COMP1811 Python Project Coursework Specification

COMP1811(2021/22)	**Paradigms of Programming**	**Contribution:** 40% of module
Module Leader/Moderator Yasmine Arafa/Andy Wicks	Practical Coursework 1: Python Project	**Deadline Date** Monday 31/01/2022
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 Monday 31/01/2022.** Submissions must be made using the Python Project Upload link under "Coursework Specification and Submission" on the Moodle page for COMP1811. Your grade for your coursework will be capped at 40% if you fail to submit by the deadline.
- 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** AND **b) a PDF version of your report**. In general, any text in the document 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.
- Make sure that 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.
This is a group project development coursework. **You are expected to work in pairs to implement your project**.

The overall task for this project is to design and develop a standalone application a **Quiz Generator system** in Python for maintaining a bank of quiz questions for different subjects covered on different modules. The system should follow the specification below. Implementing the project will give you practical experience in Object-Oriented Programming (OOP) development. You are expected to design, implement and *use* your own classes that are relevant to the application. The specification provides less overview on how to do the project as you are expected to come up with and design the solution yourself. Your project development is the culmination of all what you have learnt about Python and OOP, and will showcase all your hard work for part one of this module. The system you develop should become a strong addition to your programming portfolio. You should aim to produce a system that is well-designed, robust and useful. The GUI design should be clean, and simple to navigate. The system should operate smoothly without sluggishness or crashes and should not require instructions or a manual to use.

Scenario and System Specifications

Most of the modules you will take over your time at university have an examination, either as the assessment or as part of the assessment. You have found that taking notes in lectures helps to remember, but it does not help with understanding and assessing yourself, which is a requirement at university. Whilst you can go back over your notes before an exam, this does not help you to demonstrate the understanding that gets you the higher grades.

A bit of thinking gave you a potential solution! Maybe you could write a program that allows you to create a small quiz each week. So, you create a set of three or four questions each week which cover the main points raised for each subject. To create complex and varied quizzes, you think of creating different types of questions that are either multiple choice questions (MCQs), true-or-false (TF), or match the correct answer (BestMatch), and create a series of potential answers (of which one or more is/are correct), as well as a brief description on why each potential answer is either right or wrong.

You can then use this bank of questions when it comes to revision. Your program should also have the ability to select, say, 5 questions randomly from your bank of questions for you to take as a quick and easy way to revise. Not only are the questions selected randomly from your question bank, but the potential answers are also randomised so that you do not remember the position of which answer is correct. Your program should give you feedback after each run of the test. It will tell you your overall score as well as show the brief description of why each answer you gave was either right or wrong along with some brief statistics.

System Features and Requirements

The system must provide the following features and comply with the System Specifications outlined above. Each partner in your coursework development pair must implement one of the features below and integrate it with the others. All features will require a GUI and database/file handling.

F1 Administration Features

i. Maintain a list of modules (or topics) on a database named `question_bank.db` in a `modules` table (or a text file named `modules.txt`):

- **add a module** to the list of available modules on you `modules` db table or file:
  this feature should allow the user to enter the module code and name, and should prompt the user if the module code already exists;
- **edit an existing module**
  this feature should allow the user to select a module from the module list and edit its name;
- **delete an existing module**
  this feature should allow the user to select a module from the module list then delete the whole record.

ii. Maintain a question bank per module (topic) in the `question_bank.db` in a `questions` table (or a `questions.txt` file):

- **add a question and possible answers**:
  this feature should allow the user to select a module from the list of available modules and enter a question with the following details: the question text, the question type (MCQ, TF or BestMatch), the text for 4 or 5 possible answers and a the mark awarded for the question if all the answers are correct; then for each of the answers, the user must be able to indicate whether or not it is correct and enter a brief description explaining why (this will serve as feedback generated when the test is taken);
- **edit an existing question**
  this feature should allow the user to first select a module, then display the list of existing questions the user can select from to modify the question text and/or answer text and details;
- **delete an existing question**
  this feature should allow the user to first select a module, then display the list of existing questions for the user to select from to delete along with all its possible answers details;.

iii. Application Interactive GUI

- display a menu for selecting between the system features outlined for both F1 and F2, and include an option to exit the system;
- this should facilitate the integration of code implemented by each partner.

