# PROG 1700 PYTHON - Assignment 5

# FILE IO and TWO-DIMENSIONAL LISTS

Assignment Value: 8% of overall course mark.

Due Dates: **Program 1 - Marked by Code Review - In class on Tuesday, Nov 28th**

**Programs 2 & 3: Submit to D2L, due by 4:30 PM on Friday, Dec 1st**

Late submissions will receive the standard late submission penalty as stated in the course outline. (5% overall deduction per day late, and 0% after assignment handed back to the class.)

#### Assignment Instructions:

Use PyCharm to create console applications (.py files) in which you’ll code the answer for each of the following problems. You must create a new .py file for each question in this assignment.

#### Submissions:

Once you’ve completed all required programs for the assignment, add all .py and supporting files to a single .ZIP file. The .ZIP file name should include your name, W#, section # and Assignment 5 (ex. DoeJohn\_w0123456\_702\_Assignment5.zip).

Upload the .ZIP file to the D2L dropbox labelled **Assignment 5**, which can be found under the Assessments🡪Dropboxes link.

#### Evaluation:

To insure the greatest chance of success on this assignment, be sure to check the marking rubrics at the end of this document or in D2L. The rubrics contain the criteria your instructor will be assessing when marking your assignment.

## Program 1 – A man named Jed

## Design and write a program that reads the text from a provided text file, displays the text on-screen, makes some alterations to the text and outputs the changed text to the screen, then saves the altered text as a new file.

Begin by designing your solution to this problem in pseudocode, which will be submitted along with the program. Your solution should demonstrate an understanding of how to apply file I/O, list and looping concepts. Your program will read all the text contained in a file (provided) and output the unchanged text content to the console. Your program should then make the following alterations to the original text:

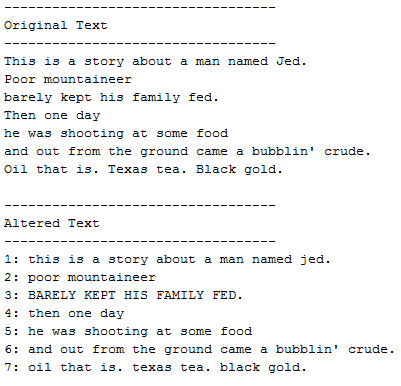
* Add a line number (starting with line 1) to the beginning of each line of text in the file.
* Your program should randomly select a line in the text and convert it to all capital letters.
* Every other line in the text (besides the all-caps version mentioned above) should be converted to all lowercase letters.

Once all text alterations are complete, output the altered text to the console, and finish by saving it as a new text file.

Every time the program is run it should pick a different random line of text and you can assume the file doesn’t contain any commas. Although a text file is provided, your finished program should work with any text-based file, not just the Jed text.

### Examples & Testing

This program has no user inputs. When run, it should produce output similar to the screenshot shown below, as well as save the altered text into a new text file. Subsequent runs of the program should capitalize different random lines of text, and overwrite the current contents of the new file.



## Program 2 – Quiz

## Design and develop a program that presents the user with a multiple choice quiz, built from questions and answers read from a specifically-formatted file.

Begin by designing your solution to this problem in pseudocode, which will be submitted along with the program. Your solution should demonstrate an understanding of how to apply file I/O, list and looping concepts. Your program should be capable of producing a quiz from any text file that uses the following format for each set of question & answers:

***<question text>,<answer A text>,<answer B text>,<answer C text>,<answer D text>,<correct answer indicator>***

In other words, each line of the quiz file will contain 6 elements: the first is the question text, the 2nd through 5th contain the answer text for the four possible multiple choice answers, and the last element contains a letter or number indicating which of the potential answers is the correct one. The answer text in the quiz file should not contain the letters/numbers used to identify each answer to the user when the quiz is being taken.

