CMSC 201 Fall 2015
Project 2 – Auto-fill

Assignment: Project 2 – Auto-fill
Due Date: Tuesday, December 8th, 2015 by 8:59:59 PM
Value: 8% of final grade

Project 2 is your final assignment in 201! Just like with Project 1, we won't be telling you exactly what to do. Instead, 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 2 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 `proj2.py`

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

Details

For this project, you will be implementing a tool called “Auto-fill”. If you’ve ever used a basic image editing software (like MS Paint), you’ve likely seen a similar function.

You will be given a text file full of characters that form an enclosed set of spaces, similar to the one below:

```
+---+-----+
|   |     |
|   |=====|
|   |     |
+---+-----+
```

The grid may be any size, but will always be fully enclosed (there will be a “border” of characters along the outer edges).

Your autofill program will ask the user to provide a point in the grid, and will “fill” the entire space it belongs to using a character, also provided by the user.
A space in the grid is only empty if it is occupied by a space character (“ ”). All other characters count as the space being occupied.

Your auto-fill function must be recursive!

Here is a **simplified** example run of the program, with the user inputs in blue. *(This does not show the full functionality your program should have.)*

```
bash-4.1$ python proj2.py
Please enter a file for input: simple.dat
+---+-----+
|   |     |
|   |=====|
|   |     |
+---+-----+
Please enter the coordinates you would like to start
filling at in the form "row,col", or Q to quit: 1,1
Please enter a symbol to fill with: X

============================================
```

```
+---+-----+
|XXX|     |
|XXX|=====|
|XXX|     |
+---+-----+
Please enter the coordinates you would like to start
filling at in the form "row,col", or Q to quit: 3,6
Please enter a symbol to fill with: O

=============================================

+---+-----+
|XXX|     |
|XXX|=====|
|XXX|OOOOO|
+---+-----+
Please enter the coordinates you would like to start
filling at in the form "row,col", or Q to quit: Q
Thank you for using the autofill program!
```

Your program should work as follows:

1. Ask the user for the name of a file that contains the board
2. Read in and store the board in a 2D list
3. Ask the user for a coordinate to start auto-filling from
    - They may also choose to quit at this point
4. Ask the user for a symbol to auto-fill with
5. Ask the user if they would like to see a step-by-step of the auto-fill
6. Auto-fill the board starting at that point, using their chosen character
    - This function must be recursive, and must make recursive calls to neighboring cells <u>in the following order</u>:
        1. The cell **above** the current one
        2. The cell **to the right** of the current one
        3. The cell **below** the current one
        4. The cell **to the left** of the current one
    - If you don't follow this order, your output won't match ours
7. Print out the resulting board (and the step-by-step, if requested)
8. Allow the user to choose a new coordinate (or to quit)

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 filename
  - The filename must end in ".dat" or ".txt"
- Getting the cells to start auto-filling from
  - **We guarantee** the user will enter either
    - The character "`Q`"
    - Two integers, separated by a comma
      (*e.g.*, `5,6` or `-1,99` or `0,0` etc.)
  - The row (the first number) must be
    - 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 (the second number) 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)
  - You must check both row and column
- Getting the symbol to use in the auto-fill
  - The symbol must be a single character
- Getting the choice for step-by-step
  - The user's answer must be either "yes" or "no" in all lowercase

Important Notes

As in Project 1, **indexing for the board starts at 0**, not at 1. Please also note the difference in the way the row and column are being taken in from the user. For Project 2, we are using "`row,column`" instead of taking the two as separate inputs.

When the user chooses to see a **step-by-step progression** of the recursive function, you should only print out the board if it has been ***updated***! Think carefully about where and when in your code you want to print the board in order for this to be correctly done. See the sample output below for an example of what this should look like.

Your autofill function **must be recursive**. Additionally, you must fill each of the 4 neighbors in the following order: **top, right, bottom, and left**. If you do not follow this order, your step-by-step output will not match ours.

There are three important things to remember about a recursive function:

1. It has at least one base case.
    - The base case(s) tell us when to stop. Once we reach a base case, we do not call the recursive function again.
2. It has at least one recursive case.
    - The recursive case(s) tell us when to call the recursive function again. We may have more than one recursive case, and we may make more than one recursive call.
    - A recursive call should pass in parameters that are at least slightly different from previous calls. (If we don't change the parameters passed in, we will keep repeating the same thing!)
3. The recursive call(s) need to approach the base case.
    - When we call our recursive case, the parameters need to not only be different, they need to bring us closer and closer to the base case. Think carefully about what this means for your recursive auto-fill function.