F2 Quiz Features

i. Generate quiz with 5 random question:

- this feature should allow the user to select a module from the list of available modules and display 5 random questions from the question bank produced in F1.ii along with their associated answers;
- the possible answers for each question must be displayed in random order and should allow the user to select, using an appropriate GUI widget, one or more correct answers depending on how many possible answers there are for that question;
- the quiz GUI should display each question on a separate frame that includes buttons to navigate between frames and an additional button for submit.

ii. Score the quiz after it has been submitted:

- the feature should collect the mark for each question, then calculate and display the overall total score after each run of the quiz;
- the feature should also display feedback along with the score which shows a brief description of why the answer given were either right or wrong;
- then store the date of the quiz and the total score achieved to a `results` table (or `results.txt` file).

iii. Generate quiz reports:

- this feature should allow the user to select a module (topic) then generate and display the following reports in appropriate GUIs:
  - an achievements report that displays: the number of times a quiz was taken for that module, the average score achieved, and the lowest and highest score achieved;
  - a question statistics report that the displays the most 2 questions repeated in the quizzes taken.
- the feature should also provide an option to save the report to file for printing.

Development

In a team of 2, you are expected to develop a standalone (desktop) version of the application. An appropriate GUI needs to be implemented to present the system features in a user friendly way. The features implemented by each partner must be fully integrated so that the system runs as one application (see what is meant by "Fully integrated" below). Your implementation must demonstrate appropriate use of the Python object-oriented concepts covered. Getting your project to function as required is important, but it is only one part of this assessment. What is most significant is the quality of your code and the OOP design features you implement.

Your implementation must define and use at least three classes and at least one appropriate inheritance hierarchy. Implementing polymorphism is option and will award you extra marks - try to think where it would be useful in your implementation. Make sure your classes apply appropriate abstraction and encapsulation. All three 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. Some of the features outlined above are very similar. Think about how you can use classes and inheritance to reduce duplication in the code. This must be discussed as a team to avoid each partner producing different versions of similar functions.

The code produced should implement all the features in a professional style that uses correct naming convention for all classes, modules, methods, functions and variables; considers appropriate OOP techniques; and should work as expected when executed. Please refer to PEP 8 for a guide to Pythonic style conventions.

Hits for Developing Your Project

The principle idea to approaching any problem-solving task is to first understand the problem, breakdown that problem into smaller chunks, decide who is doing what to solve those 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 meet up with your partner as soon as possible to discuss this, distribute the work, make decisions, decide on regular milestones (where each part must deliver and demonstrate their work) and follow up with frequent progress meeting.

1. **Understanding the problem** is often neglected as being obvious, but it isn't always the case! So as a team, come up with questions to clarify the problem and discuss them to make sure you both have the same understanding. For example, you might ask: what are you 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 the same as your partner's?
2. **Breakdown 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 requires you to read from a database so can you write a function or method to read from the database, which you can reuse 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. Agree on dates when each partner will deliver each of those parts and make sure you deliver on time and make sure you test each other's work.
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 so that you develop reusable and robust. 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 great reusable code that your partner can use.
   The coursework is released before the concepts of OOP, GUI's, and database handling are covered in the lectures. This should not set you back and you should get started with your implementation as soon as possible. For example, assume the data you will eventually get from the database is on some data structure (which you will need to do anyway) and use that data structure wherever you need the data; use the PyCharm console instead of a GUI; and define function to implement the functionally outlined in the specifications above. Once we have covered all each of the relevant topics, modify your code to read from/update a database, or use a GUI, and rework your functions as methods in classes and use those class methods instead.
   Remember to test your parts of code thoroughly as you go along and especially before inflicting them on your partner, as well as the integrated program as whole before inflicting it on other users.

"Fully integrated" or integration means bringing together various pieces of software (in your case, the parts that your partner developed) so that together they make up a single system, the Quiz Generator. Integration can be done a number of ways: a) by developing a central GUI that contains a menu or a set of buttons that link (call) the functions/methods (i.e. the features) you partner developed in one place so that the user has access to all the system features required; b) by sharing a class(es) and having your partner instantiate that class and use its methods where needed; and c) by sharing a module (file) that contains functions your partner may need and having your partner call those functions where needed.

