FIT9136 Algorithms and Programming Foundations in Python

Assignment 1

March 2023

Table of Contents

1. Key Information

Purpose	This assignment will develop your skills in designing, constructing, testing, and documenting a small Python program according to specific programming standards. This assessment is related to the following learning outcome (LO): ● **LO1:** Apply best practice Python programming constructs for solving computational problems
Your task	This assignment is an Individual task where you will write Python code for a simple application whereby you will be developing a simple card game as per the specification.
Value	**15**% of your total marks for the unit.
Due Date	**Friday, 31 March 2023, 4:30 PM (AEDT)**
Submission	● Via Moodle Assignment Submission. ● FIT GitLab check-ins will be used to assess the history of development ● Turnitin 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 and test strategy
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 7 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, you will be required to create a Python program that enables a user (player) to play a Card Game against a computer (robot). The program **must have a text interface**, and it should include all the necessary functionality as specified in this section. Your program will be evaluated based on its ease of use and clarity, including the provision of clear information and error messages for the player.

The objective of the Card Game is to allow a player and a robot to repeatedly draw cards from a deck. Victory is attained when the player's card combination conforms to the rules and surpasses that of the robot's. The task at hand requires the creation of functions to facilitate the entire gameplay process. It is imperative to carefully review the following comprehensive regulations and prerequisites for each function and aim to execute them.

2.1 Game menu function

This function is responsible for displaying the game menu at the start of the game, as well as during the game process to provide instructional suggestions. The menu should include the following six options at a minimum: Start Game, Pick a Card, Shuffle Deck, Show My Cards, Check Win/Lose, and Exit.

Aside from the menu options, you should also present all available types of suits to the player and allow them to select the desired option. It is highly recommended to include informative messages that assist users in navigating and comprehending the gameplay. Please note that this function does not return any value.

2.2 Create Deck function

The purpose of this function is to generate a deck of cards based on the provided arguments, which include the **deck**, **suits**, and **values**. The deck is a list that stores all the cards generated from a complete combination of suits and values. Suits refer to the type of card, while values refer to the card's value. It's important to note that suits must have at least two elements. For example, Given the variables suits and values defined as follows:

suits = ["♥", "♦"]

values = ["2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K", "A"]

After creating the deck, it will contain all types of cards and be saved in the deck variable. The format of each card will be as follows:

deck = ["2 of ♥", "3 of ♥", …., "J of ♥", "Q of ♥", "K of ♥", "A of ♥, "2 of ♦", "3 of ♦", …., "J of ♦", "Q of ♦", "K of ♦", "A of ♦"]

Note:

- You have the freedom to decide on the format of the cards and it does not necessarily have to be the same as the card format provided in the previous example. For instance, you could define the deck using a format such as:
    - deck = ["2♥", "3♥", ..., "J♥", "Q♥", "K♥", "A♥", "2♦", "3♦", ..., "J♦", "Q♦", "K♦", "A♦"].

  Feel free to modify the card format as needed, provided it is still a valid representation of the deck of cards.

- The **suits** variable used in this assignment is defined as a list of card suit symbols. The specific suits used may vary depending on the game being played. In this assignment, the following lists are used for suits:
    - suits1 = ["♥", "♦", "♣", "♠"]
    - suits2 = ["😃", "😈", "😵", "🤢", "😨"]
    - suits3 = ["🤡", "👹", "👺", "👻", "👽", "👾", "🤖"]

  When defining the suits variable, it is important to ensure that the symbols used are appropriate for the game being played and are consistent with the rules and conventions of the game.

- In this assignment, the **values** variable is assigned to a fixed list of card values, which includes the numbers 2 through 10, as well as the face cards J, Q, K, and A. The A value is set to 1, while the face cards have values of 11 (J), 12 (Q), and 13 (K). Specifically, the values variable is assigned to the following list:
    - values = ["2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K", "A"].

  It is important to ensure that the values used are consistent with the rules of the card game being played.

- This function does not have a return value.

2.3 Shuffle Deck function

This function takes two arguments: **deck**, which is a list representing a deck of cards, and **suits**, which is a list of the different suits used in the deck. The function shuffles the cards in the deck randomly and displays the shuffled deck to the user. After shuffling, the deck always maintains the order where the first card is the 'A' card of the first suit in the suits list, the middle position (i.e., round($\frac{n+1}{2}$, 0))) is the 'Q' card of the second suit in the suits list, and the last card is the 'K' card of the last suit in the suits list. If a required card ('A', 'Q', or 'K' of any suit) is picked by the player, that card is ignored. For example, if suits = ["♥", "♦", "♣", "♠"], after shuffling the deck might be ["A of ♥", ……, "Q of ♦"....., "K of ♠"]. This function does not return anything.

