CS2043 Final Project

Due: Tuesday March 10th 2015 at 11:59 PM on `http://cms.csuglab.cornell.edu`.

You can form groups of two students to complete this project. Please form a group on CMS and submit only one solution per group.

General Note: This assignment (and future ones) require you to have access to a Unix-like (Linux, Mac OS X, etc) machine. If you do not have such an operating system installed on your local machine, make sure to get a CSUG Lab account. Different systems have slightly different configurations. The main environment in this class is GNU/Linux.

Assignment Notes:

- You must complete the project using tools that were discussed in class.
- This project is composed of two parts.
- For each problem, you will write a script (not the output thereof), and save it to a file named with the specified problem label
  e.g. **problem1name.sh** or **problem1name.py**.
  Assume that the input is in the same directory as the script (when applicable).

  A bash script is a text file with: `#! /usr/bin/env bash`
  A python script is a text file with: `#! /usr/bin/env python`
  as the first line.
- Please also include a short README file with some feedback on the assignment.

Cows and Bulls

In this assignment we want you to gain experience writing scripts that interact with others on the network. We'll do this by implementing a network game. However, we are not interested in the game's implementation. In fact, we will point you to some source code that does that. However, we do want you to learn to write scripts that send and receive messages on the network.

What is the assignment?

moo.py: Your objective in this assignment is to write a python script that can play the game of *Bulls and Cows* with an opponent on the network. This is a two player game, where each player tries to guess a secret 4-digit number that the opponent has. A player's secret number is composed of four unique digits. So for example, 1234 is a valid secret number, but 1123 is not.

Gameplay

The game proceeds in rounds, in each round a player makes a guess for the opponent's secret, the opponent responds back with how close the guess was. More specifically, when a player makes a guess, the opponent will tell the player how many digits of the guessed number matched the secret and were in the correct position (called *bulls*), and how many digits matched the secret by were in an incorrect position (called *cows*). Examples:

- For secret 1234, guess 1873 has: one bull (the digit 1) and one cow (the digit 3)
- For secret 1234, guess 1384 has: two bulls (the digits 1 & 4) and one cow (the digit 3)
- For secret 1234, guess 6789 has: zero bulls and zero cows

Each player tries to assemble the bulls/cows data in order to figure out the other player's secret number. The first player to guess the secret number of the other wins.

You can read more about bulls and cows here:
`http://en.wikipedia.org/wiki/Bulls_and_cows`

Network interaction

Now that we talked about how the game works on pen and paper, we need to figure out how to make two programs play it over the network.

In order for two programs to understand one another, they have to agree on the format of messages they're going to exchange, where are they going to listen for messages, and how will the messages be sent to them. This information is described in what is referred to as a *communication protocol*.

First, where to look for messages? Each program on the Internet is identified by an *IP-address* and a *port number*. An IP-address is much like a building address in that it is shared by all the people who reside in that building. A port number is like the individual mailboxes where different people in the same building will differentiate their mail by their mailbox numbers. So, for an opponent to be able to send you messages, it has to know your machine's IP address and the port number on which your program is listening.

Second, how to actually send messages? The physical transport of bits from one location to another (i.e. from your program to another, or from your web browser to the server responsible for `example.com`) is handled by a *transport protocol*. There are two main transport protocols, and they are the ones used on the Internet: TCP and UDP. In this assignment we'll use UDP.

Third, the format of the messages. Here we get to define our own standard format that both players know and agree on. We'll define the format of the message next, and the game's protocol next.

For this assignment, we will follow a very simple protocol:

- Your program will send and receive messages via the UDP transport protocol: A simple protocol to exchange discrete messages (aka *datagrams*) on the Internet
- When your program starts up it will be given the port number on which to listen for incoming messages, and the port number on which to send messages to the opponent. We will assume that the opponent is also on the same machine.
- Each message is in one of the following formats:
  * `GUESS:`$wxyz$
    This message contains a player's guess.
    Examples: `GUESS:1234` , `GUESS:4567` , `GUESS:3845`
  * x`B`y`C`
    This message means that the previous guess had x bulls and y cows.
    Examples: `1B1C` , `2B1C` , `0B3C` ..etc
  * `WIN`
    This message means that you guessed the secret number correctly

Your script

- You're script will require 3 arguments when it starts: the secret, it's port number, and the opponent's port number. For example, to start with secret 1234, and local port

4000 and opponent port 4001:

```
./moo.py 1234 4000 4001
```

- When the script starts, it will start making guesses for the opponent's secret. It will print out each guess to the screen followed by what the opponent responded with.
- Your program will terminate when either it won or the opponent won.
- You can test your script by launching two instances of it and having them play against one another:

```
./moo.py 1234 4000 4001
./moo.py 9876 4001 4000
```

How to implement this

There are three parts to this assignment:

- **Checking a guess against the secret number and calculating the proper response**. This is not the focus of this assignment, and you can indeed find an implementation for this here (see if you want to come up with your own version): `http://rosettacode.org/wiki/Bulls_and_Cows#Python`
- **Sending and receiving messages on the network**. This is really two parts, the *server* which handles incoming guesses from the opponent, and the *client* which issues guesses to the opponent.

