COMP 1012 Winter 2016 Assignment 3

Due Date: Saturday, March 12, 2016, 11:59 PM

New Material Covered

- input
- testing input and asking for correction
- menus
- sets

Notes:

- When programming, follow the posted programming standards to avoid losing marks. A program has been provided to enable you to check that you are following standards.
- Name your script file as follows: `<LastName><FirstName>A4Q1.py`. For example, `LiJaneA4Q1.py` is a valid name for a student named Jane Li. If you wish to add a version number to your file, you may add it to the end of the file name. For example, `SmithRobA4Q1V2.py` is a valid name for Rob Smith's script file.
- Submit all your output cases together in one similarly named output file: e.g., `<LastName><FirstName> A4Q1_Output.txt`. To generate this file, open a new empty tab in Spyder and save it under the name above. Then copy and paste the outputs of your program runs from the console window to this file, and save it.
- You must complete the checklist for a ***Blanket Honesty Declaration*** in UMLearn to have your assignment counted. This one honesty declaration applies to all assignments in COMP 1012.
- To submit the assignment follow the instructions on the course website carefully (link: http://www.cs.umanitoba.ca/~comp1012/Handin_UMLearn.pdf). You will upload both script file and output via a dropbox on the course website. There will be a period of several days before the due date when you can submit your assignment. ***Do not be late!*** If you try to submit your assignment within 48 hours ***after*** the deadline, it will be accepted, but you will receive a penalty (roughly 1% per hour

Group Work

NOTE: This assignment allows you to work with others as a group.

If you do the assignment all on your own and hand it in, you can get full marks if you achieve about 70% on the assignment. There will be no bonuses for doing better than that. In this case, put this line into your initial doc string:

```
@with nobody
```

If you do the work with a friend, then each of you must hand in your own copy of the assignment. They do not have to be the same, but they can be, except for the inner author identification. If the solution you hand in is completely correct, you can get full marks, and similarly for your friend. In general, the maximum mark is $\min(4, 6/\sqrt{n})$ when n people work as a group.

If you work with two others ($n = 3$), then each of you must hand in your own copy of the assignment. If the solution you hand in is completely correct, you can get at maximum about 3.5 out of 4 marks. The more people you work with, the smaller is your possible mark.

If you work as part of a group also containing James Dean, Gwen Worobec and Calvin, insert lines like the following ***into your code's initial doc string*** to avoid a charge of academic dishonesty. The names deanj12, worobg and wongc23 are the UMnetIDs of your associates.

```
@with deanj12
@with worobg
@with wongc23
```

They in turn should name you and one another in their submitted assignments.

Question 1—Scrabble® Tutor [12 marks]

Description

In the Scrabble board game, a player has a random collection of seven letters, from which he/she plays a word left-to-right or top-to-bottom on a 15×15 board, to accumulate points. Except for the first word, each word must join with existing words on the board to make valid words.[1]

Each letter has a point count, based on the frequency with which it appears in common words. For example, the vowels A, E, I, O and U are very common, and they get a point count of 1, whereas letters like Q and Z are rare, and get a point count of 10. Letters are typically written along with their point count: A_1, F_4, J_8, etc.

To help Scrabble players get better at the game, you will write a program that shows them random groups of letters, and challenges them to make the highest-counting word possible using those letters during a specified time interval. The program will then display all possible words, indicating what the best choice would be.

Theory

This program will use the same basic setup as Assignment 3.

For the word list we will use twl06.txt, the Tournament Word List of 2006 for Scrabble, from http://www.freescrabbledictionary.com/twl06/. This file is not quite up to date with the version used currently in tournament play. This file has over 178 thousand 'words' acceptable in Scrabble play. A slightly modified version without the introductory heading is the one you should use. It can be found at http://cs.umanitoba.ca/~comp1012/twl06.txt. This time you should ***download it and put it in the same folder as your assignment code.*** Your program will access it on your own disk every time you read it. The code to do that is on the Python Guide, and is provided for you.

® Scrabble is a registered trademark of Hasbro, Inc., Pawtucket RI USA.

[1] Image of game in progress is by Tim Niblett under a Creative Commons Attribution 2.0 Generic licence, obtained from https://commons.wikimedia.org/wiki/File:Scrabble_(2399933493).jpg

The following table matches letters of the alphabet with the first 26 prime numbers. It also shows the points for each Scrabble letter, and how many of these letters come with the Scrabble game. For instance, the letter collection in Scrabble has 12 Es but only 1 X.

