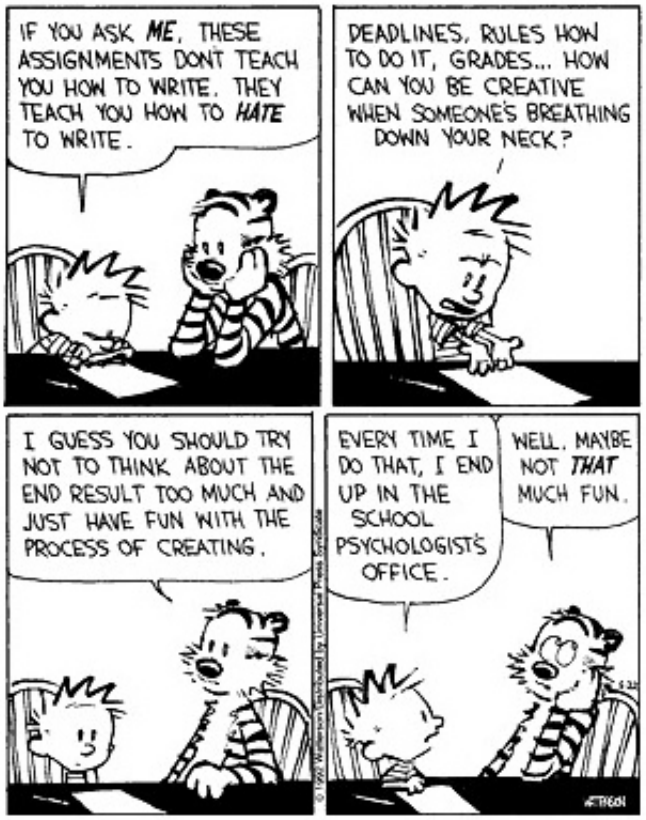
CS150 - Test Project 2
Memory
Due: Monday, May 12, at 11:59pm



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* 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 "memory" (aka concentration). See the Wikipedia page on "concentration" for more information.

When the game starts, all of the *cards* are face down. In our version, there are 16 *cards* and each of the *cards* is indicated by a number on the screen. The *cards* are laid out in four rows, each with four *cards*:

```
1  2  3  4
5  6  7  8
9  10 11 12
13 14 15 16
```

"Underneath" each of these numbers is a letter. The user does not see these, but the program will keep track of these. For example, the board above may have the following letters underneath:

```
I  D  B  F
H  B  C  D
A  A  E  E
F  H  I  C
```

The game proceeds by having the user pick two squares to look under by entering the numbers of the squares separated by a space. For example, to view 1 and 2 the user would type "1 2"; to view 7 and 15 the user would type "7 15".

The two numbers/cards that the user specified are then shown. For example, in the above case if the user entered "7 15" they would see:

```
1  2  3  4
5  6  C  8
9  10 11 12
13 14 I  16
```

If the user's selection do NOT match (like 7 and 15) then board with the cards turned over is displayed for 2 seconds and then the cards are turned back over leaving just the numbers:

```
1  2  3  4
5  6  7  8
9  10 11 12
13 14 15 16
```

The user is then prompted for another choice. If the user selects two that do match (for example 2 and 8) then those two are flipped over and they stay flipped over for the rest of the game. The user is then immediately prompted to enter an additional pair.

The game ends when the user identifies all of the matching pairs. When the game ends, the text "You win!" is displayed along with the finished board and the amount of time it took the player to solve the game. For example:

```
You win!
I  D  B  F
H  B  C  D
A  A  E  E
F  H  I  C

It took you: 28.4567864069 seconds
and 12 guesses
```

Demo

To help you understand how the game works, I've made a demo version available.

On mac:

- Go to:

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

- Download `mac.memory.zip`
- Unzip the file. This should create a directory called `memory`.
- Inside this folder, double-click on the file `memory`. This should start the memory game.

On windows:

- Go to:

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

- Download `windows.memory.zip`
- Unzip the file. This should create a directory called `memory`.
- Inside this folder, double-click on the file `memory.pyc`. This should start the memory game.

2 Requirements

- Your program must follow the specifications outlined above:
  - The displayed board must look exactly like the board above with the columns lined up correctly.
  - It must display the flipped over entries for 2 seconds *if the guess is incorrect*. If the guess is correct it immediately shows the updated board and prompts for another guess.
  - The game should stop/finish when all of the entries have been correctly guessed and are flipped over. You should then display the amount of time it took and the total number of *valid* guesses that the user made. A valid guess is any guess where the cards are flipped over, both when they match and don't match.
  - The underside of the entries will be pairs of the letters 'A' through 'H'. Each time the game is played you should get a different configuration of the letters.
- When run, your program should start playing the game automatically. If your module is imported, however, the game will not start playing.
- You can assume that the user enters two numbers separated by a space. However, you must check that the user input is valid, specifically:
  - that the numbers are valid entries on the board (1-16)
  - that the entries corresponding to the numbers entered have not already been correctly guessed (i.e. flipped over)
  - that the two numbers are different (e.g. "3 3" is NOT valid)
- Your program must make use of appropriate data structures for storing the game state.

