COMP1811 (2019/20)	**Paradigms of Programming**	**Header ID** 300911	**Contribution:** 40% of module
Module Leader/Moderator Yasmine Arafa/Andy Wicks	Practical Coursework 1 – Python Project		**Deadline Date** Monday 27/01/2020
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 27/01/20.** Submissions must be made using the Coursework link under "Coursework Specification and Upload" on the Moodle page for COMP1811. Your work will be capped if you fail to submit by the deadline.
- For this coursework you must submit a **zip file containing** all the files required to run **your Python project** (including the additional features announced during the hackathon session) **and** 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"). 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.
- 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

Your overall task for this project is to design and develop an **electronic-voting system** in Python which follows the specification below. Implementing the project will give you experience in OOP development and 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 and 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.

Please read the entire coursework specification before starting work.

Scenario

The University of Greenwich Students' Union – GSU are exploring the possibility of installing a new electronic voting system for their next Officer Team elections. GSU want to allow students to vote at dedicated electronic booths during the elections as well as the online system. They would like students to have to log in to the system during election days to cast their votes and to be able to view the results and winners once the election is over. You have been contracted to develop this voting system in Python.

verdict.co.uk/voting-technology-id/

System Specifications and Requirements

Develop an electronic-voting system for casting and counting votes, and visualising the results.

System Specifications

The GSU is led by an Officer Team of elected full-time Officers. The Officer Team consists of one President, three GSU Officers and 16 Faculty Officers (four per Faculty). Elections are generally held with a maximum of four candidates for each of these three positions. Each candidate can run for one position only.

Student voters should be able to cast their vote only during a specified election date period and only once for each of the positions available. Voting at GSU applies the "Single Transfer Vote" - STV method, which means that candidates are ranked in order of preference from 1-4. This way, if there is a tie in the number of votes, the vote moves to the second preference. A candidate wins if they receive the highest number of votes for the position they are running for. If two or more candidates receive the same number of votes, the second preference is considered. So the candidate in the tie with most votes as second preference wins. The process is repeated until there is a clear winner, or all four preferences have been exhausted (in which case the tie holds and a re-election for that position is needed). The voter must rank at least their preferred candidate (i.e. 1). Ranking the remaining three candidates is optional. Lastly, no two candidates can be given the same rank number by an individual voter.

Candidates running for each Officer Team position will be read in from a simple text file of candidates called ***GSUCandidates.txt***. The contents of this file are simply records, on separate lines, each containing the position name followed by the candidate name and separated by a single space or comma (e.g. "President Alan Smith"). Make sure that a candidate runs for one position only and eliminate those who are running for more the one position.

Students eligible to vote are also stored on a simple text file called ***StudentVoters.txt***. The contents of this file are simply records, on separate lines, each containing the student user login ID and password name and separated by a single space or comma. This is obviously not the best way of keeping login details securely, but for simplicity and as this is not a module on security this approach will suffice.

Visualisation of vote results can simply be displayed in tabular form per position as follows. Candidates must be displayed in alphabetic order by position. Graphs or chart plots are not required but are a welcome additional feature.

```
------------------------------------------------------------------------------------------------
Position: GSU President
------------------------------------------------------------------------------------------------
     Candidate        1st Preference     2nd Preference     3rd Preference     4th Preference
     ------------     ----------------   -----------------  -----------------  -----------------
     Alan Smith            80                 50                 23                 18
     Jon Adams             49                 60                 11                 7
     Lizzy Jones           57                 39                 0                  13
     Zia Ahmed             80                 36                 7                  24
------------------------------------------------------------------------------------------------
Winner: Alan Smith
Votes Received: 80
Total votes cast overall: 266
================================================================================
```

System Requirements

Development of the system is expected to be carried out in two parts. <u>Part A</u> is group work to develop the main system requirements and features, and <u>Part B</u> is individual work to develop additional system features specified during a hackathon-type development session on 27/01/2020.

1. **Part A** – **Group Work** (up to 80% of this assessment)

   Development for Part A is expected to be carried out in groups of four. Groups must be formed from within your timetabled lab sessions only. Working in a group means the workload must be equally distributed as much as possible and all members of the team should have an equal understanding of all code produced by the other members in their team.

   The system must provide the following features and comply with the System Specifications outlined. All these features, apart from A.3 and A.5, are frontend features, which means some user intervention is needed to execute them (i.e. by selecting a menu option). A.3 is a backend feature and should be executed by explicit call in your program before all other features. A.5 should be executed before the election results can be displayed.

