COMP1811 Python Project Coursework Specification

COMP1811(2022/23)	**Paradigms of Programming**	**Contribution:** 40% of module
Module Leader/Moderator Yasmine Arafa/Andy Wicks	RESIT Practical Coursework 1: Python Project	**Deadline Date** Friday 14/07/2023
This coursework should take an average student who is up to date with tutorial/lab work approximately 40 hours. Feedback and grades are normally made available within 15 working days of the coursework deadline.		
Learning Outcomes: A. Understand the programming paradigms introduced and their applicability to practical problems. B. Apply appropriate programming constructs in each programming paradigm. C. Design, implement and test small-scale applications in each programming paradigm. D. Use appropriate tools to design, edit and debug programs for each paradigm.		

Plagiarism is presenting somebody else's work as your own. It includes copying information directly from the Web or books without referencing the material, submitting joint coursework as an individual effort, copying another student's coursework, stealing coursework from another student, and submitting it as your own work. Suspected plagiarism will be investigated and if found to have occurred will be dealt with according to the procedures set down by the University. Please see your student handbook for further details of what is/isn't plagiarism.

All material copied or amended from any source (e.g., internet, books) must be referenced correctly according to the reference style you are using. Code snippets from open-source resources or YouTube must be acknowledged appropriately.

Your work will be submitted for electronic plagiarism checking. Any attempt to bypass our plagiarism detection systems will be treated as a severe Assessment Offence.

Coursework Submission Requirements

- An **electronic copy** of your work for this coursework must be fully **uploaded by 23:30 on Friday 14/07/2023** using the **RESIT Python Project Upload link** under "RESIT Coursework Specification and Submission" on the Moodle page for COMP1811. Your work WILL NOT be accepted if you fail to submit by the deadline, unless you have an EC accepted.
- For the final upload for this coursework, you must submit **two SEPARATE files**: **a)** a **zip file containing** all the files needed to run **your Python project** in PyCharm **and** a **screencast** showing your program running and explaining the code, **and b)** a **PDF version of your report**. In general, any text in the report must not be an image (i.e., must not be scanned) and would normally be generated from other documents (e.g., MS Office using "Save As ... PDF"). A penalty will apply for the use of screenshots of code in place of code text, or if two separate files are not uploaded or named correctly. There are limits on the file size.
- **Failure to submit a screencast explaining the code and development may result in a failed assessment**.
- Make sure any files you upload are virus-free and not protected by a password or corrupted, otherwise, they will be treated as null submissions.
- All coursework must be submitted as above. Under no circumstances can they be accepted by academic staff.

The University website has details of the current Coursework Regulations, including details of penalties for late submission, procedures for Extenuating Circumstances, and penalties for Assessment Offences. See http://www2.gre.ac.uk/current-students/regs

Coursework Regulations

- If you have been granted Extenuating Circumstances (ECs), you may submit your coursework up to 10 working days after the published deadline without penalty. However, this is subject to your claim being accepted by the Faculty Extenuating Circumstances Panel.
- Late submissions will be dealt with in accordance with University Regulations.
- Coursework submitted more than two weeks late may be given feedback but will be recorded as a non-submission regardless of any extenuating circumstances. The only exception to this is where your EC claim outcome has granted you a deferral.
- **Do not ask lecturers for extensions to published deadlines - they are not authorised to award an extension**.

Please refer to the University Portal for further detail regarding the University Academic Regulations concerning Extenuating Circumstances claims.

Coursework Specification

Please read the entire coursework specification before starting work. The coursework theme is similar to the original coursework, but the required features are entirely different.

This is an individual project development coursework.
You are expected to work on your own to implement the project.

Overview

The overall task for this project is to design and develop a standalone application in Python for analysing small social networks and recommending likely new friends.

The project is an extensive application that needs to be developed and tested incrementally. The implementation requires the design, implementation, and testing of an object-oriented Python program that follows the specification below and uses custom classes (classes you have defined), appropriate data structures, and file processing. The specification below provides less overview of how to do the project as you are expected to come up with and design the solution yourself.

This project development is the culmination of all you have learned about Python and OOP and will showcase your hard work for part one of this module. The system you develop should become a strong addition to your programming portfolio. Hence, you should aim to produce a system that is well-designed, robust, and useful. The system should operate smoothly without sluggishness or crashes and should not require instructions or a manual to use.

