CP1401/CP5639 Assignment 2

Task – Animal Accumulator:

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 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:

(Note: the sample output below should help explain and demonstrate all of the requirements.)

The *Animal Accumulator* is a program that simulates a collection of animals. You have a list of animals, which each generate "income" according to their name length (as everyone knows, longer animal names mean higher income… but they cost more to buy). Each day when you wait, a random "luck" value between 0-100 is generated, which determines how much income the animals generate, but if you are unlucky, a random animal will escape. You can buy new animals with the income you make, but you cannot have more than one of each animal.

The program starts with a welcome, some instructions and the option to load from a file of animals or start with three animals. Then there's a repeating menu with the following four options:

- (W)ait
  - This simulates a day starting with luck between 0 and 100. If you get less than 42 (think about constants) then a random animal from your list will escape and be deleted from the list. Animals are deleted before any income calculations.
    Each animal generates an amount of income according to the formula:

    ```
    (integer) luck / 100 * name length
    ```

    e.g., if luck is 70, a "Zebra" animal (5 characters) would generate 0.7 * 5 = 3.5 as an integer, so an income of 3.
- (D)isplay animals
  - This simply displays the animals (or “No animals.” if you have none).
- (A)dd new animal
  - You can only add animals you can afford. You can have infinite animals but you cannot have any with the same name as existing animals. New animal names cannot be empty, so error-check and repeat for empty names.
    When you input an animal, the name should be converted to title case (using Python's `.title()` string method), so if the user enters "HEARTy bass", it will become "Hearty Bass" (and would cost 11 income).
    When you add an animal, the name length (cost) is deducted from your total income.
- (Q)uit
  - This will end the main menu (follow the pattern!) and show the final details including the animals, the number of days simulated, the number of animals and the amount of income. The user will be prompted to save the animals to the text file "animals.txt". If they enter "y" then the program should save each animal to the file, one name per line.

Animals in the list should always be sorted in alphabetical order. Your program should only sort when needed – when the list has changed.

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 low luck threshold. Remember the guidelines for constants: https://github.com/CP1404/Starter/wiki/Programming-Patterns#constants
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 explanation.

    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/Style-Guide#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 the standard pattern for **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 failing mark for the functions category. Here are some possibilities for functions:
    a. displaying the animals is done the same way in multiple places
    b. adding an animal is a significant section
    c. getting an animal 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. saving and loading seem like "one job" each
    f. 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 below. Follow this to meet 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 use different output text when an animal escapes, but you could not add or remove a menu option and you could not choose to have animals never escape. Please ask if you are unsure.
    a. The sample output shows "1 animals" and "1 days". This is fine, but you are welcome to add the logic to make this grammatically correct.
    b. There's a comma at the end of the animals 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. First, begin your journal, as below.
2. Start your development 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).
3. Start coding with the main function and creating a working menu using the standard pattern. For each menu item, just print something (like "… add animal…").
4. Choose one function or section at a time to implement. E.g., start with the function to display animals and get this working, then call the function from your main menu.
5. When you do a more complex section, keep it simple first, then add complexity. E.g., when adding an animal, ignore the error-checking to start with. Get it working properly, then add error-checking.
6. When writing the function to simulate a day, start with just generating and displaying random luck, then add the calculation to determine animal incomes.
7. When writing and testing code with randomness, you can encourage your program towards what you want to test by modifying your constants or starting values. E.g., when you want to test what happens when luck is consistently low, change the maximum luck from 100 to a low number like 41 temporarily (that's how we created the 2nd sample output). Want to test what happens when you run out of animals? Change the starting list to one animal instead of three. Don't waste time running your program many many many times to hopefully get the random scenario you want to test.
8. File loading saving can be done last as it is optional for the user.

Journal:

A significant desired outcome for you in this subject is 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 to be systematic in your 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. Reflections and Lessons on Assignment 1
2. Work Entries
3. Summary

Reflections and Lessons on Assignment 1:

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

> **20/04/2023, 8:30 – 9:30am**
>
> **Work**: Wrote 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. What an awesome collection of resources we've been provided in this subject. The teaching is outstanding ;)

Summary:

Summarise the lessons you learned about the problem-solving ***process*** (not about Python code) through doing this assignment. **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 do 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 much.

