Sudoku Puzzle

Assignment 1
Semester 2, 2022
CSSE1001/CSSE7030

Due date: 24 August 2022, 20:00 GMT+10

1 Introduction

In this assignment, you will implement a text-based version of the classic Sudoku puzzle. The Sudoku puzzle is set on a 9×9 board which is split into nine 3×3 squares. To achieve the goal of the puzzle, a player must use a process of elimination to fill in the remaining empty cells on the board, using only the digits 1 - 9, so that each row, column and 3×3 square only contain one of each digit. If you would like to get a better understanding of the game, you can read a more detailed explanation here, or practice playing the game here.

2 Getting Started

Download `a1.zip` from Blackboard — this archive contains the necessary files to start this assignment. Once extracted, the `a1.zip` archive will provide the following files:

`a1.py` *This is the only file you will submit* and is where you write your code. *Do not* make changes to any other files.

`a1_support.py` *Do not modify or submit this file*, it contains a function to help you implement your tasks. You do not need to read or understand the code itself, but you should read the docstrings in order to get an idea of how the functions work. This file also contains some constants that will be useful in your implementation. In addition to these, you are encouraged to create your own constants in `a1.py` where possible.

`boards/` This folder contains a number of board files containing starting states for Sudoku games. The games can be loaded from these files into your program with the help of a function from `a1_support.py`. Feel free to try adding your own board files for further testing.

NOTE: You are not permitted to add any import statements to `a1.py`. Doing so will result in a deduction of up to 100% of your mark.

3 Terminology

In the context of this assignment:

- A *Sudoku* is a puzzle played on a 9 x 9 board.
- The *board* is comprised of 81 evenly divided cells, with 9 rows and 9 columns.

- A *cell* is a single element of the board that is either blank or contains a single digit from 1 to 9.
- A *square* is a 3 x 3 sub-board comprising 9 cells. The first square is in the top left of the board and there are 9 non-overlapping squares in total.
- A *solution* is a board in which every cell contains a single digit and each row, column and square contain each digit from 1 to 9 with no duplicates.
- The player's *goal* is to solve the puzzle by placing a digit at a blank cell so that the board's solution is achieved.

4 Gameplay

This section provides an overview of gameplay. Where prompts and outputs are not explicitly mentioned in this section, please see Section 5.

4.1 Setup

At the beginning of the game, the player is prompted with the message `"Please insert the name of a board file: "` (note the trailing space) to choose a board by providing the name of the file that contains the board. Once the filename has been provided, the board is loaded and displayed.

4.2 Solving the Puzzle

After the board is loaded and displayed, the player is prompted to enter an action with `"Please input your move: "` (note the trailing space). After each action is applied, the updated board should be displayed and the player should be prompted again. The valid actions are outlined in Table 1. An action should work the same for both lowercase and uppercase inputs.

Throughout the game, the player may overwrite cells they have previously filled; however, they are not allowed to overwrite cells filled in the original board. If the player attempts to overwrite a cell that was filled in the original board, the message `"That move is invalid. Try again!"` should be printed, and the player should be reprompted for another move. Asides from this one case, **you may assume that your program will not be tested for invalid inputs** that do not match the expected actions in Table 1, or for inputs that are otherwise invalid (e.g. out of bounds positions). The prompt should repeat until the player either wins the game or gives up by entering the `"Q"` or `"q"` action. In the case the player fills the entire board and has not won the game, the prompt should still continue until they win or give up with the relevant action.

4.3 End of Game

At the end of a game, the player is prompted to choose if they want to start a new game or stop the program with `"Would you like to start a new game (y/n)? "`. The options available at this prompt are outlined in Table 2. If they choose to start a new game, they need to select a new board. The player should be prompted with the same message to enter a board file as they were at the start of the game. Once a new board is loaded it should be displayed, and gameplay should return to that described in Section 4.2. If they choose to stop the program, the program should terminate gracefully.

Input	Description
`"H"` or `"h"`	Display a help message.
`"Q"` or `"q"`	Stop playing the current game.
`"{row} {column} {value}"`	Insert the value at the specified position. The player's input contains three digits: row ranging from 0 to 8, column ranging from 0 to 8, and value ranging from 1 to 9.
`"{row} {column} C"`	Clear the cell at the specified position. The player's input contains two digits: row ranging from 0 to 8 and column ranging from 0 to 8. This is followed by the uppercase letter 'C'.

Table 1: Potential actions that player can choose. `{quantity}` denotes that the *value* of quantity should be provided. Values not surrounded by braces should be taken as string literals.