Scenario and System Specifications

Social networks are made up of a set of members and connections (dyadic ties) between those members - for example, there exists a connection between you and everyone whom you befriend/interact with within a social network. Facebook and WhatsApp are among the many popular social networks with which you may already be a member. An interesting feature of these social media sites is they automatically recommend likely new friends (connections) for you and other members of the network. They use sophisticated algorithms to do so. However, to better understand how this feature works, your project is to simulate a social network, analyse connections between members, and implement a naive method to recommend the most likely new friend(s) to members based on the intersection of their common friends. That is the person recommended for member X as a friend is the person who has the most friends in common with member X, but who currently is not friends with member X. Your project will also analyse connections between friends and produce statistics as defined in the requirements section.

The program will need to map the connections (friendships/relations) between the network members by using appropriate data structure(s) to represent how members relate to each other. For simplicity, the only information you need about each member is their name (or member ID). The map will represent the member along with the connections to other members (the friends each member has). The input to your program is a file containing this map representation and the output described in the requirements section should be displayed on the PyCharm console. Your program should work via the console and does not need to create a graphical user interface – Graphical User Interface (GUI). There are no marks for creating a Windows-based application.

System Features and Requirements

The system must provide the following features and comply with the System Specifications outlined above. All the features below must be completed individually. All **input and output should be via the PyCharm console** and not via a GUI. The input must be validated for correctness, must prompt an error message if incorrect values are entered, and continue to ask for valid values until one is entered or "quit" is entered to stop.

Task 0 – Planning and Setup

i. **Create a plan** of how you will complete this project. The plan should show each of the features and sub-features listed below and when you expect to complete them.

ii. **Create a text file** called **`nw1`** containing a representation of a social network in the following format (which will be used as input to your program to test your code and should be read on start-up):

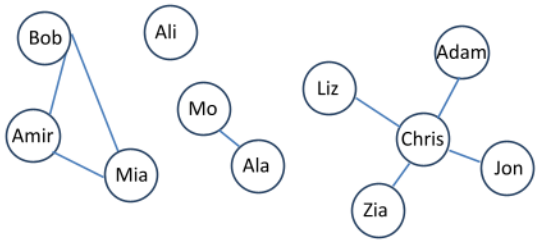
- The first line of the file is an integer representing the number of members in the social network, then the following lines are in the form of *member* and *friend* in pairs, where *member* and *friend* are represented by a member name. The member/friend pairs should be listed on separate lines in the file. A member without a friend is listed by their name only in a single line.
- There is no limit to the number of members or relations between them in the social network. The only restriction is that number of members must match that determined by the integer on the first line of the file. For simplicity, start with smaller networks and gradually test your program with larger ones.

Sample social network and the lines that should show in the `nw1` file:

```
11
Adam  Chris
Bob   Amir
Bob   Mia
Chris Liz
Jon   Chris
Chris Zia
Mia   Amir
Mo    Ala
Ali
```

The above is a representation of a social network that has 11 members with the following relations:

`Adam` is friends with `Chris`
`Amir` is friends with `Bob`, `Mia`
`Bob` is friends with `Amir`, `Mia`
`Mo` is friends with `Ala`
`Mia` is friends with `Amir`, `Bob`
`Zia` is friends with `Chris`
`Liz` is friends with `Chris`
`Jon` is friends with `Chris`
`Ali` is friends with `no one`
`Chris` is friends with `Adam, Jon, Liz, Zia`



Note: the network graph is for illustration only and you are not expected to draw it in your code.

iii. **Consider/design the classes, methods, and data structures needed** before starting to code. This coursework requires you to implement your program using classes so, start by considering suitable classes first. Think about the methods you will include in each based on the feature requirements below. Note that any program using classes can be written without classes, but there will be limited credit for such solutions!

iv. **Create additional test input files** `nw2`, `nw3`, etc. containing different social networks which you can later use to test and validate your program.

