CP1401/CP5639 Assignment 2

Task - Market Garden Simulator:

For this assessment, you are to plan and then code a medium-sized console-based program in Python 3. This assignment is designed to help you build skills using:

- As in assignment 1: Input, Processing and Output; Decision structures; Repetition structures
- New in assignment 2: Functions and random numbers; Lists; Files

The assignment also includes a **developer's journal** – a document you complete as you plan and code your program to help you focus on and improve your process and become a better developer.

Incredibly Important: This assignment must not be where you first learn about these topics. The subject is designed to systematically teach you these principles through the lectures (first), then the practicals. You should have practised each programming concept/construct many times before you start using it in your assignment. If you get to a point in your assignment where you're not sure about something, go back and learn from the subject teaching (not on the Internet). Remember:
100% of what you need to know to complete this assignment is taught in the subject.

Problem Description:

The *Market Garden Simulator* is a program that simulates the tranquil and refreshing activity of growing your own garden for fun and profit. You have a list of plants, which each generate "food" according to their name length (as everyone knows, longer plant names mean higher profit at market… but they cost more to buy). Each day when you wait and watch, it rains a random amount. This rainfall determines how much food the plants generate, but if you don't get enough rain, a random plant will die. When you have enough food, you can buy new plants. To increase the biodiversity of your garden, you can't buy plants you already have.

The program starts with a welcome, some instructions and initial plants. You can choose to either load existing plants from a text file or start with four standard plants (see output). Notice the form of the prompt for this and saving plants at the end: “Y/n” means that Yes is the default. Anything other than N or n should be interpreted as a Yes.

Then there's a menu with the following four options (read the sample output for more detail):

- (W)ait

  This simulates a day starting with rainfall between 0 and 128 mm (think about constants). If you get less than 32 mm (did someone say think about constants?) then a random plant from your list will die (and be deleted from the list). Each plant generates an amount of food according to the formula (same percentage for each plant per day):

  ```
  percent = random value between 1/3 rainfall and actual rainfall / 128
  food produced = percent * length of plant
  ```

  e.g., if rainfall is 90, then generate a random value between 30 and 90; let’s say it’s 42. This divided by maximum (128) is 0.328. This number would then multiply the length of each plant, so "Sage" plant (4 characters) would produce a result of 1.312 (0.328 * 4), as an integer so 1, and "Thai Basil" (10 characters) would produce 3.

- (D)isplay plants

  This simply displays the plants in your garden.

- (A)dd new plant

  You can only add plants you can afford. You can have an infinite number of plants.
  New plant names cannot be blank; error-check these (see expectations below).
  Plant names should be converted to title case (using Python's `.title()` string method), so if the user enters "thai BAsIL", it will become "Thai Basil".

If you already have the plant in your list, then you will be asked for the name again.
You do not need to handle plant names that aren't plants. Any non-empty name is OK.
When you add a plant, its name length is deducted from your total food.

- (Q)uit

  This will end the main menu and show the final details including the plants, the number of days simulated, the number of plants and the amount of food.
  You will then be prompted to save the plants, one per line, to a file.

Make sure you understand how the program should work (that's the "analysis" step in program development) before you plan it ("design" step), then code it ("implementation"). Don't forget to test your program thoroughly, comparing it to these requirements.

Coding Requirements and Expectations:

