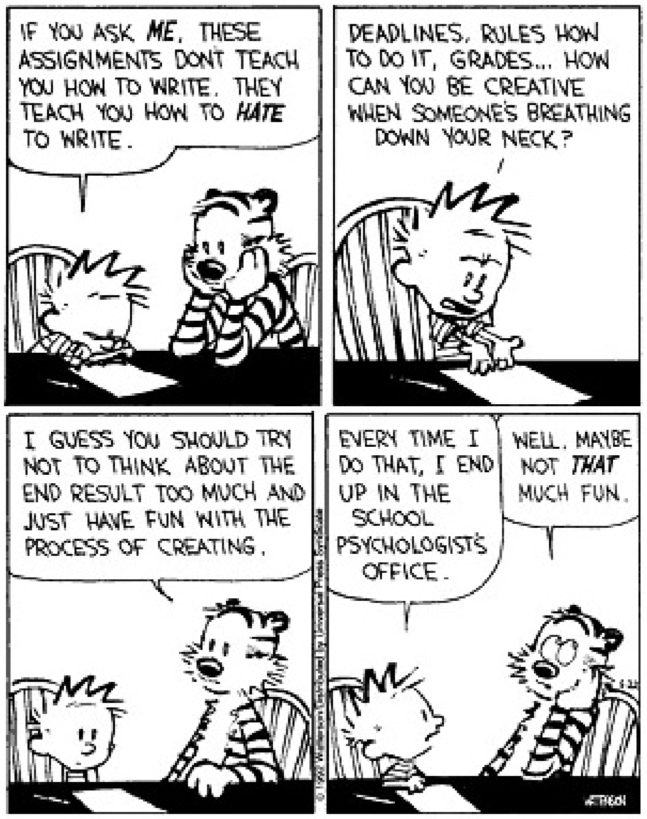
CS150 - Test Project 2: **Boggle**

Due: Friday Dec. 6, at 6pm


IF YOU ASK *ME*, THESE ASSIGNMENTS DON'T TEACH YOU HOW TO WRITE. THEY TEACH YOU HOW TO *HATE* TO WRITE.
DEADLINES, RULES HOW TO DO IT, GRADES... HOW CAN YOU BE CREATIVE WHEN SOMEONE'S BREATHING DOWN YOUR NECK?
I GUESS YOU SHOULD TRY NOT TO THINK ABOUT THE END RESULT TOO MUCH AND JUST HAVE FUN WITH THE PROCESS OF CREATING.
EVERY TIME I DO THAT, I END UP IN THE SCHOOL PSYCHOLOGIST'S OFFICE.
WELL, MAYBE NOT *THAT* MUCH FUN.

A test project is an assignment that you complete on your own, without the help of others. It is a form of take-home exam. You may consult the book, your notes, your previous assignments, the notes and examples on the course web page and the Python library documentation (linked on the course web page), *but use of any other source is forbidden.* You may not discuss these problems with anyone aside from the course instructor.

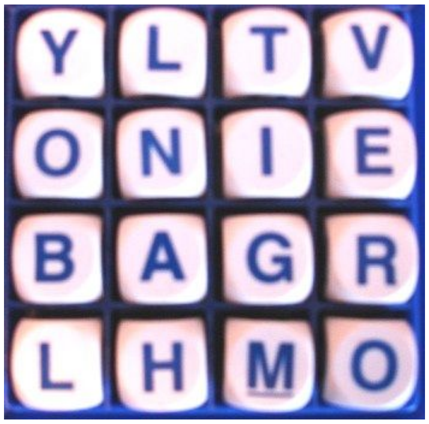
To be extra clear, you may not work with, discuss, or in any other way collaborate with anyone else.

You are encouraged to reuse code from your assignments or our class examples. Partial credit will be awarded, so try and get as far as you can.

1 The game

For this program, we will be implementing a text-based version of the game Boggle. There are many variations of this game (see the Wikipedia page for some discussion of this), so make sure to read the description below carefully.

Boggle is played with 16 dice, each of which have letters written on a side. The game is started by shaking up the dice and allowing them to fall into a four by four grid, e.g.