   A.1 Interactive GUI
   - display a menu for selecting between the frontend system features listed below and include an option to exit the voting system.

   A.2 Login with UID and password
   - use the StudentVoters.txt file to check whether the entered details are correct;
   - if UID does not exist, prompt a message indicating this person is not eligible to vote.

   A.3 Upload Candidates by GSU Position (backend feature)
   - use the ***GSUCandidates.txt*** file to upload the candidates running for each of the GSU positions.

   A.4 Cast Votes

   i) allow the user to cast votes only if the current date is within the specified election date period;

   ii) allow the user to select a GSU position, then for each selected position:
      - display the candidates running for the selected position; and allow the user to enter their vote by ranking these candidates in order of preference;
      - make sure to check for any duplication in rank numbers entered and that numbers are from 1-4 only;
      - once the vote has been entered for a particular position, allow the user to save their voting entry to the ***Votes.txt file***;

iii) allow the user to repeat this process for all the GSU positions;
iv) make sure that an individual user can cast their vote only once for each of the positions available

A.5 Count Votes (backend feature)

- count the total number of votes cast for each candidate running for each GSU position;
- find the wining candidate for each position based on the total votes they received;
- if there is a tie, use the preference votes to determine the winner.

A.6 Visualise Results

- allow the user to select a GSU position, then for each selected position:
  - display all the candidates and the number of votes received by order of preference;
  - display the outcome of the election – the winner; and
  - display the number of votes the winner received, the total number of votes cast overall as well as the percentage of the total votes received by the winner.
- display a summary showing the winning candidate for each of the GSU positions on one screen;
- output the summary of results to a text file so that it can be printed.

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. It is recommended that you approach your development by first thinking of the classes needed. There are obvious entities in this system that can be well abstracted and designed as classes in your project (e.g. Candidates, Voters, Votes, etc.). Think about the attributes and behaviours (methods) needed for each of these classes. Make sure your classes apply appropriate encapsulation. Try to use at least one appropriate inheritance hierarchy consisting of at least three classes. All three classes should be ones declared in the program and not classes from the Pythons libraries.

2. **Part B – Individual Work** (up to 20% of this assessment)

A set of additional features will be announced during a hackathon-type session held during your scheduled lab on 27/01/20. **Each member of your group will implement one of these features**. The code produced should build on top of and must integrate with the code already produced as a group for Part A.

The code must also integrate with the additional features produced by each member of your group. Your **integrated Python project must be uploaded to Moodle by the end of your timetabled lab session on 27/01/20**. It is **ESSENTIAL** that you complete the work required for Part A **BEFORE** coming to the hackathon session. Each member is expected to have FULL understanding of the entire code developed for Part A and the rationale behind its design. **You will fail the coursework if you do not attend the hackathon session - unless you have submitted a valid EC.**

The code produced in parts A and B should implement all the features 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. **Part B Python Code** – A zip file containing all the files required to run your PyCharm project that includes the integrated code produced for Part B and Part A of the requirements. **Code created by each group member must be clearly labelled with their name and student ID** in a comment. Name your Zip file as follows: <YourStudentID>_CW1PartB_COMP1811_1920. Each group member will individually upload an identical zip containing both the group code and the integrated additional features. Please try to structure your work so that it is easy for the person marking your work to run your project. Upload this zip file to Moodle to the **Part B Upload** link in the **Coursework Details and Submission** block **BY THE END OF YOUR LAB SESSION on Monday 27/01/20. You will lose marks if you do not attend the hackathon session or upload your code for Part B**.

2. **Acceptance test** (demonstration) of what you have achieved. You will be asked to demonstrate:
   - your system running,
   - what you implemented for the additional features at the end of the hackathon-type session and
   - how and where you integrated your work with the existing code.

   The demonstration will take approximately 10 minutes. As well as demonstrating your application running, you will be asked about your code and expected to show thorough understanding of it. Each member will be expected to demonstrate a good understanding of the code and the rationale behind its design.

   You are required to attend two demonstration sessions, each scheduled during your timetabled lab sessions:
   1. An **interim demonstration** on **09/12/19 ~~20~~** to demonstrate a working version of the system, which includes your development for A.3, A.4.i and at least the first two features of A.4.ii.
      For the interim demo, a GUI implementation is not required. Simply use the PyCharm console to input and display the information required for feature A.4.ii. We are most interested in getting the functionality right at this stage. The GUI is however expected for your final submission.