Input	Description
`"Y"` or `"y"` or `""`	Start a new game.
Anything else	Close the program.

Table 2: Input options for new game prompt

5 Implementation

This section outlines functions available to you in the support code, as well as functions you are **required** to implement in your solution (in `a1.py` only). You are awarded marks for the number of tests passed by your functions when they are tested *independently*. Thus an incomplete assignment with *some* working functions may well be awarded more marks than a complete assignment with faulty functions. Your program must operate *exactly* as specified. In particular, your program's output must match *exactly* with the expected output. Your program will be marked automatically so minor differences in output (such as whitespace or casing) *will* cause tests to fail resulting in a *zero mark* for that test.

Each function is accompanied with some examples for usage to help you *start* your own testing. You should also test your functions with other values to ensure they operate according to the descriptions.

In the example usage for functions you are required to write, the board variables below are assumed to be defined. You must execute these lines before testing your functions according to the examples. All example code will use these variables; any changes from a previous example are not carried over. You may adjust these or make your own for further testing if you wish.

```
board_one = "685132947734598216219764853926871534851349672473256189568427391342915768197683 42 "
board_two = "68513  477      1  1 764 5 9   7 5 48 1  9 724 3  6      42739  4 9   681 7   4  "

board_three = [
    [6, 8, 5, 1, 3, 2, 9, 4, 7],
    [7, 3, 4, 5, 9, 8, 2, 1, 6],
    [2, 1, 9, 7, 6, 4, 8, 5, 3],
    [9, 2, 6, 8, 7, 1, 5, 3, 4],
    [8, 5, 1, 3, 4, 9, 6, 7, 2],
    [4, 7, 3, 2, 5, 6, 1, 8, 9],
    [5, 6, 8, 4, 2, 7, 3, 9, 1],
    [3, 4, 2, 9, 1, 5, 7, 6, 8],
    [1, 9, 7, 6, 8, 3, 4, 2, None]
```

```
]

board_four = [
    [6, 8, 5, 1, 3, None, None, 4, 7],
    [7, None, None, None, None, None, None, 1, None],
    [None, 1, None, 7, 6, 4, None, 5, None],
    [9, None, None, None, 7, None, 5, None, 4],
    [8, None, 1, None, None, 9, None, 7, 2],
    [4, None, 3, None, None, 6, None, None, None],
    [None, None, None, 4, 2, 7, 3, 9, None],
    [None, 4, None, 9, None, None, None, 6, 8],
    [1, None, 7, None, None, None, 4, None, None]
]
```

5.1 Support Code

You may use the code provided in `a1_support.py` to implement your assignment. Do not make changes to `a1_support.py` as it could cause unexpected errors. In this file you will find some predefined named constants and a function. You should use the provided named constants in your implementation of the game. You are also provided with one function, which can be used to load a new board from a file. You must not modify this function.

5.1.1 `load_board(filename: str) -> str`

Returns a string which is a raw copy of the board, ordered left to right, top to bottom. The required parameter is the name of the file from which the game is loaded.

5.2 Required Functions

The following functions **must** be implemented in `a1.py`. They have been listed in order of increasing difficulty. It is *highly recommended* that you do not begin work on a later function until each of the preceding functions can *at least* behave as per the shown examples. You may implement additional functions if you think they will help with your logic or make your code easier to understand.

The board variable in all these functions is of the type `list[list[Optional[int]]]`. This can be described as 'a list of lists containing either integers or `None`'. We will refer to this type as `Board`, i.e. `Board = list[list[Optional[int]]]`.

5.2.1 `num_hours() -> float`

This function should return the number of hours you estimate you spent (or have spent so far) on the assignment, as a *float*. You will not be marked differently for spending more or less time on the assignment. The purpose of this function is to enable you to verify that you understand how to submit to Gradescope as soon as possible, and to allow us to gauge difficulty level of this assignment in order to provide the best possible assistance.

Ensure this function passes the relevant test on Gradescope *as soon as possible*. If the Gradescope tests have been released, you must ensure this function passes the relevant test before seeking help regarding Gradescope issues for any of the later functions. See Section 7.3 for instructions on how to submit your assignment to Gradescope.

5.2.2 `is_empty(position: tuple[int, int], board: Board) -> bool`

Returns `True` if the given (row, column) position contains `None` (i.e. it is not yet filled with a number) and `False` otherwise. You may assume that the given position is a valid position on the board.