The goal is to make as many words as possible using the 16 letters that are facing up within a given time limit. In the real version of Boggle, the words can only be constructed from sequentially adjacent cubes, however, we will not have this restriction[1].

Words are scored based on how long they are

[1]For example, `tier` and `loan` would both be a words that meets the adjacency requirements (diagonal counts as adjacent). `viola` is a word that we will count for our version, but that would not count in the real version of the game.

word length	score
1, 2	0
3, 4	1
5	2
6	3
7	5
8+	11

and the overall score is the sum of the words found.

Text-based version

We will be implementing this game via a text-based interface. When the game starts, the 4 by 4 grid of letters is shown and the user is prompted to enter a word:

```
NTEO
ETIR
EPEM
RIIO

Enter a word:
```

Once the user enters a word, four things can happen:

1. The word cannot be made from the letters on the board:

   ```
   Enter a word: rhino
   rhino cannot be made from the board letters
   ```

2. The word is not an English word:

   ```
   Enter a word: neer
   neer is not a proper word
   ```

3. The word is valid:

   ```
   Enter a word: tie
   Good word!
   ```

4. The word is valid, but has already been guessed previously:

   ```
   Enter a word: tie
   tie has already been used
   ```

Regardless of which option occurs, the board is shown again and the user is asked to enter another word.

This continues until the time limit for the game expires. When the time expires, the user is told that the game is over and the number of correct words is shown along with their Boggle score for those words.

For example, below is the output of a (short) game:

```
NTEO
ETIR
EPEM
RIIO

Enter a word: tie
Good word!

NTEO
ETIR
EPEM
RIIO

Enter a word: tie
tie has already been used

NTEO
ETIR
EPEM
RIIO

Enter a word: rhino
rhino cannot be made from the board letters

NTEO
ETIR
EPEM
RIIO

Enter a word: totem
Good word!

NTEO
ETIR
EPEM
RIIO
```

```
Enter a word: neer
neer is not a proper word

Time's up!
Words found: 2
Total score: 3
```

2 Requirements and details

http://www.willa.me/2013/01/the-boggle-effect.html

Below are the specific requirements that you must implement. If there is any question or ambiguity, please let me know. Before you submit your final version **make sure to read through all of the requirements again and make sure you've satisfied them**.

- When your program is run it should automatically start playing. If your program is imported, it should not start playing.
- The game should last for 3 minutes. Note that like our previous attempts at timing games that involve user input, the check to see if the time is up cannot happen while we're waiting

for the user to enter a word. Therefore, the actual game will likely last a bit longer than three minutes if the program is waiting for the user to enter a word.

- You must follow the output formatting outlined above.
- Your game must handle input words with any casing (i.e. lowercase, uppercase and a mixture). For example, "`TiE`" would still be a valid word for the board above.
- You must follow the specification outlined above, specifically:
  - You must check for repeat words.
  - You must check to make sure the word is a proper English word. I have included a list of English words in a file at:

    `http://www.cs.middlebury.edu/~dkauchak/classes/cs150/assignments/TP2/`

    which you should use.
  - You must check to make sure that the word can be found in the board letters. To do this, you must use either a `set` or a `dictionary`. There are two variations on how you can do this and and one will *only give you partial credit.*
    1. To receive full credit, each letter on the Boggle board may only be used once. For example, for the board:

       ```
       NTEO
       ETIR
       EPEM
       RIIO
       ```

       `tripper` would `NOT` be a valid word since there is only one 'p', but `meter` would be since there are two `e's`.
    2. I give some hints below on how to implement option 1, however, if you have trouble getting this version implemented, you may also implement a version where you only check that each letter used in the guessed word occurs in the board. In this variation, `tripper` *would* be a valid word since all of the letters in the word occur on the board. If you implement this version, you will lose 1 point from your final score.
  - You must use Boggle scoring.
  - To generate the board, randomly pick letters from the following string:

```
BOGGLE_LETTERS = "E"*19 + \
                 "T"*13 + \
                 "AR"*12 + \
                 "INO"*11 + \
                 "S"*9 + \
                 "D"*6 + \
                 "CHL"*5 + \
                 "FMPU"*4 + \
                 "GY"*3 + \
                 "W" * 2 + \
                 "BJKQVXZ" * 1
```

This will give you the same distribution over letters as the original game and makes life much easier than trying to keep track of 16 dice.