1. Make use of named **constants** as appropriate, e.g., for things that would otherwise be "magic numbers", like the maximum rainfall or low rainfall threshold for plant death. Remember the guidelines for constants: https://github.com/CP1404/Starter/wiki/Programming-Patterns#constants
   a. A very good way to test that you have used constants properly is that you should be able to change ONE value in ONE place to make the low rain threshold 20 mm… and the instructions should correctly show this. That's what constants are for.
2. You are expected to include two kinds of useful **comments** in each of your program:
   a. *Every* function should have a `"""docstring"""`.
   b. Use `# block` comments for things that might reasonably need a comment.
   c. Do not include unnecessary or many comments as these are just "noise" and make your program harder to read. See the subject teaching for how to properly write good comments: https://github.com/CP1404/Starter/wiki/Styles-and-Conventions#commenting
3. **Error checking** should follow our standard pattern:
   https://github.com/CP1404/Starter/wiki/Programming-Patterns#error-checking
   You may also notice that we have written useful functions for getting valid inputs before, e.g.,
   https://github.com/CP1401/Practicals/tree/master/prac_06#example
   So… follow what you've been taught and feel confident that you're on the right track! ☺
4. Follow what you've been taught about how to write **menus**, and know that (Q) should not be a separate option within the menu: https://github.com/CP1404/Starter/wiki/Programming-Patterns#menus
5. **Functions** should be used for sections of the program and repeated tasks as you have been taught. Follow the DRY (Don't Repeat Yourself) principle and consider turning repeated code into functions. **Do not use global variables.** Any use of global variables will result in a fail mark for the functions category. Here are some possibilities for functions:
   a. displaying the plants is done the same way in multiple places
   b. adding a plant is a significant section
   c. getting a plant name looks very similar to the kind of thing we wrote functions for in the teaching (getting a valid string)
   d. simulating a day is a nice-sized section for its own function
   e. the main menu and one-off program behaviour (like the start and end) should all be part of the main function – again, like our examples and teaching
6. **Sample output** from the program is provided. Follow this to meet requirements, but you *are allowed* to be a creative and customise the interface as you wish. So, please understand – you can change small details that don't break the requirements, but if you change it substantially you may miss some of the required aspects and lose marks. E.g., you could display the plants differently, or use different output text when a plant dies, but you could not add or remove a menu option and you couldn't choose to not have plants die. Please ask if you are unsure.
   a. The sample output shows "1 plants". This is fine, but you are welcome to add the logic to make this "1 plant".
   b. There's a comma at the end of the plants display. This is also fine. You don't need to change it – but you can if you want to.

We strongly suggest you work incrementally on this task: focus on completing small parts rather than trying to get everything working at once. This is called "iterative development" and is a common way of working, even for professionals. A suggested approach to this is described below:

1. Start with planning and pseudocode – this is important for your process as well as your journal (see below for details about recording your process in your journal).
2. Start coding with the main function and creating a working menu using the standard pattern. For each menu item, just print something (like "… add plant…").
3. Choose one function/section at a time to implement. E.g., start with the function to display plants and get this working, then call the function from your main menu.
4. When you do a more complex section, keep it simple first, then add complexity. E.g., when adding a plant, ignore the error-checking to start with. Get it working properly, then add error-checking.
5. When writing the function to simulate a day, start with just generating and displaying random rainfall, then add the calculation to determine plant food… but notice the random number or percentage are never printed… so print these as helpful debugging outputs until your function is working correctly, then remove the print statements.
6. When writing and testing code with randomness, you can encourage it towards what you want to test by modifying your constants or starting values. E.g., when you want to test what happens when it doesn't rain much, change the maximum rainfall to a low number temporarily (that's how we created the 2nd sample output). Want to test what happens when you run out of plants? Change the starting list to one plant instead of four. Don't waste time running your program many many many times to hopefully get the random scenario you want to test.
7. The file saving and loading can be done last as it is optional for the user. Start with the default plants until the program works, then add the file loading.

Journal:

A significant desired outcome for this subject is you learning to develop solutions to problems systematically and thoughtfully. That is, we are not only interested in the final product but in your *process* and the *lessons* you have learned through your experience. To encourage you in learning the systematic problem-solving process, you will record your work experiences and insights in a simple journal for this assignment – submitted as a PDF file.

There are three parts to this journal:

1. Assignment 1 Reflections and Lessons
2. Work Entries
3. Summary

Assignment 1 Reflections and Lessons:

Before you begin working on this assignment, take some time to reflect on what you learned about your process through assignment 1, including how you will make changes this time based on any feedback you received. You could consider your previous reflection exercise from the practicals: https://github.com/CP1401/Practicals/tree/master/prac_07#reflection

Work Entries:

Each time you work on the assignment, record an entry in your journal that includes:

- Date and time you worked, including duration
- What you worked on with simple details, enough that someone reading it would understand
- Any difficulties you faced and how you overcame them

Please do not include multiple entries for tiny work sessions. If you did 7 minutes, took a break then came back and did 37.5 minutes… we don't need this level of detail – just include a single entry of about 45 minutes. These entries can be quite short, but make sure your focus is on your ***process***, not your actual code/work.

