

Name: _____

Class: _____



JURONG PIONEER JUNIOR COLLEGE

JC2 Preliminary Examination 2025

**COMPUTING
Higher 2**

9569/02

22 August 2025

Paper 2 (Practical)

3 hours

Additional materials:

Cover Page
Electronic version of TASK1.csv
Electronic version of STUDENT.txt
Electronic version of STALL.txt
Electronic version of DISH.txt
Electronic version of ORDERDISH.txt
Insert Quick Reference Guide

READ THESE INSTRUCTIONS FIRST

Answer **all** the 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 task.

The total number of marks for this paper is 100.

This document consists of **11** printed pages and **1** blank page.

Instructions to candidates:

Your program code and output for each of Task 1 to 4 should be downloaded in a single .ipynb file. For example, your program code and output for Task 1 should be downloaded as TASK1_<your class>_<your name>.ipynb.

- 1 You are designing a simple text-based treasure-hunting game. After each game session, the player collects treasures. Each treasure has:
 - A name (string)
 - A value in gold coins (integer)

You need to implement a system that allows the player to sort and search their treasure inventory.

Task 1.1

Write a function `readData(filename)` that takes a string `filename` which represents the name of a csv file, reads in the contents of the file, and returns the content.

For example:

```
[["Sapphire Crown", 230], ["Rusty Dagger", 75], ["Dragon Scale", 480]]
```

Call your function `readData` with the file `TASK1.csv` and display the returned list.

[5]

Task 1.2

Write a function `quicksort(treasures)` that takes the returned list from **Task 1.1**, implements a **recursive** quicksort algorithm (sorted by its treasure name in ascending order lexicographically, and returns the sorted list.

For example:

```
[["Dragon Scale", 480]], ["Rusty Dagger", 75], ["Sapphire Crown", 230]]
```

Call your function `quicksort` with the list from **Task 1.1** and display the returned ordered list.

[6]

Task 1.3

Write a function `binarySearch(treasures, name)` that takes the returned list from **Task 1.2**, implements an **iterative** binary search on the name. If the name exists, it returns the value of the treasure. Otherwise, it returns -1.

Call your function `binarySearch` with the list from **Task 1.2** to search for "Dragon Scale" and display the returned value.

[3]

Task 1.4

Write a Python program that will:

- load the data from file TASK1.csv
- display the menu and prompt for user choice
- execute the option based on user choice by calling the appropriate functions defined in **Tasks 1.1 to 1.3**
- repeat the above steps to display the menu and execute user choice
- exit the program when user enters option 4.

For “Search Treasure by Name” option, if the treasure name cannot be found, display the message “Treasure cannot be found”.

Sample Output:

```

Welcome to Your Treasure Inventory!
1. View Inventory
2. Sort Treasures by Name
3. Search Treasure by Name
4. Exit

Choose option: 1
Sapphire Crown (Value: 230)
Rusty Dagger (Value: 75)
Dragon Scale (Value: 480)

Welcome to Your Treasure Inventory!
1. View Inventory
2. Sort Treasures by Name
3. Search Treasure by Name
4. Exit

Choose option: 2
Dragon Scale (Value: 480)
Rusty Dagger (Value: 75)
Sapphire Crown (Value: 230)

Welcome to Your Treasure Inventory!
1. View Inventory
2. Sort Treasures by Name
3. Search Treasure by Name
4. Exit

Choose option: 3
Enter treasure name to search: Dragon Scale
Found 'Dragon Scale' worth 480 gold coins!

Welcome to Your Treasure Inventory!
1. View Inventory
2. Sort Treasures by Name
3. Search Treasure by Name
4. Exit

Choose option: 4
Goodbye!

```

Run your program to test all the four menu options and display the output.

Save your Jupyter Notebook for Task 1.

[6]

[Turn over

- 2 When a user visits webpages, their browsing history is stored as a linked list in chronological order (oldest to newest). Additionally, the system supports Back and Forward navigation using two stacks, both implemented using arrays.

- The Back Stack stores the pages the user navigated. The top of Back Stack is the current page navigated.
- The Forward Stack stores the pages popped when the user presses "Back", and these can be restored using the "Forward" button.

The class `PageNode` contains two properties:

- `url` is the string data in the node
- `next` points to the next node.

The class `BrowserHistory` manages a user's browsing history as a linked list. When the user visits a webpage, a `PageNode` object is inserted. It contains one property:

- `head` points to the first node.

