# PROG 1700 PYTHON - Assignment 4

# File I/O & 2D Lists

Assignment Value: 10% of overall course mark.

Due Date: **See due date designated on the assignment dropbox on Brightspace.**

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:

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.

*Resources:*

Each of the assignment programs requires the use of provided support files, which can be downloaded from Brightspace, as a file named **PROG\_Assignment4\_Resources.zip**.

#### Submissions:

You will submit your work for this assignment via GitHub. While you will have frequent commits/pushes of your assignment code to GitHub as your work on it, the instructor needs to know which version to mark and when it was committed. So, when you have completed all assignment work, put a “Ready for Marking” comment on the last code committed into GitHub. Then, submit a simple text document to the Brightspace Dropbox that contains the git commit number/hash string (e.g. “b180b37”) that identifies that commit. It is this Dropbox submission that will be used to determine late penalties, so make sure to do so prior to the assignment deadline.

**Once you have committed the code, make sure to visit the repository page on GitHub’s website to verify that the final version has been pushed to GitHub as that is where the instructor will go to get your code.**

#### Evaluation:

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

## Program 1 – The A-Team

Design and write a program that reads the text from a provided text file into a list, 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) into a list and output the unchanged text content to the console. Your program should then make the following alterations:

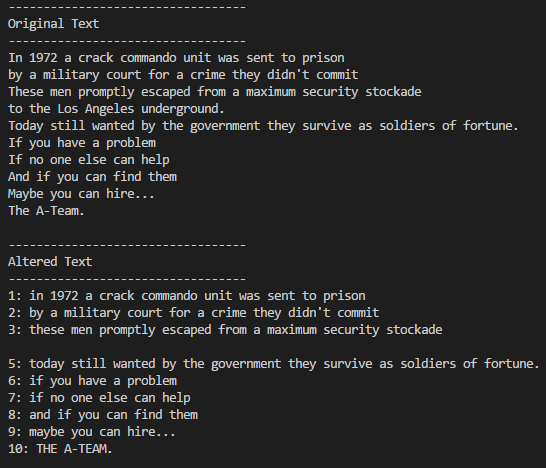
* Add a line number (starting with line number 1) to the beginning of each line of text in the file.
* Any line of text that is longer than twenty characters should be converted to all lowercase letters.
* Any line 20 or less characters long should be converted to all uppercase letters.
* Your program should randomly select a line in the text and OMIT it from any output.

Once all text alterations are complete, output the altered text to the console, and finish by saving the altered text to 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 original A-Team text.

Your solution must contain examples demonstrating your understanding of appropriate use of functions and core assignment concepts (file I/O, 2-d lists).

### 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.

Sample Output, with line 4 randomly selected  


## Program 2 – The Itsy Bitsy Aardvark

## Design and develop a program that presents the user with a “Mad Libs” type game, where a random choice of words are read from a file, then interjected into a story read from another file.

Your solution should demonstrate an understanding of how to apply file I/O, string manipulation, list and looping concepts.

Two files are provided:

* **the\_story\_file.txt** – Text file that contains an incomplete story, broken over four lines of text.
* **the\_choices\_file.csv** – A CSV file containing different series of word types. Each line has a particular format, where the descriptor is the first item, followed by 5 word choices of a certain type (ie. Noun or adjective).
* The format of each line in this file is the following:

***<phrase describing the word type for this line>,<word choice A>,<word choice B>,<word choice C>,<word choice D>,<word choice E>***

In other words, each line of the choices file will contain 6 elements: the first is a phrase describing the word type, which will be displayed in the prompt, the 2nd through 6th contain the different word choices. The word choice text in the Choices file should not contain the letters/numbers used to identify each answer to the user when the program is running.

**Examples of word choices file format:**

* an adjective,hot,cold,big,small,angry
* a verb ending in ‘ing’,running,walking,bouncing,throwing,eating

When the program is run, the user will be asked to choose a word of each type for each line of the choice file in turn. The first element in each line will be used in the on-screen prompt, while the five choices will be listed as a multiple choice list, preceded by either letters or numbers. Only valid answers (a,b,c,d or 1,2,3,4) should be accepted as valid answers before proceeding to the next word choice. 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 prompted the user to choose one word from each line, the program will read the “story” from the second file and replace the numbered placeholders (ex. \_1\_, \_2\_, etc.) with an all-uppercase version of the user-selected word choices. The completed story will be printed on screen.

Note that a single line of text from the “story” file may contain multiple placeholders.

Your solution must contain examples demonstrating your understanding of appropriate use of functions and core assignment concepts (file I/O, 2-d lists).