2.4 Pick Card function

This function takes one argument, **deck**. It randomly selects one card from the deck and returns it. The picked card is then removed from the deck. Both the player and the robot will be picking cards from the same deck.

2.5 Show Cards function

This function takes in one argument, player_cards. Its main purpose is to display all the cards that the player holds. There is no return value for this function.

2.6 Check Result function

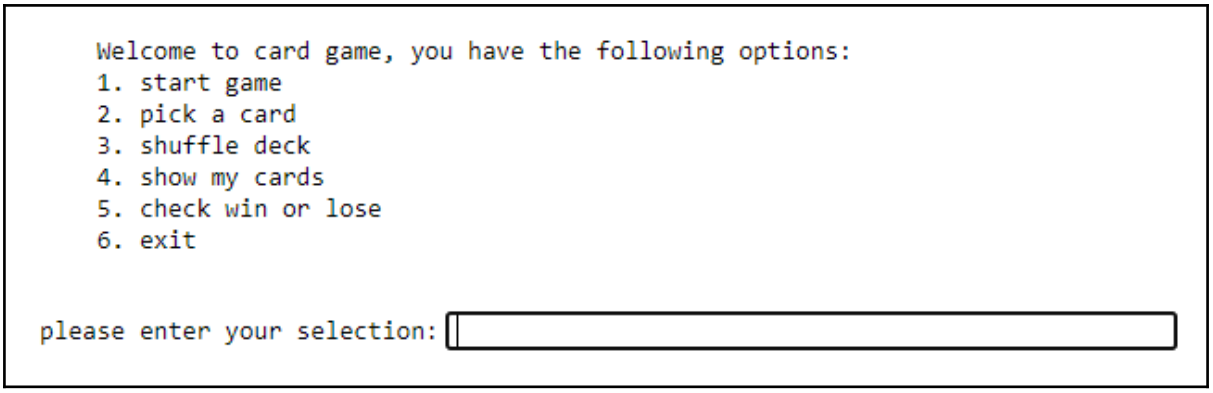
This function accepts three parameters - **player_cards**, **robot_cards**, and **suits**. Its main task is to compare the combination of cards held by the player and the robot to determine if the player has won the game. The robot is considered as another player. To win, the player must meet one of the rule in the following order of priority: Rule 1 > Rule 2 > Rule 3 > Rule 4. The rules are as follows:

1. The player holds the same value card for all the defined suits.
   For example, if the suits are ["♥", "♦", "♣", "♠"], the player should hold a set of cards that contains the following cards: ['A of ♥', 'A of ♦', 'A of ♣', 'A of ♠']. If the player's set of cards does not meet this requirement, the comparison proceeds to Rule 2. If the robot meets this requirement but the player does not, then the robot wins.
2. The player has the same values for at least the total defined suits minus one.
   For example, if the suits are ["♥", "♦", "♣", "♠"], the player should hold a set of cards that contains at least three suits with the same values, such as ['A of ♥', 'A of ♦', 'A of ♣']. If the player's set of cards does not meet this requirement, then check Rule 3. If the robot meets this requirement but the player does not, then the robot wins.
3. The player holds more cards from the suit in position 2 than the robot. Note: if the suits=["♥", "♦", "♣", "♠"], the second suit is "♦".
   For example, if the player holds cards =['A of ♥', '2 of ♦', '3 of ♣', '4 of ♦'] and the robot holds cards = ['2 of ♥', '3 of ♦', '4 of ♣', '5 of ♠'], the player wins as they have two "♦" cards and the robot only has one. If the player's set of cards doesn't meet the above requirement, then check Rule 4. If the robot meets this requirement but the player does not, then the robot wins.
4. The player holds a higher average of the card's value than robot.
   For example, if the player holds cards =['10 of ♥', '2 of ♦', '3 of ♣', '4 of ♦'] and the robot holds cards = ['2 of ♥', '3 of ♦', '4 of ♣', '5 of ♠'], the player wins as the average number of cards value (i.e., (10+2+3+4) / 4 = 4.75) is higher than the robot's average number of cards value (i.e., (2+3+4+5)/4 = 3.5). If the robot meets this requirement but the player does not, then the robot wins.

If the player's cards do not meet any of the conditions specified in rules 1 to 4, the player loses. If the robot's cards is empty and player's card is not empty, the player win.
Return a boolean value (True or False) to indicate whether the player has won or lost.