Letter	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Prime	2	3	5	7	11	13	17	19	23	29	31	37	41	43	47	53	59	61	67	71	73	79	83	89	97	101
Points	1	3	3	2	1	4	2	4	1	8	5	1	3	1	1	3	10	1	1	1	1	4	4	8	4	10
Count	9	2	2	4	12	2	3	2	9	1	1	4	2	6	8	2	1	6	4	6	4	2	2	1	2	1

To determine quickly if one word contains letters from another word, code both words as a product of prime numbers. We will call the product of letter prime codes a ***hash*** of the word. For example,

‘`bad`’ → 3 × 2 × 7 = 42.

‘`bread`’ → 3 × 61 × 11 × 2 × 7 = 28182.

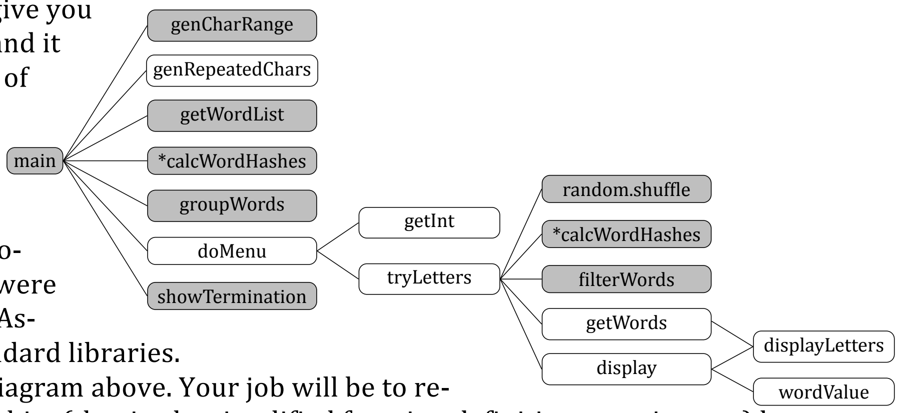
Divide 42 into 28182, and look for a remainder: 28182 % 42 = 0 since 28182 = 671 × 42.

Therefore, ‘`bad`’ *can* be made from letters of ‘`bread`’.

However, ‘`ebb`’ → 99, and 28182 % 99 = 66.

Since there *is* a remainder, ‘`ebb`’ cannot be made from ‘`bread`’ (not enough ‘b’s).

Starting Program



As in Assignment 3 we will give you a starting program. It runs, and it contains simplified versions of all the functions you have to define. You can download the starting program and begin your development using it. Some functions are provided for you, because they were completely implemented in Assignment 3 or belong to standard libraries. They appear in grey in the diagram above. Your job will be to replace the function ***stubs*** in white (that is, the simplified function definitions we give you) by ones that actually work and produce the results you want. You are strongly advised to make sure you always have a code version that runs, and to proceed in small steps.

The following sections of the assignment describe what to do in each of the function definitions. These functions are in alphabetical order by function name, because that is the order of the function definitions in the starting program.[2] This is ***not*** the order you will want to use to develop the code.

[2] Actually, function definitions that are given to you completely implemented are in alphabetical order at the top of the file, and function stubs are in alphabetical order at the bottom. There is a line of asterisks in the handout code between them. Keep it that way as you develop the stubs into full functions, to help the marker find your work.

One plausible order to do your development might be: `genRepeatedChars, doMenu`, `getInt`, `tryLetters, getWords, display, wordValue` and `displayLetters`—in other words, top-down (or rather left to right, since the structure chart has been turned on its side to make it fit). Whichever order you use, you will probably find it useful when trying to implement a function from a stub to print the incoming parameter values, and the values returned.

Last time, `getWordList` was one of the last functions to develop. Here it is already developed. How do you avoid having extraordinarily long printouts, given that the wordlist is even longer than last time? In the handout program we have set `MAX_WORD_LENGTH` to 2. That is, the program will only find 2-letter words. There aren't very many of those, so the outputs remain of reasonable size. As your program gets closer to working, you can increase `MAX_WORD_LENGTH` to a maximum of 15 to produce more words.

Function `display`

```
def display(words, label) :
```

This function takes a list or a set of words, and prints it to the console.

- First it prints a horizontal line to separate this output from previous output (see sample output).
- Then it prints a heading "Valid *words*:" or "Valid *user words*:". The italicized phrase that comes after "Valid" is the value of `label`.
- It prints out the list of words. There are several requirements, and your mark for this function will depend on how many you implement successfully.
    1. Every word in `words` must be printed.
    2. Each letter in each word should be printed with a subscript representing the letter's point value (e.g., $R_1O_1B_3E_1D_2$). A word is formatted that way by a call to `displayLetters`.
    3. Each word should be followed by its total point value in brackets (e.g., $R_1O_1B_3E_1D_2$ `(8)`).
    4. Words should be lined up in fixed width columns.
- Finally `display` prints another heading, "Best word:" and shows a single word with the maximum number of points among the words. If there are multiple words with that point total, any one of them will do.
- There is no `return` value.