Groups

You are expected to work in pairs to develop your coursework project and **you are expected to form your pair and inform your tutor by 15/10/21. Pairs must be formed from within your timetabled lab session only**. Inform your tutor by 15/10/21 if you are not able to find a partner or you are studying online and one will be assigned to you. So, once you have found a partner, arrange to meet as soon as possible on campus or on Microsoft Teams (or on any other suitable platform) to decide which feature each will be responsible for and to start planning for your project design and implementation.

Working as a pair means the workload must be equally distributed between both partners, as far as possible, and that both partners must have an equal understanding of all the code produced by the pair. The system requirements have been divided into 2 sets of partially independent features. This will make the equal distribution of effort easier for you. Each of the 2 system features can be implemented separately by each partner. So you do not need to wait around for your partner to complete their work before you can start yours. As a pair, you must nonetheless discuss and agree on the classes that will be shared (class design) and the methods needed for integration. You will also need to agree on the database tables and files that all features will need to read and update.

Deliverables

1. **Milestones and Acceptance tests** (demonstration of your code and what you have achieved).
   1.1 **Progress Milestone 1 (08/11/21)**: This is a progress check where each partner must demonstrate the GUI for at least 1 of the sub-features from the system feature they are responsible for, and at least 2 separate Python functions that implement the functionality requirements of that sub-feature. At this stage, you are expected to display the layout of the GUIs and do not need to link them with the functions created. **The milestone is not marked but it is an opportunity for you to receive formative feedback**.
   1.2 **Progress Milestone 2 (13/12/21)**: Another progress check where each partner must demonstrate the full functionality of at least 1 sub-feature from the system feature they are responsible for. Parts of the class design must be implemented at this stage. Each partner is also expected to show the implementation of at least one relevant class and how it is used. You must also show how you have or plan to integrate code developed by your partner.
   The interim demo should take up to 5 minutes. Each partner is required to individually demonstrate a running program, as well as describe the code they have written so far.
   1.3 The **final demo (due 31/01/22)** is to demonstrate the complete, fully integrated system that includes all the requirements for all 4 system features described under system requirements. The demos will take place during your lab session on 31/01/21.
   The final system demonstration should take up to 10 minutes. As a team of two, you are required to demonstrate the full system running. Individual partners must show the feature and code they implemented, and are expected to show thorough understanding of the system as a whole and the rationale behind its design, as well as demonstrate a good understanding of each other's code, and how it was integrated. This is a team demo where each partner must take part.

2. **Python Code** – A **zip file containing the entire folder for your PyCharm project** which includes the integrated code produced by all members of your group. **Code created by each group member must be clearly labelled with their student name and ID** in an in-code comment. Name the Zip file as follows: <StudentID>_COMP1811_PythonCode.zip. **Each partner must individually upload an identical zip containing the integrated features.** Please try to structure your work so that it is easy for the person marking the work to run your project.

   Upload the zip file to the Moodle Python Project Upload link under the Coursework Specification and Submission block by **23:30 on Monday 31/01/22. 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.**

3. **Python Project Report** – This is the final deliverable. The 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 coursework report template. Fill in all the sections and convert it to a PDF (using "Save As ... PDF") before you submit. Sections 4, 6 and 8 are individual work and the remainder of the sections are written by the group. **You will lose marks and are likely to fail this assessment if you do not upload your 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 Monday 31/01/22**. Name your report as follows: <StudentID>_ COMP1811_PythonProjectReport.pdf. **You will lose marks if you do not follow this file name specification.**

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 another student) then include a reference showing where the code or ideas came from and 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 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. Features implemented. [30]
   b. Use OOP techniques. [25]
   c. Quality of code. [15]
2. **Documentation. (20)**
   a. Report clarity and completeness (including clear exposition on design and decisions for OOP use. [10]
   b. Testing. [5]
   c. Objective evaluation of work. [5]
3. **Final acceptance test (demonstration). (10)**

