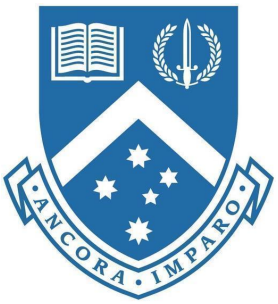

FIT9136 Algorithms and Programming Foundations in Python

Assignment 2

Last Updated: 26 August 2023

Table of Contents

1. Key Information

Purpose	This assignment will develop your skills in designing, constructing, testing, and documenting a Python program according to specific programming standards. This assessment is related to the following learning outcome (LO): ● **LO2 -** Restructure a computational program into manageable units of modules and classes using the object-oriented methodology ● **LO3 -** Demonstrate Input/Output strategies in a Python application and apply appropriate testing and exception handling techniques
Your task	This assignment is an individual task where you will work independently. Your task is to develop a simple car purchase advisor system using Python code, based on the provided specifications.
Value	**45**% of your total marks for the unit.
Due Date	**Friday, 22 September 2023, 4:30 PM (AEST)**
Submission	● Via Moodle Assignment Submission. ● FIT GitLab check-ins will be used to assess the history of development ● JPlag will be used for similarity checking of all submissions.
Assessment Criteria	This assessment includes a compulsory interview with your tutor following the submission date. At the interview you will be asked to explain your code/design/testing, modify your code, and discuss your design decisions and alternatives. Marks will not be awarded for any section of code/design/functionality that you cannot explain satisfactorily. Failure to attend the interview will result in your assessment not being marked. You will be provided with the timing of the interviews at a later date. The following aspects will be assessed: 1. Program functionality in accordance to the requirements 2. Code Architecture and Adherence to Python coding standards 3. The comprehensiveness of documented code
Late Penalties	● 10% deduction per calendar day or part thereof for up to one week ● Submissions more than 7 calendar days after the due date will receive a mark of zero (0) and no assessment feedback will be provided.
Support Resources	See Moodle Assessment page and Section 8 in this document
Feedback	Feedback will be provided on student work via ● general cohort performance ● specific student feedback ten working days post submission

2. Instruction

For this assignment, your goal is to create a **Car Purchase Advisor System** that utilises files "stock.txt" and “order.txt” located in the "data" folder. The application will feature a text interface to act as the advisor system for a car retailer. The functionalities include:

Functionality 1:

When the program starts, test data needs to be automatically generated. The generated test data needs to be reflected in the “stock.txt” file and also loaded in the program memory. For the format of the test data, you can refer to the provided example “stock.txt”. Please ensure that the test data presents a certain degree of randomness.

Functionality 2:

After generating the test data, a menu should be presented to the user with the following options:

a) Look for the nearest car retailer
b) Get car purchase advice
c) Place a car order
d) Exit

Functionality 3:

If the user selects the option for checking the nearest car retailer, the program should ask for the user’s postcode to identify the nearest car retailer (here we assume that suburbs with smaller differences in postcodes are geographically closer, for more information please refer to section 2.3.6).

Functionality 4:

If the user selects the option for getting car purchase advice, the program needs to:

a) First list all the available car retailers and prompt the users to select one to continue the advising process;
b) Then show a sub-menu with the following options:
    i) Recommend a car
    ii) Get all cars in stock
    iii) Get cars in stock by car types (the car types is a list of strings, e.g., [“AWD”, “RWD”])
    iv) Get probationary licence permitted cars in stock
c) The option i) should randomly select one car from the current stock of the car retailer, the options ii) to iv) should show the current stock of a car retailer. You need to first print the retailer ID and the retailer name, and also ensure that the printed stocked cars are well-formatted.

Functionality 5:

If the user selects the option for placing a car order, the program should ask for the retailer ID and the car ID from the user separated by a space. However, before placing the order, the program needs to check whether the current time is within the business hours of the target retailer. If not, an order should not be created and a message should be prompted to the user. For any other invalid cases, you need to handle them properly. Otherwise, in valid cases, an order should be created and stored in "order.txt" and the order details should be printed to the user.

