CP1401/CP5639 2021 SP1 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

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 spend some to 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 four plants. Then there's a repeating menu with the following four options (read the sample output to understand in more detail):

- (W)ait
    - This simulates a day starting with rainfall between 0 and 100mm (think about constants). If you get less than 30mm (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:

      `(random value between 1/2 rainfall and actual rainfall) / 100 * length`

      e.g., if rainfall is 70, then a random value between 0.35 and 0.7 would multiply the length of each plant, so "Sage" plant (4 characters) would produce a result between (0.35 and 0.7 * 4) 1.4 and 2.8 as an integer so 1 to 2, and "Thai Basil" (10 characters) would produce between 3 and 7.
- (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 and repeat for blank names.
      Notice that we have explicitly taught how to handle errors like this using the error checking 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! ☺
      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.
      When you add a plant, the name length is deducted from your 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.
      Notice that you have been taught how to write menus and you know that (Q) should not be a separate option within the menu, but rather the quit loop condition with final actions coded outside the main menu loop: https://github.com/CP1404/Starter/wiki/Programming-Patterns#menus

The sample output below will help you understand these details. There is also a video demonstration available. 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 Suggestions:

- 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: if you use a value more than once, it should probably be a constant, and if you have a constant then you must use it in all places that you reference that value.
  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.
- You are expected to include two kinds of useful comments in each of your program:
    - Every function should have a `"""docstring"""`. See the subject teaching for how to properly write docstrings.
    - Use `# block` comments for things that might reasonably need a comment.

  Do not include unnecessary or many comments as these are just "noise" and make your program harder to read.
- 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. Here are some possibilities for functions:
    - displaying the plants is done the same way in multiple places
    - adding a plant is a significant section
    - getting a plant name looks very similar to the kind of thing we wrote functions for in the teaching (getting a valid string)
    - simulating a day is a nice-sized section for its own function
    - 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
- You do not need to handle plant names that aren't plants. It's fine to have a "1401 Rocks" or "Is Monty Fun?" plant.
- Sample output from the program is provided. You should ensure that your program mostly matches this and definitely does in terms of meeting the requirements. But you *are allowed* to be a bit 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.
- The sample output shows "1 plants". This is fine, but you are welcome to add the logic to make this "1 plant".
  Also, there's a comma at the end of the plants display. This is also fine and 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 pattern you've been taught. 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 output until your function is working correctly, then remove the print statement.
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 from 100 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.

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.

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.

Include a final "**Summary**" section at the end of your journal that summarises 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.

Please note that the only reasonable way to write a journal is *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:

> **13/10/2020, 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). Notice that the menu handles uppercase and lowercase letters.

```
Welcome to the Market Garden Simulator
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 100 mm of rain per day. Not all plants can survive with less than 30.
Let's hope it rains... a lot!
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: u
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: 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: a
Enter plant name: Fern
Fern would cost 4 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: 90mm
Parsley produced 3, Sage produced 1, Rosemary produced 3, Thyme produced 2,
After 1 days, you have 4 plants and your total food is 9.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: W
Rainfall: 74mm
Parsley produced 3, Sage produced 1, Rosemary produced 3, Thyme produced 2,
After 2 days, you have 4 plants and your total food is 18.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: w
Rainfall: 38mm
Parsley produced 2, Sage produced 1, Rosemary produced 2, Thyme produced 1,
After 3 days, you have 4 plants and your total food is 24.
(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: SWEET potato
After 3 days, you have 5 plants and your total food is 12.
```

```
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: w
Rainfall: 8mm
Sadly, your Sweet Potato plant has died.
Parsley produced 0, Sage produced 0, Rosemary produced 0, Thyme produced 0,
After 4 days, you have 4 plants and your total food is 12.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: a
Enter plant name: SWEET potato
After 4 days, you have 5 plants and your total food is 0.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: w
Rainfall: 58mm
Parsley produced 4, Sage produced 2, Rosemary produced 4, Thyme produced 2, Sweet Potato produced 6,
After 5 days, you have 5 plants and your total food is 18.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: w
Rainfall: 3mm
Sadly, your Rosemary plant has died.
Parsley produced 0, Sage produced 0, Thyme produced 0, Sweet Potato produced 0,
After 6 days, you have 4 plants and your total food is 18.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: a
Enter plant name: I wonder if I could have a kauri tree
I Wonder If I Could Have A Kari Tree would cost 36 food. With only 18, you can't afford it.
After 6 days, you have 4 plants and your total food is 18.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: a
Enter plant name: Kauri Tree?
After 6 days, you have 5 plants and your total food is 7.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: d
Parsley, Sage, Thyme, Sweet Potato, Kauri Tree?,
After 6 days, you have 5 plants and your total food is 7.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: w
Rainfall: 18mm
Sadly, your Sweet Potato plant has died.
Parsley produced 0, Sage produced 0, Thyme produced 0, Kauri Tree? produced 1,
After 7 days, you have 4 plants and your total food is 8.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: W
Rainfall: 83mm
Parsley produced 4, Sage produced 2, Thyme produced 3, Kauri Tree? produced 6,
After 8 days, you have 4 plants and your total food is 23.
```

```
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: d
Parsley, Sage, Thyme, Kauri Tree?,
After 8 days, you have 4 plants and your total food is 23.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: q
You finished with these plants:
Parsley, Sage, Thyme, Kauri Tree?,
After 8 days, you have 4 plants and your total food is 23.
Thank you for simulating. Now go and enjoy a real garden.
```