Marking Breakdown

1. Code development (70)

Features Implemented [30] (group work and integration will be assessed here)

Sub-feature i (up to 8)

Sub-features have not been implemented – 0
Attempted, not complete or very buggy – 1 or 2
Implemented and functioning without errors but not integrated – 3 or 4
Implemented and fully integrated but buggy – 5 or 6
Implemented, fully integrated and functioning without errors – 7 or 8

Sub-feature ii (up to 10)

Sub-features have not been implemented – 0
Attempted, not complete or very buggy – 1 or 2
Implemented and functioning without errors but not integrated – 3 to 5
Implemented and fully integrated but buggy – 6 to 8
Implemented, fully integrated and functioning without errors – 9 or 10

Sub-feature iii (up to 12)

Sub-features has not been implemented – 0
Attempted, not complete or very buggy – 1 to 3
Implemented and functioning without errors but not integrated – 4 to 6
Implemented and fully integrated but buggy – 7 to 9
Implemented, fully integrated and functioning without errors – 10 to 12

Use of OOP techniques [25]

Abstraction (up to 10)

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 to 7
The use of classes and objects exceeds the specification – 8 to 10

Encapsulation (up to 10)

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 to 10

Inheritance (up to 5)

No inheritance has been used – 0
Classes have been inherited superficially – 1
Classes have been inherited correctly – 2 to 4
The use of inheritance exceeds the specification – 5

Bonus marks will be awarded for the appropriate use of polymorphism (bonus marks up to 10)

Quality of Code [15]

Code Duplication (up to 8)

Code contains too many unnecessary code repetition – 0
Regular occurrences of duplicate code – 1 to 3
Occasional duplicate code – 4 to 5
Very little duplicate code – 6 to 7
No duplicate code – 8

PEP8 Conventions and naming of variables, methods and classes (up to 4)

PEP8 and naming convention has not been used – 0
PEP8 and naming convention has been used occasionally – 1
PEP8 and naming convention has been used, but not regularly – 2
PEP8 and naming convention has been used regularly – 3
PEP8 convention used professionally and all items have been named correctly – 4

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)

Design (up to 10) clear exposition about the design and decisions for OOP use

The documentation cannot be understood on first reading or mostly incomplete – 0
The documentation is readable, but a section(s) are 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, demonstrates increased awareness – 5

3. Acceptance Test - Demonstration (10)

Final Demo (up to 10)

Not attended or no work demonstrated – 0
Work demonstrated was not up to the standard expected, superficial team contribution – 1 to 3
Work demonstrated was up to the standard expected, sufficient team contribution – 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 on page 7. You may also lose up to 5 additional marks if your report includes screenshots of the code required instead of text.

Assessment Criteria

These are the key points that will be taken into account 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 integrated, and the quality of their design will have an effect on 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 on more kindly than those that are not declared.

The user interface. 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 (for example for the sales receipt, shopping basket, sales reports, etc.).

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

- Lots of duplicate / near duplicate code usually indicate poor design which would benefit from refactoring.
- 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 visible behaviour of the program is implemented in the code (e.g. if asked "Where is this function carried out", they will be able to quickly and accurately find the code).
- Talk through specified fragments of code (e.g. if asked "What does this method do" they will 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.

Note that as group members are expected to have developed the code as a team, all members should be able to answer questions about any part of the code. An answer such as "my partner wrote that bit of code so, I can't answer questions about it..." may negatively affect your individual mark.

c. Group work

Group work. You will be assessed on the outcome of your joint work as a pair (i.e. the integrated system) as well as how well your pair worked. That includes: how well the team worked together to achieve objectives? team organisation (measured through observations of team effort and discussion during the labs and demos); individual commitment to working with the team and effort contribution (measured using an agreed team pro forma submitted with the final report.

d. 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, 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 but have a poor report then your mark might be in the 2:2 range or lower.

You will be assessed both on your success as a group and your individual performance. The pro forma in the report, your report and your performance during acceptance tests will be used to assess group and individual performance.