Example:

```
>>> is_empty((3, 5), board_three)
False
>>> is_empty((8, 8), board_three)
True
```

5.2.3 `update_board(position: tuple[int, int], value: Optional[int], board: Board) -> None`

Updates the board at the provided (row, column) position with the provided value. Please note that the function returns `None`, so you should be altering the board given as a parameter in this function. You may assume that the given position exists on the board. A precondition to this function is that the given position must not correspond to a filled cell in the original board.

Example:

```
>>> update_board((8, 8), 5, board_three)
>>> board_three
[[6, 8, 5, 1, 3, 2, 9, 4, 7], [7, 3, 4, 5, 9, 8, 2, 1, 6],
[2, 1, 9, 7, 6, 4, 8, 5, 3], [9, 2, 6, 8, 7, 1, 5, 3, 4],
[8, 5, 1, 3, 4, 9, 6, 7, 2], [4, 7, 3, 2, 5, 6, 1, 8, 9],
[5, 6, 8, 4, 2, 7, 3, 9, 1], [3, 4, 2, 9, 1, 5, 7, 6, 8],
[1, 9, 7, 6, 8, 3, 4, 2, 5]]
```

5.2.4 `clear_position(position: tuple[int, int], board: Board) -> None`

Updates the board to clear the cell at the provided (row, column) position. A precondition to this function is that the given position must not correspond to a filled cell in the original board. You may assume that the given position exists on the board.

Example:

```
>>> update_board((8, 8), 5, board_three)
>>> board_three
[[6, 8, 5, 1, 3, 2, 9, 4, 7], [7, 3, 4, 5, 9, 8, 2, 1, 6],
[2, 1, 9, 7, 6, 4, 8, 5, 3], [9, 2, 6, 8, 7, 1, 5, 3, 4],
[8, 5, 1, 3, 4, 9, 6, 7, 2], [4, 7, 3, 2, 5, 6, 1, 8, 9],
[5, 6, 8, 4, 2, 7, 3, 9, 1], [3, 4, 2, 9, 1, 5, 7, 6, 8],
[1, 9, 7, 6, 8, 3, 4, 2, 5]]
>>> clear_position((8,8), board_three)
>>> board_three
[[6, 8, 5, 1, 3, 2, 9, 4, 7], [7, 3, 4, 5, 9, 8, 2, 1, 6],
[2, 1, 9, 7, 6, 4, 8, 5, 3], [9, 2, 6, 8, 7, 1, 5, 3, 4],
[8, 5, 1, 3, 4, 9, 6, 7, 2], [4, 7, 3, 2, 5, 6, 1, 8, 9],
[5, 6, 8, 4, 2, 7, 3, 9, 1], [3, 4, 2, 9, 1, 5, 7, 6, 8],
[1, 9, 7, 6, 8, 3, 4, 2, None]]
```

5.2.5 `read_board(raw_board: str) -> Board`

Converts the raw board from a string of characters into a list of 9 lists. Each inner list represents one row of the board (ordered from top to bottom) and contains 9 items which can each either be an integer (if one exists in that location) or `None` if no number has been placed in that location (represented by a space, `" "`, in the `raw_board`). The format of the `raw_board` is one string, containing integers and/or spaces representing the board from left to right, top to bottom. You can assume that the string will only ever contain integers and/or space characters.

Example:

```
>>> read_board(board_one)
[[6, 8, 5, 1, 3, 2, 9, 4, 7], [7, 3, 4, 5, 9, 8, 2, 1, 6],
[2, 1, 9, 7, 6, 4, 8, 5, 3], [9, 2, 6, 8, 7, 1, 5, 3, 4],
[8, 5, 1, 3, 4, 9, 6, 7, 2], [4, 7, 3, 2, 5, 6, 1, 8, 9],
[5, 6, 8, 4, 2, 7, 3, 9, 1], [3, 4, 2, 9, 1, 5, 7, 6, 8],
[1, 9, 7, 6, 8, 3, 4, 2, None]]
>>> read_board(board_two)
[[6, 8, 5, 1, 3, None, None, 4, 7],
[7, None, None, None, None, None, None, 1, None],
[None, 1, None, 7, 6, 4, None, 5, None],
[9, None, None, None, 7, None, 5, None, 4],
[8, None, 1, None, None, 9, None, 7, 2],
[4, None, 3, None, None, 6, None, None, None],
[None, None, None, 4, 2, 7, 3, 9, None],
[None, 4, None, 9, None, None, None, 6, 8],
[1, None, 7, None, None, None, 4, None, None]]
```

5.2.6 `print_board(board: Board) -> None`

Displays the puzzle in a user-friendly format. Your output must exactly match the expected formatting (including whitespace and grammar) in order to receive marks. Ensure when testing with the below examples that your output exactly matches the output shown. Horizontal separator lines are composed of 11 hyphens. There is a single space between each row of the board and the corresponding row number, and there is a blank line between the board and the column numbers.

