CSCI 1733 – Programming Assignment 2 – 80 pts.
Due Date: Thursday, October 29, 2015

What you need to turn in:

- **Code listing:** A printed copy of your Python code solution for each problem. Remember to include your name on every page you turn in. Be sure to follow the "Code Style Guidelines" specified in the "Assignment Information and Guidelines" available on the class web page.
- **Code files:** E-mail me a copy of your Python code. Send your .py code files as attachments. Enter your name in the subject line of your email.
- **Working in Pairs:** If you want to work with one other student in our class on this assignment, this is acceptable provided that both members of the pair make a contribution to the solution. If you decide to work in a pair, turn in only one copy of the solution – clearly identify the name of each pair member on everything that you turn in.
- **Late Assignments:** Assignments are due **by the end of class** on the specified due date (both the paper copies and the e-mail copies). Assignments turned in late will be assessed a 20% penalty per day late.

1. (16 pts.) Develop and test a Python program that allows the user to convert between the metric measurements of millimeter, centimeter, meter, kilometer, and the U.S. measurements of inches, feet, yards, and miles. The program should be written so that any one measurement can be converted to another. Assume that the user will enter the measurement to be converted in the form "*number space units*", where *units* will be a standard abbreviation of one of the units (mm, cm, m, km, in, ft, yds, mi), and will enter the units that the measurement should be converted to as one of these abbreviations.
Your program should contain the following functions:
- `displayWelcome` – displays welcome message to user and what the program will do
- `getMeasurement` – prompts for and reads user input and then determines if input is valid; these will include checks for
    - check for a space between value and units
    - check that value consists of only digits and/or a period
    - check that the unit is a valid unit

  This function should return a tuple consisting of the value (as an float) and the units.
- `displayConversion` – takes the input measurement value, input measurement units, conversion units, list of units, and unit conversion factors, and displays the input measurement and units as well as the converted measurement and units.

Your program should also contain these global data structures:
- A global tuple of digit characters that is used to determine the validity of the numeric input value.
- A global tuple of units strings that is used to determine the validity of the units input value.
- A global tuple of conversion factors that is used to do conversion calculations.

Your program should not make use of string processing functions covered in our text after Chapter 5, or which are not covered in the text at all. You can make use of any of the sequence operations covered in Chapter 4 of the text.

Write your program so that output will look like this example run (user input in bold):

```
This program will convert between metric and US units of linear measure.

Measurements entered as <number><space><units>, where <units> one of:
METRIC UNITS: mm (millimeters), cm (centimeters), m (meters), km (kilometers)

US UNITS: in (inches), ft (feet), yds (yards), mi (miles)

Enter measurement: 2.35 cm
Enter units to convert to: mi
2.35 cm = 0.000015 mi

Do you wish to do another conversion? (y/n) y

Enter measurement: 2.9 km
Enter units to convert to: in
2.9 km = 114173.228346 in

Do you wish to do another conversion? (y/n) y

Enter measurement: 1 mi
Enter units to convert to: km
1.0 mi = 1.609344 km

Do you wish to do another conversion? (y/n) n
```

2. (16 pts.) Develop and test a Python program that lets two players play tic-tac-toe. Let player 1 be X and player 2 be O. The program should ask for the player's names and then prompt each by name to make moves. The program should terminate if either there is a winner, or if the game results in a tie. The tic-tac-toe board should be displayed after every move as shown in the example run, below.

Your program should contain these features:

- Use a list of characters to represent the board – initialize the list to blank strings
- Use a list of characters from '1' to '9' to represent the board locations numbering (i.e., the values the user will enter to specify their play – see sample run, below)

Your program should contain the following functions:

- `displayBoard` – takes a list and displays its content as a 3x3 array (will be used to display the contents of the board during play as well as displaying the board numbering for user play selections).
- `getMove` – takes the board list, the locations list, and a player and then displays the board locations and prompts for and reads the current user's desired move, checking it for validity and requiring re-enter if invalid. This function returns the move.
- `win` – takes the board list and a player and determines if the game has been won; it returns true for a win and otherwise returns false
- `tieGame` – takes the board list and determines if there is a tie game; it returns true for a tie game and otherwise returns false

Your program should also contain a "main" section of code that performs initialization tasks and then loops, calling the above functions, until the game is completed, at which time it prints the game results.