F1 – Feature 1: Social Network Data

i. **Open social network file**: (2 marks)
this sub-feature should allow the user to enter a file name and prompt an error message if the file cannot be opened. The process should continue to as ask for a valid file name until the file can be opened or the user types "`done`".

ii. **Validate the input:** (2 marks)

validate the social network data read from the file after it has been opened successfully. This feature should check that the number of members given on the first line of the file matches the number of unique names listed below. It should also indicate if there are fewer or more names in the data. If the network representation has errors, raise an exception and prompt the user with an appropriate error message.

iii. **Simulate the social network structure:** (5 marks)

- **Get social network data**:
  once the file is opened successfully, read the file (following the format described in `nw1`) and dynamically create an appropriate data structure named `nw` to hold the social network so that it can be used elsewhere in the code.
- **Display social network:**
  ask the user whether they want to display the social network read from the file. If they do, pretty print (display nicely formatted) the social network on the **PyCharm console**. Listing each member and their friends or none if they have no friends.
  An example of the output using the test input is as follows:

```
The current social network:
Adam  is friends with Chris
Amir  is friends with Bob, Mia
Bob   is friends with Amir, Mia
Mo    is friends with Ala
Mia   is friends with Amir, Bob
Zia   is friends with Chris
Liz   is friends with Chris
Jon   is friends with Chris
Ali   is friends with none
Chris is friends with Adam, Jon, Liz, Zia
Ala   is friends with Mo
```

  Note: the test network has 11 members so the output has 11 lines.

iv. **Delete a member to/from the network:** (6 marks)

deleting a member must include removing all occurrences of that member in the network.

F2 – Feature 2: Network Analysis

i. **Get the size of all friend groups within the network**: (8 marks)
the aim is to identify the size of the friend groups within the network. The diagram above shows examples of these: a group of one, a group of two, a group of three, and a group of 5 or more.

The output for the test network should be as follows:

```
The size of friend groups:
Number of groups with 1 member: 1
Number of groups with 2 members: 1
Number of groups with 3 members: 1
Number of groups with 4 members: 0
Number of groups with 5 members: 1
```

ii. **Identify the shape of friend groups**: (8 marks)

Identifying the friend groups within the network can be done by their shapes. From your analysis identify which of the following shapes are present in the test network data:

- A friend group with 1 member is a **singleton**,
- A friend group with 2 members is a **pair**,
- A friend group with 3 members is a **triangle** or a **tripod**.
- A friend group with 4 or more members where each member is friends with exactly two other members is a **polygon**.
- A friend group with 4 or more members where one member is friends with all the other members and the other members are only friends with the central members is a **star**. For example, Chris and his friends form a star group in the test network data.

The output for the test network should be:

```
Friend group network shapes:
1 singleton
1 pair
1 triangle
0 tripod
0 polygon
1 star
```

F3 – Feature 3: Social network analysis and statistics

i. **Determine the average number of friends among the network members**: (4 marks)

display the average number of friends on the console.

ii. **Show the members with the greatest number of friends**: (4 marks)

display the members with the greatest number of friends on the console.

Development

You are expected to develop a standalone (desktop) version of the application. Your implementation must demonstrate appropriate use of the Python object-oriented concepts covered and needs to define and use **suitable classes in a professional way**. Note that **any program using classes can be written without classes, but there will be no credit for such a solution**.

You are expected to define and use at least three classes and demonstrate an appropriate inheritance hierarchy for one. Make sure your classes apply appropriate abstraction and encapsulation rules. All the classes should be ones you have defined in the program and not classes from the Pythons modules and libraries. Remember to avoid code duplication in your programs. Think about how you can use classes and inheritance to reduce duplication in the code.

Remember getting your project to function as required is important, but it is only one part of this assessment. The quality of your code and the OOP design features you implement are the most important. **Most of the credit for this assessment is awarded to the quality of the code and the OOP design features implemented.**

The code produced should:

- implement all the features in a professional style,
- uses a correct naming convention for all classes, modules, methods, functions, and variables,
- considers the appropriate OOP techniques,
- and should work as expected when executed.

Please refer to PEP 8 for a guide to Pythonic style conventions.

Hints for Developing Your Project

The principal idea to approaching any problem-solving task is to first understand the problem, break down that problem into smaller chunks, devise a plan to solve them, and then simply carry out that plan in order to implement your project. You are advised to plan regular milestones (where each part must deliver and demonstrate their work) and follow up with frequent progress meetings with your lab tutor.

