COMP1811 **(2021/22)**	**Paradigms of Programming**	**Contribution:** 40% of overall grade for the module
Module Leader/Moderator Yasmine Arafa / Andy Wicks	Python Development Project	**Deadline Date** Friday 15/07/2022
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 Friday 15/07/22.** Submissions must be made using the Resit Coursework link under "RESIT Coursework Specification and Upload" on the Moodle page for COMP1811.
- For this coursework you must submit THREE files:
  1) a **zip file containing** all the full directory containing the files required to run **your Python project**,
  2) a **PDF version of your report**. Any text in the report MUST NOT be an image (i.e. must not be scanned). There are limits on the file size (typically 2gb), and
  3) a screencast explaining your code design and showing your programming running.
- 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.
- Your work will be marked online and comments on your work and a provisional grade will be available from the Coursework page on Moodle. A news item will be posted when the comments are available, and also when the grade is available in BannerWeb.
- 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 Extenuating Circumstances you may submit your coursework up to 10 working days after the published deadline without penalty, but 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.
- 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.

Detailed Specification

Please read the entire coursework specification before starting work.
This coursework is individual work.

Scenario

You have decided that it would be helpful to have a program on your laptop which helps organise your work whilst at university. One way to organise your study is to keep on top of coursework requirements and deadlines for each of the modules you are taking. So far you have done this manually on paper or by recording lists of coursework deadlines in MSWord files on your computer. As the number of coursework and deadlines increase, it has become tedious and time consuming for you to review specific coursework requirements and due dates. You have realised that an electronic organiser would be more efficient. There are so many new ways of doing things and so many new ways of presenting information that a good computer program would be really useful. Since you are learning Python anyway, it seems sensible to use that.

The program should help you to better record, review and keep track of coursework requirements and deadlines.

System Requirements

Your overall task for this project is to design and develop a simple **personal organiser system** in Python for recording and maintaining coursework deadlines for the modules you take at the University of Greenwich. The system should follow the specification below. Its design and implementation must consider the OOP concepts covered and you are expected to identify and implement appropriate classes that are relevant to the application. That is, make sure you consider Abstraction, Encapsulation, Inheritance and possibly Polymorphism when implementing these classes.

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.

The program should be capable of the following:

1. **maintaining a list of available modules in a table called `modules` (or on the `modules.txt` text file)**:
   a. **add a module** to the list of available modules
      this feature should allow the user to enter the module code and name, and should prompt the user if the module code already exists;
   b. **edit an existing module**
      this feature should allow the user to select a module from the module list and edit its name;
   c. **delete an existing module**
      this feature should allow the user to select a module from the module list then delete the record from file.
2. **maintaining a list of coursework expected for each module in the `coursework` table (or the `coursework.txt` file)**:
   a. **add a coursework** for a module
      this feature should allow the user to select a module from the module list and enter the following coursework details: coursework code, coursework description and due date and time, and should prompt the use if the coursework code already exists;
   b. **edit an existing coursework**
      this feature should allow the user to select a module from the module list and a coursework code from the coursework list for that module, then to edit the coursework description, due date and/or due time;

c. **delete an existing coursework**
   this feature should allow the user to select a module from the module list and a coursework code from the coursework list for that module, then to delete the record for that coursework from file.
d. **display the due date and time for all the coursework required for a module**
   this feature should allow the user to select a module from the module list then display the deadlines for each of its coursework;
e. **alert you about upcoming coursework deadlines due in the next week**
   this feature should display a list of module codes, coursework codes and deadline for coursework due in the next 7 days; the results can be displayed in a dialog box prompted to users when they first open the application or when this feature is selected from menu.

3. **maintaining a list of requirements for each coursework in the `requirements` table (or the `requirements.txt` file)**:
   a. **add a requirement** for a coursework
      (for example: a zip file of Python code, report, research, annotated bibliography, etc.)
      this feature should allow the user to select a module from the module list and a coursework code from the coursework list for that module, then enter a text description for the requirement;
   b. **edit an existing requirement** for a coursework
      this feature should allow the user to select a module from the module list, a coursework code from the coursework list for that module, and requirement from list of requirements then to edit its text description;
   c. **delete an existing requirement** for a coursework
      this feature should allow the user to select a module from the module list, a coursework code from the coursework list for that module, and requirement from list of requirements, then to delete the record for that requirement from file
   d. **display all requirements** for a coursework
      this feature should allow the user to select a module from the module list and a coursework code from the coursework list for that module then display all the requirements for that coursework.