**Examples of quiz file format:**

* What is 2 \* 2?,4,2,8,6,A
* What is capital of Canada?,Toronto,Ottawa,Pennsylvania,The Moon,B

When the quiz program is run, the user will be posed each question in turn, be presented with a lettered ranking of all four possible answers, and then be prompted to enter their choice of answer. Only valid answers (a,b,c,d or 1,2,3,4) should be accepted as valid answers before proceeding to the next question. If an invalid answer choice is entered, a message indicating an invalid choice should be displayed, and the user should be prompted to enter a new choice.

When the program has displayed all questions contained in the file, and received an answer for each, a final score will be calculated and displayed to the user, including both fractional and percentage values. Ex. **Your score is 4/5 (80%).**

### Examples & Testing

In the section below you will be presented with at least one screenshot of a successful execution of a sample solution to the program, which should help demonstrate how your input/output on the program should work. You can expect your instructor to grade your assignment by using all of these listed input values, but additional values may also be used. In other words, **you should thoroughly test your code before submitting!**

Sample OutputS

|  |  |
| --- | --- |
| **Sample output for a three-question quiz** | **Sample output for a five-question quiz** |

## Program 3 – Battleship

## Design and develop a program that replicates the functionality of the provided sample application, a simple version of the game Battleship.

Begin by designing your solution to this problem in pseudocode, which will be submitted along with the program. Your solution should demonstrate an understanding of how to apply file I/O, list and looping concepts, in a Battleship program that will work as follows:

On application start, your code will read the contents of the provided ship grid text file into a two-dimensional list in your program. This ship map will be used as the “key”, indicating the locations of the five ships used in the game. Zeros (0) indicate empty water, while ones (1) indicate part of a ship exists at that location. The ship map will remain invisible to the user during gameplay. A second map (the targeting map) will be displayed to the user each turn, and will be used to show the targeting results of the current game turn by turn. The initial display of the targeting map will be blank except for the row and column indicators (Columns A, B, C, Rows 1, 2, 3, etc.).

The user will be given 30 turns to attempt to sink all five ships. During each turn, the user will be prompted to enter a map coordinate (ex. A2, F5, B10) representing the location at which they wish to fire a missile. After each missile shot attempt, your program will evaluate whether the chosen coordinate is a hit or a miss and notify the user of the result. The targeting map will be updated to show the latest missile result and be shown to the user. A message indicating the current missile count will also be displayed, used to tell the player how many turns remain.

Only valid targeting coordinates are allowed to be entered. If an invalid coordinate value is entered, the user will be prompted to re-enter a new coordinate until a valid coordinate is entered.

The game has two ending conditions:

* If the user hits every individual location in the map that contains part of a ship before running out of missiles, they win the game.
* If the user runs out of missiles before hitting every part of every ship, they lose the game.

Your program should track the game’s progress and display either a “You win!” or “You lose!” message when either game ending condition is reached.

**Bonus Marks:** Add functionality to the game such that the user is notified when a particular ship has been definitively sunk. You’ll likely need to modify the contents of the map so that it stores more than just ones and zeros in order to identify each ship individually.

Refer to the completed sample application for guidance. Your completed program should duplicate the sample game functionality as closely as possible.

### Examples & Testing

In the section below you will be presented with at least one screenshot of a successful execution of a sample solution to the program, which should help demonstrate how your input/output on the program should work. You can expect your instructor to grade your assignment by running your program multiple times and checking for every element of standard gameplay. In other words, **you should thoroughly test your code before submitting!**