Example:

```
>>> print_board(board_three)
685|132|947 0
734|598|216 1
219|764|853 2
-----------
926|871|534 3
851|349|672 4
473|256|189 5
-----------
568|427|391 6
342|915|768 7
197|683|42  8

012 345 678
```

```
>>> print_board(board_four)
685|13 | 47 0
7  |   | 1  1
 1 |764| 5  2
-----------
9  | 7 |5 4 3
8 1|  9| 72 4
4 3|  6|    5
-----------
   |427|39  6
 4 |9  | 68 7
1 7|   |4   8

012 345 678
```

5.2.7 `has_won(board: Board) -> bool`

Returns `True` if the game is won, `False` otherwise. The game is won if the goal has been achieved; that is, every cell is filled and all rows, columns and squares on the board contain exactly one of each digit from 1 to 9.

Example:

```
>>> has_won(board_three)
False
>>> update_board((8, 8), 5, board_three)
>>> board_three
[[6, 8, 5, 1, 3, 2, 9, 4, 7], [7, 3, 4, 5, 9, 8, 2, 1, 6],
[2, 1, 9, 7, 6, 4, 8, 5, 3], [9, 2, 6, 8, 7, 1, 5, 3, 4],
[8, 5, 1, 3, 4, 9, 6, 7, 2], [4, 7, 3, 2, 5, 6, 1, 8, 9],
[5, 6, 8, 4, 2, 7, 3, 9, 1], [3, 4, 2, 9, 1, 5, 7, 6, 8],
[1, 9, 7, 6, 8, 3, 4, 2, 5]]
>>> has_won(board_three)
True
```

5.2.8 `main() -> None`

The main function should be called when the file is run, and coordinates the overall gameplay. The `main` function should be fairly short, and should utilize other functions you have written. In order to make the `main` function shorter, you should consider writing extra helper functions. In the provided `a1.py`, the function definition for `main` has already been provided, and the `if __name__ == __main__:` block will ensure that the code in the `main` function is run when your `a1.py` file is run. Do not call your `main` function outside of this block, and do not call any other function outside this block unless you are calling them from within the body of another function. The output from your `main` function (including prompts) must exactly match the expected output. Running the sample tests will give you a good idea of whether your prompts and other outputs are correct. See Section 6 for example usage.

Section 4 describes how the game should be played. The basic steps that must be implemented by this function are as follows:

1. Prompt user for board filename.

2. Load board and convert to readable format. It is recommended you do this twice so that you also have a copy of the original board stored. This way you can refer back to it later when checking if you can update a cell without changing the original board.

3. Until the game has been won:

   (a) Print the current board state in the required user-friendly format.

   (b) Prompt player for an action (see Section 4 for potential actions).

   (c) If the player requested help, print out the message "Need help?\nH = Help\nQ = Quit\nHint: Make sure each row, column, and square contains only one of each number from 1 to 9."

   (d) If the player requested to quit the game, then terminate the program gracefully.

   (e) If the player provided a position and a value (or "C" if the player wants to clear a position), first check to make sure that player is not changing a filled cell from the original board. If they are, print out the message `"That move is invalid. Try again!"` otherwise update the board accordingly.

4. Print the current board state in the required user-friendly format.

5. Notify the player that the game has been won via the required message.

6. Prompt the user for whether they'd like to play again. If yes, go to step 1. If no, terminate the program gracefully.

6 Example Gameplay

The following section provides extended instances of gameplay to demonstrate completed program behaviour.

