COMP1828	**Advanced Algorithms and Data Structures**	**Faculty Header ID:**	**Contribution: 100% of course**
Module Leader: Dr IK SOO LIM	**Designing, developing & testing a journey planner**	**Deadline Date: 5.00 pm 26th Nov 2025, UK (GMT)**	**Feedback return time: 17th Dec 2025, UK (GMT)**
This coursework should take an average student who is up-to-date with tutorial work approximately 30 hours Feedback and grades are normally made available within 3 calendar weeks of the coursework deadline (not counting holidays)			
Learning Outcomes: 1 Select and employ data structures and algorithms appropriate to a variety of problems. 2 Formulate models using appropriate algorithms and data structures. 3 Obtain programmatic solutions using appropriate software, including a high level programming language. 4 Describe and discuss the efficiency, complexity, accuracy and limitations of algorithms.			

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.

Your work will be submitted for 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 on or before the deadline date using the link on the coursework Moodle page for COMP1828.
- For this coursework you must submit a single PDF document. 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"). For mathematical notation, you can use MS Word equation tools

https://support.microsoft.com/en-us/office/write-an-equation-or-formula-4f799df7-4ca4-4670-afd3-6135768b01d0 .

- For this coursework you must also upload the source code and any additional supporting work as a single **ZIP** file.
- There are limits on the file size (see the relevant course Moodle page).
- 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 in any other form by the 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

- **Grading Criteria**

Each member must make a significant individual contribution to the technical development work as well as contributing to the overall team effort. The allocation of marks will reflect the quality of the work produced and **will be based on the breakdown provided by the team**. For example, in a team of three members A, B, and C if the breakdown provided by the team is A=100%, B=100%, and C=50%. Then for an overall project mark of 60% this will mean that A is awarded 60%, B is awarded 60%, and C is awarded 30%. If the team do not provide any breakdown or if any nonsensible breakdown is provided by the team, then the marker will allocate the same mark to each team member.

Criteria for Assessment	80-100	70-79	60-69	50-59	40-49	30-39	0-29
Content, knowledge and understanding	Demonstrates exceptional systematic understanding of problem solving, computer programming and algorithmic performance. There is exceptional evidence of engagement with all key elements.	Demonstrates an excellent systematic understanding of problem solving, computer programming and algorithmic performance. There is also excellent evidence of engagement with all key elements.	There is a very good systematic understanding of problem solving, computer programming and algorithmic performance. There is also some very good evidence of engagement with all key elements.	Has demonstrated a good understanding problem solving, computer programming and algorithmic performance. There is also some good evidence of engagement with most key elements with some omission of detail.	Has demonstrated a satisfactory level of understanding of problem solving, computer programming and algorithmic performance. There are a few notable omissions and there is limited evidence of engagement with all key elements. Overall a satisfactory attempt at this criteria.	A poor understanding of one or more of the following - problem solving, computer programming and algorithmic performance. There is insufficient evidence of engagement with the key elements. Overall an unsatisfactory attempt.	Little or no understanding of one or more of the following - problem solving, computer programming and algorithmic performance. There is very little evidence of engagement with the key elements. Overall a very unsatisfactory attempt.