1. **Understanding the problem** is often neglected as being obvious, but it isn't always the case! Come up with questions to clarify the problem and discuss them to make sure you understand the problem set. For example, you might ask: what are you being asked to do? can you restate the problem in your own words? do you have enough information for you to come up with a solution? is your understanding of the task the same as the others around you? We sometimes get wonderful work that is not what is specified and therefore gets no marks.
2. **Break down the problem**: Any big problem can be decomposed into simpler ones. This has been partly done for you by decomposing the system features into the set of sub-features outlined above. Think about how each sub-feature can be decomposed even more and make a list of all of the smaller components. Look for any patterns in those smaller decompositions (for example, more than one sub-feature may require you to read or write from the text file so could you write a function or method which does this and then call it in other sub-features?). It is a good idea to come up with a diagram that can illustrate the smaller components and how they fit together.
3. **Devise a plan**: Now that you know what needs to be done, list all the decomposed parts you identified in stage 2 above and decide who is doing what of the system features and when. Plan dates when you expect to complete each of those parts and make sure you deliver on time. Include formal testing in your plan.
4. **Carry out the plan**: In programming projects, this is the actual programming and where you put your ideas into code. When you are thinking about the code, you must think about the problem you are trying to solve and think about how you'll *design* a solution. You are required to produce an object-oriented solution and so you must think about object-oriented features that you'll need. The advantage of an OO solution is that you develop reusable and robust code. So, think about what are the real-world objects that are in the given scenario and how these may be used for implementing the sub-features to create reusable code.

Remember to test your code thoroughly as you go along. You are required to demonstrate that you have tested your program fully by using a test plan in the coursework write-up.

Individual Coursework

You are expected to work on your own to develop your coursework project. However, discussing tactics and requirements with others is recommended. Do NOT copy code from each other – that is plagiarism and both parties may be heavily penalised.

Deliverables

1. **Python Code and Screencast** – A **zip file containing the entire folder for your PyCharm project** which includes the code you produced and a screencast (video). For the screencast recording create a 7-10 minute video in which you show your program running, explain what is happening, and explain the code logic (not the code syntax). You can do this using any software you like that produces an .MP4 or WAV file, e.g. Teams. Please note that failure to include a commentary will be given a zero mark.
   Name the zip file as follows: <YourStudentID>_COMP1811_RESIT_PyCodeVideo.zip. Please try to structure your work so that it is easy for the person marking the work to run your project. **You will lose marks if you do not clearly label the code for the features you developed or do not follow the zip file name specification.**

   Upload the zip file to the Moodle **Python Project Upload link** under the Coursework Specification and Submission block by **23:30 on Friday 14/07/23.**

   There will be no demos, so the screencast is mandatory.

2. **Python Project Report** – A project **report template** is provided under Python Project in the Coursework Details and Submission block. The report should consist of **all** the sections in the report template.
3. Name your report as follows: <StudentID>_ COMP1811_RESIT_PyReport.pdf. **You will lose marks if you do not follow this file name specification. You will likely fail this assessment if you do not upload a report**.

   Upload the **PDF version of your report** to the Moodle **Python Project Upload link** under the Coursework Specification and Submission block by **23:30 on Friday 14/07/23**.

If you have borrowed code or ideas from anywhere other than the lecture notes and tutorial examples (e.g., from a book, somewhere on the web, or from another student) then include a reference showing where the code or ideas came from. Comment your code very carefully to show which bits are yours and which bits are borrowed. This will protect you against accusations of plagiarism. Be aware that the marker will look for similarities between your code and that submitted by other students so please do not share your code with any other students as this is plagiarism.

Marking Scheme

This coursework will not be marked anonymously.

Marks will be awarded for the following. Note that the marks shown are out of 100. They will be scaled to be out of 40 to give you an overall mark for this assessment.