Here is the output from a second run using a different low threshold.
Notice the different ("no plants") output at the end:

```
Welcome to the Market Garden Simulator
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 100 mm of rain per day. Not all plants can survive with less than 83.
Let's hope it rains... a lot!
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: w
Rainfall: 79mm
Sadly, your Sage plant has died.
Parsley produced 3, Rosemary produced 4, Thyme produced 2,
After 1 days, you have 3 plants and your total food is 9.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: w
Rainfall: 78mm
Sadly, your Rosemary plant has died.
Parsley produced 4, Thyme produced 3,
After 2 days, you have 2 plants and your total food is 16.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: w
Rainfall: 22mm
Sadly, your Thyme plant has died.
Parsley produced 1,
After 3 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: 40mm
Sadly, your Parsley plant has died.

After 4 days, you have 0 plants and your total food is 17.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
```

```
Choose: d

After 4 days, you have 0 plants and your total food is 17.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: a
Enter plant name:
Invalid plant name
Enter plant name: Why is there so much sadness in my garden?
Why Is There So Much Sadness In My Garden? would cost 42 food. With only 17, you can't afford it.
After 4 days, you have 0 plants and your total food is 17.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: a
Enter plant name: Ficus elastica
After 4 days, you have 1 plants and your total food is 3.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: a
Enter plant name: Fig
After 4 days, you have 2 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: P
P would cost 1 food. With only 0, you can't afford it.
After 4 days, you have 2 plants and your total food is 0.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: w
Rainfall: 44mm
Sadly, your Fig plant has died.
Ficus Elastica produced 5,
After 5 days, you have 1 plants and your total food is 5.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: w
Rainfall: 64mm
Sadly, your Ficus Elastica plant has died.

After 6 days, you have 0 plants and your total food is 5.
(W)ait
(D)isplay plants
(A)dd new plant
(Q)uit
Choose: q
You finished with no plants
After 6 days, you have 0 plants and your total food is 5.
Thank you for simulating. Now go and enjoy a real garden.
```

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) to score well. 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 2021-1 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. Please 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.