Sample Output:

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. Notice what happens in various situations and don't make up different requirements not specified or shown here.

```
Welcome to the Animal Accumulator.
Animals cost and generate income according to their name length (e.g., a Zebra costs 5).
Each day, animals generate income based on luck. Sometimes they escape.
You can buy new animals with the income your animals generates.
Would you like to load your animals from animals.txt (Y/n)? n
You start with these animals:
Antelope, Fox, Zebra,

After 0 days, you have 3 animals and your total income is 0.
(W)ait
(D)isplay animals
(A)dd new animal
(Q)uit
Choose: a choice
Invalid choice.
After 0 days, you have 3 animals and your total income is 0.
(W)ait
(D)isplay animals
(A)dd new animal
(Q)uit
Choose: a
Animal name: fox
You already have Fox.
Animal name: x
You can't afford X.
After 0 days, you have 3 animals and your total income is 0.
(W)ait
(D)isplay animals
(A)dd new animal
(Q)uit
Choose: w
Today's lucky number is 58.
Antelope earned 4, Fox earned 1, Zebra earned 2,
After 1 days, you have 3 animals and your total income is 7.
(W)ait
(D)isplay animals
(A)dd new animal
(Q)uit
Choose: A
Animal name:
Invalid animal name.
Animal name: aarDVArk
You can't afford Aardvark.
After 1 days, you have 3 animals and your total income is 7.
(W)ait
(D)isplay animals
(A)dd new animal
(Q)uit
Choose: a
Animal name: hippo
Added Hippo.
```

```
After 1 days, you have 4 animals and your total income is 2.
(W)ait
(D)isplay animals
(A)dd new animal
(Q)uit
Choose: w
Today's lucky number is 72.
Antelope earned 5, Fox earned 2, Hippo earned 3, Zebra earned 3,
After 2 days, you have 4 animals and your total income is 15.
(W)ait
(D)isplay animals
(A)dd new animal
(Q)uit
Choose: d
Antelope, Fox, Hippo, Zebra,
After 2 days, you have 4 animals and your total income is 15.
(W)ait
(D)isplay animals
(A)dd new animal
(Q)uit
Choose: w
Today's lucky number is 41.
Sadly, your Hippo has escaped.
Antelope earned 3, Fox earned 1, Zebra earned 2,
After 3 days, you have 3 animals and your total income is 21.
(W)ait
(D)isplay animals
(A)dd new animal
(Q)uit
Choose: w
Today's lucky number is 82.
Antelope earned 6, Fox earned 2, Zebra earned 4,
After 4 days, you have 3 animals and your total income is 33.
(W)ait
(D)isplay animals
(A)dd new animal
(Q)uit
Choose: w
Today's lucky number is 11.
Sadly, your Antelope has escaped.
Fox earned 0, Zebra earned 0,
After 5 days, you have 2 animals and your total income is 33.
(W)ait
(D)isplay animals
(A)dd new animal
(Q)uit
Choose: a
Animal name: a pet for mrs arbuckle
Added A Pet For Mrs Arbuckle.
After 5 days, you have 3 animals and your total income is 11.
(W)ait
(D)isplay animals
(A)dd new animal
(Q)uit
Choose: w
Today's lucky number is 14.
Sadly, your Zebra has escaped.
A Pet For Mrs Arbuckle earned 3, Fox earned 0,
After 6 days, you have 2 animals and your total income is 14.
(W)ait
(D)isplay animals
(A)dd new animal
(Q)uit
Choose: w
Today's lucky number is 10.
Sadly, your Fox has escaped.
A Pet For Mrs Arbuckle earned 2,
After 7 days, you have 1 animals and your total income is 16.
(W)ait
(D)isplay animals
(A)dd new animal
(Q)uit
Choose: w
Today's lucky number is 79.
A Pet For Mrs Arbuckle earned 17,
After 8 days, you have 1 animals and your total income is 33.
(W)ait
(D)isplay animals
(A)dd new animal
(Q)uit
Choose: d
```

```
A Pet For Mrs Arbuckle,
After 8 days, you have 1 animals and your total income is 33.
(W)ait
(D)isplay animals
(A)dd new animal
(Q)uit
Choose: w
Today's lucky number is 42.
A Pet For Mrs Arbuckle earned 9,
After 9 days, you have 1 animals and your total income is 42.
(W)ait
(D)isplay animals
(A)dd new animal
(Q)uit
Choose: w
Today's lucky number is 60.
A Pet For Mrs Arbuckle earned 13,
After 10 days, you have 1 animals and your total income is 55.
(W)ait
(D)isplay animals
(A)dd new animal
(Q)uit
Choose: w
Today's lucky number is 79.
A Pet For Mrs Arbuckle earned 17,
After 11 days, you have 1 animals and your total income is 72.
(W)ait
(D)isplay animals
(A)dd new animal
(Q)uit
Choose: w
Today's lucky number is 45.
A Pet For Mrs Arbuckle earned 9,
After 12 days, you have 1 animals and your total income is 81.
(W)ait
(D)isplay animals
(A)dd new animal
(Q)uit
Choose: w
Today's lucky number is 63.
A Pet For Mrs Arbuckle earned 13,
After 13 days, you have 1 animals and your total income is 94.
(W)ait
(D)isplay animals
(A)dd new animal
(Q)uit
Choose: w
Today's lucky number is 55.
A Pet For Mrs Arbuckle earned 12,
After 14 days, you have 1 animals and your total income is 106.
(W)ait
(D)isplay animals
(A)dd new animal
(Q)uit
Choose: w
Today's lucky number is 5.
Sadly, your A Pet For Mrs Arbuckle has escaped.

After 15 days, you have 0 animals and your total income is 106.
(W)ait
(D)isplay animals
(A)dd new animal
(Q)uit
Choose: w
Today's lucky number is 19.
After 16 days, you have 0 animals and your total income is 106.
(W)ait
(D)isplay animals
(A)dd new animal
(Q)uit
Choose: d
No animals.
After 16 days, you have 0 animals and your total income is 106.
(W)ait
(D)isplay animals
(A)dd new animal
(Q)uit
Choose: a
Animal name: what will I buy with all these 106 income$? :)
Added What Will I Buy With All These 106 Income$? :).
After 16 days, you have 1 animals and your total income is 60.
```

```
(W)ait
(D)isplay animals
(A)dd new animal
(Q)uit
Choose: a
Animal name: fox
Added Fox.
After 16 days, you have 2 animals and your total income is 57.
(W)ait
(D)isplay animals
(A)dd new animal
(Q)uit
Choose: a
Animal name: hippo
Added Hippo.
After 16 days, you have 3 animals and your total income is 52.
(W)ait
(D)isplay animals
(A)dd new animal
(Q)uit
Choose: d
Fox, Hippo, What Will I Buy With All These 106 Income$? :),
After 16 days, you have 3 animals and your total income is 52.
(W)ait
(D)isplay animals
(A)dd new animal
(Q)uit
Choose: q
You finished with these animals:
Fox, Hippo, What Will I Buy With All These 106 Income$? :),
After 16 days, you have 3 animals and your total income is 52.
Would you like to save your animals to animals.txt (Y/n)? OK, let's save this thing
Saved.
Thank you for playing the Animal Accumulator :)
```