Cognitive/Intellectual Skills	Demonstrates exceptional use of a critical analysis of information leading to the proposal of a robust and detailed solution. There is exceptional evidence of reflection that identifies the strengths and weakness of the approaches undertaken.	Demonstrates an excellent use of a critical analysis of information leading to the proposal of a robust and detailed solution. There is also excellent evidence of reflection and judgement based on the interpretation of the results obtained.	Demonstrates a very good use of a critical analysis of information leading to the proposal of a detailed solution. There is also some very good evidence of reflection and judgement based on the interpretation of the results obtained.	Demonstrates some good critical analysis of information leading to the proposal of a detailed solution. There are some exposed weaknesses of cognitive skills. There is also some good evidence of reflection and judgement based on the interpretation of the results obtained.	Has shown some satisfactory level of critical analysis of information. There is evidence of reflection and judgement based on the interpretation of the results obtained at a threshold pass level.	Has shown little use of techniques to undertake a critical analysis of information. The reflection and judgement based on the interpretation of results is weak and lacks detail.	Has shown little or no use of techniques to undertake a critical analysis of information. The reflection and judgement based on the interpretation of results is very weak and lacks detail.
Communication, Organisation and Presentation **Graduate Employability and Application of Skills**	Demonstrates exceptional use of argument and language which effectively communicates information to the target audience. The structure and flow of the report is clear and of an exceptional quality. There is exceptional evidence of the qualities of transferrable skills necessary for employment that required personal judgement and successful experimentation.	Demonstrates excellent use of argument and language which effectively communicates information to the target audience. The structure and flow of the report is clear and of an excellent quality. There is excellent evidence of the qualities of transferrable skills necessary for employment that required personal judgement and successful experimentation.	Demonstrates a very good use of argument and language which effectively communicates information to the target audience. The structure and flow of the report is clear and overall is very good. There is also very good evidence of the qualities of transferrable skills necessary for employment that required personal judgement and mostly successful experimentation.	There is good use of argument and language which communicates information to the target audience. The structure and flow of the report is mostly coherent and overall is good. There is also some good evidence of the qualities of transferrable skills necessary for employment.	The use of argument and language which communicates information to the target audience is mostly acceptable with some shortcomings in the grammar. The structure and flow of the report is barely acceptable with some presentation issues. There is also some evidence of the qualities of transferrable skills necessary for employment.	The use of argument and language which communicates information to the target audience is mostly at a substandard level. The structure and flow of the report is unacceptable with some presentation issues. There may also be little evidence of the qualities of transferrable skills necessary for employment.	The use of argument and language which communicates information to the target audience is at a substandard level. The structure and flow of the report is unacceptable with significant presentation issues. There may also be little/no evidence of the qualities of transferrable skills necessary for employment.

Referencing, sourcing, acknowledging and coverage	The exceptional use of appropriate references reflects clear and detailed understanding of the referenced works and its contents from a variety of sources.	The excellent use of appropriate references reflects clear and detailed understanding of the referenced works and its contents referenced works.	The use of references reflects a very good understanding of the cited work and its contents. Some references may not be the most recent.	The use of references reflects a good understanding of the cited work and its contents. Some references may not be the most recent or are taken from a narrow range of sources.	The use of references reflects a satisfactory understanding of the cited work and its contents. Some references may not be appropriate or the most recent or are taken from a narrow range of sources.	The use of references reflects a poor understanding of the cited work and its contents. The references may not be sufficient or appropriate or the most recent or are taken from a narrow range of sources.	Little or no cited work. The references may not be appropriate or the most recent.

Designing, developing and testing solutions for the London Underground system

Group work:

This coursework requires collaborative group efforts, and effective communication among group members to be successful. The demonstration of teamwork accounts for up to 20% of the overall coursework grade. This portion of the grade can be achieved relatively easy, regardless of the actual coursework completion. Securing this 20% could be crucial for students on the threshold of passing.

Documentation of weekly communications between group members is required, including:

1. Weekly email exchanges
2. A record of a weekly Teams meeting
3. Details of each member's cumulative contribution percentage

If a member's cumulative contribution credit reaches 100%, that member's individual grade will match the group's overall grade. Members with contributions under 100% will receive a proportional percentage of the group grade.

Any disagreements concerning contribution percentages or other issues should be resolved within the group. The resolution process may be included in the final report if necessary.

Further instructions and guidelines will be provided in due course.

Data:

The London Underground comprises an extensive network of stations across Greater London. A standard map of this system is available in PDF format from the Transport for London (TfL) website at https://tfl.gov.uk/maps/track/tube . Accompanying this is an Excel spreadsheet titled "London Underground Data.xlsx", which contains the journey times in minutes between adjacent stations, based on previously collected data. Please note:

1. While this data set may contain errors or omissions, you should assume it to be correct for the purposes of this assignment.
2. The provided times do not include waiting periods at stations, or the time taken for passengers to board or alight from trains.