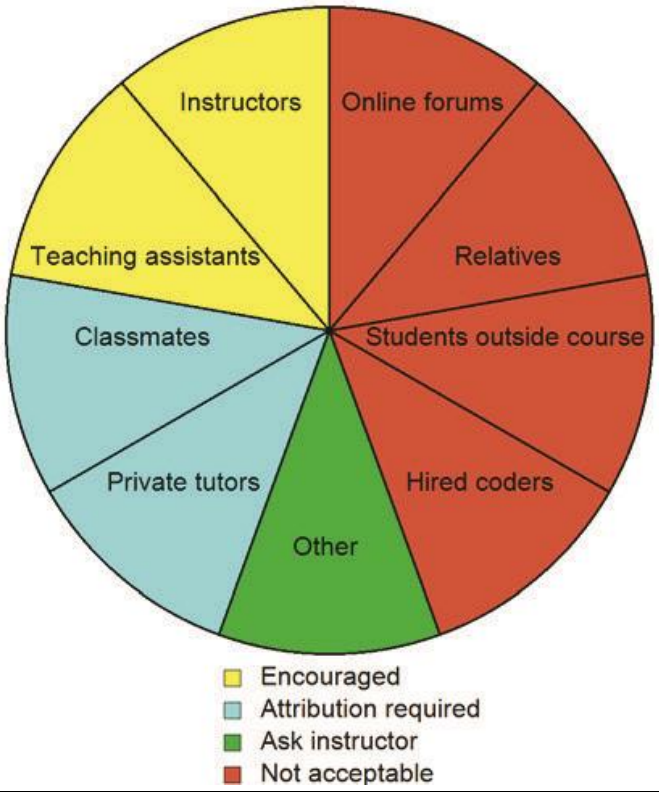
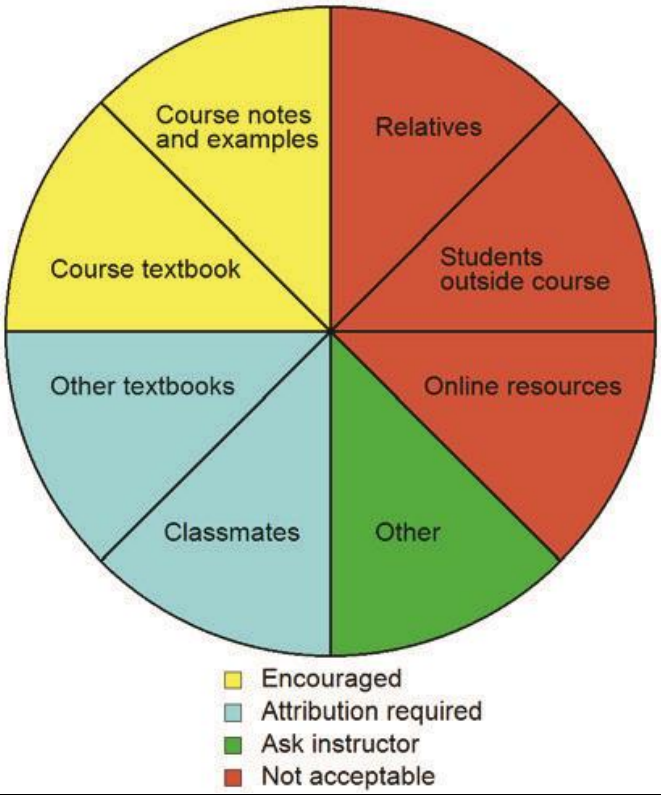
Note that the assignment allows multiple submissions, but we will only look at and assess the most recent submission. If you need to resubmit, then submit your entire assignment again.

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 class in order to food 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.

Criteria	**Exemplary (9, 10)**	**Good (7, 8)**	**Satisfactory (5, 6)**	**Limited (2, 3, 4)**	**Very Limited (0)**
Journal 15%	Significant number of entries showing good problem-solving process including starting with planning; well-written entries with appropriate detail; summary highlights useful lessons.	Exhibits aspects of exemplary (left) and satisfactory (right)	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 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 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 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 variable, constant, function names are not appropriate, meaningful or consistent.
Use of code constructs 15%	Appropriate and efficient code use, including good choices for variables, constants, processing, decision and repetition.		Mostly appropriate code use but with problems, e.g. unnecessary/repeated code, missing constants, poor choice of decision or repetition.		Many significant problems with code use.
Use of functions 15%	Functions and parameters are appropriately used, functions are well reused to avoid code duplication.		Functions used but not well, e.g. breaching SRP, incorrect use of parameters, unnecessary duplication, global variables or main code outside main function.		No functions used or functions used very poorly.
Commenting 10%	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 5%	All formatting meets PEP8 standard, including indentation, horizontal spacing and consistent vertical line spacing. PyCharm shows no formatting warnings.		Multiple problems with formatting reduce readability of code. PyCharm shows formatting warnings.		Readability is poor due to formatting problems. PyCharm shows many formatting warnings.