More Sample Output

This shows a second run where the user chooses to load the animals from the file saved in the previous run, then not save at the end. Notice the "no animals" display after quitting.

```
Welcome to the Animal Accumulator.
Animals cost and generate income according to their name length (e.g., a Zebra costs 5).
Each day, animals generate income based on luck. Sometimes they escape.
You can buy new animals with the income your animals generates.
Would you like to load your animals from animals.txt (Y/n)? y
Loaded.
You start with these animals:
Fox, Hippo, What Will I Buy With All These 106 Income$? :),

After 0 days, you have 3 animals and your total income is 0.
(W)ait
(D)isplay animals
(A)dd new animal
(Q)uit
Choose: w
Today's lucky number is 31.
Sadly, your Fox has escaped.
Hippo earned 1, What Will I Buy With All These 106 Income$? :) earned 14,
After 1 days, you have 2 animals and your total income is 15.
(W)ait
(D)isplay animals
(A)dd new animal
(Q)uit
Choose: W
Today's lucky number is 18.
Sadly, your What Will I Buy With All These 106 Income$? :) has escaped.
Hippo earned 0,
After 2 days, you have 1 animals and your total income is 15.
(W)ait
(D)isplay animals
(A)dd new animal
(Q)uit
Choose: w
Today's lucky number is 32.
Sadly, your Hippo has escaped.

After 3 days, you have 0 animals and your total income is 15.
(W)ait
(D)isplay animals
```

```
(A)dd new animal
(Q)uit
Choose: d
No animals.
After 3 days, you have 0 animals and your total income is 15.
(W)ait
(D)isplay animals
(A)dd new animal
(Q)uit
Choose: q
You finished with no animals.
After 3 days, you have 0 animals and your total income is 15.
Would you like to save your animals to animals.txt (Y/n)? N
Thank you for playing the Animal Accumulator :)
```

Submission:

Write your pseudocode and code in a single Python file called **`a2_animals.py`**.