The class `BrowserHistory` contains the following methods:

- a constructor to set the head pointer to `None`
- `visit_page(url)` to add the node with `url` data to the end of the linked list
- `print_history()` to output all `url` data from first visited to last visited.

Task 2.1

Write program code to declare the class `PageNode` and its constructor.

Write the program code to declare the class `BrowserHistory` and its methods.

[8]

The class `BrowserNavigator` contains the following properties:

- `current_page` is the URL string of the currently viewed page
- `back_stack` is the stack array that stores up to 10 URL strings of pages visited before the current one
- `forward_stack` is the stack array that stores up to 10 URL strings of pages that were navigated away from by going back
- `back_top` is the top pointer of the back stack
- `forward_top` is the top pointer of the forward stack.

The class `BrowserNavigator` contains the following methods:

- a constructor to set the properties to appropriate `NULL` values
- `go_back()` to push the current page onto the forward stack, then pop the top page from the back stack and assign it to the current page
- `go_forward()` to push the current page onto the back stack, then pop the top page from the forward stack and assign it to the current page
- `visit_new_page(url)` to push the current page onto the back stack, clear the forward stack, and then assign the URL string to the current page.

Task 2.2

Write the program code to declare the class `BrowserNavigator` and its methods. [10]

Task 2.3

Write the main program to:

- declare a new instance of `BrowserHistory`
- declare a new instance of `BrowserNavigator`
- update the browser history and browser navigator when the user
 - o visits `"jpjc.coursemology.org"`, `"jpjc.moe.edu.sg"`, and followed by `"web.whatsapp.com"`
 - o goes back twice
 - o goes forward once
- output the URL of the current page
- output the URLs of the full browsing history. [6]

Save your Jupyter Notebook for Task 2.

- 3 You are developing a locker management system for a local gym. Each locker is associated with a unique locker ID, which is an integer. These IDs are not guaranteed to be sequential or small — customers may be assigned or choose IDs such as 101, 203, 550, etc.

To efficiently store and retrieve locker access information regardless of how large or scattered the IDs are, the system uses a hash table array of fixed size **37**.

For privacy and storage efficiency, all codewords are first encrypted using a Caesar cipher (+3 shift) and then stored in the hash table.

However, since multiple locker IDs may hash to the same index, your system must use **linear probing** to resolve collisions.

Task 3.1

Write a function `encrypt(code)` to:

- encrypt the codeword `code` using a Caesar cipher with +3 shift which wraps around after 'z' (e.g., 'z' → 'c')
- return the encrypted string.

[5]

Task 3.2

Write a function `hashfunction(locker_id)` to:

- implement a hash function using $\text{hash_index} \leftarrow \text{locker_id} \bmod 37$
- return the hash index.

[1]

Task 3.3

A locker record structure is used to store a locker ID and an encrypted codeword.

Write a function `add(locker_id, hta)` to:

- get user to input a valid codeword (only lowercase alphabetic characters are allowed)
- use your function from **Task 3.1** to encrypt the codeword
- store the locker record (i.e.: `locker_id` and the encrypted codeword) in the hash table array `hta`
- return the updated hash table array.

[8]

Task 3.4

Write a procedure `search(hta)` to:

- get the user to enter his/ her locker ID and codeword
- use your function from **Task 3.1** to encrypt the codeword
- perform a hash table search
- display "LOCKER NOT FOUND" if the locker ID does not exist in the hash table
- display "INCORRECT CODE" if the locker ID is found but the encrypted codeword in the locker record is not a match
- display "LOCKER IS OPEN" if the locker ID is found and the encrypted codeword in the locker record is a match

[6]

Task 3.5

Write a program to:

- initialise a hash table array with None as the NULL values
- add locker (locker ID 36 and user input codeword "jurongpioneer")
- add locker (locker ID 73 and user input codeword "computing")
- search locker (user input locker ID 73 and codeword "computing")
- search locker (user input locker ID 74 and codeword "computing").

Test your program and show the output.

[4]

Save your Jupyter Notebook for Task 3.

- 4 JP canteen operates various food stalls in school and wishes to implement a web-based food ordering system to better understand students' food taste and facilitate food ordering. Students can browse dishes online, place food orders, and collect their meals when ready. The system should allow the storage of student details, stall and dish information, and order history.

There are a total of **four** tables, and the table descriptions are as follows:

- student(studentID, name, phoneNo)
- stall(stallID, stallName)
- dish(dishID, stallID*, dishName, price, availability)
- orderDish(studentID*, dishID*, orderTime, orderDate, quantity)

The primary key is indicated by underlining one or more attributes.