2.7 Play Game function

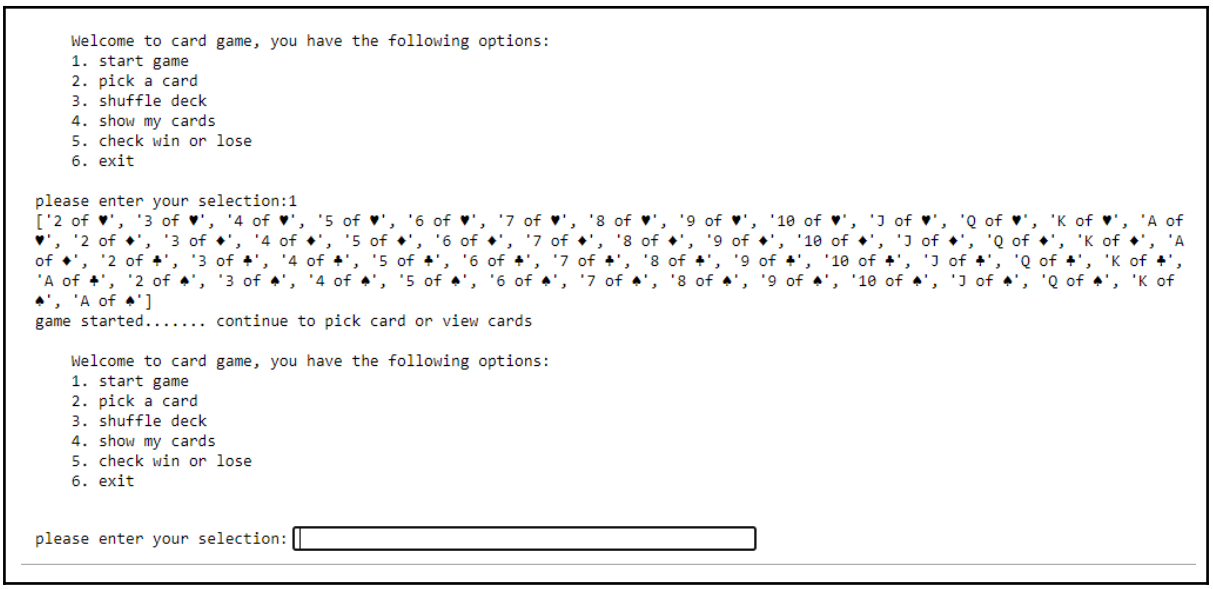
The play game function is responsible for managing all aspects of the game play. When the play game function is invoked, it should display a menu that allows the user to select menu options and card suit types by typing a corresponding number or letter. The menu should be easily readable and user-friendly. Once the user selects an option, the program should execute the relevant functions or display additional prompts as needed. The user should be able to return to the menu at any time or exit the program if desired. The accompanying image provides a basic example of the menu, but you are required to incorporate additional features as specified in the assignment description.



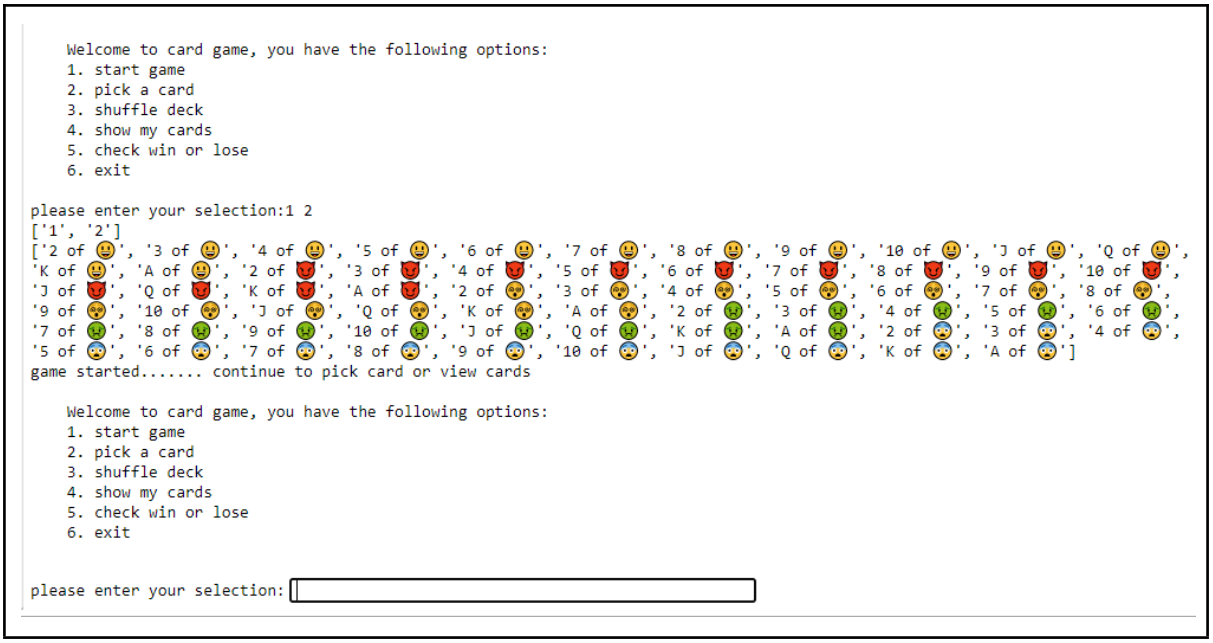
To start a game, the user can input one or two inputs. When providing two inputs, the first input should specify the game options and the second input should specify the suits to use. If no second argument is provided, the program will use a default suit type.

