COSC 101 Homework 6 Spring 2016

Due date: **Monday, April 11, 10:00 a.m.**

In this assignment, you will be writing a two-person word search game. There are several parts to this task; all of them use concepts you learned in the second part of the semester, i.e. lists and string methods, files and dictionary and recursion.

You may work in pair as long as you are together all the time you are working on the homework and that you come to see me Friday afternoon to report on the collaborative process.

Starter code

For this assignment, you are giving starter code: some parts of the program have been written, such as the `doc_string`s that describe the role of helper functions, which are used to solve the search word game problem. Make sure you read through the provided code and understand each piece and how all of them work together.

You will have to complete many helper functions and the `play_game` function.

There are two files:

- `word_search_program.py` contains the main program.
  It handles all the user input and printing, and the game loop.
- `word_search_functions` contains helper functions used by `word_search_program.py`.
  The `word_search_program.py` main function and others depend on these helper functions.

The image below shows the console output in the first iteration of the word search game

```
**************************************
**       Where's That Word?         **
**************************************
   0  1  2  3  4  5  6  7  8  9
0  r  m  h  l  z  x  c  e  u  q
1  b  x  m  i  c  h  e  l  l  e
2  m  n  n  e  j  l  u  a  p  v
3  c  a  e  l  l  e  h  c  i  m
4  x  d  y  d  a  n  a  g  b  z
5  x  i  n  i  a  r  b  p  r  r
6  v  c  t  z  e  v  b  k  i  z
7  j  g  f  a  v  q  w  j  a  n
8  q  u  o  t  j  e  n  h  n  a
9  i  u  m  x  d  d  b  x  n  d

The remaining words to be found:
brian dan jen michelle paul

Player One, please enter a word:
```

Figure 1: Board output and prompting Player One input

Part I: `word_search_program.py`

Open `word_search_program.py` in IDLE. The first line is an import statement. It imports all the functions of `word_search_functions.py` file. (Note: `word_search_functions.py` must be in the same directory as `word_search_program.py`.)

main

Now scroll to the bottom of the file to read the `main` function, which is made of two assignment statements and a function call to the game loop function, `play_game`.

- The first assignment statement creates a static game board. The puzzle is represented as a string, and the main program assigns that string to variable `puzzle`. Notice that the letters in the puzzle are the same as what you see in the grid below the title `Where's That Word?` in the image shown above.
- The second assignment statement creates a list of the possible words from which the player can guess.
- The last statement is a function call, `play_game` is executed to play the whole game once. When that function exits, the game will be over.

other defs

Now scroll to the top of the file and read the function declarations and `doc_string`s.

Functions `get_num_rows` and `get_num_cols` need to count how many rows and columns there are for the given game board, the string `puz`. Find how it can be done from the parameter format, write examples in the `doc_string` to show you understand their role. The function bodies for now are returning *dummy* values.

Task 1 Fix the two functions using string functions (should be one line each).

Function `print_puzzle` role is to print the board with row and column numbers and proper spacing between columns as shown on the first page image.

Task 2 Complete `print_puzzle` body.

Function `print_words` should print the remaining words to be guessed.

Task 3 Complete `print_words` body.

Function `game_over` should return `True` if there are no more words to guess: parameter is the empty list, `False` otherwise.

Task 4 Complete `game_over` body in **one line**.

The last function `get_direction_calculate_score` does many things and we will leave it for now. Read the implemented method `take_turn`, whose body is commented and you will need after.

Open the `word_search_functions` file now and complete the tasks described in Part II. In PartIII you will complete `get_direction_calculate_score` and uncomment the body of `take_turn`.

Part II: `word_search_functions.py`

Basic helpers

At the very top, capitalized names indicate declaration of global constants. There are four of them, use them to make your code most robust.

Task 5 Complete `get_current_player` body.

Task 6 Complete `get_row` and `get_column` bodies to correspond to the written `doc_string`s.

Task 7 Rewrite the function `reverse` to be recursive. It should not have a loop but `reverse` should call itself recursively.

Task 8 Similarly **rewrite** the function `contains` to be a recursive function. It should not use `in` or a loop.

Now you will call these helper functions in your `word_search_program` when you need to get a row or a column out of the puzzle, reverse a row or a column, and check whether one string is inside another string.

Scoring mechanism

The number of points earned for finding a word depends on the word's direction and on the number of words remaining to be found. Each direction has a constant multiplicative factor shown below. (These are ordered by how difficult it is to find a word in each direction.)

Direction	Factor
`FORWARD`	1
`DOWN`	2
`BACKWARD`	3
`UP`	4

- If there are five or more words remaining to be guessed (including the current guess), the number of points for a move is simply five times the appropriate factor.
- If the number of words remaining to be guessed is fewer than five, then the total points earned for finding a word is 10 minus the number of words left (including the current guess), all multiplied by the factor for the appropriate direction.
- On top of that there is an extra bonus of 25 points for finding the last remaining word from the list.

Task 9 In `word_search_functions.py` according to the scoring mechanism just describe write the function and `doc_string` for `calculate_score`. This function takes as parameters

- a puzzle string,
- a direction (one of the '`up`', '`down`', '`forward`' and '`backward`' strings),
- the guessed word,
- the row or column number and
- the number of words remaining to be guessed (including this guess) and returns the appropriate score if this guess is found in this direction and location at this point in the game (not having been found before); Otherwise, return 0.

You should use constants, helper functions you wrote and possibly new ones if necessary.

Part III: Implementing the game loop

Now you are ready to complete in `word_search_program.py` the `get_direction_calculate_score` function. `get_direction_calculate_score` has to do many things

- prompts the current player for a direction and a row or column (handle invalid input),
- figures out how many words are left, and
- calls `calculate_score`, which is a function that you just completed in `word_search_functions.py`.

Task 10 Complete `get_direction_calculate_score` body.

Task 11 After uncommenting the body of `take_turn` complete `play_game` to run the whole program, not only print the board. As long as the game isn't over, `play_game` should

- print the puzzle and the list of remaining words,
- figure out the name of the current player, has them take a turn to guess a word, and
- update that player's score.

At the end of the game, the results should be printed:

- '`Player One wins!`',
- '`Player Two wins!`' or
- '`Tie game!`'
  as appropriate based on the score. In this game, the highest score wins.

Task 12 Test your program, make sure you handle user-errors nicely. The program should not crash if a word isn't in the list. Change the code where appropriate.

Part IV: Dynamic input

Look at the folder `1`, it contains two files `puzzle.txt` and `words.txt`

Task 13 Extend the program to propose the user to enter a number (the name of a directory) to play a search game of their choice. If so instead of the static, hardcoded `puzzle` string and `word` list values you used, the two files content provide the word search game configuration.

For Fun

If you wish to create new directories, you may use the following site. How should you print the header numbers for a large grid? You may

```
    1   2   3   4   5   6   7   8   9   1   1   1
                                        0   1   2
 1  .   .   .   .   .   .   .   .   .   .   .   .
 2  .   .   .   .   .   .   .   .   .   .   .   .
 3  .   .   .   .   .   .   .   .   .   .   .   .
 4  .   .   .   .   .   .   .   .   .   .   .   .
 5
 6
 7
 8
 9
10
11
12
13
```

Thought question: How would you program your own word search game just from the `wordlist.txt` file? Players could decide the grid size and you auto-generate a game!

Feel free to submit a second version of your game if you decide to implement such feature.