Function `displayLetters`

```
def displayLetters(text) :
```

This function is used to show the point values of all letters in `text`. Notes:

- Every character in `text` must be a letter from 'A' to 'Z'. These are the only characters for which point values are defined. For example, `text` cannot be a phrase with blanks in it.
- The point value of every character from 'A' to 'Z' is stored in the global constant POINTS.
- The subscript characters from $_0$ to $_{10}$ are stored in order in the global variable `subscripts`. The subscripts are just special Unicode characters of small size that are placed slightly below the font baseline. The `main` function has already created POINTS and `subscripts` by the time this function is called.
- The `return` value is the list of converted characters with subscripts; if `text` has the value `'ROBED'` on entry, the return value should be $R_1O_1B_3E_1D_2$.

Function `doMenu`

```
def doMenu(hashes, wordGroups) :
```

This function asks the user which action to perform next, and then it carries out that single action. It provides error correction, so it will keep asking until a valid action is selected. It should follow the example shown in the notes for a menu. See the sample output to see what the menu interaction looks like. Here are the choices to offer the user.

- `a) change the number of random letters`
  The two choices allowed are 7 or 8. Users have seven letters on their racks, but they have to fit the word onto the board, which often means using one letter from another word already placed. This function should call `getInt` appropriately to acquire the user's choice with error correction.
- `b) change the time limit`
  The default time limit is 20 seconds, but this is quite long when looking for Scrabble words. Any time between 1 and 120 seconds should be acceptable. This function should again call `getInt` appropriately to acquire the user's choice with error correction.
- `c) try a new set of letters`
  In this choice, the user is presented with a random sequence of letters from which to make words. The function should call `tryLetters` to carry out this choice.
- `q) quit`
  The function should do nothing in this case.

The parameters of `doMenu` are provided to pass down to `tryLetters` when it is called. After the single action is complete, `doMenu` should ***return the user's choice*** in uppercase (`'A', 'B', 'C' or 'Q'`) to the calling code.

Function `genRepeatedChars`

```
def genRepeatedChars(chars, repeats) :
```

This function is used to generate a list of all the Scrabble tiles. As a simple example, if there are five letters, A, B, C, D and E, with 9 A tiles, 2 B tiles, 2 C tiles, 4 D tiles and 12 E tiles, the parameters of this function will be `chars 'ABCDE'` and `repeats [9, 2, 2, 4, 12]`. The return value will be a list: `['A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'B','B', 'C', 'C', 'D', 'D', 'D', 'D', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E', 'E']`. Assume without checking that the parameters are a string with unique characters, and a list of non-negative integers.

Function `getInt`

```
def getInt(lo, hi, what) :
```

This function is provided in several versions in the notes. Follow the guidelines there about which one to use.

Function `getWords`

```
def getWords(letters) :
```

This function repeatedly prompts the user to enter a word made up of the letters from the parameter.

- The repetition ends after the number of seconds specified by the global variable `timeOut`. You can determine how much time has elapsed since the first input request using the library function `time.time`. It returns the time since some point in the past, in seconds. Simply record the time before the input loop, then record it again each time through the loop. If the difference from the start is more than `timeOut` seconds, drop out of the loop.

- Display the letters to the user as formatted by `displayLetters`, so the user can see the point count for each letter.
- No error-checking is required. Whatever the user enters is considered valid at this point. Record all entries in a list or set, in uppercase.
- Return a ***set*** of user entries, without duplicates. The easiest way to remove duplicates from a list of strings is to convert it to a set: `stringSet = set(stringList)`. This works because every entry in a set must be unique. Duplicates are discarded.

Function `tryLetters`

```
def tryLetters(hashes, wordGroups) :
```

This function generates a string of random letters, gets user input by calling `getWords`, and displays the results.

- By the time `tryLetters` is called, `main` has produces a global list of letters called `allLetters`. It uses `genRepeatedChars` to do so, and so `allLetters` contains 98 letters[3] representing the tiles in a Scrabble game. They can be in any order.
- To generate a random sample of the letters, call `random.shuffle(allLetters)`. Then use a slice to take the required number of letters from the shuffled list, and convert it to a string.
- As in Assignment 3, call `calcWordHashes` to hash the string with the chosen letters (remember that `calcWordHashes` requires an argument providing a list of words, but we have just one string).
- As in Assignment 3, call `filterWords` to find all the word groups from the input file that can be made from letters in the random sample.
- In this program we are interested in individual words, not word groups, so expand all the word groups and make a big set (called `allWords`) containing all the words that can be made from letters in the random sample,.
- Call `getWords` to get a set of the words identified by the user and save it as `userWords`.
- The invalid words provided by the user are the entries in `userWords` that are not in `allWords`. They could be valid words that can't be made from these letters, or strings that are not words at all. The set difference operation `aSet - bSet` finds all the entries in set `aSet` that are not in set `bSet`. Print out the invalid words.
- The valid words provided by the user are those that are in both `userWords` and `allWords`. The set intersection operation `aSet.intersection(bSet)` finds all the entries that are in both aSet and `bSet`.
- Call `display` to print out the valid words neatly and to find the best one (that is, the one with the highest point count).
- Call `display` again to print the list of all words, and to find the best one.