1. **Code development (70)**
   a. Feature correctness and completeness. [40]
   b. Use of OOP techniques. [20]
   c. Quality of code. [10]
2. **Documentation. (20)**
   a. Report clarity and completeness (including a clear exposition on design and decisions for OOP use. [10]
   b. Testing. [5]
   c. Objective evaluation of work. [5]
3. **Screencast. (10)**

Marking Breakdown

1. Code development (70)

Features correctness and completeness [40]

Feature 1 (up to 16)

Sub-features have not been implemented – 0
Attempted, not complete, or very buggy – 1 to 5
Implemented and functioning without errors but not integrated – 6 to 9
Implemented and fully integrated but buggy – 10 to 14
Implemented, fully integrated, and functioning without errors – 15 to 16

Feature 2 (up to 16)

Sub-features have not been implemented – 0
Attempted, not complete, or very buggy – 1 to 5
Implemented and functioning without errors but not integrated – 6 to 9
Implemented and fully integrated but buggy – 10 to 14
Implemented, fully integrated, and functioning without errors – 15 to 16

Feature 3 (up to 8)

Sub-features have not been implemented – 0
Attempted, not complete, or very buggy – 1 to 3
Implemented and functioning without errors but not integrated – 4 or 5
Implemented and fully integrated but buggy – 6 or 7
Implemented, fully integrated, and functioning without errors – 8

Use of OOP techniques [20]

Abstraction (up to 8)

No classes have been created – 0
Classes have been created superficially and not instantiated or used – 1 or 2
Classes have been created but only some have been instantiated and used – 3 or 4
Useful classes and objects have been created and used correctly – 5 or 6
The use of classes and objects exceeds the specification – 7 or 8

Encapsulation (up to 8)

No encapsulation has been used – 0
Class variables and methods have been encapsulated superficially – 1 to 3
Class variables and methods have been encapsulated correctly – 4 to 6
The use of encapsulation exceeds the specification – 7 or 8

Inheritance (up to 4)

No inheritance has been used – 0
Classes have been inherited superficially – 1 or 2
Classes have been inherited correctly – 3
The use of inheritance exceeds the specification – 4

Bonus marks will be awarded for the appropriate use of polymorphism (bonus marks up to 5)

Quality of Code [10]

Code Duplication (up to 4)

Code contains too many unnecessary code repetitions – 0
Regular occurrences of duplicate code – 1
Occasional duplicate code – 2
Very little duplicate code – 3
No duplicate code – 4

PEP8 Conventions and naming of variables, methods, and classes (up to 3)

PEP8 and naming convention has not been used – 0
PEP8 and naming convention have been used occasionally – 1
PEP8 and naming convention have been used, but not regularly – 2
PEP8 convention used professionally, and all items have been named correctly – 3

In-code Comments (up to 3)

No in-code comments – 0
Code contains occasional in-code comments – 1
Code contains useful and regular in-code comments – 2
Thoroughly commented, good use of docstrings, and header comments describing.py files – 3

2. Documentation (20)

Report clarity and completeness (up to 10)

(must include clear exposition about the design and decisions for OOP use and implementation)

The documentation cannot be understood on first reading or is mostly incomplete – 0
The documentation is readable, but a section(s) is missing – 1 to 3
The documentation is complete – 4 to 6
The documentation is complete and of a high standard – 7 to 10

Testing (5)

Testing has not been demonstrated in the documentation – 0
Little white box testing has been documented – 1 or 2
White box testing has been documented for all the coursework – 3 or 4
White box testing has been documented for the whole system – 5

Evaluation (5)

No evaluation was shown in the documentation – 0
The evaluation shows a lack of thought – 1 or 2

The evaluation shows thought – 3 or 4
The evaluation shows clear introspection and demonstrates increased awareness – 5

3. Acceptance Test - Screencast (10)

Screencast (up to 10)

Not submitted or no work demonstrated – 0
Work demonstrated was not up to the standard expected – 1 to 3
Work demonstrated was up to the standard expected – 4 to 7
Work demonstrated exceeded the standard expected – 8 to 10

Note: you will lose up to 5 marks if you do not follow the zip or report file name specification defined in the deliverables section. You may also lose up to 5 additional marks if your report includes screenshots of the code required instead of text. **You may fail the assessment overall if you do not submit the mandatory screencast (acceptance test)**.