Game 1

```
Please insert the name of a board file: boards/board_one.txt
685|13 | 47 0
7  |   | 1  1
 1 |764| 5  2
-----------
9  | 7 |5 4 3
8 1|  9| 72 4
4 3|  6|    5
-----------
   |427|39  6
 4 |9  | 68 7
1 7|   |4   8

012 345 678
Please input your move: 0 5 2
685|132| 47 0
7  |   | 1  1
 1 |764| 5  2
-----------
9  | 7 |5 4 3
8 1|  9| 72 4
```

```
4 3|  6|    5
-----------
   |427|39  6
 4 |9  | 68 7
1 7|   |4   8

012 345 678
Please input your move: 0 6 9
685|132|947 0
7  |   | 1  1
 1 |764| 5  2
-----------
9  | 7 |5 4 3
8 1|  9| 72 4
4 3|  6|    5
-----------
   |427|39  6
 4 |9  | 68 7
1 7|   |4   8

012 345 678
Please input your move: 3 7 3
685|132|947 0
7  |   | 1  1
 1 |764| 5  2
-----------
9  | 7 |534 3
8 1|  9| 72 4
4 3|  6|    5
-----------
   |427|39  6
 4 |9  | 68 7
1 7|   |4   8

012 345 678
Please input your move: 6 8 1
685|132|947 0
7  |   | 1  1
 1 |764| 5  2
-----------
9  | 7 |534 3
8 1|  9| 72 4
4 3|  6|    5
-----------
   |427|391 6
 4 |9  | 68 7
1 7|   |4   8

012 345 678
Please input your move: 0 1 4
That move is invalid. Try again!
```

```
685|132|947 0
7  |   | 1  1
 1 |764| 5  2
-----------
9  | 7 |534 3
8 1|  9| 72 4
4 3|  6|    5
-----------
   |427|391 6
 4 |9  | 68 7
1 7|   |4   8

012 345 678
Please input your move: 1 0 C
That move is invalid. Try again!
685|132|947 0
7  |   | 1  1
 1 |764| 5  2
-----------
9  | 7 |534 3
8 1|  9| 72 4
4 3|  6|    5
-----------
   |427|391 6
 4 |9  | 68 7
1 7|   |4   8

012 345 678
Please input your move: H
Need help?
H = Help
Q = Quit
Hint: Make sure each row, column, and square contains only one of each number
from 1 to 9.

685|132|947 0
7  |   | 1  1
 1 |764| 5  2
-----------
9  | 7 |534 3
8 1|  9| 72 4
4 3|  6|    5
-----------
   |427|391 6
 4 |9  | 68 7
1 7|   |4   8

012 345 678
Please input your move: Q
```

Game 2

```
Please insert the name of a board file: boards/board_one_winnable.txt
685|132|947 0
734|598|216 1
219|764|853 2
-----------
926|871|534 3
851|349|672 4
473|256|189 5
-----------
568|427|391 6
342|915|768 7
197|683|42  8

012 345 678
Please input your move: 8 8 4
685|132|947 0
734|598|216 1
219|764|853 2
-----------
926|871|534 3
851|349|672 4
473|256|189 5
-----------
568|427|391 6
342|915|768 7
197|683|424 8

012 345 678
Please input your move: 8 8 C
685|132|947 0
734|598|216 1
219|764|853 2
-----------
926|871|534 3
851|349|672 4
473|256|189 5
-----------
568|427|391 6
342|915|768 7
197|683|42  8

012 345 678
Please input your move: 8 8 5
685|132|947 0
734|598|216 1
219|764|853 2
-----------
926|871|534 3
851|349|672 4
473|256|189 5
-----------
```

```
568|427|391 6
342|915|768 7
197|683|425 8

012 345 678
Congratulations, you won!
Would you like to start a new game (y/n)? n
```

7 Assessment and Marking Criteria

This assignment assesses course learning objectives:

1. apply program constructs such as variables, selection, iteration and sub-routines,
2. read and analyse code written by others,
3. read and analyse a design and be able to translate the design into a working program, and
4. apply techniques for testing and debugging.

7.1 Functionality

Your program's functionality will be marked out of a total of 6 marks. Your assignment will be put through a series of tests and your functionality mark will be proportional to the number of tests you pass. You will be given a *subset* of the functionality tests before the due date for the assignment.

You may receive partial marks within each section for partially working functions, or for implementing only a few functions.

You need to perform your *own* testing of your program to make sure that it meets *all* specifications given in the assignment. Only relying on the provided tests is likely to result in your program failing in some cases and you losing some functionality marks. Note: Functionality tests are automated, so string outputs need to match *exactly* what is expected.
Your program must run in the Python interpreter (the IDLE environment). Partial solutions will be marked but if there are errors in your code that cause the interpreter to fail to execute your program, you will get zero for functionality marks. If there is a part of your code that causes the interpreter to fail, comment out the code so that the remainder can run. Your program must run using the Python 3.10 interpreter. If it runs in another environment (e.g. Python 3.8 or PyCharm) but not in the Python 3.10 interpreter, you will get zero for the functionality mark.