You must implement ALL three features on your own. Your implementation must demonstrate appropriate use of the Python language features and 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.

The user interface should implement appropriate GUI features. The code should implement all the requirements in a professional style that uses correct naming convention for all classes, 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.

Deliverables

1. **Python Code** – Include the whole directory which contains all the files required to run your PyCharm project that includes all the .py files and folders in you PyCharm project folder including the "venv" folder.

   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 considered to be plagiarism.

2. **Report** – The report should consist of **all** the sections described in the coursework report template provided on Moodle. Fill in all the sections and convert it to a PDF (using "Save As ... PDF") before you submit. Make sure you rename the report template to <YourStudentID>_RESIT_PythonCW_Report_Sept20.pdf. **You will lose marks and are likely to fail the coursework if you do not upload your report**.

3. **Screencast** – The screencast must be 6-8 minutes long and should show all the functions running as well as provide a brief exposition about the code design. Save your recording as <YourStudentID>_COMP1811_RESIT_PythonProject_Screencast_2021.mp4

Upload a **zip file** containing the **Python code** in your PyCharm project **AND** a **PDF version of your report** by **23:30 on 15/07/22** to the "Python Project CW Upload" area under the "RESIT Coursework Specification and Submission" block on the Moodle page for COMP1811. Remember to name your Zip file as follows: <YourStudentID >_COMP1811_RESIT_PythonCode_2022.zip, your report as <YourStudentID >_COMP1811_ RESIT_PythonProject_Report_2022.pdf, and your screencast as: <StudentID>_COMP1811_RESIT_ PythonProject_Screencast_2022.mp4.

Marking Scheme

This coursework will not be marked anonymously.

Marks will be awarded for the following. Note that marks shown are out of 100.

1. **Code development (65)**
   a. Features implemented. [30]
   b. Use of OOP techniques. [20]
   c. Quality of code. [15]
2. **Documentation. (25)**
   a. Clear explanation of code design and decisions for OOP use, and overall report clarity and completeness [15]
   b. Testing. [5]
   c. Objective evaluation of work. [5]
3. **Acceptance test (screencast)** [10]

Marking Breakdown

1. Code development (65)

a. Features Implemented [30]

Feature 1 (up to 6)

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

Feature 2 (up to 14)

Sub-features have not been implemented – 0
Attempted, not complete or very buggy – 1 to 4
Implemented and functioning without errors but not integrated – 5 to 8
Implemented and fully integrated but buggy – 9 to 11
Implemented, fully integrated and functioning without errors – 12 or 14

Feature 3 (up to 10)

Sub-features has 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

b. 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 to 5
Useful classes and objects have been created and used correctly – 6 or 7
The use of classes and objects exceeds the specification – 8

Encapsulation (up to 8)

No encapsulation has been used – 0
Class variables and methods have been encapsulated superficially – 1 or 2

Class variables and methods have been encapsulated correctly – 3 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
Classes have been inherited correctly – 2 or 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)

c. 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 briefly describing each .py file – 3

2. Documentation (25)

a. Design (up to 15) clear explanation of code 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 and superficial explanation of code design – 1 to 5
The documentation is complete, and the explanation of code design is clear and mostly thorough – 6 to 10
The documentation is complete and of a high standard, and code design is clear and thorough – 11 to 15

b. 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

c. 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 – Screencast demo [10]

Not attended 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

Grading Criteria

Judgement made for the items in the marking scheme will be made based on the following criteria. 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.