Assessment Criteria

These are the key points that will be considered when marking your work:

a. Software Development (Python Project)

Features implemented. The number of features, from those listed in the requirements section above, that you have successfully implemented, and the quality of their design will influence your overall mark. The features implemented must run smoothly and produce the expected results without errors, bugs, or system crashes.

Reliability of the code. Does it run correctly, or does it crash or give incorrect results? Are exceptions handled properly? Bugs that you admit on your bug list (see deliverables) will be looked at more kindly than those that are not declared.

The user interface. Input and output should be implemented via the console only. This is not a module about user interface design, but credit will be given for making your application as pleasant an experience as possible for the user. Visualising results does not need to use graphics, simply present these in tabular form with appropriate spacing.

OOP Features. Does the code make use of classes? Is the choice of classes appropriate? The number of classes implemented. Are they well abstracted and encapsulated? Are the classes instantiated and used elsewhere in the code? Is inheritance used, i.e., does the code derive new class(es) based on an existing class?

Quality of the code. For example the inclusion of meaningful comments, the use of sensible naming standards (e.g., for variables and methods), and code layout. Follow the Python naming conventions in PEP 8.
Points to consider when designing and developing your code:

- Lots of duplicate/near duplicate code usually indicates poor design which would benefit from being turned into functions.
- Features of the Python language (e.g., OOP) and functions should be used appropriately.
- Code decomposition into appropriate modules.
- How easy would it be to add /change features? e.g., if making a relatively small enhancement to the functionality would require a lot of change to the code, then this is usually an indication of poor design.
- Thorough exception handling.

b. Acceptance tests

Ability to answer questions. All members of the team should be able to:

- Identify how the visible behaviour of the program is implemented in the code (e.g., if asked "Where is this function carried out", you should be able to quickly and accurately find the code).
- Talk through specified fragments of code (e.g., if asked "What does this method do" you should be able to give an overview of its purpose and then go through it line by line explaining its logic and behaviour).
- Explain the reasons for specified design decisions and discuss alternative possibilities.

c. Report

Your report will be assessed on the following criteria.

- Are all the required sections included and completed properly?
- Is the report clear, accurate, and easy to read?
- Does the report give an accurate representation of what you have achieved?
- Is the evaluation realistic and does it show deep introspection and that you have really thought about your project and performance, and relevant issues and have given a balanced view of positive and negative points?

Grading Criteria

The criteria for assessing the items in the marking scheme will be made based on the following benchmarks. To be eligible for the mark in the left-hand column, you should at least achieve what is listed in the right-hand column. Note that you may be awarded a lower mark if you don't achieve all the criteria listed. For example, if you achieve all the criteria listed for a 2:1 mark (60% - 69%) but have a poor report then your mark might be in the 2:2 (50% - 59%) range or lower.

Grade Rang	Grading Criteria
> 80% ***Exceptional***	**Software development**: Completed all features and sub-features to an exceptional or outstanding standard (all required sub-features implemented, no obvious bugs, outstanding code quality, no duplicate code, OOP, and Python language features accurately applied and used). **Acceptance test**: Able to show outstanding and thorough knowledge and systematic understanding of OOP concepts and Python language features at the demonstrations and in the report, including deep critical discussions of design choices and possible alternative implementation choices. **Overall report**: Outstanding (required sections completed accurately and clearly; easy to read; coherent, insightful arguments for design justification and use of OOP). The evaluation section of the report shows a deep analysis of the design and implementation, including an account of the decisions made. Outstanding, insightful points relevant to the development and design produced, productivity, errors, learning, and time management, thorough introspection, realistic and insightful reflection. An accurate self-assessment.
70-79% ***Excellent***	**Software development**: Completed all the features and sub-features to an excellent standard (required sub-features implemented, no obvious bugs, excellent code quality, no duplicate code, OOP concepts, and Python language features accurately applied and used). **Acceptance test**: Able to show excellent, detailed knowledge and systematic understanding of OOP concepts and Python language features at the acceptance test demonstration and in the report, including deep critical justification of design choices. **Overall report**: Excellent (required sections completed accurately and clearly; easy to read, well justified design decisions and clear argument with good comparative reasoning). The evaluation section of the report contains an excellent analysis of the design and implementation, including an account of the decisions made. Interesting points relevant to development produced, productivity, errors, learning, and time management, detailed introspection and interesting reflections. Accurate self-assessment.
60-69% ***Very Good***	**Software development**: Completed all the features and sub-features to a very good standard (required sub-features implemented, few if any bugs, very good code quality, may contain duplicate code, very good use of Python language features, very good OOP implementation – very good understanding of concepts but implementation may contain some design flaws). **Acceptance test**: Able to show very good knowledge and systematic understanding of OOP concepts and Python language features at the acceptance test demonstration and in the report, including very good critical justification of design choices. **Overall report**: All required sections completed accurately and clearly; a good argument for design justification and OOP use. The evaluation section of the report makes very good points relevant to at least four out of the following matters: development (system produced), productivity, errors, learning, and time management. Introspection shows some reasonable thought. Mostly accurate self-assessment.