Summary:

Summarise the lessons you learned about the problem-solving process (not about Python code) through doing this assignment. **This is the most important part** – where you **reflect** on your process and show what you have learned. Did anything you considered or changed based on assignment 1 reflection prove important or useful? How are you improving in terms of following a systematic development process? What would you differently next time?

Please note that the only reasonable way to write a journal is regularly, ***as*** you develop your solution. A journal that is completed at the end of the assignment after you've finished everything is not a journal and will not aid your learning experience as much.

Here is a sample journal entry that shows a 'satisfactory' level:

> **20/04/2022, 8:30 – 9:30am**
>
> **Work**: Pseudocode for main function; nearly completed start and menu section.
>
> **Challenges**: Took a few goes to remember how to deal with lists and functions in pseudocode (not Python). Checked the "Pseudocode Guide" and followed the examples, which helped.

Sample Output (see also the video demonstration):

It should be clear which parts below are user input (not printed, but entered by the user).

```
Welcome to my garden.
Plants cost and generate food according to their name length (e.g., Sage plants cost 4).
You can buy new plants with the food your garden generates.
You get up to 128 mm of rain per day. Not all plants can survive with less than 32.
Enjoy :)
Would you like to load your plants from plants.txt (Y/n)? n
You start with these plants:
Parsley, Sage, Rosemary, Thyme,

After 0 days, you have 4 plants and your total food is 0.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: d
Parsley, Sage, Rosemary, Thyme,
After 0 days, you have 4 plants and your total food is 0.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: p
Invalid choice
After 0 days, you have 4 plants and your total food is 0.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: a
Enter plant name:
Invalid plant name
Enter plant name: Sage
You already have a Sage plant.
Enter plant name: OK
Ok would cost 2 food. With only 0, you can't afford it.
After 0 days, you have 4 plants and your total food is 0.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: w
Rainfall: 62mm
Parsley produced 1, Sage produced 0, Rosemary produced 1, Thyme produced 1,
```

```
After 1 days, you have 4 plants and your total food is 3.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: W
Rainfall: 66mm
Parsley produced 3, Sage produced 2, Rosemary produced 4, Thyme produced 2,
After 2 days, you have 4 plants and your total food is 14.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: a
Enter plant name: BroccoLINI
After 2 days, you have 5 plants and your total food is 4.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: w
Rainfall: 76mm
Parsley produced 2, Sage produced 1, Rosemary produced 2, Thyme produced 1, Broccolini produced 3,
After 3 days, you have 5 plants and your total food is 13.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: a
Enter plant name: broccoli
After 3 days, you have 6 plants and your total food is 5.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: d
Parsley, Sage, Rosemary, Thyme, Broccolini, Broccoli,
After 3 days, you have 6 plants and your total food is 5.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: w
Rainfall: 96mm
Parsley produced 1, Sage produced 1, Rosemary produced 2, Thyme produced 1, Broccolini produced 2, Broccoli
produced 2,
After 4 days, you have 6 plants and your total food is 14.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: w
Rainfall: 63mm
Parsley produced 1, Sage produced 0, Rosemary produced 1, Thyme produced 1, Broccolini produced 2, Broccoli
produced 1,
After 5 days, you have 6 plants and your total food is 20.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: a
Enter plant name: Now I have lots
After 5 days, you have 7 plants and your total food is 5.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: w
Rainfall: 69mm
Parsley produced 2, Sage produced 1, Rosemary produced 2, Thyme produced 1, Broccolini produced 3, Broccoli
produced 2, Now I Have Lots produced 4,
After 6 days, you have 7 plants and your total food is 20.
```

```
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: d
Parsley, Sage, Rosemary, Thyme, Broccolini, Broccoli, Now I Have Lots,
After 6 days, you have 7 plants and your total food is 20.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: q
You finished with these plants:
Parsley, Sage, Rosemary, Thyme, Broccolini, Broccoli, Now I Have Lots,
After 6 days, you have 7 plants and your total food is 20.
Would you like to save your plants to plants.txt (Y/n)? Oh, yes please :)
Saved.
Thank you for simulating. Now enjoy some real plants.
```

