

TEMASEK JUNIOR COLLEGE

2025 JC2 PRELIMINARY EXAMINATION

Higher 2



COMPUTING

9569/02

Paper 2 (Lab-based)

28 August 2025

3 hours

Additional Materials:

- Removable storage device
- Electronic version of `Artefacts.txt` data file
- Electronic version of `level_order.py` Python file
- Electronic version of `Busroutes.db` data file
- Electronic version of `Route.csv` data file
- Electronic version of `Service.csv` data file
- Electronic version of `Stop.csv` 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.

This document consists of **9** printed pages, **3** blank pages and **1** Insert.

Instruction to candidates:

Your program code and output for each of Task 1, 2, 3 and 4 should be saved in a single `.ipynb` file using Jupyter Notebook. 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`

A museum maintains information about its artefacts in a text file `ARTEFACTS.txt` in the following format:

`[(artefactID 1, rating 1), (artefactID 2, rating 2), ...]`

where:

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

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 a function `task1_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 from 1000 to 9999, outputting an appropriate error message if data is invalid
- validates that each `rating` is an integer from 1 to 100, 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 1.2

Write a function `task1_2(artefacts_list)` that:

- takes a list of artefact tuples 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 valid artefacts.

[1]

Task 1.3

Write a function `task1_3(artefacts_list, target_rating)` that:

- uses your function from **Task 1.2** to sort the valid artefacts by rating in **descending** order
- implements a recursive binary search to find **all** the artefacts with `target_rating`
- returns the tuple `(ID, rating)` 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]

Task 1.4

Write a function `task1_4(artefacts_list)` that:

- takes a list of artefact tuples as a parameter
- calculates and returns a tuple containing:
 - the average historical significance rating, rounded to 2 decimal places
 - the count of artefacts with ratings above the average
 - a list containing the IDs of the 3 least significant artefacts
- returns `None` if the input list is empty.

[5]

Test your function with the valid artefacts and display the output.

[3]

Save your Jupyter Notebook.

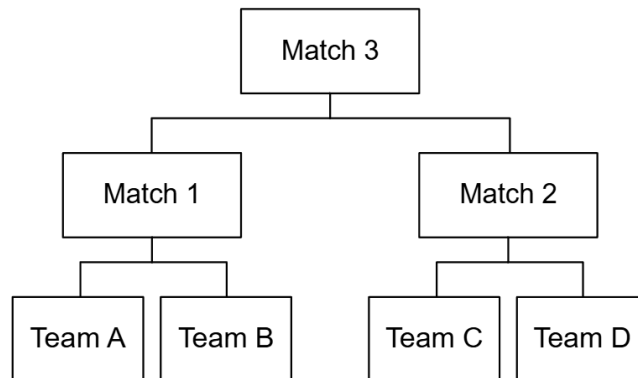
2 Name your Jupyter Notebook as:

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

Write a program to represent the matches in a sports tournament as a binary tree where:

- each leaf node represents a participating team
- each internal node represents a match between two winners from lower rounds
- the root node represents the final match (champion).

An example of a binary tree used to represent matches in a sports tournament involving 4 participating teams is as follows:



The program uses object-oriented programming to implement the binary tree and its nodes.

The class `Node` contains three attributes:

- `team`: name of the team or match stored at the node
- `left_ptr`: pointer to the left child of the node, `None` if there is no left child
- `right_ptr`: pointer to the right child of the node, `None` if there is no right child

The class `Node` has a constructor that initialises the three attributes.

The class `TournamentTree` is a binary tree that represents the matches in the tournament and contains one attribute:

- `root`: pointer to the root node of the tree, `None` if the tree is empty.

The class `TournamentTree` has the following methods:

- a constructor that initialises the `root` pointer to an appropriate value
- `create_tree()` that takes the number of teams, `num_of_teams`, as a parameter and create the binary tree representing the matches. You may assume that the number of teams will not exceed 26
- `level_order()` that displays the names of the matches or teams by levels of the binary tree
- `pre_order()` that displays the names of the matches or teams by performing pre-order traversal on the binary tree

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 declare the class `Node` and its constructor. [2]

Task 2.2

Write program code to declare the class `TournamentTree` and its constructor. [2]

Task 2.3

Amend your code for **Task 2.2** to declare the `TournamentTree` method `create_tree()` and add in the code provided for `level_order()` from the file `level_order.py`

In a new cell, write program code to

- initialise a new `TournamentTree`
- call `create_tree()` to create a tree with 8 participating teams, named Team A - H
- call `level_order()` to display the tree by level. [8]

Test your program and show the output. [1]

Task 2.4

Amend your code for **Task 2.2** to declare the `TournamentTree` method `pre_order()`

In a new cell, write program code to call `pre_order()` to perform pre-order traversal. [2]

Test your program and show the output. [1]

Save your Jupyter Notebook.

3 Name your Jupyter Notebook as:

TASK3_<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 3.1
Program code
```

Output:

Task 3.1

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 coordinates (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 coordinates (x , y).

[5]

Task 3.2

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`. [9]

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

Task 3.3

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_board(board)` that:

- takes in a `SlidingGame` instance,
- carries out the above algorithm to shuffle the board. [3]

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

Task 3.4

Write program code to implement the number-sliding game on the shuffled board from **Task 3.3** by:

1. Displaying the board
2. Prompting the player to pick from valid tiles to slide
3. Sliding the tile, or re-prompting the player if their input is invalid
4. Ending the game when the board is complete and showing the number of moves used.
5. Otherwise, repeat from step 1. [5]

Save your Jupyter Notebook.

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. He created the database, `Busroutes.db`, to store the data and to allow them to run searches for bus route information. The database has 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
- `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

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 4.2

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 168.

[2]

Save your Jupyter Notebook.

Task 4.3

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_3_<your name>_<centre number> <index number>.py

with any additional files/ subfolders in a folder named:

TASK_4_3_<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_3_<your name> <centre number> <index number>_1.html

TASK_4_3_<your name> <centre number> <index number>_2.html

[2]

- End of Paper -

[BLANK PAGE]

[BLANK PAGE]

[BLANK PAGE]