Please note that the program will continue to run until the user chooses to exit the application.

To develop this application, you will need to create four classes, each defined in a separate Python file (i.e. ***.py**), and one main Python file to execute the program. All the required files can be found in the **RENAME_ME.zip** file, which is located in the Assessments section on Moodle. **It is important to note that the addition of supplementary classes exceeding the prescribed four may incur a penalty in the form of mark deduction.**

2.1. Car Class

<table>
<tr><td colspan="2">Contains all the operations related to a car.</td></tr>
<tr><td>Required Class Variables</td><td>● N/A (You can add if you need)</td></tr>
<tr><td>Required Methods</td><td>
<table>
<tr><td colspan="2">2.1.1. __init__()
Constructs a car object.</td></tr>
<tr><td>Arguments</td><td>● car_code (must be unique and in the format of two uppercase letters plus 6 digits, e.g., MB123456)
● car_name
● car_capacity (Each car has a maximum seating capacity)
● car_horsepower (in kilowatts)
● car_weight (in kilograms) - the car_weight is the tare weight of the vehicle (the weight of an empty standard vehicle with all of its fluids and specifically 10 litres of fuel in the tank)
● car_type (values should be one of “FWD”, “RWD” or “AWD”)</td></tr>
<tr><td>Returns</td><td>● N/A</td></tr>
</table>
*All arguments of the constructor must have a default value.
<table>
<tr><td colspan="2">2.1.2. __str__()
Return the unit information as a formatted string.</td></tr>
<tr><td>Arguments</td><td>● N/A</td></tr>
<tr><td>Returns</td><td>● a string in the following format:
"car_code, car_name, car_capacity, car_horsepower, car_type"</td></tr>
</table>
</td></tr>
</table>

2.1.3. probationary_licence_prohibited_vehicle() Return whether the vehicle is a prohibited vehicle for probationary licence drivers	
Arguments	● N/A
Returns	● A boolean value of True or False

Note:

- *A car with a Power to Mass ratio greater than 130 kilowatt per tonne is a prohibited vehicle for probationary licence drivers*
- *The formula to calculate the Power to Mass ratio is:* $RoundUp(\frac{Power\ (kilowatts)}{Weight\ (kilograms)}) \times 1000$*, where weight is the tare weight of the vehicle.*

2.1.4. found_matching_car() Return whether the current vehicle is the one to be found based on a car_code.	
Arguments	● car_code (the car_code of the car to be searched)
Returns	● True (the car_code matches the current car's car_code) ● False (the car_code does not match the current car's car_code)

2.1.5. get_car_type() Return the car_type of the current car.	
Arguments	● N/A
Returns	● A string value of the car_type

2.2. Retailer Class

<table>
<tr><td colspan="2">Contains all the operations related to a retailer.</td></tr>
<tr><td>Required Class Variables</td><td>● N/A (You can add if you need)</td></tr>
<tr><td>Required Methods</td><td>
<table>
<tr><td colspan="2">2.2.1. __init__()
Constructs a Retailer object.</td></tr>
<tr><td>Arguments</td><td>● retailer_id (must be unique integer of 8 digits)
● retailer_name (can only consist of letters and the whitespace character)</td></tr>
<tr><td>Returns</td><td>● N/A</td></tr>
</table>
*All arguments of the constructor must have a default value.
<table>
<tr><td colspan="2">2.2.2. __str__()
Return the retailer information as a formatted string.</td></tr>
<tr><td>Arguments</td><td>● N/A</td></tr>
<tr><td>Returns</td><td>● a string in the following format:
"retailer_id, retailer_name"</td></tr>
</table>
<table>
<tr><td colspan="2">2.2.3. generate_retailer_id()
Generate a randomly generated unique retailer ID that is different from the existing retailer IDs and set it as the retailer_id (8 digits number)</td></tr>
<tr><td>Arguments</td><td>● list_retailer (A list of all existing retailers)</td></tr>
<tr><td>Returns</td><td>● N/A</td></tr>
</table>
</td></tr>
</table>

2.3. CarRetailer Class

Contains all the operations related to the CarRetailer Class. This class inherits from the Retailer class.