Mandatory Use of Library Code:

• For specific subtasks outlined later in the coursework specification, you must use the Python library code available at https://mitp-content-server.mit.edu/books/content/sectbyfn/books_pres_0/11599/clrsPython.zip .
• For example, if your application code requires a binary search algorithm, you must use or call the binary search function from this library code only.
• This library is structured according to the chapter numbers from one of the required course textbooks, "Introduction to Algorithms" (4th edition). You can easily navigate and identify code for specific algorithms within the library using the book content, whose online copy is available from the library (https://ebookcentral.proquest.com/lib/gre/detail.action?docID=6925615) or its table of contents (https://mitp-content-server.mit.edu/books/content/sectbyfn/books_pres_0/11599/4e_toc.pdf), which lists chapters and sections.
• For these subtasks, using any other library code, including code you've written yourself, is strictly prohibited.
• Please note: Failure to comply with this requirement may result in your maximum score being limited to 50%.
• This mandatory use is a result of the School's directive to limit potential AI misuse and other challenges in coursework design. Without this requirement, the coursework specification would not be approved during the moderation process.
• In professional software development, using pre-existing libraries for data structures and algorithms is common practice, often resulting in faster and more reliable outcomes. Your focus should be on selecting the most appropriate data structures and algorithms for the project at hand.

A Note on Group Collaboration and Project Integration

Given the size of your group and the scope of this project, different members may find it efficient to work on different tasks in parallel. While this independent work can speed up development, a key consideration for a successful project may be how the separate parts are ultimately integrated.

To help facilitate this process, it is recommended that your group agrees on common data formats and interfaces *early* in the process. For instance, the member(s) working on Task 2 (Journey Planner) would benefit from knowing how the network data will be represented and accessed. Your group may find it useful to decide on a standard format for your network data structure (e.g., how to represent stations and connections). This allows members working on later tasks to write their code assuming this standard, even before the data parsing for the full network is complete.

This approach can allow for independent development while helping to ensure that the different parts of your project can be integrated smoothly for the final submission. Agreeing on how the code will connect early on is a common strategy for managing collaborative projects.

TASK 1:

Operational Station Status System

Your group is tasked with designing and evaluating a system to track the real-time operational status of stations in the London Underground network. For a large transport network, this set of 'operational' stations changes frequently due to maintenance or incidents, requiring a system that can handle dynamic updates to its list of station names efficiently.

The primary function of your system is to answer one key query with maximum speed: "Is station *X* currently operational?". Given a station name as input, your system must determine if that name is present in the current set of operational stations and return a clear status indicating whether it is operational or not.

(1a) Manual versus Code-Based Execution of a Data Organisation Strategy:
[For this specific subtask (1a), you must use the previously mentioned Python library by calling its functions to implement the core mechanics of your chosen data structure.]

- **Data Structure Selection:** To manage the daily list of operational stations, the data structure must be efficient for frequent updates and fast status checks. Your choice must demonstrate strong performance for **insertion, deletion, and membership testing (i.e., checking if a given station name is present in the set)**. Select a single data structure and justify your choice based on its theoretical performance for all three of these operations.
- **Create a Simple Dataset:** Create a simple (and artificial) list of 5 "operational" station names for a given day (e.g., 5 stations named A, B, C, D, and E; or represented by integers 0, 1, 2, 3, and 4).
- **Manual Application:** Manually trace the process of **populating** your chosen data structure with each of the 5 station names from your simple dataset. After each addition, you must **clearly represent the internal state** of the structure (e.g., visually or textually). Then, manually trace the steps required to **check the status** of one specific station name (i.e., determine if it is present) within your final, populated structure.
- **Code Implementation and Verification:**
    - Implement a Python **programme** that builds this small data structure using the library, populating it with the same 5 station names.
    - Execute a status check for the same station you traced manually.
    - Verify that the final internal state of the structure from your manual trace is consistent with the one created by your code, and that the status check process is analogous. Explain any discrepancies.

(1b) Empirical Measurement and Application

[For this specific subtask (1b), you must reuse the code from the previous subtask (1a) as much as possible]