Write your program so that output will look like this example run (user input in bold):

```
This program will allow two players to play the game of tic-tac-toe.
Player 1 has 'X', and player 2 has 'O'.
Enter the name of player 1: Ed
Enter the name of player 2: Bob
Ed you start. You are playing 'X'
 -  -  -
 -  -  -
 -  -  -

Using the board positions shown below...

 1  2  3
 4  5  6
 7  8  9

Ed, enter your move: 1
 X  -  -
 -  -  -
 -  -  -

Using the board positions shown below...

 1  2  3
 4  5  6
 7  8  9
```

```
Bob, enter your move: 5
 X  -  -
 -  O  -
 -  -  -

Using the board positions shown below...

 1  2  3
 4  5  6
 7  8  9

Ed, enter your move: 9
 X  -  -
 -  O  -
 -  -  X

Using the board positions shown below...

 1  2  3
 4  5  6
 7  8  9

Bob, enter your move: 7
 X  -  -
 -  O  -
 O  -  X

Using the board positions shown below...

 1  2  3
 4  5  6
 7  8  9

Ed, enter your move: 3
 X  -  X
 -  O  -
 O  -  X

Using the board positions shown below...

 1  2  3
 4  5  6
 7  8  9

Bob, enter your move: 2
 X  O  X
 -  O  -
 O  -  X

Using the board positions shown below...

 1  2  3
 4  5  6
 7  8  9

Ed, enter your move: 6
 X  O  X
 -  O  X
 O  -  X

Ed, you win!
```

3. (16 pts.) Modify the Horse Racing program from Section 6.3 of our text to allow individuals to enter their name to "register themselves" to place bets. The program should be modified so that races can be consecutively run without having to restart the program. Before each race, bets can be placed by registered bettors. Each bet is on which horse will win. The payout will be based on the rules of pari-mutuel betting described below.

Example of pari-mutuel betting:
Suppose each horse has a certain amount of money wagered on it (assuming eight horses):