Required Class Variables

- N/A (You can add if you need)

Required methods

2.3.1. __init__()
Constructs a CarRetailer object.

Arguments	● retailer_id (must be unique integer of 8 digits) ● retailer_name (can only consist of letters and whitespace) ● carretailer_address (the address format will be street address followed by the state and postcode, e.g., “Wellington Road Clayton, VIC 3800”) ● carretailer_business_hours (a tuple of floats representing start and end hours in 24hr, e.g., (8.5, 17.5) - business hours are from 8:30AM **inclusive** to 5:30PM **inclusive**), the business hours should be within the range of 6:00AM **inclusive** to 11:00PM **inclusive**. ● carretailer_stock (a list of *car_code*s indicating the available cars from the retailer; the default value should be an empty list)
Returns	● N/A

**All arguments of the constructor must have a default value.*

2.3.2. __str__()
Return the car retailer information as a formatted string.

Arguments	● N/A
Returns	● a string in the following format: "retailer_id, retailer_name, carretailer_address, carretailer_business_hours, carretailer_stock"

2.3.3. load_current_stock() Load the current stock of the car retailer according to the *retailer_id* from the *stock.txt* file and store the *car_code*s of the Cars in a list; this list should be saved to *carretailer_stock*.	
Arguments	● path (the path to the stock file)
Returns	● N/A

2.3.4. is_operating() Return a boolean value to indicate whether the car retailer is currently operating (i.e., within working hours)	
Arguments	● cur_hour (A float value indicating current hour in 24H format, e.g., 12.5 - 12:30PM)
Returns	● True (is operating) / False (is not operating)

2.3.5. get_all_stock() Returns the information of all available cars currently in stock at the car retailer	
Arguments	● N/A
Returns	● A list of Car objects

2.3.6. get_postcode_distance() Return the absolute difference of the postcode input by the user and that of the car retailer. You need to extract the postcode from *carretailer_address* first.	
Arguments	● postcode (An integer)
Returns	● The absolute difference between the two postcodes as integers

2.3.7. remove_from_stock()
Remove a car from the current stock at the car retailer. The car stock should be consistent with the *stock.txt* file. A boolean value should be returned to indicate whether the removal is successful.

Argument	● car_code (the car_code of the car to be removed)
Returns	● True (successful) / False (unsuccessful)

Note:

- *Unsuccessful cases: car not existing in the current stock*

2.3.8. add_to_stock()
Add a car to the current stock. The car stock should be consistent with the *stock.txt* file. A boolean value should be returned to indicate whether the adding is successful.

Argument	● *car (a Car object)*
Returns	● True (successful) / False (unsuccessful)

Note:

- *Unsuccessful cases: car already existing in the current stock*

2.3.9. get_stock_by_car_type()
Return the list of cars in the current stock by specific car_type values.

Argument	● car_types (the list of car_type values of the cars to be fetched)
Returns	● A list of Car objects

2.3.10. get_stock_by_licence_type()
Return the list of cars in the current stock that are not forbidden by the driver's licence type.

Argument	● licence_type (a string value of either "L" (Learner Licence), "P" (Probationary Licence), or "Full" (Full Licence))
Returns	● A list of Car objects

Note:

- *The current regulation only forbids certain cars for probationary licence, but no restrictions are placed for learner licence and full licence.*

	2.3.11. car_recommendation() *Return a car that is randomly selected from the cars in stock at the current car retailer.*
Argument	● N/A
Returns	● A Car object

	2.3.12. create_order() *Return an order object of a created order. When an order is created, the car needs to be removed from the current stock of the car retailer. Such updates need to be reflected in "stock.txt". The created order needs to be appended to "order.txt".*
Argument	● car_code (the car_code of the car to be ordered)
Returns	● An Order object

2.4. Order Class

Contains all the operations related to an order.

Required Class Variables

- N/A (You can add if you need)

Required Methods

2.4.1. __init__()
Constructs an order object.