  The following page from the python documentation has a good example how to write UDP client/server loops in Python:
  `http://docs.python.org/library/socketserver.html#socketserver-udpserver-example`
- **Making good guesses**. Your program can blindly make guesses until it stumbles upon the correct secret number, or it can try to learn something from what the opponent responds with. This part is optional, and we will leave it up to you. It would be cool to implement some simple calculations to reduce the number of guesses.

Notes:

- Test your program by running two instances of it against each other on your machine (the same machine) as I showed earlier.

- If you get stuck, there is a wealth of information on how to write network programs in Python on the web.
- Your program as described above can only play with an opponent on the same machine. However, if you're feeling adventurous, you can modify your script to take in an opponent's address (hostname or IP address) and you can easily play this game against anybody on the Internet! Isn't that cool!

Bash, what is in the theater?

themovies.sh: In this project we will build a command line app in Bash that fetches the list of the top 10 box office movies from the Web, presents a menu to the user, and allows them to choose one of the movies in the list to read its synopsis. The interface looks like this

```
Movies.com Top 10 Box Office
  Movies.com Top 10 Box Office
  1. Fifty Shades of Grey - $22.26M
  2. Kingsman: The Secret Service - $18.35M
  3. The SpongeBob Squarepants Movie: Sponge Out of Water - $16.57M
  4. McFarland, USA - $11.02M
  5. The DUFF - $10.81M
  6. American Sniper - $10.05M
  7. Hot Tub Time Machine 2 - $5.96M
  8. Jupiter Ascending - $3.81M
  9. The Imitation Game - $2.53M
  10. Paddington - $2.44M


Choose a movie (1-10) >
```

When the user chooses a number between 1 and 10 and presses enter, the app displays the corresponding movies synopsis on the screen (or an empty synopsis if the user enters an invalid option).

```
Movies.com Top 10 Box Office
  Movies.com Top 10 Box Office
  1. Fifty Shades of Grey - $22.26M
  2. Kingsman: The Secret Service - $18.35M
  3. The SpongeBob Squarepants Movie: Sponge Out of Water - $16.57M
  4. McFarland, USA - $11.02M
  5. The DUFF - $10.81M
  6. American Sniper - $10.05M
  7. Hot Tub Time Machine 2 - $5.96M
  8. Jupiter Ascending - $3.81M
  9. The Imitation Game - $2.53M
  10. Paddington - $2.44M


Choose a movie (1-10) > 2

Movie 2
Synopsis

    "Mark Millar and Dave Gibbons' comic series is adapted for the big
       screen in this Matthew Vaughn-directed action thriller. The story
        centers on a secret agent who recruits a juvenile delinquent
       into a top-secret spy organization. Together, they battle a tech
       genius with diabolical ambitions. ~ Jeremy Wheeler, Rovi"
```

```
Press enter to return
```

When users finish reading the synopsis, they can press enter to return to the initial screen (the menu without the synopsis) and choose another movie.

Here is our strategy. First, we are going to use the command curl to fetch the data from the website http://www.movies.com/rss-feeds/top-ten-box-office-rss. The syntax is:

```
curl http://www.movies.com/rss-feeds/top-ten-box-office-rss 2> /dev/null
```

The data we will retrieve contains multiple records, each with a title field, which stores the movies title, and a description field, which stores the movies synopsis. We are going to extract the movie titles from the data and print them on the screen. In each record, the movie titles are embedded in a line with the following format:

```
<title><![CDATA[1. Fifty Shades of Grey - $22.26M]]></title>
```

From this line, we want to extract and print the string

```
1. Fifty Shades of Grey - $22.26M
```

When we print the strings for all movies, we will produce a nice numbered menu. The movies corresponding synopsis is embedded in a description field. The line has the following format:

```
<description><![CDATA[Director Sam Taylor-Johnson's big screen version...]]></description>
```

From this line, we want to extract the string

```
Director Sam Taylor-Johnson's big screen version...
```

While we will print the titles directly, we are going to store the synopses in a Bash array. Then, we will print them on request.
Hint 1: To create an array from a set of strings separated by the newline character we will first redefine the default field separator for Bash as the newline character:

```
IFS=$\n
```

For example, if we have a file called actors.txt with the following lines

```
Dakota Johnson
Jamie Dornan
```

The command `array=($(cat actors.txt))` will be equivalent to
`array=([0]=Dakota [1]=Johnson [2]=Jamie [3]=Dornan)`

On the other hand, after resetting the default field separator to the newline character, this command will be equivalent to

```
array=([0]="Dakota Johnson" [1]="Jamie Dornan")
```

Hint 2: Write an infinite while loop to repeat each cycle, i.e., (1) fetch and print the titles, (2) ask the user for input, (3) show a synopsis on request, and repeat from (1) after the user presses enter.
Hint 3: The `sed` flag -n in combination with the function `p` (see `man sed`) might prove handy for this project, e.g., `sed -n s/Movie/Flix/p` .
Style tip: Ideally, your script should not produce any temporary files. Some of your users may not have permission to write on disk.

Remember your best friends are the Python documentation: `http://www.python.org/doc/`, the `man` tool and your favorite search engine.
If you wish to ask questions, use the class discussion board on Piazza.