### 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 while choosing words:** | **Sample output of incorrect choices attempts:** |
| **First sample output of a completed story:** | |
| **Second sample output of a completed story:** | |

## 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.

Your solution must contain examples demonstrating your understanding of appropriate use of functions and core assignment concepts (file I/O, 2-d lists).

**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 provided 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 – The A-Team** | |  |  |  |  |  |
| **Criteria** | **Insufficient (0 pts)** | **Needs Development**  **(1-2 pts)** | **Sufficient (3-4 pts)** | **Excellent (5 pts)** | **Mark** | **X** |
| **Pseudocode** | Little to no effort was made, or contains too many errors / omissions. | A reasonable effort was made, but there are multiple areas for improvement. | A good effort was made, but at least one error or omission exists. | A comprehensive effort was made to plan out the program using pseudocode. |  |  |
|  |
| **File IO - Read** | Little to no effort was made, or contains too many errors / omissions. | A reasonable effort was made, but there are multiple areas for improvement. | A good effort was made, but at least one error or omission exists. | Program reads the file in correctly and stores the text in a list |  | 2 |
| **File IO - Write** | Little to no effort was made, or contains too many errors / omissions. | A reasonable effort was made, but there are multiple areas for improvement. | A good effort was made, but at least one error or omission exists. | Program creates second file as output. |  | 2 |
| **Uppercasing** | Little to no effort was made, or contains too many errors / omissions. | A reasonable effort was made, but there are multiple areas for improvement. | A good effort was made, but at least one error or omission exists. | Any line of text shorter than the specified number of characters is forced to uppercase and outputted correctly, both on-screen and to the output file. No use of magic numbers for 20 char line limit. |  |  |
| **Lowercasing** | Little to no effort was made, or contains too many errors / omissions. | A reasonable effort was made, but there are multiple areas for improvement. | A good effort was made, but at least one error or omission exists. | Any line of text longer than the specified number of characters is forced to lowercase and outputted correctly, both on-screen and to the output file. No use of magic numbers for 20 char line limit. |  |  |
| **Line numbers** | Little to no effort was made, or contains too many errors / omissions. | A reasonable effort was made, but there are multiple areas for improvement. | A good effort was made, but at least one error or omission exists. | Line numbers are added to each line, displayed correctly and as expected. |  |  |
| **Random Omission** | Little to no effort was made, or contains too many errors / omissions. | A reasonable effort was made, but there are multiple areas for improvement. | A good effort was made, but at least one error or omission exists. | A random line is selected each time the program is run. The selected line is omitted from being output to the screen or file. |  |  |
| **Use of Functions & Core Concepts** | Little to no effort was made, or contains too many errors / omissions. | A reasonable effort was made, but there are multiple areas for improvement. | A good effort was made, but at least one error or omission exists. | Solution contains at least one example demonstrating strong understanding of appropriate function use, including use of parameters and return values, and with no global variable use. (Using or adapting the standard main() function does not count) and file I/O and 2-d list concepts. |  |  |
| **Comments & Best Coding Practices**  (At least 60% of the functional requirements must be complete) | Little to no effort was made, or contains too many errors / omissions. | A reasonable effort was made, but there are multiple areas for improvement. | A good effort was made, but at least one error or omission exists. | 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 |  |  |
|  |  |  |  | **Total:** |  | **/55** |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Program 2 – Itsy Bitsy Aardvark** | | |  |  |  |  |
| **Criteria** | **Insufficient (0 pts)** | **Needs Development**  **(1-2 pts)** | **Sufficient (3-4 pts)** | **Excellent (5 pts)** | **Mark** | **X** |
| **File I/O** | Little to no effort was made, or contains too many errors / omissions. | A reasonable effort was made, but there are multiple areas for improvement. | A good effort was made, but at least one error or omission exists. | Program reads both provided files correctly. Contents of the word choices file are stored properly in a 2-d list. Contents of the story file are read into a 1-d list. File format is followed correctly. |  |  |
|  | 2 |
| **Word Choice Prompts** | Little to no effort was made, or contains too many errors / omissions. | A reasonable effort was made, but there are multiple areas for improvement. | A good effort was made, but at least one error or omission exists. | Program correctly uses the first element of each line in the on-screen prompt, to tell the user which word type is currently being used. Formatting of prompt is correct. |  |  |
| **Choice Enumerations** | Little to no effort was made, or contains too many errors / omissions. | A reasonable effort was made, but there are multiple areas for improvement. | A good effort was made, but at least one error or omission exists. | Each word choice option pulled from the choices file is preceded by a letter or number as expected. Letters or numbers are NOT in data file. |  |  |