Arguments	• order_id (must be unique strings) • order_car (the car object related to this order) • order_retailer (the retailer object related to this order) • order_creation_time (the UNIX timestamp of the order creation), for example, 1/1/2023 00:00:01 in UNIX timestamp is 1672491601
Returns	• N/A

**All arguments of the constructor must have a default value.*

2.4.2. __str__()
Return the user information as a formatted string.

Arguments	• N/A
Returns	• a string in the following format: "order_id, order_car.car_code, order_retailer.retailer_id, order_creation_time"

2.4.3. generate_order_id()
Return a unique order ID. For the order ID:

Step 1: You first need to generate a random string of 6 lowercase alphabetic characters;
Step 2: For every second character in the string (i.e., the 2nd, 4th, 6th, etc. character), convert it into uppercase letter;
Step 3: For each character in the string from *Step 2*, get the ASCII code of the letter using `ord()`;
Step 4: Calculate the ASCII code to the power of 2 for each character and get the remainder of the powered ASCII code divided by the length of `str_1`;
Step 5: Use the remainder from *Step 4* as the index to obtain the corresponding character from `str_1` for each character in *Step 2*;

Step 6: Append each of the characters from Step 5 *n* times to the string updated in Step 2. (*n* is the index of the character in the string from Step 2.

Step 7: Append the car_code and the order creation time to the string generated in Step 6.

Example:

1. "python"
2. "pYtHoN"
3. p →112, Y→89, t→116, H→72, o→111, N →78
4.
    a. p: 112^2 →12544%9 →7
    b. Y: 89^2 →7921%9 → 1
    c. t: 116^2 →13456%9 →1
    d. H: 72^2 →5184%9 →0
    e. o: 111^2 →12321%9 →0
    f. N: 78^2 →6084%9 →0
5.
    a. 7→&
    b. 1 →!
    c. 1 →!
    d. 0 →~
    e. 0 →~
    f. 0 →~
6.
    a. & refer to 'p' with index 0 in "pYtHoN" →nothing will be appended
    b. ! refer to 'Y' with index 1 in "pYtHoN" → "pYtHoN!"
    c. ! refer to 't' with index 2 in "pYtHoN" → "pYtHoN!!"
    d. ~ refer to 'H' with index 3 in "pYtHoN" → "pYtHoN!!!~~~"
    e. ~ refer to 'o' with index 4 in "pYtHoN" → "pYtHoN!!!~~~~~~~"
    f. ~ refer to 'N' with index 5 in "pYtHoN" → "pYtHoN!!!~~~~~~~~~~~~"

    So the final output of step 6 is "pYtHoN!!!~~~~~~~~~~~~"
7. With a car_code (e.g., "MB123456") and an order creation time (e.g., 1672491601), the final output will be "pYtHoN!!!~~~~~~~~~~~~MB1234561672491601"

Arguments	• *car_code* (the car_code related to the current order)
Local Variables	• `str_1 = "~!@#$%^&*"`
Returns	• A unique string representing the order ID.

2.5. Main File

In this file, you will be creating three functions: **main_menu()**, **generate_test_data()**, and **main()**.

- The **main_menu()** function will display all available operations for users to choose from. This function accepts zero parameters and has no return value.
- The **generate_test_data()** function will generate test data for the program, including a total of twelve cars and three car retailers (each car retailer with four cars in their stock). This function accepts zero parameters and has no return value.
- The **main()** function will handle all program logic, including user input, calling class methods, and handling validations.
    - In the **main()** function, you should maintain a list of all existing retailer objects. When creating a new retailer object, you should create the CarRetailer object with a default retailer ID value of -1, and then immediately call the **generate_retailer_id()** of the newly created retailer object to generate a unique retailer ID for the retailer object.

  This function accepts zero parameters and has no return value.

The design and implementation is up to you but must include the menu items outlined in section 2 using the classes and methods outlined.

You must ensure that your menu and control logic handles exceptions appropriately.

You can break down your code to several functions if you wish but you still need to call the extra defined functions in the main function. In the **if __name__=="__main__"** part, only call **main()** function. Your tutor will only run your main.py file. For each operation that the user performs, try to give enough instructional messages.

