Pre-WA1 Practice Paper 2 (50 marks, recommended duration – 1 hour 15 minutes)

Answer **all** questions.

All tasks must be done in the computer laboratory. You are not allowed to bring in or take out any pieces of work or materials on paper or electronic media or in any other form.

Approved calculators are allowed.

Save each task as it is completed.

The use of built-in functions, where appropriate, is allowed for this paper unless stated otherwise.

The number of marks is given in brackets [ ] at the end of each question or part question.

Your program code and output for each of Task 1 to 2 should be saved in a single .ipynb file. For example, your program code and output for Task 1 should be saved as

`TASK1_<your name>_<centre number>_<index number>.ipynb`

1 Name your Jupyter Notebook as

`TASK1_<your name>_<centre number>_<index number>.ipynb`

In a competition, the overall score obtained by a participant is derived from the individual scores given by nine judges. As the nine judges enter the scores of a participant into the score processing system, they are stored in a nested list of the following format:

```
scores = [['judge_1', score_1], …, ['judge_9', score_9]]
```

where `judge_x` is a string and `score_x` is a float.

Judges are allowed to amend their scores once before the overall score is released. All changes are stored in a nested list of the following format:

```
amendments = [['judge_m', score_m], …, ['judge_k', score_k]]
```

where `judge_x` is a string and `score_x` is a float.

The task is to write functions for the score processing system to compute the scores obtained by the participants.

For each of the sub-tasks, add a comment statement at the beginning of the code, using the hash symbol '#' to indicate the sub-task the program code belongs to, for example:

```
In [1] : # Task 1.1
         Program code

         Output:
```

Task 1.1

The use of built-in sorting functions for this sub-task is prohibited.

Write a function `average_score(lst)` that:

- takes in a nested list, `lst`, containing nine sub-lists, each storing the score given by a judge
- ignores the highest and lowest scores in `lst`
- returns the score obtained by a participant as the average of the remaining seven scores. [5]

Write further program code to test the function with the following list of scores given to a participant by the nine judges:

```
scores = [['J1', 2.0], ['J2', 5.0], ['J3', 8.0], ['J4', 8.5],
          ['J5', 8.0], ['J6', 9.5], ['J7', 7.5], ['J8', 9.0],
          ['J9', 6.0]]
```
[2]

Task 1.2

Write a function `change_score(lst, new_lst)` that:

- takes in a nested list, `lst`, containing nine sub-lists, each storing the original score given by a judge
- takes in a second nested list, `new_lst`, where each sub-list stores the amended score made by the judge
- replaces the original scores in `lst` with the amended scores in `new_lst` where necessary
- uses the function in **Task 1.1** to return the new score obtained by a participant [5]

Write further program code to test the function with the following arguments:

```
scores = [['J1', 2.0], ['J2', 5.0], ['J3', 8.0], ['J4', 8.5],
          ['J5', 8.0], ['J6', 9.5], ['J7', 7.5], ['J8', 9.0],
          ['J9', 6.0]]

amendments = [['J8', 8.5], ['J2', 4.5], ['J7', 8.0]]
```
[2]

Save your Jupyter notebook for Task 1.

2 Name your Jupyter Notebook as

`TASK2_<your name>_<centre number>_<index number>.ipynb`

A text file, `TOP20SONG.TXT`, contains information for a list of top 20 songs in the last one year. Each line contains comma-delimited data that shows the rank, artist's name, song title and the number of weeks it was on the chart.

The task is process information on the songs.

For each of the sub-tasks, add a comment statement at the beginning of the code, using the hash symbol '#' to indicate the sub-task the program code belongs to, for example:

```
In [1] : # Task 2.1
         Program code

         Output:
```

Task 2.1

Write program code to:

- read the song information from the text file
- create a tuple `(Rank, Song_Title, Weeks_On_Chart)` for each song and append to a list [6]

Task 2.2

Write program code to display the ranks, song titles of all the 20 songs and the number of weeks they were on the chart.

The output should be in displayed in columns of fixed width.

An example is as follows:

```
Rank  Song Title                          Weeks On Chart
=================================================
1     Break My Soul                       8
2     As It Was                           19
3     Bad Habit                           6
…     …                                   …
```

[4]

Task 2.3

The `.sort()` built-in function can be used with the `lambda` function to sort a nested list based on a particular element. For example, to sort a nested list based on ascending order of the second element, the following syntax can be used.

```
sample_list.sort(key = lambda x: x[1])
```

Write program code to:

- sort the list in **Task 2.1** in descending order of the number of weeks on chart
- write the sorted list as comma-separated values into a file `top20songs.csv`, where the information on one song will form one single record. [8]

Save your Jupyter notebook for Task 2.