Sample Gameplay

|  |  |
| --- | --- |
| **Beginning of a new game, prior to any missile shots:** | |
| **After an unsuccessful missile shot (a Miss):** | **After a successful missile shot (a Hit):** |
| **Player wins the game:** | **Player loses the game:** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Program 1 – A man named Jed** | |  |  |  |  |
| **Criteria** | **Unsatisfactory (0 pts)** | **Partially Correct (1 pt)** | **Excellent (2 pts)** | **Marks** | **X** |
| **Pseudocode** | Little to no effort was made to plan out the program using pseudocode | A reasonable but incomplete effort was made to plan out the program using pseudocode | A comprehensive effort was made to plan out the program using pseudocode |  |  |
| **File IO - Read** | Program does not read in file contents | Program reads the file but does not load the lines into a list | Program reads the file in correctly and stores the text in a list |  |  |
|  |  |  |  |  | 2 |
| **File IO - Write** | Program does not produce second text file with altered contents. | Program attempts to create second file with contents but an error exists. | Program creates second file as output. |  | 2 |
| **Random Uppercase** | Not implemented or contains too many errors. | A line number is randomly selected but errors exist or is not outputted to the second file | A line number is randomly selected and outputted correctly in the output file |  |  |
| **Lowercase** | Not implemented or contains too many errors. | All lines are forced to lowercase but not outputted to the second file | All lines are forced to lowercase and are outputted to the second file (except for the randomly selected line) |  |  |
| **Line numbers** | Not implemented or contains too many errors. | Each line in the output file is preceded by a line number but errors or inaccuracies exist | All numbers greater than the average are displayed correctly and as expected. |  |  |
| **Code Efficiency** | Code demonstrates little to no understanding of applicable concepts in developing an efficient solution. | Code demonstrates a reasonable, but incomplete, understanding of applicable concepts in developing an efficient solution. | Code demonstrates a strong understanding of applicable concepts in developing an efficient solution. |  |  |
| **Comments & Best Coding Practices**  (At least 60% of the functional requirements must be complete) | Little to no organizational or explanatory comments used  No apparent naming convention was followed or was inconsistently applied  Source code was poorly formatted | Some organizational or explanatory comments are used, some are meaningful and easily understood  A naming convention was used for part of the program, but deviated often  Effort was made to format the code, but improvements could be made | Organizational or explanatory comments are used extensively, most are meaningful and easily understood  A consistent naming convention was used for most of the program and deviated very little  Source code was clean, consistently well-formatted and easy to read |  | 2 |
|  |  |  | **Total:** |  | **/22** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Program 2 – Quiz** | |  |  |  |  |
| **Criteria** | **Unsatisfactory (0 pts)** | **Partially Correct (1 pt)** | **Excellent (2 pts)** | **Marks** | **X** |
| **Pseudocode** | Little to no effort was made to plan out the program using pseudocode | A reasonable but incomplete effort was made to plan out the program using pseudocode | A comprehensive effort was made to plan out the program using pseudocode |  |  |
| **File IO** | Not implemented or contains too many errors. | Implemented, but contains at least one error, or accomplished using inappropriate methods. | Program reads the file and stores all quiz questions and answers properly in a 2-d list. Program accommodates quiz files with any number of questions/answers, provided the file format is followed. |  |  |
| **Questions** | Not implemented or contains too many errors. | Implemented, but contains at least one error, or accomplished using inappropriate methods. | The question portion for every question read from the list is correctly identified and displayed as expected in an interactive quiz program.  All questions are displayed using a readable, user-friendly format. |  |  |
| **Answers** | Not implemented or contains too many errors. | Implemented, but contains at least one error, or accomplished using inappropriate methods. | The answer portion for every question read from the list is correctly identified and displayed as expected in an interactive quiz program.  Answer letters (a,b,c, etc.) are dynamically assigned in code, not statically added to the quiz file.  All answers are displayed using a readable, user-friendly format. |  |  |
| **User Choices** | Not implemented or contains too many errors. | Implemented, but contains at least one error, or accomplished using inappropriate methods. | User is asked to enter a choice of answer as expected, using a descriptive, user-friendly prompt.  Only valid letters (a, b, c, d) can be accepted as choices. Letters of both cases are accepted. |  |  |