Now here is a second run where there's not a lot of rain. This shows you what should happen when you have no plants.

```
Welcome to my garden.
Plants cost and generate food according to their name length (e.g., Sage plants cost 4).
You can buy new plants with the food your garden generates.
You get up to 128 mm of rain per day. Not all plants can survive with less than 32.
Enjoy :)
Would you like to load your plants from plants.txt (Y/n)? y
Loaded.
You start with these plants:
Parsley, Sage, Rosemary, Thyme, Broccolini, Broccoli, Now I Have Lots,

After 0 days, you have 7 plants and your total food is 0.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: d
Parsley, Sage, Rosemary, Thyme, Broccolini, Broccoli, Now I Have Lots,
After 0 days, you have 7 plants and your total food is 0.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: w
Rainfall: 36mm
Parsley produced 1, Sage produced 0, Rosemary produced 1, Thyme produced 1, Broccolini produced 2, Broccoli
produced 1, Now I Have Lots produced 3,
After 1 days, you have 7 plants and your total food is 9.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: W
Rainfall: 1mm
Sadly, your Sage plant has died.
Parsley produced 0, Rosemary produced 0, Thyme produced 0, Broccolini produced 0, Broccoli produced 0, Now I
Have Lots produced 0,
After 2 days, you have 6 plants and your total food is 9.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: w
Rainfall: 10mm
Sadly, your Parsley plant has died.
Rosemary produced 0, Thyme produced 0, Broccolini produced 0, Broccoli produced 0, Now I Have Lots produced
0,
After 3 days, you have 5 plants and your total food is 9.
(W)ait
```

```
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: w
Rainfall: 30mm
Sadly, your Rosemary plant has died.
Thyme produced 1, Broccolini produced 2, Broccoli produced 1, Now I Have Lots produced 3,
After 4 days, you have 4 plants and your total food is 16.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: W
Rainfall: 23mm
Sadly, your Broccolini plant has died.
Thyme produced 0, Broccoli produced 0, Now I Have Lots produced 1,
After 5 days, you have 3 plants and your total food is 17.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: a
Enter plant name: I think this is a dry season...
I Think This Is A Dry Season... would cost 31 food. With only 17, you can't afford it.
After 5 days, you have 3 plants and your total food is 17.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: w
Rainfall: 3mm
Sadly, your Broccoli plant has died.
Thyme produced 0, Now I Have Lots produced 0,
After 6 days, you have 2 plants and your total food is 17.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: W
Rainfall: 14mm
Sadly, your Now I Have Lots plant has died.
Thyme produced 0,
After 7 days, you have 1 plants and your total food is 17.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: w
Rainfall: 43mm
Thyme produced 1,
After 8 days, you have 1 plants and your total food is 18.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: W
Rainfall: 40mm
Thyme produced 1,
After 9 days, you have 1 plants and your total food is 19.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: w
Rainfall: 13mm
Sadly, your Thyme plant has died.

After 10 days, you have 0 plants and your total food is 19.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
```

```
Choose: d

After 10 days, you have 0 plants and your total food is 19.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: w
Rainfall: 21mm
After 11 days, you have 0 plants and your total food is 19.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: W
Rainfall: 12mm
After 12 days, you have 0 plants and your total food is 19.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: a
Enter plant name: sage
After 12 days, you have 1 plants and your total food is 15.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: w
Rainfall: 27mm
Sadly, your Sage plant has died.

After 13 days, you have 0 plants and your total food is 15.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: q
You finished with no plants
After 13 days, you have 0 plants and your total food is 15.
Would you like to save your plants to plants.txt (Y/n)? No, please don't save. I'd lose all my plants!
Saved.
Thank you for simulating. Now enjoy some real plants.
```

The contents of the file, plants.txt, will simply be the plant names, one per line, e.g.,

```
Parsley
Sage
Rosemary
Thyme
```

Submission:

Write your pseudocode and code in a single Python file called **`a2_garden.py`**.

Note: You are only required to write **pseudocode** for your main function and the function for simulating a day. You are welcome and encouraged to do it for the whole program, but we will only mark these two functions' pseudocode. However, your main function must be substantial and appropriately designed (e.g., do not use a "menu" function). Remember, "main should look like the whole program", with the details in the functions:
https://github.com/CP1404/Starter/wiki/Programming-Patterns#main-program-structure

Copy and follow the structure provided below, that is, starting with a module docstring at the very top containing your own details and your pseudocode, then your solution code below.

```
"""
CP1401 2022-1 Assignment 2
Market Garden Simulator
Student Name: XX
Date started: XX

Pseudocode:


"""
```

Your journal must be saved as a PDF file called **`a2_journal.pdf`**. Follow the example provided above and include an appropriate heading and your name.