   2. The **final acceptance test** on **27/01/20** where you will demonstrate the additional features implemented during the hackathon type session. These features must be demoed integrated with the code you will have already produced for Part A.

3. **Complete Python Code and Report** – This is the **final deliverable**. After submitting the work for the hackathon session, you have the opportunity to further work on Part A and to finalise your report. Upload a zip file containing the final PyCharm project (integrated code for Parts A and B) and a **PDF version of your report** to Moodle to the **Final Upload** link under the **Coursework Details and Submission** block **by Monday 27/01/20 by 23:30 at the latest.** Name your Zip file as follows: <YourStudentID>_CW1ProjectFinal_COMP1811_1920.

   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.

   The report should consist of **all** the sections described in the coursework report template provided. 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 the coursework if you do not upload your report or final code**.

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. Features implemented. (15)
2. Use OOP techniques. (30)
3. Quality of code. (20)
4. Documentation. (10)
5. Testing. (5)
6. Objective evaluation of work. (5)
7. Acceptance tests (demonstrations) (15 - interim demonstration (5) and final acceptance test (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.

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.

> 80% ***Exceptional***	• Completed features A.1-A.6 and B 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 at the acceptance test demonstration and in the report, including critical discussions of design choices and possible alternative implementation choices. • Group worked well together to achieve objectives. Well organised and each member made equally significant contribution to the project. • Overall report – outstanding (required sections completed accurately and clearly, easy to read, structured and coherent arguments for design justification and use of OOP, insightful). • Section 8 of report (evaluation) – 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 features A.1-A.6 and B 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 at the acceptance test demonstration and in report, including critical justification of design choices. • Group worked well together to achieve objectives. Well organised and each member made equally significant contribution to the project. • Overall report – excellent (required sections completed accurately and clearly, easy to read, well justified design decisions and clear argument, comparative reasoning). • Section 8 of report (evaluation) – 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 features A.1-A.6 and B 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 at the acceptance test demonstration and in report, including reasonable critical justification of design choices. • Group 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 group meetings organised and attended to achieve objectives. • Overall report – very good (required sections completed accurately and clearly, good argument for design justification and OOP use). • Section 8 of report (evaluation) – 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 features A.1-A.5 and B implemented to a good standard (required features implemented, 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 at the acceptance test demonstration and in report, including some rationale for design choices. • Group 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 group meetings held or attended to achieve objectives. • Overall report – good (required sections completed accurately, mostly clear, some reasonably balanced argument for design justification and OOP use). • Section 8 of report (evaluation) – 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 (Strong 3rd)***	• Completed features A.1-A.4 and B implemented to a good standard (at least 3 required features implemented for A.4, few if any bugs, contains some duplicate code, acceptable attempt at applying OOP concepts and Python language features, contains some design flaws). • 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 worked together some of the time, with some occurrences of communication breakdown or failure to collaborate when appropriate. Few group meetings organised or attended. • Overall report – acceptable (required sections completed, mostly accurate and clear, superficial justification for design choices and OOP use). • Section 8 of report (evaluation) – 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 features A.1-A.4 and B implemented to a reasonable standard (at least 2 required features 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 at the acceptance test demonstration and in report, understanding is less systematic showing little rationale for design choices. • Group did not collaborate or communicate well. Members often worked independently, with weak regard to objectives or priorities. Very few group meetings organised or attended. • Overall report – acceptable (required sections completed, mostly accurate, clumsy language, descriptive account of design choices and OOP use).

	• Section 8 of report (evaluation) – 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 at the acceptance test demonstration and in report. Some evidence of systematic understanding showing confused rationale for design choices. • Group did not collaborate or communicate well. Members mostly worked independently, without regard to objectives or priorities. Very few group meetings organised or attended. • Overall report – mostly completed to an acceptable standard. Some sections are confused. • Section 8 of report (evaluation) – 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 at the acceptance demonstration and in report. No or little evidence of rationale for design choices. • Group did not collaborate or communicate. Members worked independently, without regard to objectives or priorities. No meetings attended. • Overall report – mostly in-completed, at an un-acceptable standard or missing. • Section 8 of report (evaluation) – 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. They apply to all levels in Part A as well as Part B.

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.

Additional feature implemented. Are the additional features (announced during the hackathon-like session) implemented and running correctly? Emphasis will be on design, functionality and **integration** with the code produced in Part A.

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 the vote results does not need to use graphics, simply present this in tabular form with appropriate spacing.

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. Acceptance test

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. 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?