In regards to the tasks listed above, please adhere to the method names as provided in the instructions. Additionally, you are permitted to add more class variables, methods in classes and functions. However, it is crucial to ensure that all necessary methods and functions have been implemented and that they have been invoked in the application. Any unused code present in the application will lead to mark deductions.

Please keep in mind:

- In terms of validation, it is essential to perform all validation of the user inputs in the main file outside of any methods. It is not acceptable to include any validation inside of methods unless it is required locally. Additionally, all operations involving data should be updated to files immediately.
- It is also important to understand the concept of inheritance. The child class inherits all the methods and attributes from the parent class, and the idea of overriding or

adding new methods comes into play when additional methods or attributes are required in the child class.

- Furthermore, the format of the data saved in *stock.txt* should conform to the format specified in the **str()** method for the CarRetailer Class. The format of the data saved in *order.txt* should conform to the format specified in the **str()** method for the Order Class.

2.6. User Manual

Please provide user instructions in a file named **userManual_studentID.pdf**, which should clearly explain how to run your application (main.py). Your marker will first run your main.py file without looking at the user manual. Therefore, please ensure that your program is informative. In cases when the marker cannot run your program based on the instructional messages in your program (which may incur penalty), they will follow the user manual to run the application. So please ensure they are clear and easy to follow. Please be concise in your writing and keep the PDF document to a maximum of one page in length.

3. Git Management

GIT STORAGE

Your work MUST be saved in your local working directory (local repository) in the Assignment 2 folder and **regularly pushed to the FIT GitLab server** (remote repository) to build a clear history of development of your application. Any submission with less than four pushes to the FIT GitLab server will incur a grade penalty. Please note **four** pushes is a minimum, in practice we would expect significantly more. All commits must include a meaningful commit message which clearly describes what the particular commit is about.

Students must regularly check that their Git pushes have been successful by logging in to the web interface of the FIT GitLab server; you must not simply assume they are working. Before submission, via Moodle, you must log in to the web interface of the GitLab server and ensure your submission files are present on the GitLab server.

GIT automatically maintains a history of all files pushed to the server, you MUST not add a version name to your files, please ensure you use the same name for all versions of a particular file.

If you have problems in pushing to the remote repository, you should make a backup of your work by moving your current local repository to a new folder, and then re-clone your repository.

4. Do and Do NOT

Do	Do NOT
• Maintain appropriate citing and referencing[1], • Get support early from this unit and other services within the university, • Apply for special consideration or for extensions[2] early if needed.	• Leave your assignment in draft mode (assignments in draft mode will not be marked), • Submit late (10% daily penalty applies)[3], • Attempt to submit after 7 days of the due date (they will not be accepted), unless you have special consideration.

4.1. Important NOTES

- DO NOT use absolute paths and follow the exact structure for the path as provided in the examples in each section. All path issues that cause your program crash or exception will lead to no mark for functionality.
- You must implement all required methods but you may add additional methods if you require but ensure they will be used in the application.
- Please make sure your file reading/writing operations include encoding type **utf8**. Changing the application running environment could cause issues if you do not have the encoding type. Any character encoding issues/exceptions will cause serious mark deduction.
- If one method is not working and it hinders the program from continuing running to show other functionalities, the following functionalities will receive no marks. For example, if your program can generate test data but the user can not select any option in the menu, the functionality/s that needs to be marked after the display menu will receive no marks and you will only get marks for the generating test data. Therefore, it is important to finish each functionality one by one and make sure they can work properly.
- Add correct validation and output messages to make your code more user-friendly to users.
- The assignment must be done using the **PyCharm**, **Python Version 3.10**.
- The Python code for this assignment must be implemented according to the PEP 8-Style Guide for Python Code.

[1] https://www.monash.edu/library/help/citing-and-referencing/citing-and-referencing-tutorial
[2] https://www.monash.edu/exams/changes/special-consideration
[3] e.g.: The original mark was 70/100, submitting 2 days late results in 50/100 (20 mark deduction). This includes weekends and public holidays.