1	2	3	4	5	6	7	8
$30.00	$70.00	$10.00	$50.00	$110.00	$40.00	$150.00	$40.00

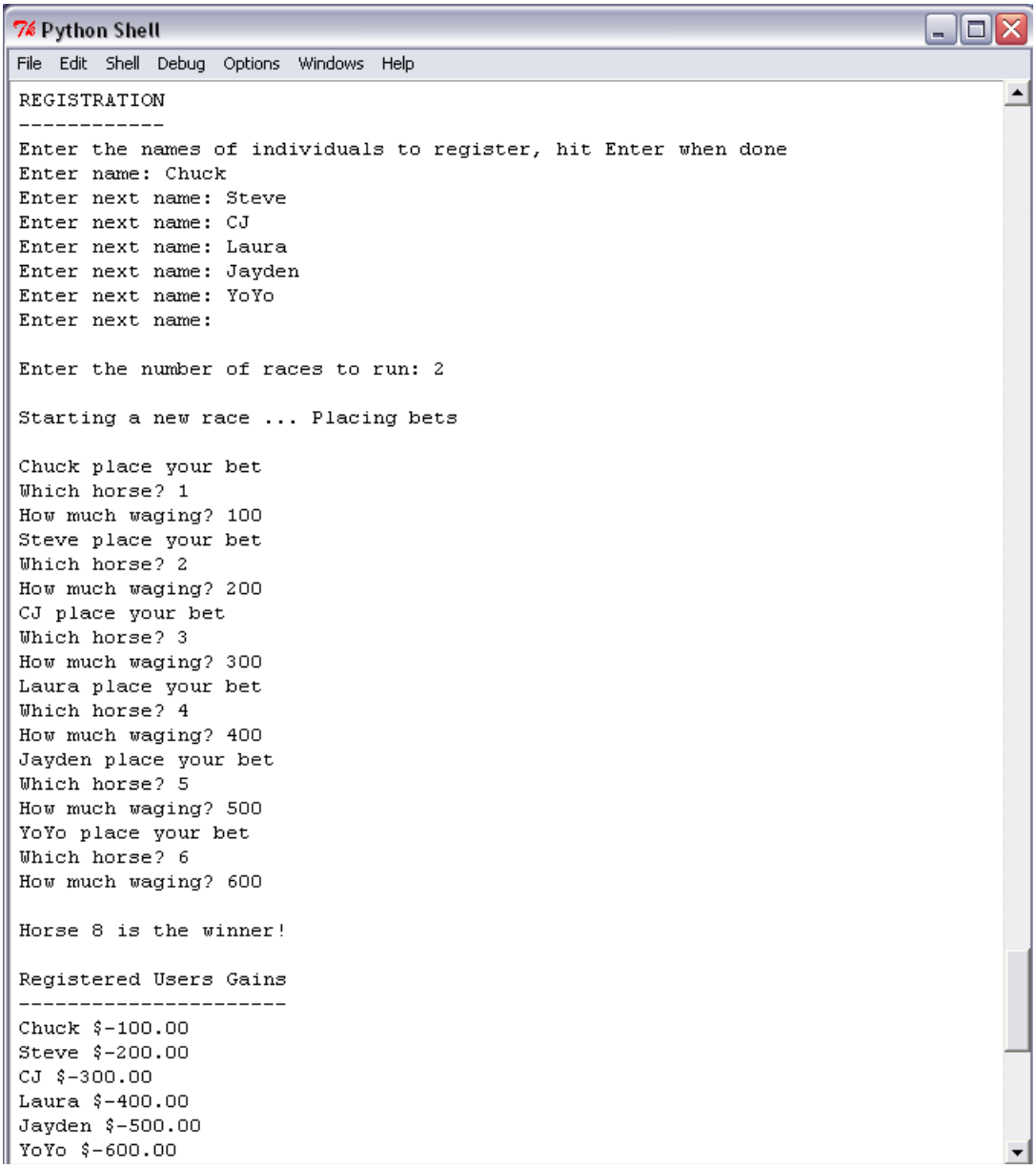
Thus, the total pool of money on the event is $500. Following the start of the event, no more wagers are accepted. Suppose that the race is run and that Horse 4 is the winner. The payout is now calculated. First, the commission for the wagering company is deducted from the pool. For example, with a commission rate of 15%, the commission is: $500 × 0.15 = $75. The remaining amount in the pool, $425, is now distributed to those who bet on Horse 4. First determine the payout per dollar wagered: $425/ $50 = $8.50 per dollar wagered. Now, use this figure to determine the payout to each bettor who bet on Horse 4. For example, if one winning bettor had bet $20 on Horse 4 and a second winning bettor had bet $30 on Horse 4, then the first bettor would receive $20*$8.50 = $170 and the second bettor would receive $30*$8.50 = $255. Note that these payouts include the amount each bettor wagered and so the total gain for the first winning bettor would be $170 – $20 = $150 and for the second winning bettor it would be $255 –$30 = $225. For all the other bettors, their losses would be the amount each wagered.

Your program should contain the following new functions:

- `registerBetters` – will prompt for and read the names of the bettors, storing them into a list, and then return the list.
- `getAllBets` – will accept the list of bettors and the `num_horses` constant and will then prompt for and read each betters horse selection and bet amount, storing them all into a list of tuples where each tuple has their bet horse number and bet amount, and then return the list.
- `updateGains` – will accept the winning horse number, the bets list, and the bettor's gains list, and will do all the calculations based on the result of the race (i.e., calculate the total pool, the update the bettor's gains list); assume that the commission is used is 15%.

