NANYANG JUNIOR COLLEGE

JC2 PRELIMINARY EXAMINATIONS

Higher 2

COMPUTING 9569/02

Paper 2 (Lab-based) **28th August 2025**

3 Hours

Additional Materials: Removable storage device
Electronic version of `Route.csv` data file
Electronic version of `Service.csv` data file
Electronic version of `Stop.csv` data file
Electronic version of `ARTEFACTS.txt` data file
Insert Quick Reference Guide

READ THESE INSTRUCTIONS FIRST

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.

Note that up to **6** marks out of 100 will be awarded for the use of common coding standards for programming style.

The number of marks is given in brackets [ ] at the end of each question or part question.
The total number of marks for this paper is 100.

[Turn over

Instruction to candidates:

Your program code and output for each of Task 1, 2 and 3 should be downloaded in a single `.ipynb` file. For example, your program code and output for Task 1 should be downloaded as

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

Make sure that each of your `.ipynb` files shows the required output in Jupyter Notebook.

1 Name your Jupyter Notebook as:

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

A number-sliding game consists of a 3-by-3 game board and 8 number tiles, as shown below:

1	2	3
4	5	6
7	8	

There is one blank on the board, into which an adjacent tile can be slid horizontally or vertically.

The coordinates of the game board comprise two integers (`x`, `y`), with `y` representing the row index and `x` representing the column index. Row indices increase downwards, while column indices increase rightwards. In the above board, the number 1 has coordinates (0, 0), the number 2 has coordinates (1, 0), the number 4 has coordinates (0, 1), and so on.

The board is represented by a `Board` class that encapsulates game data.

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

Write program code to declare the class `Board`. Each `Board` stores the following information:

- `data` – a 9-item array containing the numbers in the board.
  The blank tile is represented by the number 0.
- `blank` – the coordinates of the blank.

The class `Board` should be initialised without any arguments and should contain initialised board data identical to the above diagram. It should have the following methods:

- `get(x, y)` returns the number at coordinate (`x`, `y`).
  Coordinates can be converted into an array index `i` using the formula `i = y * w + x`, where `w` is the width of the board.
- `set(x, y, n)` stores the number `n` at coordinate (`x`, `y`). [5]

Task 1.2

The `SlidingGame` class inherits from `Board`, and implements the following methods:

- `is_valid(x, y)` returns `True` if (`x`, `y`) is a valid coordinate and the tile at (`x`, `y`) can slide into the blank, otherwise it returns `False`.
  A tile can slide if it is horizontally or vertically adjacent to the blank.
  (An array index can be converted into coordinates using `y = i // w` and `x = i % w`, where `w` is the width of the board.)
- `slide(x, y)` slides the tile at coordinate (`x`, `y`) into the blank, and returns `True`.
  After this is successfully done, the blank should now be at coordinate (`x`, `y`).
  If this is not valid or possible, then `False` should be returned instead.
- `options()` returns a list of `(x, y)` coordinates representing tiles that can slide.
- `is_complete()` returns `True` if the board is complete, otherwise it returns `False`.
  A board is complete if:
  - the blank is at the end of the array, and
  - the array (except for the last tile) is sorted in ascending order.
- `display()` displays the board contents as a 3-by-3 grid.

Write program code to declare the class `SlidingGame`. [10]

Test your code by instantiating the `SlidingGame` class. [1]

Task 1.3

The number-sliding game can be shuffled into a starting state, starting from a complete board, by:

1. checking for valid options,
2. picking a random move from step 1 and executing it,
3. repeating steps 1 and 2 until the first array value is no longer 1.

Write a function `shuffle_game(game)` that:

- takes in a `SlidingGame` instance,
- carries out the above algorithm to shuffle the board. [4]

Run your program, displaying the state of the board before and after the shuffle. [2]

Task 1.4

Write program code to implement a number-sliding game which:

1. displays the board
2. prompts the player to pick from valid tiles to slide
3. slides the tile, or re-prompts the player if their input is invalid
4. ends the game when the board is complete, and shows the player the number of moves used
5. otherwise, repeats from step 1. [5]

Save your Jupyter Notebook for Task 1.

2 Name your Jupyter Notebook as:

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

A museum maintains information about its artefacts in the following format:

`[(artefactID,rating),…]`

where:

- `artefactID` is a single uppercase letter followed by 4 digits (e.g., A1234, B5678)
- `rating` is an integer between 1 and 100 representing the historical significance

The museum's artefacts are provided in `ARTEFACTS.txt` and may be copied from the file to be used in your code. The data format provided in `ARTEFACTS.txt` is as follows:

`[(A1234,85),(B1234,85),(C1234,85),…]`

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 a function `task2_1(artefacts_list)` that:

- takes a list of tuples `artefacts_list`
- validates that each `artefactID` starts with an uppercase letter followed by a 4-digit number between 1000-9999 inclusive, outputting an appropriate error message if data is invalid
- validates that each `rating` is between 1 and 100 inclusive, outputting an appropriate error message if data is invalid
- returns a list of tuples containing valid `(artefactID, rating)` pairs. [4]

Test your function with the artefacts list in `ARTEFACTS.txt` and output:

- the total number of valid artefacts read
- the first 5 artefacts in the list. [1]

Task 2.2

Write a function `task2_2(artefacts_list)` that:

- takes a list of artefactID-rating pairs as a parameter
- implements a quicksort algorithm to sort the artefacts by rating in descending order
- uses the first element as the pivot
- returns the sorted list. [6]

Test your function with the artefacts list in `ARTEFACTS.txt`. [1]

Task 2.3

Write a procedure `task2_3(artefacts_list, target_rating)` that:

- uses your function from **Task 2.2** to sort the artefacts by rating in descending order
- implements a binary search to find an artefact with `target_rating`
- returns an `(artefactID, rating)` pair if found or `None` if not found. [8]

Test your function with:

- `target_rating = 85`
- `target_rating = 100`

Output an appropriate message for each test case. [2]

Save your Jupyter notebook for Task 2.

3 Name your Jupyter Notebook as:

`TASK3_<your name>_<centre number>_<index number>.ipynb`

A game engine uses a circular queue for its particle effects renderer. The queue uses a statically allocated array for particles. Each particle has a colour, represented as an integer from 0 – 255. A value of –1 represents a null value in a pointer.

The program uses three global variables:

- `particles` the array of particles
- `head` the index of the first particle of the queue, initialised to –1
- `tail` the index of the last particle of the queue, initialised to –1

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

The function `initialise_queue(n)` takes `n` the size of the queue as a parameter. It initialises `particles` as a 1-dimensional list of `n` integers, initialised to –1.

Write the function `initialise_queue()`. [2]

Task 3.2

The function `add_particle(colour)` takes the colour of a particle as a parameter and adds it to the queue at the tail. If the queue is full, the function displays "Queue full" and does nothing.

The function `render()` displays the colour of the particle at `head` and removes the particle from the queue. If the queue is empty, the function does nothing.

Write the functions `add_ particle()` and `render()`. [8]

Task 3.3

Test your program by initialising a queue of 20 particles. Populate the queue with 25 colours from 0 to 24. [2]

Render colours from the particle queue until it is empty. [1]

Save your Jupyter notebook for Task 3.

4 Name your Jupyter Notebook as:

`TASK4_<your name>_<centre number>_<index number>.ipynb`

A student has records about bus services in Singapore stored in comma-separated value format. They want to create a suitable database to store the data and to allow them to run searches for bus route information. The database will have three tables: a table to store data about bus stops, a table about bus services, and a table about the bus routes. The fields in each table are:

`Service`:

- `ServiceNo` – The bus service (e.g. 118A)
- `Operator` – The bus operator company (e.g. SBST)
- `Category` – The type of route

`Stop`:

- `RoadName` – The road that the bus stop is on
- `Description` – A short description of the bus stop location
- `BusStopCode` – The unique code of the bus stop

`Route`:

- `ServiceNo` – The bus service of the route
- `Operator` – The bus operator
- `Direction` – The direction of the route (1 or 2 only)
- `StopSequence` – The stop's numerical sequence (first stop = 1, second stop = 2, ...)
- `BusStopCode` – The unique code of the bus stop

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 4.1
         Program code
         Output:
```

Task 4.1

Write a Python program that uses SQL code to create the database `BUSROUTES` with the three tables given. Define the primary and foreign keys for each table. [9]

Task 4.2

The text files `Route.csv`, `Service.csv` and `Stop.csv` store the comma-separated values for each of the tables in the database.

Write a Python program to read in the data from each file and then store each item of data in the correct place in the database. [6]

Task 4.3

Write a Python program to input a bus service number and return its routes, showing:

- the direction of the route
- the bus stop code of each stop on the route
- the road name of the bus stop
- the description of the bus stop
- in ascending order by direction and by stop sequence. [6]

Test your program by running the application with bus service `506`. [2]

Save your Jupyter Notebook for Task 4.

Task 4.4

Write a Python program and the necessary files to create a web application that allows the user to input a bus stop code. The web application displays the bus services that stop at that bus stop, along with the bus operator. If the bus stop code is invalid, it displays an appropriate error message instead.

Save your Python program as:

`TASK_4_4_<your name>_<centre number>_<index number>.py`

with any additional files/ subfolders in a folder named:

`TASK_4_4_<your name>_<centre number>_<index number>` [7]

Run the web application with the following inputs:

- `01111`
- `01112`

Save the webpage output for each input as:

`TASK_4_4_<your name>_<centre number>_<index number>_1.html`

`TASK_4_4_<your name>_<centre number>_<index number>_2.html` [2]

-- END OF PAPER --