- **Empirical Performance Measurement:**
  - **Data Generation:** Develop or source Python code to generate artificial datasets of *n* unique "operational station names". For simplicity and to ensure uniqueness, you can represent these station names as integers from 0 to *n*-1.
  - **Time Measurement:** For a given dataset of size *n*, build the data structure and measure the **average time per status check** over a large number of random queries.
  - **Performance Analysis:** Perform this time calculation for datasets of different sizes: e.g., *n* = 1000, 5000, 10000, 25000, 50000.
  - **Plotting and Comparison:** Plot a graph of average time per check versus dataset size *n*. Compare this empirical graph with the **theoretical time complexity for this operation that you discussed in your justification in subtask (1a)**. Assess their alignment and explain any discrepancies.
- **Application with London Underground Data:**
  - **Data Acquisition and Implementation:** Obtain a complete and unique list of all station names from the provided London Underground Data.xlsx file. You may use any method to do this, such as writing your own parsing script, manually copying the data, or using an external application. **You must briefly describe your chosen method in your report.** Once you have this list, populate an instance of your chosen data structure (from subtask 1a) with all of these real station names. For this part, assume all stations in the dataset are "operational".
  - **Testing:** Demonstrate that your system works by performing status checks that test for both the presence and absence of a station name. To ensure authenticity, instead of typing the results, your report must include **screenshots of your programme's actual output** for these tests. The screenshots must clearly show the input (the station name being tested) and the corresponding output from your system. Provide screenshots for at least three queries in total, ensuring you test both a **successful lookup** for a valid station name (e.g., 'Victoria') and an **unsuccessful lookup** for a name that is not in the dataset, such as a misspelled station name (e.g., 'Paddinton') or a fictional one. The output shown in your screenshots should clearly indicate the status (e.g., 'Operational' or 'Not Found').

Task 2:

Journey Planner Based on Journey Time

Your group is tasked with developing a Python programme for the London Underground tube system's route planner. It must be capable of determining the most efficient route between any two stations, where efficiency is defined by the shortest possible journey duration in minutes.

Given a pair of starting and destination stations, your programme should provide two key outputs:

1. A detailed, ordered list of stations indicating the shortest journey from start to destination.
2. The total duration of that journey in minutes.

(2a) Manual versus Code-Based Execution of a Shortest Path Algorithm:

[For this specific subtask (2a), you must use the previously mentioned Python library by calling its functions to implement the core mechanics of your chosen data structure.]

- **Data Structure and Algorithm Selection:** To find the shortest path, you must first choose a suitable data structure to represent the network of stations and the connections between them. You must also select an appropriate algorithm for finding the shortest path between two points in a network where connections have associated journey times. Justify your choices for both the data structure and the algorithm based on their theoretical properties and suitability for this task.
- **Create a Simple Dataset:** Create a simple (and artificial) dataset representing a tube network with only a few stations (e.g., 5 stations named A, B, C, D, and E; or represented by integers 0, 1, 2, 3, and 4) and assign journey durations in minutes for the connections between adjacent stations.
- **Manual Application:** Manually apply your chosen shortest path algorithm to your simple dataset. Choose a pair of stations and determine the shortest path in terms of total journey duration, executing the algorithm step-by-step using pen and paper and documenting the process.
- **Code Implementation and Verification:**
    - Implement a Python **programme** that builds this small data structure and finds the shortest path using the library.
    - Execute a shortest path search for the same pair of stations you traced manually.
    - Verify that the path and total duration from your manual computation and the code-based execution produce identical outcomes. Explain any discrepancies.

(2b) Empirical Measurement and Application

[For this specific subtask (2b), you must reuse the code from the previous subtask (2a) as much as possible]

- **Empirical Performance Measurement:**
  - **Data Generation:** Develop or source Python code to generate artificial tube network datasets with *n* stations and connections between them that have associated journey durations.
  - **Time Measurement:** For a given network of *n* stations, employ your Python code to find the shortest path between two randomly selected stations. Measure the **average time per shortest path calculation** over a large number of random pairs.
  - **Performance Analysis:** Perform this average time calculation for networks of different sizes: e.g., *n* = 100, 200, ..., 900, and 1000.
  - **Plotting and Comparison:** Plot a graph of average time per calculation versus network size *N*. Compare this empirical graph with the **theoretical time complexity for this operation that you discussed in your justification in subtask (2a)**. Assess their alignment and explain any discrepancies.
- **Application with London Underground Data:**
  - **Data Acquisition and Implementation:** Using the station names and connections from the London Underground Data.xlsx file, build a complete representation of the network using the data structure you selected in subtask (2a). Your chosen algorithm may require that there is at most one direct connection between any two stations. Therefore, before running your algorithm, you must simplify any multiple links found in the data into a single connection (e.g., by keeping the one with the minimum journey time).
  - **Testing:** Demonstrate that your route planner works by finding the shortest path for at least two different journeys on the real network to test its correctness on different scales. Your report needs to include tests for both a **short journey** (e.g., 'Covent Garden' to 'Leicester Square') and a **longer journey that spans a significant distance across the network** (e.g., 'Wimbledon' to 'Stratford'). To ensure authenticity, your report needs to include **screenshots of your programme's actual output** for these tests. The screenshots must clearly show the input stations and the resulting path and total journey time.