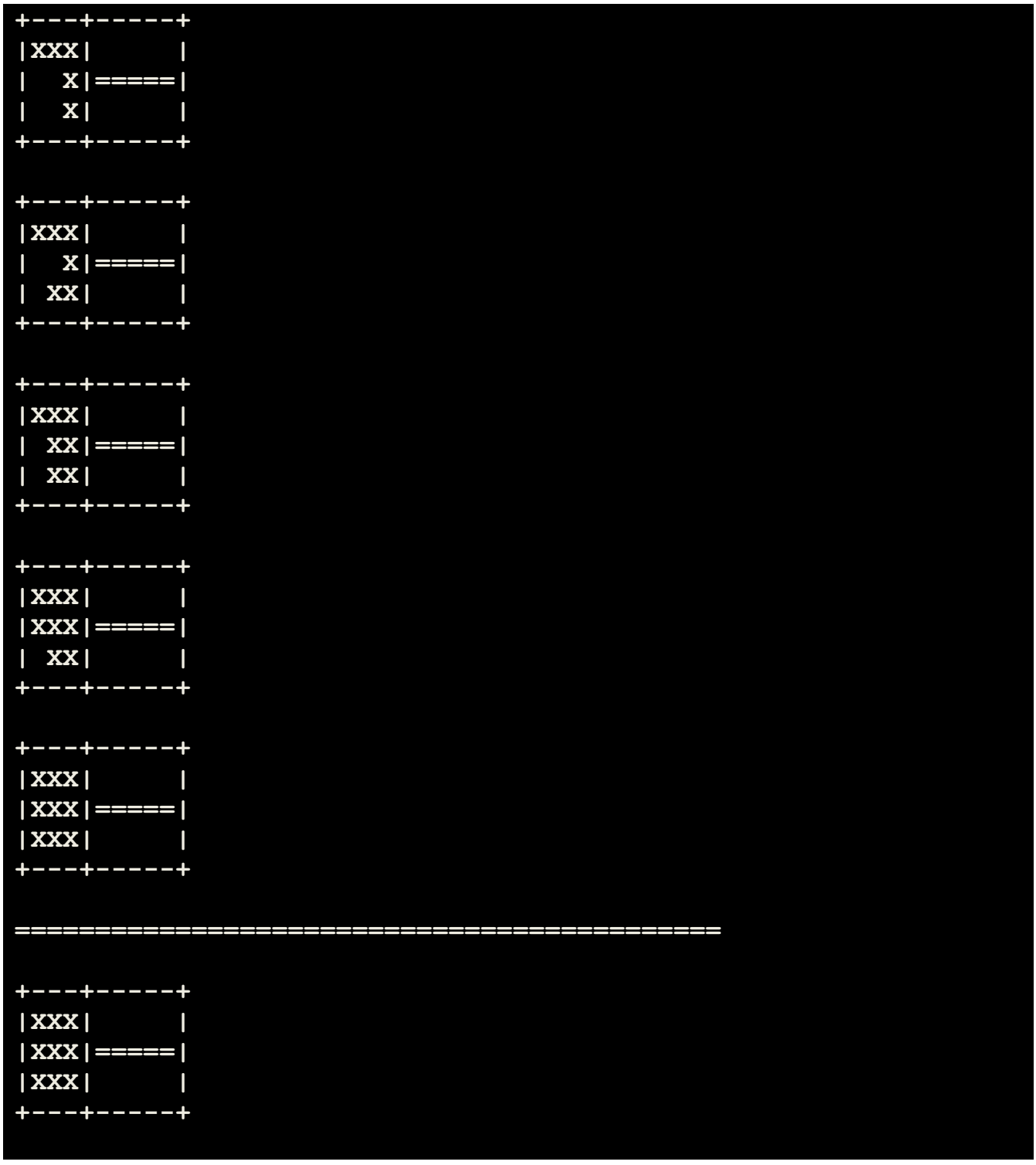
Sample Output and Input

Here is some **sample output**, with the user input in blue. The sample output continues on to the following pages. We have also provided **sample input** files following this.