- The allowed libraries are **random**, **math**, **os**, **string, re, time**. You will not receive marks for the components if you use any other libraries apart from the mentioned library.
- Commenting on your code is an essential part of the assessment criteria. In addition to inline and function commenting on your code, you should include comments at the beginning of your program file which specify your name, student ID, the creation date, the last modified date of the file, as well as a high-level description of the file.
- This assignment cannot be completed in a few days and requires students to apply what they learnt each week as we move closer to the submission date. Please remember to show your progress weekly to your tutor.
- You must keep up to date with the Moodle Ed Assignment 2 forum where further clarifications may be posted (this forum is to be treated as your client). In cases of specification updates, apart from making announcements on Ed, an updated specification file will also be uploaded to Moodle. Please ensure that you regularly check the specification files.
- Please be careful to ensure you do not publicly post anything which includes your reasoning, logic or any part of your work to this forum, doing so violates Monash plagiarism/ collusion rules and has significant academic penalties. Use private posts or email your allocated tutor to raise questions that may reveal part of your reasoning or solution.
- If any aspect of the assignment specifications is unclear or not explicitly addressed, please ensure to verify with the teaching team and request clarification.
- **In this Assessment, you must NOT use generative artificial intelligence (AI) to generate any materials or content in relation to the assessment task.**

5. Submission Requirements

The assignment must be submitted by **Friday, 22nd September 2023, 4:30 pm (AEST)**.

The following files are to be submitted and must exist in your FIT GITLab server repo:

- A series of `.py` files (i.e., order.py, car.py, retailer.py, car_retailer.py and main.py)
    - A template, RENAME-ME.zip, is available on the Moodle Assessments page. **You have to use this template**.
- A **userManual_studentID.pdf** file**.**

The above files must be compressed to a .zip file named **ass2_studentID.zip** and submitted via Moodle. The **.py** files must also have been pushed to your remote repository (at FIT GitLab server) with an appropriate history as you develop your solutions. Please ensure your commit messages are meaningful. You are **NOT** required to push the history of the manual file to the FIT GitLab server. **DO NOT** push the .zip file.

- No submissions will be accepted via email.
- Please note we **cannot mark any work on the GitLab Server**, you need to ensure that you submit correctly via Moodle since it is only in this process that you complete the required student declaration without which work cannot be assessed.
- It is your responsibility to **ENSURE** that the submitted files are the correct files. We strongly recommend after uploading a submission, and prior to actually submitting in Moodle, that you download the submission and double-check its contents.
- Please **carefully** read the documentation under the "**Special Consideration**" and "**Assignment Task Submission**" on the Moodle Assessments page which covers things such as extensions, correct submission, and resubmission.
- Please note, if you need to resubmit, you cannot depend on your tutors' availability, for this reason, please be VERY CAREFUL with your submission. **It is strongly recommended that you submit several hours before the deadline to avoid such issues.**
- **Marks will be deducted for any of these requirements that are not strictly complied with.**

6. Academic Integrity

Students are expected to be familiar with the University Academic Integrity Policy and are particularly reminded of the following:

Section 1.9:

Students are responsible for their own good academic practice and must:

- undertake their studies and research responsibly and with honesty and integrity;
- credit the work of others and seek permission to use that work where required;
- not plagiarise, cheat or falsify their work;
- ensure that their work is not falsified;
- not resubmit any assessment they have previously submitted, without the permission of the chief examiner; appropriately acknowledge the work of others;
- take reasonable steps to ensure that other students are unable to copy or misuse their work; and
- be aware of and comply with University regulations, policies and procedures relating to academic integrity.

and **Section 2.9:**

Unauthorised distribution of course-related materials: Students are not permitted to share, sell or pass on to another person or entity external to Monash:

2.9.1 any course material produced by Monash University (such as lecture slides, lecture recordings, class handouts, assessment requirements, examination questions; excluding Handbook entries) as this is a breach of the Copyright Compliance Policy and such conduct may be a copyright law infringement subject to legal action; or

2.9.2 any course-related material produced by students themselves or other students (such as class notes, past assignments), nor to receive such material, without the permission of the chief examiner. The penalties for breaches of academic misconduct include

- a zero mark for the assessment task
- a zero mark for the unit
- suspension from the course
- exclusion from the University.