The attribute indicated with * denoted a foreign key.

It is given that:

- students are allowed to order different dishes in a single transaction. For example, when a student orders **three** different dishes in a single transaction, **three** separate records with the same value for studentID, orderTime, orderDate and quantity will be inserted into the table orderDish
- the primary keys stallID and dishID of the tables stall and dish are auto-incremented integers.
- orderDate and orderTime in table orderDish are represented in the format of YYYY-MM-DD and HH:MM respectively.

Task 4.1

Create a SQL file named TASK4_1_<yourclass>_<yourname>.sql to show the SQL code to create database canteen.db with the **four** tables given. Define the primary and foreign keys for each table.

[4]

Task 4.2

The files `STUDENT.txt`, `STALL.txt`, `DISH.txt` and `ORDERDISH.txt` contain information about the students, stalls, dishes and order history respectively for insertion into the canteen database. Each row in the four files is a comma-separated list of information.

FOR `STUDENT.txt`, information about each student is given in the following order:

NRIC, Name, PhoneNo

Due to the sensitive nature of the National Registration Identity Card (NRIC), and because it can be used to access a range of personal data, it has been decided that a unique identifier for each student will be obtained by combining the last four characters of the NRIC with the first four characters of the student's name, with white spaces removed. All letters in `studentID` will be made uppercase.

For example, a student with `NRIC = "S2345678B"` and `name = "Low Jia Ying"` will be given a `studentID` of `"LOWJ678B"`.

For `STALL.txt`, information about each stall is given in the following order:
`stallID`, `stallName`.

For `DISH.txt`, information about each dish is given in the following order:
`dishID`, `stallID`, `dishName`, `price` and `availability`.

For `ORDERDISH.txt`, information about the dishes ordered by individual students over time.

Write a Python program to insert all information from the **four** files into the **four** separate tables of the canteen database, `canteen.db` based on the table descriptions given above. Run the program. [8]

Save your program code as

`Task4_2_<yourclass>_<yourname>.py`

Task 4.3

Write SQL code to show the dish names and prices of all the **available** dishes with a specified stall name. Run this query with the store name "Spicy Delight".

[3]

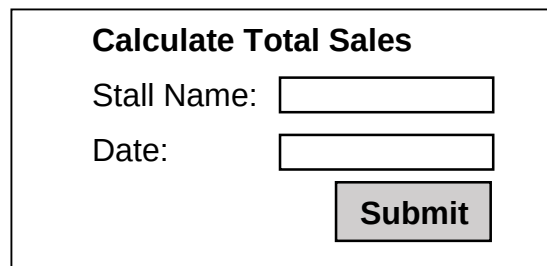
Save this code as.

Task4_3_<yourclass>_<yourname>.sql

Task 4.4

The stall owners of JP Canteen requested that they would like to know their total sales amount collected on a particular date.

Write a Python program and the necessary files to create a web application. The web application accepts **two** input Stall Name and Date from the web form in separate textboxes. The form design of the web application is shown below:



The form is titled "Calculate Total Sales". It contains two input fields: "Stall Name:" and "Date:". Below these fields is a "Submit" button.

[3]

Save your program code as

Task4_4_<yourclass>_<yourname>.py

With any additional files/subfolders as needed in a folder named

Task4_4_<yourclass>_<yourname>

Run the web application and save the output of the program as

Task4_4_<yourclass>_<yourname>.html

Task 4.5

Write an SQL query that sums up the total sales amount collected by a stall specified by stallName on a given date.

The results of the query should be shown on a webpage in a table that displays the following data:

- Stall ID
- Stall Name
- Date
- Total Sales Collected

The program should return a HTML document that enables the web browser to display a table with required data. The resulting web page is accessible by clicking on the "Submit" button on the web form in **Task 4.4**.

The program should return an HTML document that enables the web browser to display a table with the required data.

[6]

Save your program code as

Task4_5_<yourclass>_<yourname>.py

With any additional files/subfolders as needed in a folder named

Task4_5_<yourclass>_<yourname>

Run the web application and save the output of the program as

Task4_5_<yourclass>_<yourname>.html

Task 4.6

Run your web application by entering the following input into the web form through its "Submit" button.

- Stall Name = "Nasi Padang"
- Date = "2025-07-04"

[2]

Run the web application and save the output of the program as

Task4_6_<yourclass>_<yourname>.html

END OF PAPER

BLANK PAGE