



YISHUN INNOVA JUNIOR COLLEGE
JC 2 PRELIMINARY EXAMINATION
Higher 2

CANDIDATE
NAME

CG

INDEX NO

COMPUTING

Paper 2 (Lab-based)

9569/02

2 Sep 2025

3 hours

Additional Materials: Removable storage device with the following files:

- Prelim_template.ipynb
- DATASET.TXT
- secured_module.py
- a folder Q5_WebApp containing sportsclub.db and
SessionId_2504_ATTENDANCE.TXT

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.

The number of marks is given in brackets [] at the end of each task.

The total number of marks for this paper is **100**.

This document consists of **18** printed pages.

2

Instruction to candidates:

Your program code and output for Questions 1, 2, and 3 should be saved in a single `.ipynb` and named as

`Prelim_<your class>_<your name>.ipynb`

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:

In [2]:

```
#Task 1.2  
Program Code
```

Output:

In [3]:

```
#Task 1.3  
Program Code
```

Output:

- 1 More than 27000 spectators are expected to attend the National Day Parade (NDP). Each Singaporean can apply for 2, 4, or 6 tickets, with seats grouped together in the same row.

The seats are grouped into four coloured zones: Red (R), Green (G), Blue (B), and Yellow (Y). Each zone has 10 areas labelled from A to J. Each area has 20 rows (R01–R20) of 40 seats (01–40).

For example, a seat code 'B-E20-26' refers to a seat in the Blue Zone, Area E, Row 20 and Seat 26.

During the NDP, volunteers will be stationed at the entrances to assist in scanning the spectators' NRIC cards to retrieve their seat allocations and direct them to their seat locations.

Task 1.1

Write program code for a function `hash(string)` to compute and return the hash value for a given string, using the following algorithm:

- For each character in the string, calculate its ASCII value to the power of its position. The leftmost character in the string is at position 1
- Sum all these values
- Return the total modulo 1000. [2]

Task 1.2

Write program code to read the ticket information from `DATASET.TXT` and store all the information in a tuple of tuples named `data`. Each line in the text file contains an NRIC number, the number of tickets allocated, and the first seat code, separated by commas.

Each individual tuple should contain an NRIC number, the number of tickets allocated, and the first seat code. The output of `data` is similar to the following:

For example,

```
data = (('S9383870Z', 4, 'B-E20-36'),
        ('T1583391I', 2, 'B-C12-14'))
```

[4]

Task 1.3

Write program code for a function `store(data)` to:

- create a hash table of size 1000
- for each individual tuple in `data`, compute the hash value for the NRIC number and use it as an index to locate the position in the hash table
- if the position is empty, store the individual data tuple at that position
- otherwise, at that position, create a list to store multiple individual tuples. [5]

To further enhance the security of personal data, a secured hash function `secured_hash(string)` is used to generate a secured hash table for the actual NDP event.

The following methods are provided in the Python file `secured_module.py`:

- `secured_hash(string)`: to securely hash the string and return an integer between 0 and 999 (inclusive)
- `data()`: to return the secured hash table of size 1000 containing all the ticket holders' information.

Task 1.4

Write program code for the function `query(tbl, nric)` to search the secured hash table `tbl` using the NRIC number and return the individual data in a tuple in the following format:

`(NRIC number, number of tickets allocated, the first seat code)`

If the NRIC number is not found in `tbl`:

- display the following statement:
`'No ticket was allocated to this NRIC number.'`
- return `None`.

Test Cases

```
>>> query(secured_hash_table, 'S9114928B')
No ticket was allocated to this NRIC number.
```

```
>>> query(secured_hash_table, 'S9114928A')
('S9114928A', 4, 'Y-I19-28') [5]
```

Task 1.5

Write program code for a function `display(tup)` to display the information stored in the tuple `tup`.

