CMSC 201 Fall 2015
Project 1 – Conway's Game of Life

Assignment: Project 1 – Conway's Game of Life
Due Date: Tuesday, November 17th, 2015 by 8:59:59 PM
Value: 8% of final grade

Project 1 is the first assignment where we won't be telling you exactly what to do! You will get the chance to make your own decisions about how you want your program to handle things, what its functions should be called, and how you want to go about designing it.

Project 1 will also be ***substantially longer*** than any of the single homework assignments you've completed so far, so make sure to take the time to plan ahead, and don't do any "cowboy" coding!

Remember to enable Python 3 before you run your programs:

```
     /usr/bin/scl enable python33 bash
```

Instructions

For this assignment, you'll need to follow the class coding standards, a set of rules designed to make your code clear and readable. The class coding standards are on Blackboard under "Course Documents" in a file titled "CMSC 201 - Python Coding Standards."
You will **lose major points** if you do not following the 201 coding standards.

A very important piece of following the coding standards is writing a complete **file header comment block**. Make sure that your file has a comment block at the top (see the coding standards document for an example).

NOTE: Your filename for this project must be `proj1.py`

NOTE: **You must use `main()` in your file.**

Details

For this project, you will be coding a simple cellular automata game, called Conway's Game of Life. In this game, you have a grid where pixels can either be on or off (dead or alive). In the game, as time marches on, there are simple rules that govern whether each pixel will be on or off (dead or alive) at the next time step. These rules are as follows:

Any **live** cell with fewer than <u>two</u> live neighbors dies.
Any **live** cell with <u>two or three</u> live neighbors lives on to the next generation.
Any **live** cell with <u>more than three</u> live neighbors dies.
Any **dead** cell with <u>exactly three</u> live neighbors becomes a live cell.

To begin, you will ask the user for the size of the game board (rows first, then columns). Next prompt them for any cells they would like to be "alive" when the game begins. Finally, ask the user how many iterations of the game (number of time steps) they would like to see run. You should then display these iterations.

"Live" cells are to be represented with the character "`A`" (capital 'a'), and "dead" cells are to be represented with the character "`.`" (a period).

HINT: It would be a good idea to:

- *Store your board in a 2D list (make sure you don't mix up column and row!)*
- *Have a function called* **`nextIteration()`** *that takes the current board in as a parameter, and that returns a new board with the next iteration*
- *Have a function called* **`printBoard()`** *that takes in the current board as a parameter and prints out the board's contents*

<u>TIP</u>: This would be a <u>very</u> good time to use incremental programming! Incremental development is when you are only working on a small piece of the code at a time, and testing that the piece of code works before moving on to the next piece. This makes it a lot easier to fix any mistakes.

If you want to make sure you've done everything correctly, the Wikipedia page (https://en.wikipedia.org/wiki/Conway's_Game_of_Life) shows some other examples of the game.

Input Validation

For this project, we will require that you validate input from the user. You can assume that the user will enter the right type of input, but not that they will enter a correct value. In other words, a user will always give an integer when you expect one, but it may be a negative or otherwise invalid value.

You will need to validate the following things:

- Getting the numbers of rows and columns
  - Rows must be 1 or greater
  - Columns must be 1 or greater
- Getting the cells to make "alive"
  - The row must be either
    - The character "q" (for quit)
    - A valid index for the number of rows your board has (*i.e.*, it starts at index 0, and goes up to index row_size – 1)
  - The column must be
    - A valid index for the number of columns your board has (*i.e.*, it starts at index 0, and goes up to index column_size – 1)
- Getting the number of iterations to run
  - Must be 0 or greater
    (They can choose to run no iterations, in which case you would only print the starting board)

We have provided additional sample output below showing how this should work in your program.

Sample Output

Here is some **sample output**, with the user input in blue. The sample output continues on to the following page.

```
bash-4.1$ python proj1.py
Please enter number of rows:     5
Please enter number of columns: 5

Please enter the row of a cell to turn on (or q to exit): 3
Please enter a column for that cell: 3

Please enter the row of a cell to turn on (or q to exit): 2
Please enter a column for that cell: 3

Please enter the row of a cell to turn on (or q to exit): 4
Please enter a column for that cell: 3

Please enter the row of a cell to turn on (or q to exit): q

How many iterations should I run? 3
Starting Board:

.....
.....
...A.
...A.
...A.

Iteration 1:

.....
.....
.....
..AAA
.....
```

```
Iteration 2:

.....
.....
...A.
...A.
...A.

Iteration 3:

.....
.....
.....
..AAA
.....
```

Sample Output

Here is some **sample output** that shows input validation, with the user input in blue. The sample output continues on to the following page.

```
bash-4.1$ python proj1.py
bash-4.1$ python proj1.py
Please enter number of rows:    0
        That is not a valid value; please enter a number
        greater than or equal to 1
Please enter number of rows:    -1
        That is not a valid value; please enter a number
        greater than or equal to 1
Please enter number of rows:    5
Please enter number of columns: 10

Please enter the row of a cell to turn on (or q to exit): 99
        That is not a valid value; please enter a number
        between 0 and 4 inclusive...

Please enter the row of a cell to turn on (or q to exit): 6
        That is not a valid value; please enter a number
        between 0 and 4 inclusive...

Please enter the row of a cell to turn on (or q to exit): 4
Please enter a column for that cell: 6

Please enter the row of a cell to turn on (or q to exit): 4
Please enter a column for that cell: 7

Please enter the row of a cell to turn on (or q to exit): 4
Please enter a column for that cell: 9

Please enter the row of a cell to turn on (or q to exit): q

How many iterations should I run? -11
        That is not a valid value; please enter a number
        greater than or equal to 0

How many iterations should I run? 1
```

```
Starting Board:

..........
..........
..........
..........
......AA.A

Iteration 1:

..........
..........
..........
..........
..........
```

Submitting

Once your Project 1 is complete, it is time to turn it in with the **`submit`** command.

Don't forget to complete the header block comment for your file! Make sure that you updated the header block's file name and description.

You must be logged into your GL account, and you must be in the same directory as the Project 1 file. To double check this, you can type **`ls`**.

```
linux1[3]% ls
proj1.py
linux1[4]% 
```

To submit your files, we use the **`submit`** command, where the class is **`cs201`**, and the assignment is **`PROJ1`**. Type in (all on one line)
`submit cs201 PROJ1 proj1.py`
and press enter.

```
linux1[4]% submit cs201 PROJ1 proj1.py
Submitting proj1.py...OK
linux1[5]% 
```

If you don't get a confirmation like the one above, check that you have not made any typos or errors in the command.

You can **double-check that your file was submitted** by using the **`submitls`** command. Type in **`submitls cs201 PROJ1`** and hit enter.

And you're done!