In addition, you will need to add new code to the "main" section of the program that implements the new features (i.e., letting the user select to run more than one race, get the bettor names, get the bets for each race, display the results displaying the total gains/losses for each bettor). This new code will include the addition of the new lists described above that are used by the new functions as well as calls to the new functions.

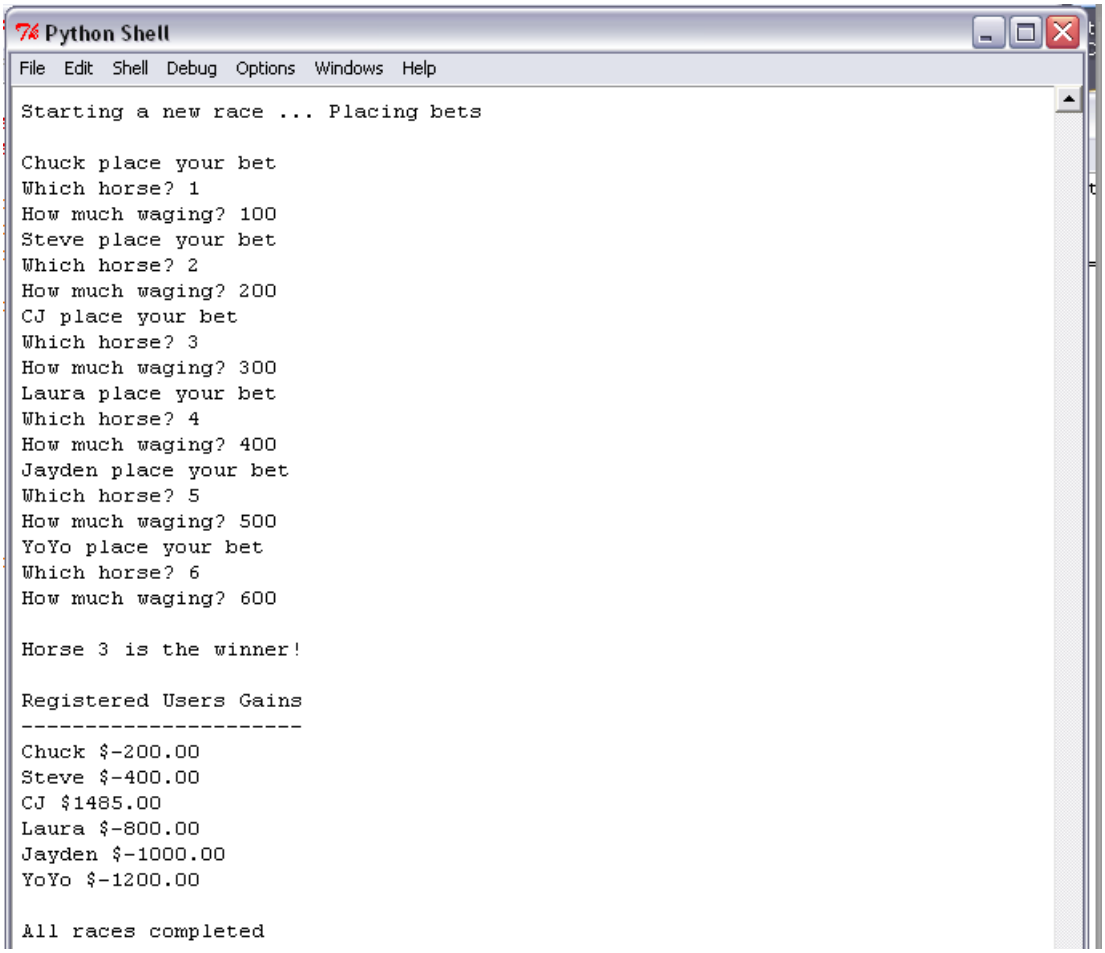
On the next several pages, example output windows from running the program are shown.


