CMSC 201 Spring 2018

Project 2 – Battleship

Assignment: Project 2 – Battleship
Due Date:
Design Document: Friday, April 13th, 2018 by 8:59:59 PM
Project: Friday, April 20th, 2018 by 8:59:59 PM
Value: 80 points

Collaboration: For Project 2, **collaboration is not allowed** – you must work individually. You may still come to office hours for help, but you may not work with any other CMSC 201 students.

Make sure that you have a complete file header comment at the top of each file, and that all of the information is correctly filled out.

```
# File:    FILENAME.py
# Author:  YOUR NAME
# Date:    THE DATE
# Section: YOUR DISCUSSION SECTION NUMBER
# E-mail:  YOUR_EMAIL@umbc.edu
# Description:
#    DESCRIPTION OF WHAT THE PROGRAM DOES
```

For Project 2 you will have to turn in a "design document" in addition to the actual code. The design document is intended to help you practice deliberate construction of your program and how it will work, rather than coding as you go along, or starting without a plan.

Instructions

For this project, you will be creating a single program, but one that is bigger in size and complexity than any individual homework problem. This assignment will focus on manipulating lists, calling functions, and handling mutability appropriately.

The design for Project 2 is entirely up to you – suggestions are provided within the project description, but you are not required to use them.

At the end, your Project 2 file must run without any errors.
It must also be called proj2.py (case sensitive).

Additional Instructions – Creating the proj2 Directory

During the semester, you'll want to keep your different Python programs organized, organizing them in appropriately named folders (also known as directories).

You should create a directory in which to store your Project 2 files. We recommend calling it **`proj2`**, and creating it inside a newly-created directory called **`Projects`** inside the **`201`** directory.

If you need help on how to do this, refer back to the detailed instructions in Homework 1.

Objective

Project 2 is designed to give you practice with two-dimensional lists, mutability, and creating and calling functions. You'll need to use practically everything you've learned so far, and will need to do some serious thinking about how all of the pieces you need to create should fit together.

Remember to enable Python 3 before running and testing your code:

```
scl enable python33 bash
```

Task

For this project, you will be coding a very simplified game of Battleship, in which there is only one player, and the locations of the ships are "hardcoded" (that is, they are the same every game). There are only five ships, and the program must update the board to reflect the user's actions, and stop the game when the user has either run out of moves, or successfully "sunk" all five ships.

Coding Standards

Prior to this assignment, **you should be familiar with the entirety of the Coding Standards**, available on Blackboard under "Assignments" and linked on the course website at the top of the "Assignments" page.

You should be commenting your code, and using constants in your code (not magic numbers or strings).
Any strings with a meaning (*e.g.*, ship names) should be constants!
Any numbers other than 0 or 1 are magic numbers!
(See the note on the numbers for ship positioning.)

You will **lose major points** if you do not follow the 201 coding standards.

If you have questions about commenting, whitespace, or any other coding standards, please come to office hours.

Additional Specifications

For this assignment, you must create and call **at least six individual functions**, not including `main()`. All other design decisions are up to you.

You may *not* import any libraries or use any library functions! Doing so will lose you a *very* large number of points.

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 user's choice for each move, both row and column
  - Rows and columns must both be between 0 and 9, inclusive

Formatting

When printed, your board must appear ***exactly*** like the one shown in the sample output, including the row and column headings, and the dividing lines.

You can look at the sample output to see what a board should look like, but a description has also been provided below with exact character counts and descriptions.

The board shown in the sample output uses hyphens (*i.e.*, dashes) for the horizontal dividing lines, and vertical bars (*i.e.*, pipes) for the vertical dividing lines. Each square is three characters wide: a single character in the middle, and a space on either side. The row headings follow this same format (space, number, space; and then a vertical pipe). The column headings start five characters in, and are separated by three spaces.

Details

For this project, you will be coding a simplified game of Battleship. The board for the game is a 10x10 grid, which simulates an area of the ocean. On the board, there are five ships, but the user cannot see where they are. The objective of the game is for the player to sink all five ships; they are only provided with 40 torpedoes, and hence have only 40 guesses to do so.

For each turn, the player will fire a single torpedo by selecting the row and column they want to strike. The board should be updated with an "O" if the player hit "empty" ocean, and with an "X" if they hit a ship.

- If they hit the same spot twice, they must be informed that they have already struck that area. (This does count as a turn, and "costs" them a torpedo as any other move would.)
- If they hit a ship, the program must check to see if the entire ship has been sunk. If this is the case, a message must be printed out stating which ship was sunk.