3 Implementation thoughts

As always, I strongly suggest an incremental approach to developing this program. Before you start programming, develop a design that describes what functions you will need, the parameters each function should take and how each function will work. You won't turn this in, but it will save you a lot of time and grief if you think through the design of the program first.

As you start to code, work incrementally: pick one function and get it working before moving on.

Read through this entire section before you start the program since I give some hints about implementing certain parts.

3.1 Representation

As you're thinking about how you will write your program, think about what information you need to store and update as the game progresses and how you are going to represent this information. For example, for hangman we had a number of pieces of information:

- The word the user was trying to guess
- The current version of the guessed word
- The letters that had been guessed so far
- The number of guesses that the user had left

What information do we need to keep track of for this game? As you think about this, think about how you will store this data (as a string, a list, a set, a dictionary, etc.). You should choose an option that is convenient and efficient. You will be graded based on your choices (for example, if you try and do it all with strings, you will not get full credit).

3.2 Game flow

Besides the representation, also think about the general flow of the game. There will be some sort of loop where you repeat some steps over and over as long as the game is in session. Think about what will happen during one "turn":

- the current version of the board should be displayed
- get the guess from the user
- check the validity of the user's guess
- check whether the guess is a match or not and act accordingly

This is a rough skeleton of the flow of the game. Flush out the details a bit more before you start coding and it will make your life much easier down the road.

3.3 One route

There are many ways of implementing this and getting it working. Here is one approach:

1. Pick your board representation and figure out a way to generate a random board configuration. For each number, the board will need to keep track of which letters are associated with each letter.

   Hint: the `random` class has a function called `shuffle` that takes any list and randomly shuffles all of the elements in the list. For example, try shuffling [‘A’, ‘A’, ‘B’, ‘B’]?

2. Decide how you're going to keep track of which numbers on the board have been correctly guessed. Write some code to display the board. The board should be printed out as 4 lines. If a square/number has not been correctly guessed, then the number will be printed. If the square/number has been correctly guessed, then the letter will be printed.

   To check that you have this working correctly, manually add numbers that have been "correctly guessed" and make sure that the board is displayed correctly by your code.

3. Get the input from the user and then regardless of whether they match or not, show the updated version of the board (i.e. with the letters flipped over the selected numbers), then revert back to the original board after the 2 second pause.

   Warning: For this program, it is important that you either run the program from the command-line OR run it in debug mode. If you run the program with the green arrow you will not be able to get the user interface to work properly.

   Hints:

   - In the `time` module there is a function called `sleep` that takes the number of seconds as a parameter and the program stops executing for that number of seconds.
   - You cannot "delete" text that you've already printed on the screen. However, one way to make it look like the text has been deleted is to print out a number of blank lines (e.g. 100). This causes anything that was on the screen to disappear. If you do this every time before you display the board, it will appear to the user like the board is simply being updated.
   - There are many ways of updating the board to reflect the current guess and then undoing them if the guess is not right. Think about the best approach for your implementation.

4. Update the functionality to differentiate when the pairs match vs. when the pairs don't match. To help you when you're debugging, you can print out what the "underneath" letters are, though make sure to remove this printing before you submit.

5. Add checking to make sure that the user enters valid numbers. *You may assume that they enter two integers separated by a space.* You must however check to see if they are valid board spaces and whether or not they have already been selected. As long as either of these cases applies, then you should prompt the user to enter another selection.

   If you write a separate function that checks all of these different scenarios in one place, then it should make your code much simpler.

6. Add functionality to check if the user has won and print "You win!" when the game is won. Print the time it took the user to solve the problem. At this point you should have a working version of the memory game.

   Hint: During testing, it may be easier if you make your game easier (for example, just a 2 by 2 board).

When you're done

Make sure that your program is properly commented:

- Your name and section number (A or B) should be at the top of the file.
- You should have a docstring comment for each module at the top of the file

- You should have comments after the module docstring stating your name, course (including section number), assignment number and the date.
- Each function should have an appropriate *docstring*
- Other miscellaneous comments to make things clear

In addition, make sure that you've used good ***style***.

Submission procedure

Submit your .py file through the normal course submission mechanism with the course number as "TP2".

Grading

		points
style/comments		15
	if/while/for	
	variable naming	
	comments/docstrings	
	formatting	
	parameters	
	additional functions	
	representation choices	
	misc	
functionality		
	board displays correctly	6
	board is randomly initialized	3
	loops until game completed	4
	properly handles match	4
	properly handles non-match	4
	handles improper user input	6
	times game	3
total		45