Follow/copy 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.

Note: You are only required to write **pseudocode** for your main function and your 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.

```
"""
CP1401 2023-1 Assignment 2
Animal Accumulator
Student Name: XX
Date started: XX

Pseudocode:


"""
```

Save your journal 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.

Attach two separate files (.py and .pdf) to your submission as this makes it considerably easier for markers to access your work (and it's easier for you!).

Do not submit **`animals.txt`**.

Submit your Python file and PDF file by uploading them on LearnJCU under Assessments 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 all files for your assignment again.

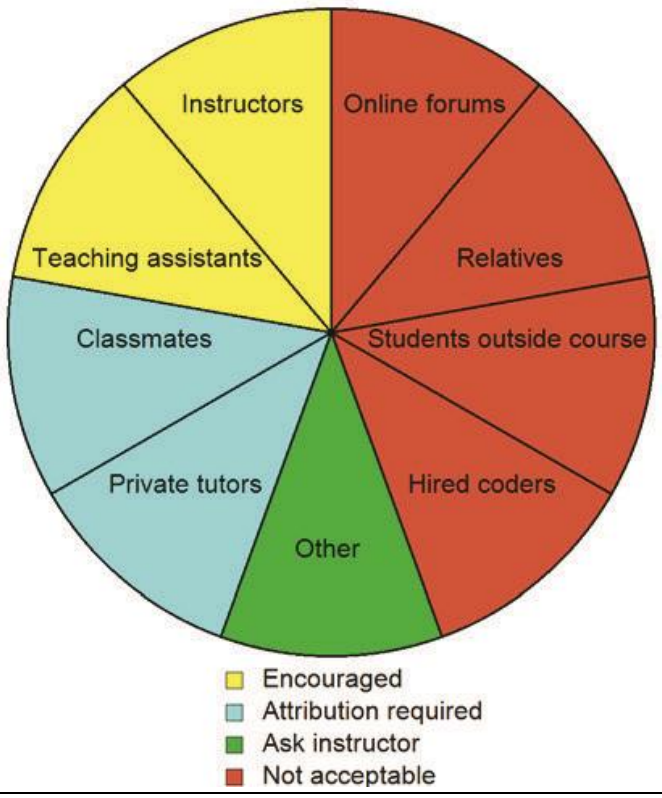
Integrity:

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.

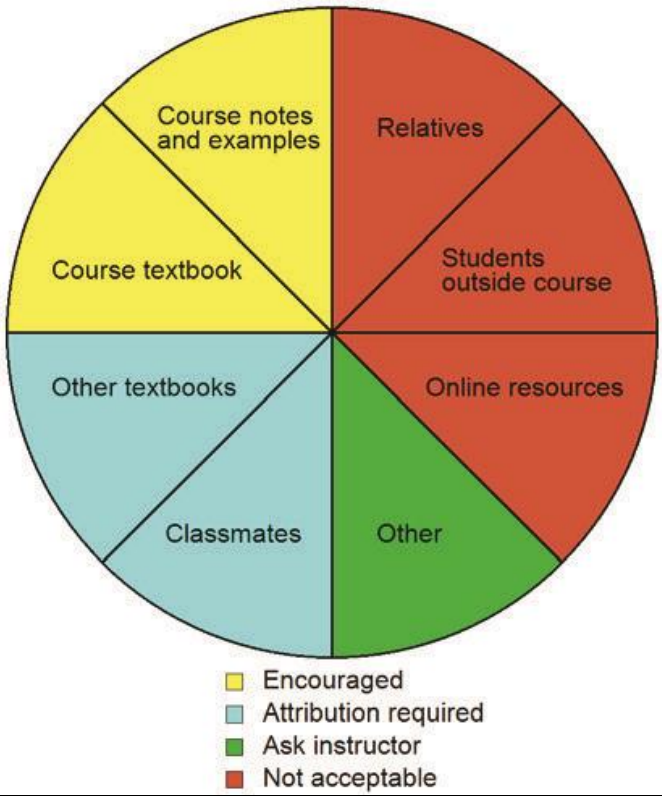
The goals of this assignment include helping you gain understanding of fundamental programming concepts and skills, and future subjects will build on this learning. Therefore, it is important that you develop these skills to a high level by completing the work and gaining the understanding yourself. 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 teaching (lectures, practicals, textbook and other guides provided in the subject) contains all the information you need for this particular assignment. You should not use online resources (e.g., Stack Overflow, ChatGPT, or other sources) 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 (1-4)	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.