For example:

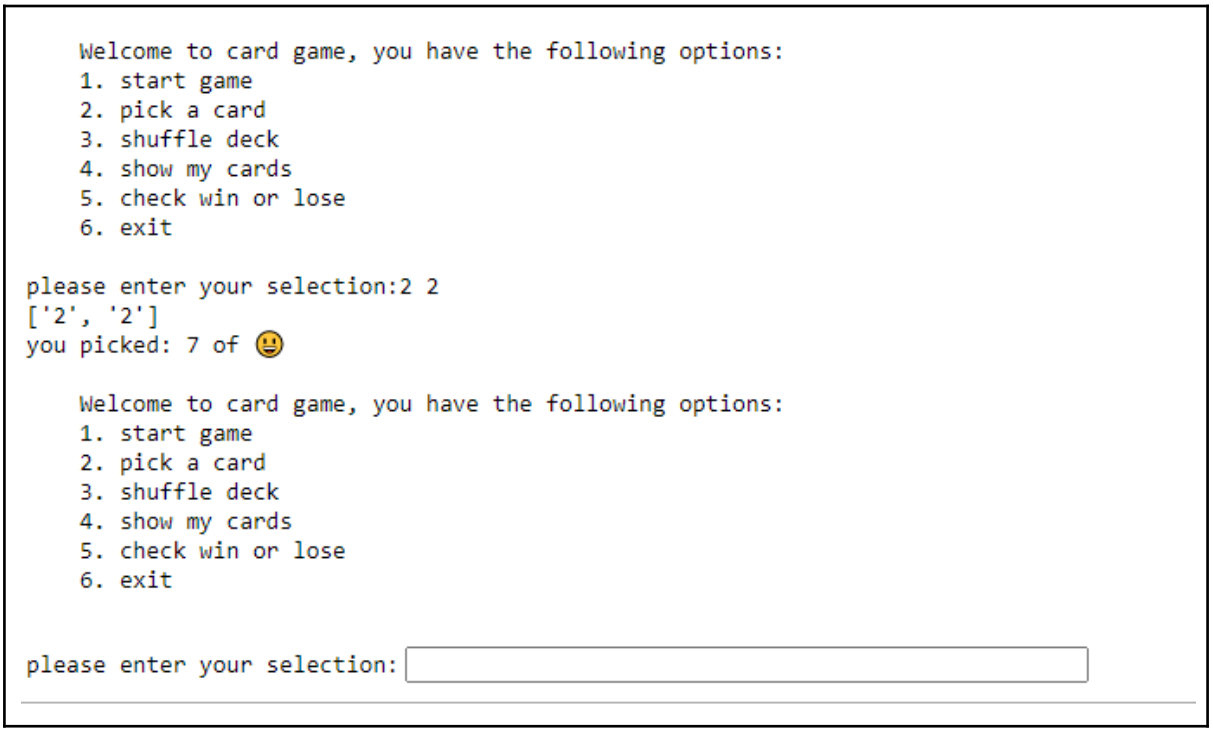
1. If the player enters "1" as input, the program should execute the "1. start game" option and activate the corresponding functions. As shown in the accompanying image, a prompt message should inform the user that the game has started, and that the deck has been shuffled and printed out.



2. if the user enters "1 2" as input, the program should execute the "1. start game" option and use the second suit type. Please refer to the accompanying image for an example of how the program should respond to two inputs.



Note that the suits type can only be changed when selecting the "1. Start Game" option from the menu. If the user enters a two-argument command, such as "2 2", this indicates that the second option in the menu is selected, and the second argument should be ignored. For example, as shown in the accompanying image, if the user enters "2 2" as input, the "pick a card" function should be executed, but the second argument "2" should be ignored. The program should be designed to handle such scenarios and ensure that the correct functions are called and the appropriate actions are taken based on the user inputs.