50-59% ***Good***	**Software development**: Completed all but one of the features or two of the sub-features to a good standard (few bugs, contains some duplicate code, good OOP attempt and use of Python language features, contains some design flaws). **Acceptance test**: Able to show good knowledge and mostly accurate systematic understanding of OOP concepts and Python language features at the acceptance test demonstration and in the report, including a good rationale for design choices. **Overall report**: Most sections completed accurately; mostly clearly written - may contain typos and grammatical errors, some reasonably balanced arguments about design justification and OOP use. The evaluation section of the report addresses points relevant to at least three out of the following matters reasonably: development (system produced), productivity, errors, learning, and time management. Contains limited introspection. Sound accurate self-assessment.
46-49% ***Satisfactory (Strong 3rd)***	**Software development**: Completed at features 1 and 2 to a working standard (contains minor bugs and some duplicate code, basic attempt at implementing OOP and Python language features – classes defined but not used; contains some design flaws). **Acceptance test**: Able to show satisfactory knowledge of OOP concepts and Python language features at the acceptance test demonstration and in the report, understanding is less systematic showing unbalanced rationale for design choices. **Overall report**: Acceptable. Most required sections completed; mostly accurate and clear with some typos and grammatical error; superficial justification for design choices and OOP use. The evaluation section of the report addresses points relevant to at least three out of the following matters reasonably: development (system produced), productivity, errors, learning, and time management. Mostly descriptive; superficial introspection. Flawed self-assessment.
40-45% ***Satisfactory (Weak 3rd)***	**Software development**: Completed at least feature 1 and most of feature 2 to a working standard (contains some bugs, some duplicate code, basic attempt at implementing OOP concepts and Python language features – classes defined but not used; contains some design flaws). **Acceptance test**: Able to show fairly satisfactory knowledge of OOP concepts and Python language features at the acceptance test demonstration and in report, understanding is less systematic showing little rationale for design choices. **Overall report**: Acceptable (required sections completed, mostly accurate, clumsy language, descriptive account of design choices and OOP use). The evaluation section of the report addresses points relevant to at least two out of the following matters reasonably: development (system produced), productivity, errors, learning, and time management. Mostly descriptive; superficial introspection. Flawed self-assessment.
35-39% ***Fail***	**Software development**: Reasonable attempt at some of the sub-features for the selected feature from F1-F2 but not fully working and maybe buggy (at least 4 required sub-features implemented, contains errors and bugs, significant duplicate code, basic or no attempt at implementing OOP concepts and Python language features – too few or no classes defined or used; contains system design and logic flaws). **Acceptance test**: Confused knowledge of OOP concepts and Python language features at the acceptance test demonstration and in report. Some evidence of systematic understanding showing confused rationale for design choices. **Overall report**: Mostly completed to an acceptable standard. Some sections are incomplete, inaccurate, confused and/or missing. The evaluation section of the report is mostly descriptive but showing some thought. Inaccurate self- assessment or missing.
<35% ***Fail***	**Software development**: Little or no attempt at implementing required sub-features or very buggy. No or flawed attempt at implementing OOP concepts and Python language features. **Acceptance test**: Not able to show satisfactory knowledge or systematic understanding of OOP concepts and Python language features at the acceptance demonstration and in report. No or little evidence of rationale for design choices. **Overall report**: Mostly incomplete, at an un-acceptable standard or missing. Evaluation is descriptive, showing a lack of thought or missing. Inaccurate self-assessment or missing.