Test Case

```
>>> tup = ('S9383870Z', 4, 'B-E20-36')
>>> display(tup)
```

Output:

```
'4 tickets allocated to NRIC: xxxxx870Z, and the seats are located
at the Blue Zone, Area E, Row 20, and Seats from 36 to 39.'
```

 [2]

- 2 A teacher has 20 students in her class. The names and dates of birth, in dd-mm-yyyy format, are stored in a list of tuples `students`:

```
students = [('Emma', '12-05-2004'), ('Evelyn', '12-05-2008'),  
            ('Amelia', '14-02-2006'), ('Ava', '14-02-2004'),  
            ('Lucas', '29-08-2003'), ('Benjamin', '19-09-2005'),  
            ('Olivia', '07-01-2003'), ('Grace', '09-12-2003'),  
            ('Ethan', '19-09-2003'), ('Michael', '07-01-2009'),  
            ('Noah', '12-05-2006'), ('James', '22-03-2003')]
```

Task 2.1

Write program code for the function `merge_sort(seq)` using the Merge Sort algorithm to return a new list of the students sorted in ascending order of their names. [6]

Task 2.2

Write program code for the function `sort_dob(seq)` using the function `merge_sort(seq)` written in **Task 2.1** to return a new list of the students sorted in descending order of their ages. [4]

- 3 A beginner archer practises his shots on a 9 x 9 target board. The board is laid out as a grid with rows and columns numbered from 1 to 9. The bullseye is located at the centre of the board, at coordinates (5, 5) .

	1	2	3	4	5	6	7	8	9
1	•	•	•	•	•	•	•	•	•
2	•	•	•	•	•	•	•	•	•
3	•	•	•	•	•	•	•	•	•
4	•	•	•	•	•	•	•	•	•
5	•	•	•	•	O	•	•	•	•
6	•	•	•	•	•	•	•	•	•
7	•	•	•	•	•	•	•	•	•
8	•	•	•	•	•	•	•	•	•
9	•	•	•	•	•	•	•	•	•

Bullseye at (5,5)

A target board is displayed with 9 rows of 9 dots, with labels 1 to 9 at the top and left side of the grid. The top left corner dot has $x=1$ and $y=1$. The bullseye at position (5, 5) is marked with the uppercase letter 'O'.

Task 3.1

Write program code to initialize a 2D array to represent the grid.

Write program code for the function `print_grid()` to display the labels, the dots, and the bullseye 'O' using the 2D array grid created. [4]

The archer begins with an initial skill level represented by a variable d , which limits how far the arrow can land from where the archer aims. The archer begins with an initial skill level of $d=5$, and he chooses where to aim by entering the x - and y -coordinates (x_1, y_1) .

The simulation program generates the actual strike point of the arrow (x_2, y_2) where:

- x_2 is a random integer between (x_1-d) and (x_1+d) (inclusive)
- y_2 is a random integer between (y_1-d) and (y_1+d) (inclusive).

Depending on the outcome, the output can be one of the following:

- If the strike point (x_2, y_2) is outside the grid (i.e. not in the range 1 to 9 for both x and y), an output message 'You have missed the target board. Try again.' is printed.
- If the strike point is within the grid, the grid is printed with the hit coordinates (x_2, y_2) marked with an 'X', if it is not the bullseye 'O'
- If the strike point is on the bullseye 'O', the grid is printed with the bullseye 'O' replaced with a 'H'.

Task 3.2

Write program code to:

- prompt the archer to enter the x - and y -coordinates as his aiming coordinates
- generate the strike point (x_2, y_2)
- if the strike point is within the grid, display the grid marking the strike point with an 'X'
- if the strike point is outside the grid, display the appropriate message. [5]

To compute the distance between (x_1, y_1) and (x_2, y_2) , the following formula can be used.

$$\text{distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Task 3.3

Write program code to:

- compute the distance between (x_1, y_1) and (x_2, y_2) using the formula above
- if the distance is less than d , update d to this smaller value to simulate the archer improving his aiming precision.

[2]

Task 3.4

Write program code to simulate the beginner archer practising with the simulation program until he hits the bullseye.