7.2 Code Style

The style of your assignment will be assessed by a tutor. Style will be marked according to the style rubric provided with the assignment. The style mark will be out of 4.

The key consideration in marking your code style is whether the code is easy to understand. There are several aspects of code style that contribute to how easy it is to understand code. In this assignment, your code style will be assessed against the following criteria.

- Readability

  - Program Structure: Layout of code makes it easy to read and follow its logic. This includes using whitespace to highlight blocks of logic.
  - Descriptive Identifier Names: Variable, constant, and function names clearly describe what they represent in the program's logic. Do not use Hungarian Notation for identifiers. In short, this means do not include the identifier's type in its name, rather make the name meaningful (e.g. employee identifier).
  - Named Constants: Any non-trivial fixed value (literal constant) in the code is represented by a descriptive named constant (identifier).

- Algorithmic Logic
  - Single Instance of Logic: Blocks of code should not be duplicated in your program. Any code that needs to be used multiple times should be implemented as a function.
  - Variable Scope: Variables should be declared locally in the function in which they are needed. Global variables should not be used.
  - Control Structures: Logic is structured simply and clearly through good use of control structures (e.g. loops and conditional statements).

- Documentation:
  - Comment Clarity: Comments provide meaningful descriptions of the code. They should not repeat what is already obvious by reading the code (e.g. `# Setting variable to 0`). Comments should not be verbose or excessive, as this can make it difficult to follow the code.
  - Informative Docstrings: Every function should have a docstring that summarises its purpose. This includes describing parameters and return values (including type information) so that others can understand how to use the function correctly.
  - Description of Logic: All significant blocks of code should have a comment to explain how the logic works. For a small function, this would usually be the docstring. For long or complex functions, there may be different blocks of code in the function. Each of these should have an in-line comment describing the logic.

7.3 Assignment Submission

You must submit your assignment electronically via Gradescope (`https://gradescope.com/`). You **must** use your UQ email address which is based on your student number (e.g. s4123456@student.uq.edu.au) as your Gradescope submission account.

When you login to Gradescope you may be presented with a list of courses. Select CSSE1001/CSSE7030. You will see a list of assignments. Choose **Assignment 1**. You will be prompted to choose a file to upload. The prompt may say that you can upload any files, including zip files. You **must** submit your assignment as a single Python file called `a1.py` (use this name – all lower case), and *nothing* else. Your submission will be automatically run to determine the functionality mark. If you submit a file with a **different name**, the tests will **fail** and you will get **zero** for functionality. Do **not** submit **any** sort of archive file (e.g. zip, rar, 7z, etc.).

Upload an initial version of your assignment *at least* one week before the due date. Do this even if it is just the initial code provided with the assignment. If you are unable access Gradescope, contact the course helpdesk (csse1001@helpdesk.eait.uq.edu.au) *immediately*. Excuses, such as you were not able to login or were unable to upload a file will not be accepted as reasons for

granting an extension.

When you upload your assignment it will run a **subset** of the functionality autograder tests on your submission. It will show you the results of these tests. It is your responsibility to ensure that your uploaded assignment file runs and that it passes the tests you expect it to pass.

Late submissions of the assignment will **not** be marked. Do not wait until the last minute to submit your assignment, as the time to upload it may make it late. Multiple submissions are allowed, so ensure that you have submitted an almost complete version of the assignment *well* before the submission deadline of 20:00. Your latest, on time, submission will be marked. Ensure that you submit the correct version of your assignment.

In the event of exceptional personal or medical circumstances that prevent you from handing in the assignment on time, you may submit a request for an extension. See the course profile for details of how to apply for an extension.

Requests for extensions must be made **before** the submission deadline. The application and supporting documentation (e.g. medical certificate) must be submitted via my.UQ. You must retain the original documentation for a minimum period of six months to provide as verification, should you be requested to do so.

7.4 Plagiarism

This assignment must be your own individual work. By submitting the assignment, you are claiming it is entirely your own work. You **may** discuss general ideas about the solution approach with other students. Describing details of how you implement a function or sharing part of your code with another student is considered to be **collusion** and will be counted as plagiarism. You **may not** copy fragments of code that you find on the Internet to use in your assignment.

Please read the section in the course profile about plagiarism. You are encouraged to complete *both* parts A and B of the academic integrity modules *before* starting this assignment. Submitted assignments will be electronically checked for potential cases of plagiarism.