DO NOT zip/compress your files together. Submit two separate files (.py and .pdf) as this makes it considerably easier for markers to access your work (and it's easier for you!).

Submit your single Python file and single PDF file, by uploading them on LearnJCU under Assessments as instructed by the date and time specified on LearnJCU. Submissions received after this date will incur late penalties as described in the subject outline.

Note that the assignment allows multiple submissions, but we will only look at and assess the most recent submission. If you need to resubmit, then submit your entire assignment again.

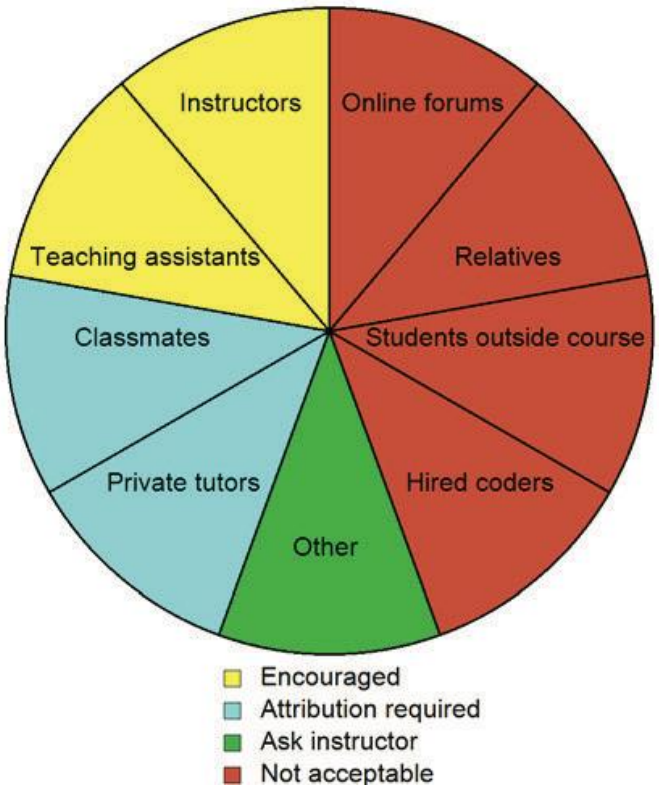
Integrity:

The work you submit for this assignment must be your own. Submissions that are detected to be too similar to that of another student or other work (e.g., code found online) will be dealt with according to the College procedures for handling plagiarism and may result in serious penalties.

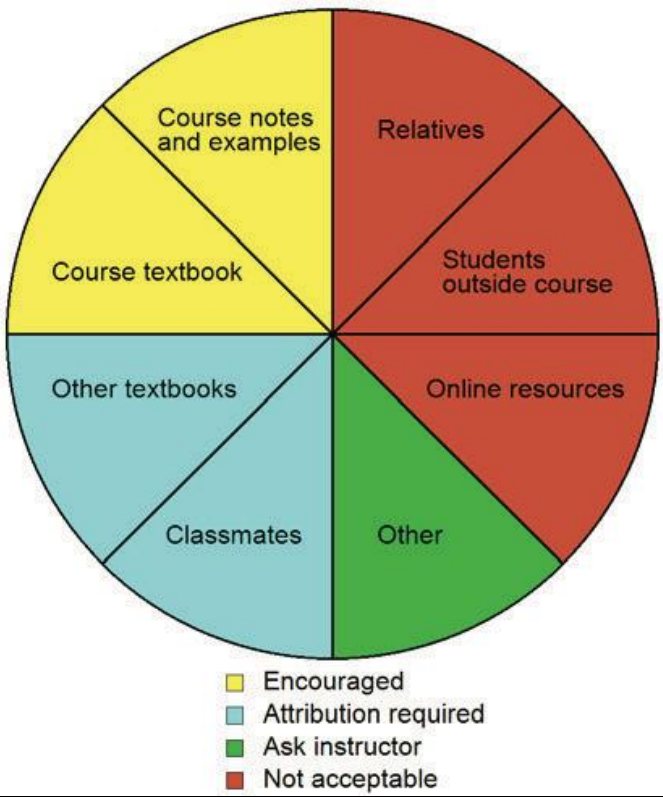
The goals of this assignment include helping you gain understanding of fundamental programming concepts and skills, and future subjects will build on this learning. Therefore, it is important that you develop these skills to a high level by completing the work and gaining the understanding yourself. You may discuss the assignment with other students and get assistance from your peers, but you may not do any part of anyone else's work for them and you may not get anyone else to do any part of your work. Note that this means **you should never give a copy of your work to anyone or accept a copy of anyone else's work, including looking at another student's work or having a classmate look at your work**. If you require assistance with the assignment, please ask **general** questions in #cp1401 in Slack, or get **specific** assistance with your own work by talking with your lecturer or tutor.

