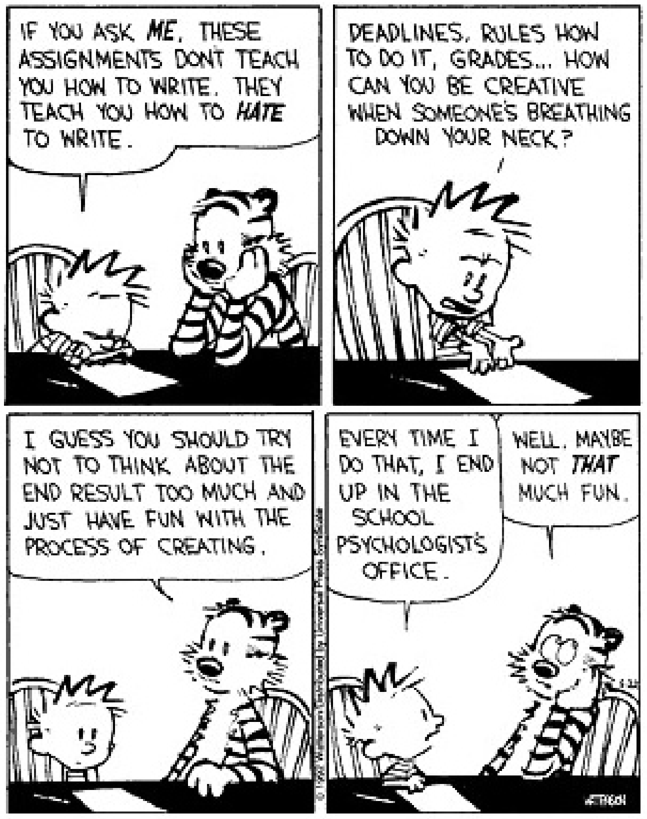
CS150 - Test Project 2
Mancala
Due: Friday Dec. 7, at 6pm


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.

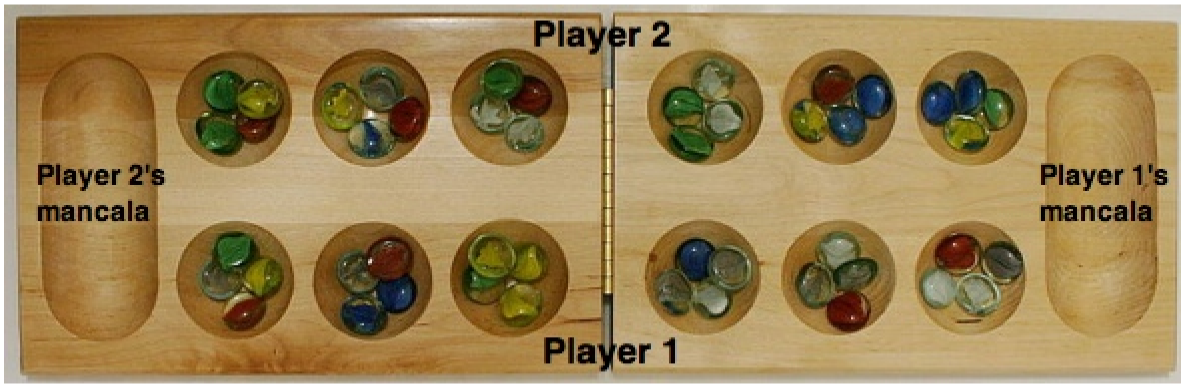
A test project is an assignment that you complete on your own, without the help of others. It is a form of take-home exam. You may consult the book, your notes, your previous assignments, the notes and examples on the course web page and the Python library documentation (linked on the course web page),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

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 mancala. There are many variations of the game (see the Wikipedia page for some discussion of this), but the variant we'll be implementing works as follows.

The game is played with two players. Each player has six "bowls" on their side and their "mancala" is a larger bowl to their right. For example, here is a board:



When the game starts, each of the 12 bowls contains four stones in them. A player's turn consists of the following:

- Pick one of the non-empty bowls on the player's side.
- Pick up all of the stones in that bowl and, moving counterclockwise, drop one stone in each bowl starting with the one to the right. This includes all bowls and the two mancalas.
- *If* the last stone place is in one of the 12 main bowls *and* that bowl is empty, then the player takes all of the stones from the bowl directly across from the finishing bowl and places them in his/her mancala.

The game ends when one of the players cannot make a move (i.e. has no more stones on his/her side). The winner is the person with the most stones in their mancala when the game finishes.

This is all we will be implementing, though there are many other variants of the game.[1]

Text-based version

We will be implementing this game via a text-based interface. Each time it is a player's turn, they will be shown the current board and a set of corresponding numbers for each bowl for use in choosing a move. For example, if it's player 1's turn, they would see:

[1]One common variant that makes the game much more interesting is that you get another turn if you finish in your own mancala.

```
--------------------
Current board:
  4  4  4  4  4  4
0                  0
  4  4  4  4  4  4
--------------------

Select a move:
  -  -  -  -  -  -
-                  -
  1  2  3  4  5  6

Player 1 -- from 1 to 6:
```

The game then waits for player 1 to enter their selection.