Where a penalty or disciplinary action is applied, the outcome is recorded and kept for seven years, or for 15 years if the penalty was excluded.

7. Marking Guide

Your work will be marked as per the following:

- **Classes Implementation - 45 marks**
  - Car Class - 7 marks
  - Retailer Class - 6 marks
  - CarRetailer Class - 22 marks
  - Order Class - 10 marks

- **Main File Design - 7 marks**
  - Error Handling and Validation
    - Does the program handle different types of errors gracefully?
    - Are user inputs validated to prevent unexpected behaviour or crashes?
    - Are error messages clear and helpful to users or developers?
  - User Interface
    - Are any user prompts or messages clear and informative?
  - Logical Flow:
    - Does the code follow a logical sequence of steps that align with the program's functionality?
    - Do all defined methods and functions have been invoked in the application?
  - Control Structures:
    - Are conditional statements (if, else, match) used effectively to guide program flow?
    - Are loops used appropriately to repeat actions based on specific conditions?

- **Application Functionality - 26 Marks**
  - Your tutor will run your main.py to check all the basic functionalities listed in section **2. Instruction**:
    - Functionality 1 - 4 marks
    - Functionality 2 - 2 marks
    - Functionality 3 - 3 marks
    - Functionality 4 - 12 marks
    - Functionality 5 - 5 marks
  - If your program crashes or if any functionalities do not execute correctly, you will receive zero (0) marks for those specific functionalities. For instance, if Functionality 4 encounters an error and its assigned Functionality score is 12 marks, you will be awarded zero (0) marks for Functionality 4. It's important to note that if the failure of any Functionality impacts the assessment of other Functionalities, those affected Functionalities will also receive a mark of zero (0) if they cannot be properly evaluated.

- **User Manual - 2 Marks**

    - Is there clear documentation on how to run the program and utilise its features?

- **Interview (compulsory) - 20 Marks**
    - Missing Interview: If you do not attend the interview, you will receive zero marks for the entire Assignment 1.
    - Interview Score of 5 Marks: If you score 5 marks in the interview, 50% of the marks initially assigned to the function implementation will be deducted. For example, if you initially scored 80 marks out of 80 for the Functions Implementation and your interview mark is 5, you will only receive 35 marks for the Functions Implementation when calculating your overall score.
    - Interview Scores of 10, 15, or 20 Marks: If you score 10, 15, or 20 marks in the interview, this mark will be considered as your interview score and added to your final mark.
- **Penalty - up to 20 marks**
    - Missing requirements as listed in section 4 (Do and DO NOT) and section 5 (Submission Requirements).

Final Assignment Mark Calculation:

- 80 marks from the Assignment 2 code submission => 36/45 **PLUS**
- 20 marks from the interview => 9/45

Total: 100 marks, recorded as a grade out of 45[4]

[4] The penalty here is excluded from the late submission.

8. Getting help

8.1. English language skills

if you don't feel confident with your English.

- Talk to English Connect: https://www.monash.edu/english-connect

8.2. Study skills

If you feel like you just don't have enough time to do everything you need to, maybe you just need a new approach.

- Talk to a learning skills advisor: https://www.monash.edu/library/skills/contacts

8.3. Things are tough right now

Everyone needs to talk to someone at some point in their life, no judgement here.

- Talk to a counsellor: https://www.monash.edu/health/counselling/appointments (friendly, approachable, confidential, free)

8.4. Things in the unit don't make sense

Even if you're not quite sure what to ask about, if you're not sure you won't be alone, it's always better to ask.

- Ask in Ed
- Attend a consultation

8.5. I don't know what I need

Everyone at Monash University is here to help you. If things are tough now they won't magically get better by themselves. Even if you don't exactly know, come and talk with us and we'll figure it out. We can either help you ourselves or at least point you in the right direction.