[3] In fact a Scrabble game contains 100 tiles, but two of them are blank and can be used for any letter. This complication is ignored in this program.

Function `wordValue`

```
def wordValue(word) :
```

This function returns the total value of the parameter `word`, found by adding up the point counts of all the letters in `word`.

Hand-in

You will hand in your Python 3 program script file. Also hand in the text output produced by your code for a session in which you demonstrate all the menu operations, and also demonstrate error detection for the menu input and for integer input.

Sample Output

```
WELCOME TO SCRABBLE® TUTOR!
'Scrabble' is a registered trade mark of Hasbro, Inc.

Test your ability to find Scrabble words!

Reading words from twl06.txt
178691 words read: AA, AAH, AAHED, AAHING, AAHS, AAL, ...


Choose one of the following and enter the letter:
    a) change the number of random letters
    b) change the time limit
    c) try a new set of letters
    q) quit
a


Enter int n so that 7 <= n < 9 for number of letters to play: 6

*** 6 is too small
Enter int n so that 7 <= n < 9 for number of letters to play: 9

*** 9 is too big
Enter int n so that 7 <= n < 9 for number of letters to play: 7


Choose one of the following and enter the letter:
    a) change the number of random letters
    b) change the time limit
    c) try a new set of letters
    q) quit
b


Enter int n so that 1 <= n < 121 for seconds to guess words: 121

*** 121 is too big
Enter int n so that 1 <= n < 121 for seconds to guess words: 0

*** 0 is too small
Enter int n so that 1 <= n < 121 for seconds to guess words: two minutes

*** two minutes is not an int
Enter int n so that 1 <= n < 121 for seconds to guess words: 20


Choose one of the following and enter the letter:
    a) change the number of random letters
    b) change the time limit
    c) try a new set of letters
    q) quit
c
Time limit: 20 seconds

Enter a word made from letters in          $P_3I_1P_3R_1C_3O_1B_3$ pip
Enter a word made from letters in          P₃I₁P₃R₁C₃O₁B₃ prip

Enter a word made from letters in          P₃I₁P₃R₁C₃O₁B₃ cob

Enter a word made from letters in          P₃I₁P₃R₁C₃O₁B₃ proc

Enter a word made from letters in          P₃I₁P₃R₁C₃O₁B₃ crib

Invalid user words: PRIP, PROC

--------------------------------------------------------------------------------
Valid user words:
                  C₃O₁B₃ (7)              C₃R₁I₁B₃ (8)              P₃I₁P₃ (7)

Best word:                              C₃R₁I₁B₃ (8)

--------------------------------------------------------------------------------
Valid words:
                  C₃O₁B₃ (7)              O₁R₁B₃ (5)              P₃R₁O₁ (5)
                  R₁O₁C₃ (5)          B₃O₁R₁I₁C₃ (9)            C₃R₁I₁B₃ (8)
                  C₃O₁R₁ (5)              R₁I₁B₃ (5)              O₁R₁C₃ (5)
                  P₃O₁P₃ (7)                P₃I₁ (4)              P₃I₁P₃ (7)
                  B₃O₁P₃ (7)              B₃R₁O₁ (5)              O₁B₃I₁ (5)
                  R₁O₁B₃ (5)            B₃R₁I₁O₁ (6)              B₃I₁O₁ (5)
                  P₃I₁C₃ (7)                O₁R₁ (2)            B₃I₁R₁O₁ (6)
                    O₁P₃ (4)            P₃R₁O₁P₃ (8)              C₃O₁P₃ (7)
                    B₃I₁ (4)              R₁I₁P₃ (5)                O₁I₁ (2)
                C₃R₁O₁P₃ (8)              P₃O₁I₁ (5)            C₃O₁I₁R₁ (6)
                    B₃O₁ (4)

Best word:                            B₃O₁R₁I₁C₃ (9)



Choose one of the following and enter the letter:
    a) change the number of random letters
    b) change the time limit
    c) try a new set of letters
    q) quit
end

Invalid response 'END'; try again
Choose one of the following and enter the letter:
    a) change the number of random letters
    b) change the time limit
    c) try a new set of letters
    q) quit
q

Programmed by Instructors
Date: Sun Feb 28 14:05:15 2016
End of processing
```