TASK 3:

Journey Planner Based on Number of Stops

Your group is tasked with developing an alternative version of the Python programme for the London Underground tube system's route planner. It must be capable of determining the route with the fewest number of stops between any two stations, which represents the simplest journey in terms of stations visited.

Given a pair of starting and destination stations, your programme should provide two key outputs:

1. A detailed, ordered list of stations indicating the journey with the fewest stops.
2. The total number of stops for that journey.

(3a) Manual versus Code-Based Execution of a Fewest Stops Algorithm:
[For this specific subtask (3a), you must use the previously mentioned Python library by calling its functions to implement the core mechanics of your chosen data structure.]

- **Data Structure and Algorithm Selection:** To find the path with the fewest stops, you must first choose a suitable data structure to represent the network of stations and the connections between them. You must also select an appropriate algorithm for finding the shortest path where the "length" of the path is measured by the number of connections. Justify your choices for both the data structure and the algorithm based on their theoretical properties and suitability for this task.
- **Create a Simple Dataset:** Create a simple (and artificial) dataset representing a tube network with only a few stations (e.g., 5 stations named A, B, C, D, and E; or represented by integers 0, 1, 2, 3, and 4) and the connections between adjacent stations.
- **Manual Application:** Manually apply your chosen algorithm to your simple dataset. Choose a pair of stations and determine the path with the fewest number of stops, executing the algorithm step-by-step using pen and paper and documenting the process.
- **Code Implementation and Verification:**
    - Implement a Python **programme** that builds this small data structure and finds the path with the fewest stops using the library.
    - Execute a search for the same pair of stations you traced manually.
    - Verify that the path and total number of stops from your manual computation and the code-based execution produce identical outcomes. Explain any discrepancies.

(3b) Empirical Measurement and Application

[For this specific subtask (3b), you must reuse the code from the previous subtask (3a) as much as possible]

- **Empirical Performance Measurement:**
  - **Data Generation:** Develop or source Python code to generate artificial tube network datasets with *n* stations and connections between them.
  - **Time Measurement:** For a given network of *n* stations, employ your Python code to find the path with the fewest stops between two randomly selected stations. Measure the **average time per calculation** over a large number of random pairs.
  - **Performance Analysis:** Perform this average time calculation for networks of different sizes: e.g., *n* = 100, 200, ..., 900, and 1000.
  - **Plotting and Comparison:** Plot a graph of average time per calculation versus network size *n*. Compare this empirical graph with the **theoretical time complexity for this operation that you discussed in your justification in subtask (3a)**. Assess their alignment and explain any discrepancies.

- **Application with London Underground Data:**
  - **Data Acquisition and Implementation:** Using the station names and connections from the London Underground Data.xlsx file, build a complete representation of the network. Your chosen algorithm may require that there is at most one direct connection between any two stations. Therefore, before running your algorithm, you must simplify any multiple links found in the data into a single connection.
  - **Testing:** Demonstrate that your route planner works by finding the path with the fewest stops for the **same two journeys** you tested in Task 2 (e.g., 'Covent Garden' to 'Leicester Square' and 'Wimbledon' to 'Stratford'). To ensure authenticity, your report must include **screenshots of your programme's actual output** for these tests. The screenshots must clearly show the input stations and the resulting path and total number of stops. In your report, compare these paths to the ones you found in Task 2 and analyse any differences.

TASK 4:

Analysis of the Core Network Backbone

Your group is tasked with analysing the core infrastructure of the London Underground. In a complex network, some connections are essential to keep the system connected, while others provide valuable alternative routes. Imagine a scenario where, for major engineering works, as many line sections as possible need to be closed temporarily without isolating any stations. The specific operational requirement is to identify the set of line sections that can be closed while adhering to one critical rule: travel between any two stations in the network must remain possible, even if the journey becomes longer. Your task is to find this maximum set of closable sections, which by extension identifies the essential "backbone" network that must be kept open.

(4a) Manual versus Code-Based Execution of a Core Network Algorithm:
[For this specific subtask (4a), you must use the previously mentioned Python library by calling its functions to implement the core mechanics of your chosen data structure.]