Let's say player 1 chooses bowl 3. The move is made, the board is updated and then Player 2 is shown their options:

```
--------------------
Current board:
  4  4  4  4  4  4
0                  1
  4  4  0  5  5  5
--------------------

Select a move:
  13 12 11 10 9  8
-                  -
  -  -  -  -  -  -

Player 2 -- from 8 to 13:
```

Notice a few things:

- The bowl that was selected by player 1 is empty.
- The stones from that bowl were circulated to the right, one per bowl including the mancalas.
- The moves shown are customized based on whose turn it is.

Finally, just to make sure that the rules are clear, let's say that player 2 now selects bowl 13. The resulting board state will be the following:

```
--------------------
Current board:
  0  4  0  4  4  4
5                 1
  5  5  1  5  5  5
--------------------

Select a move:
  -  -  -  -  -  -
-                 -
  1  2  3  4  5  6

Player 1 -- from 1 to 6:
```

and it will be player 1's turn again. Notice that because player 2 ended in an empty bowl (i.e. bowl 3) they captured all of the stones in the bowl that was directly across from bowl 3 (i.e. bowl 11), moving those stones to player 2's mancala.

Requirements

http://www.clubpenguinsaraapril.com/2009/07/mancala-game-in-club-penguin.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 with Player 1 going first. If your program is imported, it should not start playing.
- Your board and moves should be displayed exactly as above. In particular:
  - You must insert dashes ('-') for moves that are not eligible.
  - The bowls on both sides must line up.
  - You must properly handle double digit bowl numbers, both in the select a move portion as well as in the current board.
- The game should alternate turns between player 1 and player 2 until the game is finished.
- When the game is finished, you must print out the final board configuration, which player won and the scores for each of the players. The scores for a player are the number of stones in their mancala.
- When a player enters a move, you *may* assume that they enter a valid number (e.g. 1-6 for player 1 and 8-13 for player 2.
- You must check to make sure that the bowl entered by a player is not empty. If it is, you should prompt them again to select a move until they select a valid bowl. For example:

```
--------------------
Current board:
  0  4  0  4  4  4
5                  1
  5  5  0  6  5  5
--------------------

Select a move:
  13 12 11 10 9  8
-                  -
  -  -  -  -  -  -

Player 2 -- from 8 to 13: 11
That bowl is empty!
Player 2 -- from 8 to 13: 11
That bowl is empty!
Player 2 -- from 8 to 13: 11
That bowl is empty!
Player 2 -- from 8 to 13:
```

Implementation Suggestion and Hints

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

The design

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

- How are you going to keep track of the board state? Things you need to do with the board are:
  - Print it out.
  - Make moves. A move involves moving the pieces around the board counterclockwise. Make sure your board supports this functionality in a reasonable way.
  - Figure out if particular spaces are empty.
- What are the main things your program will need to do? Can you break it into smaller parts? 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 board with the initial configuration.
2. Now that you have a basic board, see if you can get it displayed. Try to get the initial board configuration displaying properly, then check to make sure you can handle different variations that include double digit numbers.
3. Figure out what you need to do to make a move and add some code to change the board to make a given move. Test to make sure that moves from both sides work. It can be helpful to draw a picture here with your board on a piece of paper to help yourself visualize what should be happening.
4. Figure out how to print the move choices for each player.

5. Try and put together a basic version of the game with what you've implemented so far, which takes turns between the two players and allows the players to pick piles and makes the specified moves. You'll need to alternate turns between the players, but try and reuse as much functionality as you can between the two players (i.e. you shouldn't have two separate copies of everything).
6. Add in move checking to make sure that a user doesn't select an empty bowl. If they do, repeatedly ask them to pick another bowl.
7. Add in functionality to check if the move ended in an empty bowl. If they did, you'll need to figure out what the bowl directly across from it is and then move those to the player's mancala.
8. Right now, your game could play forever. Add functionality to check for a win and stop if the game is over. Print out who the board and who won.
9. Review your code and the style guidelines below and make sure your code has good style :)

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.

- [**0.5 points**] Add in an extra turn for a player if they end in their mancala.
- [**0.5 points**] Don't show options that are empty in the "move" board.
- [**0.5 points**] Check to make sure that the user actually enters a valid number appropriate for the given player (e.g. 1-6 for player 1) and re-prompt if they do not. The `isdigit` function for strings may be useful.

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.

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.

Extreme board gaming: `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	5
	moves display correctly	3
	moves work properly	5
	alternates between players correctly	2
	properly checks for empty bowl selection	5
	properly handles landing in empty bowl	5
	loops until game completed	5
	prints winner	3
	runs vs. import	2
extra credit		1
total		53 (+1)