> 80% ***Exceptional***	• Completed all features 1 to 3, implemented to an outstanding standard or above (all required features implemented, no obvious bugs, outstanding code quality, no duplicate code, OOP and Python language features accurately applied). • Able to show outstanding and thorough knowledge and systematic understanding of OOP concepts and Python language features in the code and report, including critical discussions of design choices and possible alternative implementation choices. • Overall report – outstanding (required sections completed accurately and clearly, easy to read, structured and coherent arguments for design justification and use of OOP, insightful). • Evaluation section of report – outstanding (insightful points made relevant to development (system produced), productivity, errors, learning, and time management; thorough introspection, realistic and insightful reflection).
70-79% ***Excellent***	• Completed all features 1 to 3, implemented to an excellent standard (required features implemented, no obvious bugs, excellent code quality, no duplicate code, OOP concepts and Python language features accurately applied). • Able to show excellent, detailed knowledge and systematic understanding of OOP concepts and Python language features in the code and report, including critical justification of design choices. • Overall report – excellent (required sections completed accurately and clearly, easy to read, well justified design decisions and clear argument, comparative reasoning). • Evaluation section of report – excellent (interesting points made relevant to development (system produced), productivity, errors, learning, and time management; detailed introspection and interesting reflections).
60-69% ***Very Good***	• Completed all features 1 to 3, implemented to a very good standard (required features implemented, few if any bugs, very good code quality, may contain duplicate code, very good attempt applying OOP concepts and Python language features, may contain some design flaws). • Able to show very good knowledge and systematic understanding of OOP concepts and Python language features in code and report, including reasonable critical justification of design choices. • Overall report – very good (required sections completed accurately and clearly, good argument for design justification and OOP use). • Evaluation section of report – very good (very good points made relevant to at least three out of the following matters: development (system produced), productivity, errors, learning, and time management; introspection shows some reasonable thought).
50-59% ***Good***	• Completed all features 1 to 2, implemented to a good standard (at least 3 requirements implemented for feature 3, few if any bugs, contains some duplicate code, good attempt at applying OOP concepts and Python language features, contains some design flaws). • Able to show good knowledge and mostly accurate systematic understanding of OOP concepts and Python language features in code and report, including some rationale for design choices. • Overall report – good (required sections completed accurately, mostly clear, some reasonably balanced argument for design justification and OOP use). • Evaluation section of report – good (addresses points relevant to at least three out of the following matters: development (system produced), productivity, errors, learning, and time management; limited introspection).
45-49% ***Satisfactory***	• Completed all features 1 and 2, implemented to a good standard (required features implemented, few if any bugs, contains some duplicate code, acceptable attempt at applying OOP concepts and

(*Strong 3rd*)	Python language features, contains some design flaws). • Able to show satisfactory knowledge of OOP concepts and Python features in code and report, understanding is less systematic showing unbalanced rationale for design choices. • Overall report – acceptable (required sections completed, mostly accurate and clear, superficial justification for design choices and OOP use). • Evaluation section of report – acceptable (addresses points relevant to at least two out of the following matters: development (system produced), productivity, errors, learning, time management, mostly descriptive; superficial introspection).
40-45% ***Satisfactory*** (*Weak 3rd*)	• Completed all features 1 and 2, implemented to a reasonable standard (at least 2 requirements for each feature implemented with a few minor bugs, contains some duplicate code, acceptable attempt at applying OOP concepts and Python language features, contains some design flaws). • Able to show fairly satisfactory knowledge of OOP concepts and Python language features in code and 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). • Evaluation section of report – acceptable (addresses points relevant to at least one out of the following matters: development (system produced), productivity, errors, learning, time management, mostly descriptive; superficial introspection).
35-39% ***Fail***	• Good attempt at some features although maybe buggy. • Confused knowledge of OOP concepts and Python language features in code and report. Some evidence of systematic understanding showing confused rationale for design choices. • Overall report – mostly completed to an acceptable standard. Some sections are confused. • Evaluation section of report – mostly descriptive but showing some thought.
<35% ***Fail***	• Little or no attempt at features or very buggy. • Not able to show satisfactory knowledge of or systematic understanding OOP concepts and Python language features in code and report. No or little evidence of rationale for design choices. • Overall report – mostly in-completed, at an un-acceptable standard or missing. • Evaluation section of report – descriptive showing a lack of thought or missing.

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 (listed in the requirements section above) that you have successfully implemented, and the quality of their design will have an effect on your overall mark.

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.

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

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. 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?