3 Implementation Suggestions and Hints

There are many ways of implementing this program. Below are some suggestions.

The design

As always, I strongly suggest that you spend an hour or two thinking about the program before you write it. Some things to think about are:

- How are you going to create the board?
- How are you going to check if the word is valid, i.e. in the list of valid English words, not a repeat and made up of valid letters? Think particularly hard about this last question.
- What are the main things your program will need to do? Can you break it into smaller parts (i.e. functions)? Walkthrough the flow of the game and think about how the parts will interact. You probably won't get it right the first time, but it's good to think about the program and make some notes before starting coding.

One approach to implementation

Here is an incremental approach to implementation. Even if you don't do it this way I strongly urge you to take *some* incremental approach. Do **not** move on to the next step until you have that particular step working and tested!

1. Figure out how you are going to represent the board. Generate a random board and get it displaying on the screen.
2. Get a basic game playing loop going that displays the board to the user, prompts the user for a word and repeats this until the time expires.
3. Incrementally add in the different guess "correctness" checks.
4. Add in code to keep track of the number of words correct and the scoring and then print out this information when you're done.
5. Review your code and the style guidelines below and make sure your code has good style :) Consider breaking up larger chunks of code into functions and make sure the way you've broken things up is reasonable.
6. Have fun playing Boggle!

Extra Credit

You may add any extra credit functionality that you like that does not make the game simpler. Below are some possible suggestions. For any extra credit options that you add, make sure to include them in the comments at the top of the program. You may do up to 1 point of extra credit.

- [**0.5 points**] Create a turtle graphics representation of the board when the game starts.
- [**1point**] Implement the real Boggle checking, where only adjacent letters are considered correct.
- [**1 point**] Allow the user to ask for a hint and somehow suggest a word (or better, part of a word).
- [**? points**] Use your imagination!

Be careful about style when adding extra credit features!

Style

A third of the points for the test project come from style, so it's important that you think about it before submitting. Here are some of the things that I will be looking for:

- Did you properly document your code? This means a program-level docstring, function docstrings and some comments for complicated parts of the code.
- Did you use good names for functions and variables?
- Is your code readable? This includes avoiding long lines that wrap as well as using both spaces and empty lines to make your code easier to read.
- Did you use global variables, i.e. variables used inside a function that were declared outside a function? You shouldn't have! These should be passed as parameters.
- Did you properly break your code into functions? Each function should represent roughly a single task. If you find yourself putting more than one task in a function, break it into two separate functions.
- Did you reuse code well? If you find yourself copying and pasting code, or writing code in multiple places that does roughly the same thing, then figure out a way for that code to be shared. Sometimes it takes a bit of work, but this will be something I'm looking for in particular for this test project.
- Did you use a good data representation? For example, if you're asking "set"-like questions (e.g. `in`) don't use a list.

- Did you use constants appropriately? Similarly, did you avoid having hard-coded numbers in your code? Some of this is reasonable, but it should be avoided in cases of things that might change in the future.
- Are your if/else constructs properly setup?
- Are you doing anything that is grossly inefficient? We haven't talked about this a lot, but certain things, like repeatedly reading a file, etc. fall under this category.

4 Submission Procedure

For this assignment you should just have a single .py file. Submit in via the normal course submission on the course web page.

If you're board (*sic*): `http://xkcd.com/chesscoaster/`

Grading

		points
style/comments		18
	if/while/for	
	variable naming	
	comments/docstrings	
	formatting	
	parameters	
	additional functions	
	representation choices	
	misc	
functionality		
	board displays correctly	2
	proper output	3
	correct random letters	2
	run vs. import properly	2
	timing and looping	2
	lower vs. upper case	1
	repeat words	3
	proper English word check	3
	word on board check	6
	Boggle scoring	3
	misc. correctness	6
extra credit		1
total		53 (+1)