| **Data Validation** | Little to no effort was made, or contains too many errors / omissions. | A reasonable effort was made, but there are multiple areas for improvement. | A good effort was made, but at least one error or omission exists. | Program does not accept any invalid entries, and correctly re-prompts the user when an invalid entry is attempted. If appropriate, letter casing should be handled correctly. |  |  |
| **Story Output** | Little to no effort was made, or contains too many errors / omissions. | A reasonable effort was made, but there are multiple areas for improvement. | A good effort was made, but at least one error or omission exists. | At minimum, generic story text, containing placeholders, is properly output to the screen, and is well-formatted. |  |  |
| **Word Interjection** | Little to no effort was made, or contains too many errors / omissions. | A reasonable effort was made, but there are multiple areas for improvement. | A good effort was made, but at least one error or omission exists. | All text placeholders are replaced with the correct user-selected words. All interjected words are capitalized as required. |  | 2 |
| **Use of Functions & Core Concepts** | Little to no effort was made, or contains too many errors / omissions. | A reasonable effort was made, but there are multiple areas for improvement. | A good effort was made, but at least one error or omission exists. | Solution contains at least one example demonstrating strong understanding of appropriate function use, including use of parameters and return values, and with no global variable use. (Using or adapting the standard main() function does not count) and file I/O and 2-d list concepts. |  |  |
| **Comments & Best Coding Practices**  **(At least 60% of the functional requirements must be complete)** | Little to no effort was made, or contains too many errors / omissions. | A reasonable effort was made, but there are multiple areas for improvement. | A good effort was made, but at least one error or omission exists. | 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 |  |  |
|  |  |  |  | **Total:** |  | **/50** |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Program 3 – Battleship** | |  |  |  |  |  |
| **Criteria** | **Insufficient (0 pts)** | **Needs Development**  **(1-2 pts)** | **Sufficient (3-4 pts)** | **Excellent (5 pts)** | **Mark** | **X** |
| **Pseudocode** | Little to no effort was made, or contains too many errors / omissions. | A reasonable effort was made, but there are multiple areas for improvement. | A good effort was made, but at least one error or omission exists. | A comprehensive effort was made to plan out the program using pseudocode |  | 2 |
| **Map File Reading** | Little to no effort was made, or contains too many errors / omissions. | A reasonable effort was made, but there are multiple areas for improvement. | A good effort was made, but at least one error or omission exists. | Provided map file is unedited and is read into a two-dimensional list as expected. |  |  |
|  |  |
| **Target Input** | Little to no effort was made, or contains too many errors / omissions. | A reasonable effort was made, but there are multiple areas for improvement. | A good effort was made, but at least one error or omission exists. | 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** | Little to no effort was made, or contains too many errors / omissions. | A reasonable effort was made, but there are multiple areas for improvement. | A good effort was made, but at least one error or omission exists. | No invalid targeting coordinates accepted by the program. Appropriate error message displayed to user. Missile count not increased with invalid inputs. |  |  |
| **Board Display** | Little to no effort was made, or contains too many errors / omissions. | A reasonable effort was made, but there are multiple areas for improvement. | A good effort was made, but at least one error or omission exists. | Game board and missile result is displayed.  Game board is accurately updated after each missile firing. |  |  |
| **Missile Count** | Little to no effort was made, or contains too many errors / omissions. | A reasonable effort was made, but there are multiple areas for improvement. | A good effort was made, but at least one error or omission exists. | Game accurately keeps track and displays missile count as expected. |  |  |
| **Game End Conditions** | Little to no effort was made, or contains too many errors / omissions. | A reasonable effort was made, but there are multiple areas for improvement. | A good effort was made, but at least one error or omission exists. | 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). |  |  |
| **Use of Functions & Core Concepts** | Little to no effort was made, or contains too many errors / omissions. | A reasonable effort was made, but there are multiple areas for improvement. | A good effort was made, but at least one error or omission exists. | Solution contains at least one example demonstrating strong understanding of appropriate function use, including use of parameters and return values, and with no global variable use. (Using or adapting the standard main() function does not count) and file I/O and 2-d list concepts. |  |  |
| **Comments & Best Coding Practices**  (At least 60% of the functional requirements must be complete) | Little to no effort was made, or contains too many errors / omissions. | A reasonable effort was made, but there are multiple areas for improvement. | A good effort was made, but at least one error or omission exists. | 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 |  |  |
|  |  |  |  | **Total:** | **/50** |