Your program code should include the following:

- print a new target grid at the beginning of the training
- prompt the archer to enter the aiming coordinates or choose to quit the training by entering $(0, 0)$ as his aiming coordinates
- generate the coordinates of the strike point
- print an updated grid with the strike point marked with 'X' if it is within the grid
- calculate the distance between the aiming coordinates and the strike point and update the value of d , if necessary
- repeat prompting the archer to enter new aiming coordinates until he hits the bullseye or when he chooses to quit the training
- display the number of attempts made by the archer when he hits the bullseye or chooses to quit.

[12]

- 4 A programmer implemented a queue data structure with Object-Oriented Programming (OOP) to design a messaging system, Message of the Day (MOTD), for teachers to create and broadcast announcements that will be shown to students on their personal learning devices (PLDs).

The class `Message` contains three attributes:

- `title` the title of the message
- `content` the contents of the message
- `next` points to the next `Message` object.

Task 4.1

Write program code to declare the class `Message`, its constructor, and its methods.

[3]

The class `MOTD` contains one attribute:

- `root` the pointer pointing to the first message object in the queue.

The class `MOTD` has the following methods initially:

- a constructor that initialises the pointer `root` to `None` when the queue is empty.
- `add_msg(title, content)` adds a `Message` object to the queue and returns `True`
- `display()` prints an ordered list of the titles of the messages, according to the `Message` objects in the queue. If there are no messages in the queue, print 'No message in system.'.

Task 4.2

Write program code to declare the:

- class `MOTD`
- constructor, `add_msg()`, and `display()` methods.

[7]

The table below contains the messages to be added to the MOTD queue according to the following order:

	Title	Content
1	'Title A'	'Content A'
2	'Title B'	'Content B'
3	'Title C'	'Content C'
4	'Title D'	'Content D'
5	'Title E'	'Content E'

Task 4.3

Write program code to:

- initialise a new MOTD queue as `motd1`
- add the five messages to the `motd1` queue

Test your program and use the `display()` method to display all the messages in the `motd1` queue.

Test Case

```
>>> motd1.display()
```

Output:

```
Title A -> Title B -> Title C -> Title D -> Title E
```

[2]

To ensure that more important messages are displayed first, a priority rating ranging from 1 to 5 is added in the MOTD system. The value 1 denotes the most important and 5 denotes the least important. If there are two or more messages with the same priority rating, the messages will be displayed according to the order they were added to the MOTD queue.

When the students log in to the computer network, the MOTD application will display the messages automatically on their PLDs.

Task 4.4

Modify the class `Message` by adding a new attribute `priority` which is the priority rating of the message, and its associated methods.

Modify the class `MOTD` by adding an additional method:

- `add_msg_priority(title, content, priority)` adds a `Message` object to the queue in the order of the priority, from 1 to 5, and returns `True`
- `broadcast()` removes the messages one by one according to their order in the queue and display the messages, with the titles and contents. [9]

The table below contains the messages to be added to the MOTD queue according to the following order:

	Title	Content	Priority
1	'Title A'	'Content A'	5
2	'Title B'	'Content B'	3
3	'Title C'	'Content C'	1
4	'Title D'	'Content D'	2
5	'Title E'	'Content E'	3

Task 4.5

Write program code to:

- initialise a new MOTD queue as `motd2`
- use `add_msg_priority()` to add the five messages to the `motd2` queue

Test your program by:

- calling `display()` on `motd2` queue
- calling `broadcast()` on `motd2` queue
- calling again `display()` on `motd2` queue

The following three test cases are to be executed consecutively:

Test Case

```
>>> motd2.display()
```

Output:

```
Title C -> Title D -> Title B -> Title E -> Title A
```

Test Case

```
>>> motd2.broadcast()
```

Output:

```
1. Title C: Content C
2. Title D: Content D
3. Title B: Content B
4. Title E: Content E
5. Title A: Content A
Last message printed.
```

Test Case

```
>>> motd2.display()
```

Output:

```
No messages in system.
```

[2]

- 5 A recreational sports club in the college organises training sessions for three new sports and encourages student members to participate regularly.

The three new sports are Lawn Bowls, Pickleball and Tchoukball.

The attendance, the sessions' information and members' data are stored in a relational database.