| **Score Calculation** | Not implemented or contains too many errors. | Implemented, but contains at least one error, or accomplished using inappropriate methods. | Each input answer is correctly determined to match the designated correct answer in the list.  Final score displayed as both fraction and percentage.  Score properly calculated using correct answers from all expected questions. |  | 2 |
| **Code Efficiency** | Code demonstrates little to no understanding of applicable concepts in developing an efficient solution. | Code demonstrates a reasonable, but incomplete, understanding of applicable concepts in developing an efficient solution. | Code demonstrates a strong understanding of applicable concepts in developing an efficient solution. |  |  |
| **Comments & Best Coding Practices**  (At least 60% of the functional requirements must be complete) | Little to no organizational or explanatory comments used  No apparent naming convention was followed or was inconsistently applied  Source code was poorly formatted | Some organizational or explanatory comments are used, some are meaningful and easily understood  A naming convention was used for part of the program, but deviated often  Effort was made to format the code, but improvements could be made | Organizational or explanatory comments are used extensively, most are meaningful and easily understood  A consistent naming convention was used for most of the program and deviated very little  Source code was clean, consistently well-formatted and easy to read |  | 2 |
|  |  |  | **Total:** |  | **/20** |
| **Program 3 - Battleship** | | |  |  |  |
| **Criteria** | **Unsatisfactory (0 pts)** | **Partially Correct (1 pt)** | **Excellent (2 pts)** | **Marks** | **X** |
| **Pseudocode** | Little to no effort was made to plan out the program using pseudocode | A reasonable but incomplete effort was made to plan out the program using pseudocode | A comprehensive effort was made to plan out the program using pseudocode |  | 2 |
| **Map File Reading** | Not implemented or contains too many errors. | Implemented, but contains at least one error, or accomplished using inappropriate methods. | Provided map file is unedited and is read into a two-dimensional list as expected. |  |  |
| **Target Input** | Not implemented or contains too many errors. | Implemented, but contains at least one error, or accomplished using inappropriate methods. | Targeting coordinates entered by user using two separate prompts for rows and columns, one using letters A-J (either case accepted), the other using numbers 1-10. |  |  |
| **Input Validation** | Not implemented or contains too many errors. | Implemented, but contains at least one error, or accomplished using inappropriate methods. | No invalid targeting coordinates accepted by the program. Appropriate error message displayed to user. Missile count not increased with invalid inputs. |  |  |
| **Board Display** | Not implemented or contains too many errors. | Implemented, but contains at least one error, or accomplished using inappropriate methods. | Game board and missile result is displayed.  Game board is accurately updated after each missile firing. |  |  |
| **Missile Count** | Not implemented or contains too many errors. | Implemented, but contains at least one error, or accomplished using inappropriate methods. | Game accurately keeps track and displays missile count as expected. |  |  |
| **Game End Conditions** | Not implemented or contains too many errors. | Implemented, but contains at least one error, or accomplished using inappropriate methods. | Game end conditions exist and are applied. User is notified if they won (by sinking all ships) or if they lost (by running out of missiles). |  |  |
| **Code Efficiency** | Code demonstrates little to no understanding of applicable concepts in developing an efficient solution. | Code demonstrates a reasonable, but incomplete, understanding of applicable concepts in developing an efficient solution. | Code demonstrates a strong understanding of applicable concepts in developing an efficient solution. |  |  |
| **Comments & Best Coding Practices**  (At least 60% of the functional requirements must be complete) | Little to no organizational or explanatory comments used  No apparent naming convention was followed or was inconsistently applied  Source code was poorly formatted | Some organizational or explanatory comments are used, some are meaningful and easily understood  A naming convention was used for part of the program, but deviated often  Effort was made to format the code, but improvements could be made | Organizational or explanatory comments are used extensively, most are meaningful and easily understood  A consistent naming convention was used for most of the program and deviated very little  Source code was clean, consistently well-formatted and easy to read |  | 2 |
|  |  |  | **Total:** |  | **/22** |

Assignment 5 Total: \_\_\_\_/64