3 Name your Jupyter notebook as

`Task3_<your name>_<centre number>_<index number>.ipynb`

A computer program can generate a simple Sudoku puzzle using a 4 × 4 two-dimensional array.

An example of this puzzle is:

4	3	2	1
1	2	4	3
3	4	1	2
2	1	3	4

The first step to creating this puzzle is to develop a program to display the 4 × 4 two-dimensional array as a grid. This program will display the grid as:

```
4 3 2 1
1 2 4 3
3 4 1 2
2 1 3 4
```

For each of the sub-tasks, add a comment statement at the beginning of the code, using the hash symbol '#' to indicate the sub-task the program code belongs to, for example:

```
In [1] : # Task 3.1
         Program code

         Output:
```

Task 3.1

Create a program design that will declare, initialise and display the example puzzle shown. This design will:

- make use of top-down design
- include the data structure to represent the puzzle as a grid
- initialise the grid using the values shown
- make use of appropriate procedures and/or functions. [6]

Task 3.2

Write program code to display the puzzle designed in **Task 3.1**. [6]

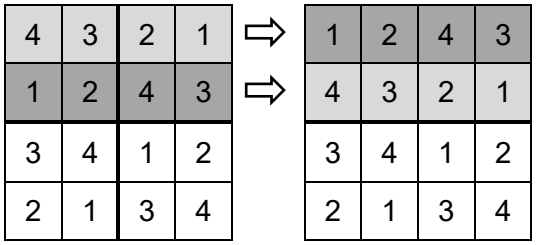
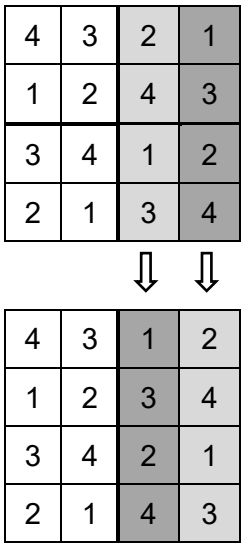
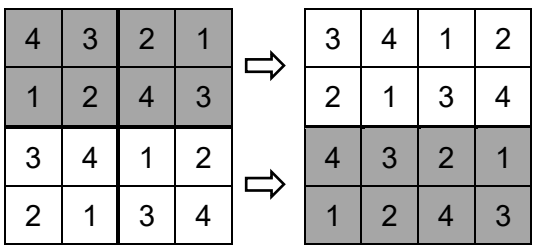
The puzzle is said to be valid if it follows these rules:

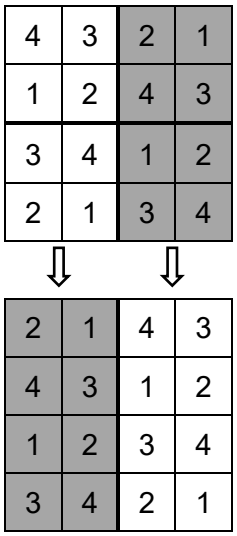
- It consists of four quadrants.
- The numbers in each quadrant must add up to ten.
- Each horizontal and vertical row of the puzzle must also add up to ten.
- No number can be repeated in the same row, same column or same quadrant of the puzzle.

A good strategy for creating puzzles is to start with a valid 'base' puzzle and perform transformations on it to create new puzzles.

You will write program code to create new valid puzzles.

Each puzzle created will have **two** randomly selected transformations, from a possible four, performed on it. The following are the four possible transformations that can be carried out.

Transformation	Explanation	
1	Swaps two rows in the same quadrants	
2	Swaps two columns in the same quadrants	
3	Swaps the top and bottom quadrant rows entirely	

4	Swaps the left and right quadrant columns entirely	

Task 3.2

Write additional program code, with **internal commentary** to identify each transformation.

The program code will:

- create a method of selecting, at random, two of the four possible transformations to be applied to the puzzle
- call a sub-program for each of the required transformations
- randomly select which rows will be transformed for transformations 1 and 2, for example, either the top or bottom two rows (for transformation 1) OR either the left-most or right-most two columns (for transformation 2) respectively
- display the puzzle before each transformation is applied and after the final transformation. Before each transformation, it will also display the name of the transformation being carried out. For example:

```
4321
1243
3412
2134

Transformation 1: Swaps two rows in the same quadrants
1243
4321
3412
2134

Transformation 4: Swaps the left and right quadrant columns
4312
2143
1234
3421
```

Run your program to obtain the outputs for each of the transformation. [18]

Save your Jupyter notebook for Task 3.