The database `sportsclub.db` has the following tables:

Member:

- `Id` – unique code for each member, for example 'M2503'
- `Name` – the name of the member
- `Class` – the class of the member

Session:

- `Id` – unique integer code that is auto-generated for each session, for example 2501
- `Date` – the date of the session, for example '2025-03-26'
- `Sport` – the sport training conducted during the session

Attendance:

- `MemberID` – the unique member code
- `SessionID` – the unique session code
- `Status` – the attendance of the member for the session, i.e. '1' for present and '0' for absent.

Task 5.1

A training session (session id: 2504) was conducted on 14th Feb 2025 for the sport Lawn Bowls and the attendance was stored in a text file `SessionID_2504_ATTENDANCE.TXT`.

Write Python program code to insert the session information and the attendance records into their respective tables in the database.

Save your Python program as `Task5_1_<your name>.py`.

[4]

A web application is to be designed to allow the teacher to view and update the attendance.

The default webpage `index.html` will display the following menu:

Sports Club Attendance

1. [Members List](#)
2. Choose a sport to display the attendance:
3. [Absentees List](#)
4. [Mark Attendance](#)

Task 5.2

Write a Python program and the necessary files to create a web application to display **only the first option** (1. `Members List`) in the menu for the teacher to view the list of members in the Sports Club.

The program should return an HTML document `members.html` that enables the web browser to display a table containing the Class and Name of the members.

Save your Python program as:

`Task5_2_<your name>.py`

with any additional files / subfolders in a folder named:

`Task5_2_<your name>`

Run the web application.

Save the webpage output as:

`Task5_2_<your name>_output.html`

[6]

Task 5.3

Modify your Python program and the default webpage `index.html` to include the **second option** (2. Choose a sport ...) in the menu for the teacher to select to view the attendance for a particular sport.

The program should return an HTML document `attendance.html` that enables the web browser to display a table containing the following data:

- session dates
- member names
- attendance statuses.

Save your Python program as:

`Task5_3_<your name>.py`

with any additional files / subfolders in a folder named:

`Task5_3_<your name>`

Run the web application.

Save the webpage output as:

`Task5_3_<your name>_output.html`

[5]

Task 5.4

Modify your Python program and the default webpage `index.html` to include the **third option** (3. Absentees List) in the menu for the teacher to select to view the list of absentees for all the sessions.

The program should return an HTML document `absent.html` that enables the web browser to display a table containing the following data:

- member names
- member classes
- session dates

The data in the table should be sorted by the members' names.

Save your Python program as:

`Task5_4_<your name>.py`

with any additional files / subfolders in a folder named:

`Task5_4_<your name>`

Run the web application.

Save the webpage output as:

`Task5_4_<your name>_output.html`

[3]

During the training session, the teacher will mark the attendance of the members.

An example of the webpage is shown below:

Members List for Attendance Marking

Session Date (yyyy-mm-dd):

Sport:
Please Select

Class	Name	Present (?)
25A21	Yi Xuan	☐
25A21	Ravi	☐
25A21	Zhi Yong	☐
25S31	Aisyah	☐
25S31	Nurul	☐
25S31	Jun Hao	☐
25S32	Azman	☐
25S32	Jia Hui	☐
25S32	Anjali	☐

Submit

Task 5.5

Modify your Python program and the default webpage `index.html` to include the **fourth option** (4. Mark Attendance) in the menu for the teacher to view the members' list and mark the attendance.

The program should return an HTML document `mark.html` that enables the web browser to display the following:

- a text box to enter the session's date in the `yyyy-mm-dd` format
- a list to select one of the three sports
- a table with three columns:
 - member classes
 - member names
 - a column for the teacher to indicate if the member is present for the training session.

The data in the table should be sorted by the members' classes.

The submitted session information and the attendance records should be updated into their respective tables in the database.

After updating the database, the program should return an HTML document `success.html` that enables the web browser to display the following message:

"Attendance Updated."

Save your Python program as:

`Task5_5_<your name>.py`

with any additional files / subfolders in a folder named:

`Task5_5_<your name>`

Run the web application.

Save the attendance taking webpage output as:

`Task5_5_<your name>_output.html`

[8]

END OF PAPER