Python Shell
File Edit Shell Debug Options Windows Help
REGISTRATION

Enter the names of individuals to register, hit Enter when done
Enter name: Chuck
Enter next name: Steve
Enter next name: CJ
Enter next name: Laura
Enter next name: Jayden
Enter next name: YoYo
Enter next name:
Enter the number of races to run: 2
Starting a new race ... Placing bets
Chuck place your bet
Which horse? 1
How much waging? 100
Steve place your bet
Which horse? 2
How much waging? 200
CJ place your bet
Which horse? 3
How much waging? 300
Laura place your bet
Which horse? 4
How much waging? 400
Jayden place your bet
Which horse? 5
How much waging? 500
YoYo place your bet
Which horse? 6
How much waging? 600
Horse 8 is the winner!
Registered Users Gains

Chuck $-100.00
Steve $-200.00
CJ $-300.00
Laura $-400.00
Jayden $-500.00
YoYo $-600.00


Horse Race Simulation Program with Handicapping
10
9
8
7
6
5
4
3
2
1
We Have a Winner!


Python Shell
File Edit Shell Debug Options Windows Help
Starting a new race ... Placing bets
Chuck place your bet
Which horse? 1
How much waging? 100
Steve place your bet
Which horse? 2
How much waging? 200
CJ place your bet
Which horse? 3
How much waging? 300
Laura place your bet
Which horse? 4
How much waging? 400
Jayden place your bet
Which horse? 5
How much waging? 500
YoYo place your bet
Which horse? 6
How much waging? 600
Horse 3 is the winner!
Registered Users Gains
Chuck $-200.00
Steve $-400.00
CJ $1485.00
Laura $-800.00
Jayden $-1000.00
YoYo $-1200.00
All races completed


Horse Race Simulation Program: Pari-mutal Betting
10
9
8
7
6
5
4
3
2
1
We Have a Winner!

4. (16 pts.) In this problem, you will use stacks to model a simulation of Instant Runoff Voting (IRV) in an election. IRV is a voting system for single-winner elections that guarantees majority winners in a single round of voting. Here is how IRV works:

- IRV uses ranked ballots to simulate a traditional runoff in a single round of voting. Voters rank candidates in order of preference. They may rank as many or as few candidates as they wish, with lower rankings never counting against higher rankings.
- First choices are tabulated. If a candidate receives a majority of first choices, he or she is elected.
- If no candidate receives a majority of first choices, the candidate receiving the fewest first choices is eliminated. Ballots cast for the eliminated candidate are now counted toward those voters' second choices.

This process continues until one candidate receives a majority and is elected.

You will consider a simple scenario in which there is an election involving only five candidates. Assume that all voters generate a ranked vote of five ranked preferences. For our scenario, let's assume we have candidates #1, #2, #3, #4, and #5.

Write a Python program that uses the stack module from Chapter 7 of our text to simulate an election. Each ranked vote will consist of a stack of the voter's ranked preferences – they are placed on the stack from 5th to 1st preferences so that the top of the stack contains their first vote preference. To simulate an election, you will create a list of size given by the user of ranked votes (i.e., stacks of the ordered vote preferences). The program will then tally the votes from this list of votes and report the totals for each candidate. If no candidate has more than 50% of the votes, the candidate with the least votes will be removed from the running and the second round vote totals will be calculated. This process will continue until a candidate has over 50% of the vote and is declared the winner. Note: The process of removing the lowest candidate from a given vote is simply facilitated by popping that vote off the top of the given vote's stack.