Every user can pick **a maximum of 6 cards**. Once the number of cards reaches 6, the program should display the final result and automatically restart the game. When a user

picks a card, the program should also randomly generate a card for the ***robot_cards*.** The program could pick no card for the ***robot_cards*,** which means the length of ***robot_cards*** and ***player_cards*** could be different***.*** The program should keep running until the user selects “6. Exit” option. The user should only input number 1-6 and only option 1 can accept a second argument for ***suits*** type. No return value.

In completing the tasks outlined above, you are free to modify the function names and variables as needed to follow your own naming conventions. This can include changes to the name, type, or scope of the variables, as well as the names of the functions themselves. However, it is important to ensure that the changes are consistent and coherent throughout the code, and that they do not impact the functionality or readability of the program. If you are unsure about the appropriateness of a specific naming convention, please consult the instructor or teaching assistants for guidance.

3. 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.

3.1. Important Notes:

- If any exception/errors happen when running each function, you will lose 50% of allocated function logic marks. For example, if the total mark of one task is 10 marks and any exception happens when running this task, then the maximum mark you can get is 5 instead of 10 in the function logic.
- Add correct validation and output messages to make your code more user-friendly to users.
- For each function, add test code with correct/incorrect input/output. All the test code should be commented upon when submitting the assignment.
- This is an individual assignment and must be completed on your own. You must attribute the source of any part of your code that you have not written yourself. Please note the section on Academic Integrity in this document.
- The assignment must be done using the **Jupyter Notebook**, **Python Version 3.9**.
- The Python code for this assignment must be implemented according to the PEP 8-Style Guide for Python Code.
- The allowed libraries are **random** and **math**. You will receive penalties if you use any other libraries.
- 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, and the last modified date of the program, as well as a high-level description of the program.

---

[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.

- This assignment cannot be completed in a few days and requires students to apply what we learn 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 1 forum where further clarifications may be posted (this forum is to be treated as your client).
- 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.
- **In this Assessments, you must NOT use generative artificial intelligence (AI) to generate any materials or content in relation to the assessment task.**

4. Submission Requirements

The assignment must be submitted by **Friday, 31 March 2023, 4:30 PM (AEDT)**.

The following files are to be submitted on Moodle:

- A Jupyter notebook file (i.e., .ipynb file) that you created to implement your assignment (i.e, code and documentation). Name the file **ass1_studentID.ipynb**
- A PDF file. Use Jupyter Notebook to export a PDF file (Read the instruction provided on Week 2 Applied Class Activities, section 2.2. "**How to Export a Jupyter Notebook to a PDF file?**". Note, The pdf file cannot be an image pdf file. Make sure all the text in the pdf file can be selected and copied). Name the file **ass1_studentID.pdf**

Do not zip these files into one zip archive, submit two independent files. The **.ipynb** file must also have been pushed to the FITGitLab server with an appropriate history as you developed your solutions (a minimum of four pushes, however, we would strongly recommend more than this). Please ensure your committed comments are meaningful.

- 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 due to avoid such issues.**
- **There is no restriction on having extra functions. Do NOT create redundant functions**
- **Marks will be deducted for any of these requirements that are not strictly complied with.**

5. 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.

6. Marking Guide

Your work will be marked as per the following:

- Game Menu Function - 3 Marks
- Create Deck Function - 5 Marks
- Shuffle Deck Function - 10 Mars
- Pick Card Function - 4 Marks
- Show Cards Function - 3 Marks
- Check Result Function - 15 Marks
- Play Game Function - 15 Marks
- Code Architecture and Adherence to Python coding standards - 5 Marks
- Comprehensiveness of documented code and test strategy - 10 Marks
- Interview (compulsory) - 20 Marks
    - Missing interview = 0 mark for the Assignment 1
    - If interview mark <=10 marks then 50% of the mark for the other parts (cannot answer) will be deducted
- Optimised Python program - 10 Marks
    - All the above functions works without any error/ single minor mistake
    - The game is easy to follow with clear information/error messages to the player.
    - The code is efficiently crafted for example using a loop instead of a series of if-else statements, using appropriate data type.
    - Have some creativity in the functionality design
- Penalty - up to 20 marks
    - Missing submission requirements

7. Getting help

7.1. English language skills

if you don't feel confident with your English.

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

7.2. 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

7.3. 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)

7.4. 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: https://edstem.org/au/courses/8843/discussion/
- Attend a consultation: https://lms.monash.edu/course/view.php?id=141449&section=21

7.5. 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.