The subject materials (lecture notes, practicals, textbook and other guides provided in the subject) contain all of the information you need for this particular assignment. You should not use online resources (e.g., Stack Overflow or other forums) to find resources or assistance as this would limit your learning and would mean that you would not achieve the goals of the assignment - mastering fundamental programming concepts and skills.

Assistance: Who can you get help from? Use this diagram to determine from whom you may seek help with your programs.



Resources: Where can you get code from? Use this diagram to determine where you may find code to use in your programs.



Marking Scheme:

Ensure that you follow the processes and guidelines taught in the subject to produce high quality work. Do not just focus on getting your code working. This assessment rubric provides you with the characteristics of exemplary to very limited work in relation to task criteria, covering the outcomes:

- SLO1 - apply problem-solving techniques to develop algorithms in the IT context
- SLO2 - apply basic programming concepts to develop solutions

Criteria	Exemplary (9, 10)	Good (7, 8)	Satisfactory (5, 6)	Limited (2, 3, 4)	Very Limited (0)
Journal SLO1 20%	Good first reflection showing learning; significant number of entries showing good problem-solving process including starting with planning; summary highlights useful lessons.	Exhibits aspects of exemplary (left) and satisfactory (right)	Some first reflection; reasonable number of entries but not enough; process is not exemplary (e.g., planning or testing are missing or not in appropriate order); summary lacks insight and clear lessons.	Exhibits aspects of satisfactory (left) and very limited (right)	No journal or trivial effort.
Algorithms SLO1 10%	Clear, well-formatted, consistent and accurate pseudocode that completely and correctly solves the problem.		Some but not many problems with algorithm (e.g., incomplete solution, inconsistent use of terms, inaccurate formatting).		Many problems or algorithm not done.
Correctness SLO2 20%	Program works correctly for all functionality required.		Program mostly works correctly for most functionality, but some required aspects are missing or have problems.		Program works incorrectly for all functionality required.
Identifier naming SLO2 10%	All variable, constant and function names are appropriate, meaningful and consistent.		Multiple variable, constant or function names are not appropriate, meaningful or consistent.		Many names are not appropriate, meaningful or consistent.
Use of code constructs SLO1, SLO2 15%	Appropriate and efficient code use following the standard patterns, including good choices for variables, constants, processing, decision and repetition.		Mostly good code use but with problems, e.g., not following patterns, unnecessary or repeated code (DRY), missing constants, poor choice of decision or repetition.		Many significant problems with code use.
Use of functions SLO2 15%	Functions and parameters are appropriately used, functions are well reused to avoid code duplication.		Functions used but not well, e.g., breaching SRP, poor use of parameters, unnecessary duplication, main code outside main function.		No functions used or functions used very poorly. **Any use of global variables will result in < 5.**
Commenting SLO2 5%	Every function has a docstring, some helpful block/inline comments, module docstring contains all details, no 'noise' comments.		Some noise (too many/unhelpful comments) or missing some function docstrings or incomplete module docstring.		Commenting is very poor either through having too many comments (noise) or too few comments.
Formatting SLO2 5%	All formatting meets PEP8 standard, including indentation and spacing. PyCharm shows no formatting warnings.		Multiple problems with formatting reduce readability of code. PyCharm shows formatting warnings.		Readability is poor. PyCharm shows many formatting warnings.