Your program should import the stack module given in the text and make use of it. Access to a stack can only be made using the functions in the stack module. In addition, your program should contain these functions:

- `count_votes` – this function will take a list of ranked votes and will return a list of the vote counts for each of the five candidates.
- `pop_votes` – this function will take list of ranked votes and a list of the candidates remaining in the election and will return the modified list of ranked votes with the candidates not still remaining having been popped of the top of each of the ranked votes in the list.
- Use the following `simulate_election` function to generate the list of ranked votes (it uses the `random` module function `sample`, to generate a randomly ordered list of 5 values from the list [1,2,3,4,5] – in our program, this represents one randomly generated ranked vote):

```
def simulate_election(n):
    vote_list=[]
    for i in range(0,n):
        vote_list.append(random.sample([1,2,3,4,5],5))
    return vote_list
```

Your program's main section should allow the user to run repeated vote simulations. It should handle the display of the simulations using the above functions.

Write your program so that output will look like this example run (user input in bold):

```
This program simulates an IRV election with five candidates
Enter the number of votes to be simulated (or -1 to stop):10000
Round 1  vote totals:
Candidate 1 : 2029   20.3 %
Candidate 2 : 2004   20.0 %
Candidate 3 : 1918   19.2 %
Candidate 4 : 2018   20.2 %
Candidate 5 : 2031   20.3 %
Candidate 5  had the most votes
Candidate 3  had the least votes

Round 2  vote totals:
Candidate 1 : 2502   25.0 %
Candidate 2 : 2452   24.5 %
Candidate 4 : 2519   25.2 %
Candidate 5 : 2527   25.3 %
Candidate 5  had the most votes
Candidate 2  had the least votes

Round 3  vote totals:
Candidate 1 : 3321   33.2 %
Candidate 4 : 3324   33.2 %
Candidate 5 : 3355   33.6 %
Candidate 5  had the most votes
Candidate 1  had the least votes

Round 4  vote totals:
Candidate 4 : 4978   49.8 %
Candidate 5 : 5022   50.2 %
Candidate 5  had the most votes
Candidate 4  had the least votes

Candidate 5  is the winner

Try another simulation?
Enter the number of votes to be simulated (or -1 to stop):-1
```

5. (16 pts.) Develop and test a Python program that reads in any given text file and displays the total number of sentences, words, and characters there are in the file.

Here are program specifications and requirements:

- Assume that a word is terminated by a delimiting character (a space, comma, semicolon, colon, period, question mark, exclamation point, double quotation mark, right parenthesis). But, even if a word is followed by one of the above special characters, assume that there is a space between every word.
- Assume there are no digit characters in the file.
- Count a hyphenated word as one word.
- Count a word with an embedded single quotation mark as one word.
- Assume that a sentence ends with a period, question mark, or exclamation point.
- Assume that a sentence can appear on more than one line in the file, but if it does, the space character separating all words in the sentence will be either at the end of the first line or the beginning of the second line.
- The total character count should include spaces and special characters, but not the newline character, '\n'.

Your program should contain these functions:

- `getFile` – will prompt for and read the input file to be opened, open the file (raising an exception if it cannot be opened), and return the file name and the opened file object.
- `countSentenceWords` – takes a line read from the file and returns the number of sentences and words found in the line as a tuple.
- `countAll` – takes the opened file object and returns the number of sentences, words and characters in the file as a tuple.

In addition, your program should contain a "main" section that will display the program greeting and call the other functions to open and read the file, perform the counts, and then display the results.

To test your program, create text files using Notepad (or any text file processor) to use as input files for the program.

Write your program so that output will look like this example run (user input in bold):

```
This program will display the number of sentences, words, and number
of characters (including special characters) in a specified text file.
Enter input file name (with extension): testfilebackup.txt

File testfilebackup.txt contains ...
4 sentences
18 words
87 characters
```

Here is the content of the file `testfilebuackup.txt`:

```
Hi how are you?
This is a (very)
 good day.
The file's end is coming!
The last pseudo-line.
```