Please note that when we attempt to perform an auto-fill starting at the point 0,0 nothing happens – that is the top left corner, which is already occupied by a "**+**" character, meaning it can't be filled. A coordinate in the grid is only considered empty if it is occupied by a space character ("**.**").

```
bash-4.1$ python proj2.py
Please enter a file for input: box.doc
That is not a valid filename -- please enter a filename
     that ends in ".dat" or ".txt": simple.dat
+---+-----+
|   |     |
|   |=====|
|   |     |
+---+-----+
Please enter the coordinates you would like to start
filling at in the form "row,col", or Q to quit: -1,3
        -1 is not a valid row value; please enter a number
        between 0 and 4 inclusive.
Please enter the coordinates you would like to fill at
        in the form "row,col": -1,99
        -1 is not a valid row value; please enter a number
        between 0 and 4 inclusive.
        99 is not a valid column value; please enter a number
        between 0 and 10 inclusive.
Please enter the coordinates you would like to fill at
        in the form "row,col": 1,13
        13 is not a valid column value; please enter a number
        between 0 and 10 inclusive.
Please enter the coordinates you would like to fill at
        in the form "row,col": 0,0
Please enter a symbol to fill with: %
Would you like to show each step of the recursion?
Enter "yes" or "no": no
```

```
==========================================

+---+-----+
|   |     |
|   |=====|
|   |     |
+---+-----+

Please enter the coordinates you would like to start
filling at in the form "row,col", or Q to quit: 1,1
Please enter a symbol to fill with: X
Would you like to show each step of the recursion?
Enter "yes" or "no": yes

+---+-----+
|X  |     |
|   |=====|
|   |     |
+---+-----+

+---+-----+
|XX |     |
|   |=====|
|   |     |
+---+-----+

+---+-----+
|XXX|     |
|   |=====|
|   |     |
+---+-----+

+---+-----+
|XXX|     |
|  X|=====|
|   |     |
+---+-----+
```


+---+-----+
XXX	
X	=====
X	
+---+-----+
+---+-----+
XXX	
X	=====
XX	
+---+-----+
+---+-----+
XXX	
XX	=====
XX	
+---+-----+
+---+-----+
XXX	
XXX	=====
XX	
+---+-----+
+---+-----+
XXX	
XXX	=====
XXX	
+---+-----+
==========
+---+-----+
XXX	
XXX	=====
XXX	
+---+-----+

```
Please enter the coordinates you would like to start
filling at in the form "row,col", or Q to quit: 1,6
Please enter a symbol to fill with: O
Would you like to show each step of the recursion?
Enter "yes" or "no": help
The choice "help" is not valid.
Would you like to show each step of the recursion?
Enter "yes" or "no": no

==========================================

+---+-----+
|XXX|OOOOO|
|XXX|=====|
|XXX|     |
+---+-----+

Please enter the coordinates you would like to start
filling at in the form "row,col", or Q to quit: 3,9
Please enter a symbol to fill with: ***
        The symbol "***" is not a single character.
Please enter a symbol to fill with:
        The symbol "" is not a single character.
Please enter a symbol to fill with: *
Would you like to show each step of the recursion?
Enter "yes" or "no": yes

+---+-----+
|XXX|OOOOO|
|XXX|=====|
|XXX|    *|
+---+-----+

+---+-----+
|XXX|OOOOO|
|XXX|=====|
|XXX|   **|
+---+-----+
```

```
+---+-----+
|XXX|OOOOO|
|XXX|=====|
|XXX|  ***|
+---+-----+
+---+-----+
|XXX|OOOOO|
|XXX|=====|
|XXX| ****|
+---+-----+
+---+-----+
|XXX|OOOOO|
|XXX|=====|
|XXX|*****|
+---+-----+

=============================================

+---+-----+
|XXX|OOOOO|
|XXX|=====|
|XXX|*****|
+---+-----+

Please enter the coordinates you would like to start
filling at in the form "row,col", or Q to quit: Q
Thank you for using the autofill program!
```

Sample Input

You can download a number of sample input files from my public directory by running the following command.

```
cp /afs/umbc.edu/users/k/k/k38/pub/cs201/proj2files/* .
```

Make sure you run this command from inside your proj2 folder! And don't forget to include the period at the end of the command.

Hints and Advice

TIP: **This would be a very 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.

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!)
- Reuse some of your older code in this project. For example:
  - The function to print the board should be very, very similar to the one from Project 1.
  - Code to check that a filename ends in "`.dat`" or "`.txt`" should have been something you wrote for a previous homework.
  - (This is why we break code down into functions! It makes it easier to reuse general code later!)
- Make sure you follow the directions closely!

Submitting

Once your Project 2 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 2 file. To double check this, you can type **`ls`**.

```
linux1[3]% ls
proj2.py
linux1[4]%
```

To submit your files, we use the **`submit`** command, where the class is **`cs201`**, and the assignment is **`PROJ2`**. Type in (all on one line)
`submit cs201 PROJ2 proj2.py`
and press enter.

```
linux1[4]% submit cs201 PROJ2 proj2.py
Submitting proj2.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 PROJ2`** and hit enter.

And you're done!