If the player wins (sinks all five ships), the program should end at that point. If the player loses (uses all 40 torpedoes without sinking all five ships), the program should stop, and should print out the board so that the user can see where the ships were placed. (See the sample output for an example of what this should look like.)

Each ship has a specific length, a specific character, and an assigned set of spaces that it should occupy in the game board. You must use these when setting up your board.

Ship name	Character	Length	Orientation	Start	End
Aircraft Carrier	A	5	Vertical	2, 7	6, 7
Battleship	B	4	Horizontal	2, 0	2, 3
Cruiser	C	2	Horizontal	0, 3	0, 4
Destroyer	D	3	Horizontal	9, 5	9, 7
Submarine	S	3	Vertical	5, 4	7, 4

NOTE: You may use "magic numbers" for the start and end of the ships when you are first initializing the board. (The character and length should still be set as constants if you find them necessary.)

Hints and Advice

It would be a good idea to:

- Store your board in a 2D list (make sure you don't mix up column and row!)
  - You only need to have one board, but if you choose to use two separate boards, that is acceptable
- Remember that lists are passed by reference, and that any in-place changes to a list made within a function remain when control is returned to the calling function
- Use constants for the different characters used on the board, as well as any other numbers or strings you deem necessary
- Have a function called **`printBoard()`** that takes in the current board as a parameter and prints out the board's contents
  - Remember that the program needs to be able to print the board two ways: as the user sees it during gameplay, and with all of the information displayed at the end of a loss
- When printing the board, remember to make use of either concatenation or the **`end=""`** parameter in the **`print()`** function, to control when line breaks and spaces are printed to the screen.

If you would like to practice playing Battleship so that you understand it, I've found a similar game online that will allow you to play a two-player game against a computer. (Do note that the game board and rules are slightly different, so make sure to read the project instructions carefully.)

TIP: This would be a very good time to use incremental development! 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.

Project

The project is worth a total of 80 points. Of those points 10 will be based on your design document, 10 will be based on following the coding standards, and the other 60 will be based on the functionality and completeness of your project.

Design Document

The design document will ensure that you begin seriously thinking about your project early on. This will not only give you important experience doing design work, but will help you gauge the number of hours you'll need to set aside to be able to complete the project. **Your design document must be called design2.txt.**

For Project 2, you are creating the design entirely on your own.
You **may NOT work with another student** to "brainstorm" a solution or discuss any general approaches or requirements. If you need assistance with the design document, come to office hours.

Your design document must have four separate parts:

1. A file header, similar to those for your assignments
2. Constants
    a. A list of all the constants your program will need, including a short comment describing what each "group" of constants is for (*e.g.,* menu options, meaning of indexes, etc.)
3. Function headers
    a. A complete function header comment for each function you plan to create, including the name, description, inputs, and outputs
4. Pseudocode for main()
    a. A brief but descriptive breakdown of the steps your main() function will take to completely solve the problem; note function calls under the relevant comment (if applicable)

Your design can follow the same general format as the design for Project 1.

Your **`design2.txt`** file will be compared to the **`proj2.py`** file that you submit. Minor changes to the design are allowed. A minor change might be the addition of another function, or a small change to **`main()`**.

Major changes between the design and your project will lose you points. This would indicate that you didn't give sufficient thought to your design.
(If your submitted design doesn't work, it is generally better to lose the points on the design, and to have a functional program, rather than turning in a broken program that follows the design. The decision is ultimately up to you.)

To submit your design document, use

```
linux1[4]% submit cs201 PROJ2_DESIGN design2.txt
Submitting design2.txt...OK
linux1[5]%
```

Sample Output

The sample output is available as a separate file under "Assignments" on Blackboard, and is called "sample2.txt".

(Yours does not have to match the sample output exactly, but it should be similar.)

Submitting

Once your **`proj2.py`** or **`design2.txt`** file is complete, it is time to turn it in with the **`submit`** command. (You may also turn the design or project in multiple times, as you reach new milestones or complete each piece. To do so, run **`submit`** as normal.)

To submit your design file (which is due Friday, April 13th, 2018 by 8:59:59 PM), use the command:

```
linux1[4]% submit cs201 PROJ2_DESIGN design2.txt
Submitting design2.txt...OK
linux1[5]%
```

To submit your project file (which is due Friday, April 20th, 2018 by 8:59:59 PM), use the command:

```
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 check that your homework was submitted by following the directions in Homework 0. Double-check that you submitted your homework correctly, since **an empty file will result in a grade of zero for this assignment.**