Grade Range	Grading Criteria
> 80% ***Exceptional***	**Software development**: Completed all sub-features for a selected feature from F1-F2 to an exceptional or outstanding standard (all required sub-features implemented, no obvious bugs, out-standing 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. **Group work**: Pair worked well together to achieve objectives. Extensive commitment to the team, ensuring partner engagement and feature integration. Well organised and made significant contribution to system design and development. **Overall report**: Outstanding (required sections completed accurately and clearly; easy to read; coherent, insightful arguments for design justification and use of OOP). Evaluation (section 8 of report): Deep analysis of the design, implementation and group decisions. Outstanding, insightful points relevant to the development and design produced, productivity, errors, learning, and time management; thorough introspection, realistic and insightful reflection. Accurate self-assessment.
70-79% ***Excellent***	**Software development**: Completed all the sub-features for a selected feature from F1-F2 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 report, including deep critical justification of design choices. **Group work**: Pair worked well together to achieve objectives. Excellent commitment to the team and facilitating partner engagement and system feature integration. Well organised and made significant contribution to system design and development. **Overall report**: Excellent (required sections completed accurately and clearly; easy to read, well justified design decisions and clear argument with good comparative reasoning). Evaluation (section 8 of report): Excellent analysis of the design, implementation and group decisions. 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 sub-features for a selected feature from F1-F2 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 report, including very good critical justification of design choices. **Group work**: Pair worked well together most of the time, with only a few occurrences of communication breakdown or failure to collaborate when appropriate. A good number of team meetings organised and attended to achieve objectives. Made significant contribution to system design and development.

Grade Range	Grading Criteria
	Overall report: All required sections completed accurately and clearly; good argument for design justification and OOP use. Evaluation (section 8 of report): 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 at least 3 of the sub-features for the selected feature from F1-F2 to a good standard (at least 2 sub-features implemented, 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 report, including good rationale for design choices. **Group work**: Pair worked well together most of the time, with only a few occurrences of communication breakdown or failure to collaborate when appropriate. An adequate number of team meetings held or attended to achieve objectives. Made partial contribution to system design and development. Sub-features implemented are fully integrated within overall system. **Overall report**: Most sections completed accurately; mostly clearly written - may contain typos and grammatical error; some reasonably balanced arguments about design justification and OOP use. Evaluation (section 8 of report): Reasonably addresses points relevant to at least three out of the following matters: 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 least 3 of the sub-features for the selected feature from F1-F2 to a working standard (at least 2 required sub-features implemented, 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 report, understanding is less systematic showing unbalanced rationale for design choices. **Group work**: Pair worked together some of the time with frequent occurrences of communication breakdown or failure to collaborate when appropriate. Few team meetings organised or attended. Features implemented may be only partially integrated with system or not working accurately. **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. Evaluation (section 8 of report): Reasonably addresses points relevant to at least three out of the following matters: 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 2 sub-features for the selected feature from F1-F2 to a working standard (at least 1 required sub-feature fully implemented, 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. **Group work**: Pair did not collaborate or communicate well. Members often worked independently with weak regard to objectives or priorities. Very few team meetings organised or attended. Features implemented may be only partially integrated or not working accurately. **Overall report**: acceptable (required sections completed, mostly accurate, clumsy language, descriptive account of design choices and OOP use). Evaluation (section 8 of report): Reasonably addresses points relevant to at least two out of the following matters: development (system produced), productivity, errors, learning, and time management. Mostly descriptive; superficial introspection. Flawed self-assessment.

Grade Range	Grading Criteria
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 1 requited sub-feature 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. **Group work**: Pair did not collaborate or communicate well. Partners mostly worked independently, without regard to objectives or priorities. Very few team meetings organised or attended. Features implemented not integrated or not working accurately. **Overall report**: Mostly completed to an acceptable standard. Some sections are incomplete, inaccurate, confused and/or missing. Evaluation (section 8 of report): 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. **Group work**: Pair did not collaborate or communicate. Partners worked independently, without regard to objectives or priorities. No meetings attended. Very little or no attempt at integration. **Overall report**: Mostly incomplete, at an un-acceptable standard or missing. Evaluation (section 8 of report): Descriptive, showing a lack of thought or missing. Inaccurate self-assessment or missing.