- **Algorithm Selection:** To solve the problem of identifying the maximum number of closable connections while maintaining full network connectivity, you will need to select an appropriate algorithm. Justify your choice of algorithm based on its theoretical properties and its suitability for solving this specific problem.
- **Create a Simple Dataset:** It is recommended you reuse the simple, weighted dataset from Task 2(a).
- **Manual Application:** Manually apply your chosen algorithm to your simple dataset. Document the process step-by-step, showing how connections are evaluated and a final set of essential, open connections is determined.
- **Code Implementation and Verification:**
    - Implement a Python **programme** that builds the small data structure and identifies the core network backbone using the library.
    - Execute the algorithm on the same dataset you traced manually.
    - Verify that the set of connections in your manually computed backbone and the code-based version are identical. Explain any discrepancies.

(4b) Empirical Measurement and Application

[For this specific subtask (4b), you must reuse the code from the previous subtask (4a) as much as possible]

- **Empirical Performance Measurement:**
  - **Data Generation:** Using your data generator, create artificial network datasets with *n* stations and weighted connections.
  - **Time Measurement:** For a given network of *n* stations, measure the **average time required to compute the core network backbone**.
  - **Performance Analysis:** Perform this time calculation for networks of different sizes: e.g., *n* = 100, 200, ..., 900, and 1000.
  - **Plotting and Comparison:** Plot a graph of average time versus network size *n*. Compare this empirical graph with the **theoretical time complexity for this operation that you discussed in your justification in subtask (4a)**. Assess their alignment.

- **Application with London Underground Data:**
  - **Core Backbone Calculation:** Using the simplified, weighted network representation from the previous tasks, run your implementation to find the core backbone of the real London Underground network. This creates a new, "reduced" network consisting only of the essential connections.
  - **Testing and Impact Analysis:** Demonstrate your system works by analysing the results. Your report needs to include the following, using **screenshots of your programme's actual output** for authenticity:
    1. The **total journey time (total weight)** of the final core network backbone.
    2. A list of **at least 10 "redundant" connections**—those present in the original network but **not** part of the essential backbone (i.e., the connections that can be closed).
    3. An **impact analysis**: Choose one of the longer journeys you calculated in Task 2. Check if its path uses any of the "redundant" connections you identified. If it does, re-calculate the shortest path for that same journey on the "backbone-only" network. In your report, show both the original and the new path. Analyse the difference in journey time and discuss what this reveals about the role of redundant connections in a real-world transport network (e.g., providing efficiency, shortcuts, or resilience).

DELIVERABLES

Deliverable 1: PDF Report

- Use the provided template: GroupX_ID1_ID2_ID3_ID4_ID5.doc.

- Replace 'X' in the filename with your group number and 'ID1', 'ID2', etc. with your group members' student ID numbers.

- Save or export the report as a PDF before submission. This PDF will be your main coursework document.

The report should adhere to the structure of the supplied template. Refer to the template for specific content requirements. The marking scheme is as follows:

- Task 1: 20 marks
- Task 2: 20 marks
- Task 3: 20 marks
- Task 4: 20 marks
- Weekly progress journal: 20 marks

Total: 100 marks

The weekly progress journal should include dated entries, evidence of regular group communication, and other relevant details; see the report template

Properly cite any content in your report that is not your own original work. A reference list alone is insufficient; you must clearly indicate within the text where and how each reference is used. **Failure to properly cite sources may result in an academic misconduct investigation.**

Additional coursework guidance may be provided separately.

Deliverable 2: Well-Commented Python Source Code

- Please upload the fully functional source code as a ZIP file.
- Ensure the code is well-commented and self-contained, meaning it should run without any additional downloads.
- After unzipping, executing the main code files should produce the outputs mentioned in the report.
- **The ZIP file for Deliverable 2 should be uploaded separately from the report.**
- Note: **Failing to submit the source code might limit your maximum score to 50%**.

Your solution should be crafted using the Python language. Where the usage of the library code is not mandatory, you are free to use supplementary resources. However, always credit any resource not created by your team. **Failing to cite external materials might lead to an academic misconduct investigation.**

It's highly recommended to start this coursework promptly once it is accessible. Should any instructions be unclear, reach out to the Module Leader. You can use email, visit during office or lab hours, or schedule an appointment at your earliest convenience.