```
"Peach fuzz on my face
Lookin, on the case
Tryna find a hella taste"
```

Each line is of the form "blah blah ending", where

- the ending words `my-face/case/taste` all have the same "rhythm", i.e., they all end roughly the same way (in this case, they all end sounding like "ase"), and
- all lines use up the same number of beats when sung.

So suppose you were the song-writer, and you knew three things: (a) "Peach fuzz on my face" has 4 beats and ends with the rhythm "ase", and (b) You want to the next line to be "Lookin', on the XXX", and (c) The *prefix* "Lookin', on the" takes up 3 beats. Then you know you want to replace the XXX with some ending word that has the rhythm "ase" and takes 1 beat.

I'm going to give you two files.
(1) The "song skeleton" file, of the form:

```
[Prefix word(s)]::[beats in prefix]::[ending word(s) or XXX]
```

(2) The "endings" file, of the form:

```
[ending word(s)]::[rhythm]::[beats]
```

For instance, the song skeleton file could be:

```
Peach fuzz on::2::my face
Lookin,::1::XXX
Tryna find a hella::3::XXX
```

The endings file could be:

```
my face::ase::2
on the case::ase::3
taste::ase::1
```

You want to output a rhyming song. Specifically, whenever there is an XXX in the song skeleton, you want to replace it with an ending that (a) matches the rhythm of the ending from the previous line, and (b) ensures the total number of beats in this line matches the total beats in the previous line.

An instance of such reasoning is as follows:

```
Peach fuzz on::2::my face
(rhythm "ase" because the ending is "my face"; beats: 2 from
prefix, 2 from "my face" gives 4 beats)

Lookin'::1::XXX
(prefix "Lookin'" has 1 beat, so XXX needs to be filled
with ending of rhythm "ase" that has 4-1=3 beats;
so "on the case" would work)

Tryna find a hella::3::XXX
(Prefix "Tryna find a hella" has 3 beats, so XXX needs
to be filled with ending of rhythm "ase" that has 4-3=1 beats;
so "taste" would work)
```

You want to output the completed song.

Your algorithm will look something like this:

- Traverse the file one line at a time, and if the ending is not XXX, keep doing the following:
    1. Locate the ending word(s) in the endings file
    2. Identify the total number of beats in the prefix word(s) and the ending word(s).
    3. Identify the rhythm from the ending word(s).
- When you encounter a line with XXX as the ending, do the following:

    1. Identify the number of beats in the prefix word(s)
    2. Locate the word(s) in the endings file that has the same rhythm as the rhythm of the previous line, and a number of beats such that adding it to the end of the line with XXX will result in both lines having the same number of beats.
    3. Concatenate the word(s) to the end of the line with XXX, and print the completed line
- Repeat until you have processed the entire file, and print the completed song.

Note 1: Song skeletons may have some lines with no ending, e.g.,

```
Its like Im 17 again::4::
```

In such cases, you should just print that line of the song, but don't use it to set the rhythm for the next line.

Note 2: Beats per line can change over the course of a song! Always use the previous line's beats to fill in the XXX's.