



RAFFLES INSTITUTION

2025 YEAR 6 PRELIMINARY EXAM

CANDIDATE
NAME

CLASS

25

COMPUTING

9569/02

Paper 2 (Lab-based)

3 hours

Additional materials: Electronic version of `student_codes.txt` data file
Electronic version of `questions.csv` file
Electronic version of `client.ipynb` file
Electronic version of `lost_items.json` file
Electronic version of `scheduler.py` file
Electronic version of `announcement.txt` 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 task.

The total number of marks for this paper is 100.

FOR EXAMINER'S USE				
Task 1	Task 2	Task 3	Task 4	
				TOTAL
				100

This document consists of **15** printed pages and **1** blank page.

Instructions to candidates:

Your program code and output for each of Task 1 and 2 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>_<index number>.ipynb`

1 Name your Jupyter Notebook as:

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

Your school is organising an Inter-House Sports Carnival, where students from five different houses compete in friendly games.

Student participants information is stored in `student_codes.txt`. Each student is identified by a code that combines their house name and student number, using the format: `<HouseCode><StudentNumber>`

where

- `<HouseCode>` is one of the 5 houses, in 2 letter code:
 - BW representing Bailey-Waddle
 - BB representing Buckle-Buckley
 - HH representing Hadley-Hullet
 - MR representing Morrision-Richardson
 - MT representing Moor-Tarbet
- `<StudentNumber>` is an integer between 1 and 99

Examples of student codes: "BW12", "MR5", "HH88"

The organising committee has decided not to group students by house. Instead, the games are organised into two teams based on student numbers:

- Students with even student numbers will join Team Blue
- Students with odd student numbers will join Team Red

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:

Your task is to write a program that processes a list of student participants, sorts them into their respective teams using **Insertion Sort**, and generates matchups for parallel rounds of competition.

Task 1.1

Write a Python function, `validate_student_code(code_list)`, that takes in a list of student codes and validates them according to these rules:

- The code must begin with a valid house code (BW, BB, HH, MR or MT)
- It must be followed by a number between 1 and 99
- Any invalid codes should be rejected with an appropriate message printed on the console.
- Ensures no duplicates

Examples of invalid codes: `MR-5`, `HH`, `MT0`, `mr99`

The function should return a list of valid codes. Test your program with appropriate test data.

[4]

Task 1.2

Write a function, `insertion_sort_by_number(code_list)`, to sort a list of student codes using your own Insertion Sort algorithm.

Sort the list by the numeric part of the code, e.g. "MR5" is sorted as 5.

If two student codes have the same number, it breaks the tie by alphabetically sorting the house code. For example, `MT5` and `HH5`, the sorted order should be `HH5` followed by `MT5`.

Do not use built-in sort functions like `sorted()` or `.sort()`

This function will be reused in Task 1.3.

[4]

Task 1.3

Write code to process `student_codes.txt`, and return two lists containing sorted codes of the Team Blue and Team Red.

Use Task 1.1 and 1.2 appropriately to answer this question.

[4]

Task 1.4

The games will proceed in parallel rounds to encourage maximum interaction between students from different houses.

Each round is formed as follows:

- The first student from Team Blue is matched with the last student from Team Red
- The second from Team Blue is matched with the second-last from Team Red, and so on

If one team has more students than the other, the unmatched students are paired with "BYE".

For instance,

```
Team Blue = ["BB8", "BW12", "HH14"]
```

```
Team Red = ["HH3", "MR5"]
```

The matchups would be as follows:

```
Round 1: BB8 vs MR5
```

```
Round 2: BW12 vs HH3
```

```
Round 3: HH14 vs BYE
```

Write a function `generate_reverse_matchups(team_blue, team_red)` to produce this output. Run your function.

[5]

2 Name your Jupyter Notebook as:

TASK2_server_<your name>_<index number>.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 2.1
        Program code
```

Output:

A company is developing a knowledge-based game that allows users to test their general knowledge through a five-question quiz. The system runs on a client-server model using socket programming. The server retrieves questions from a SQLite database, receives and validates user responses, calculates scores, and stores them. After the quiz, the server returns a sorted leaderboard to the client, displaying the top three performers. You are to implement core functionalities of this game system.

Task 2.1

Write a program that uses SQL to create a SQLite database named `quiz.db`. The database will be used to store questions from `questions.csv` and scores of quiz users.

Create a table `question` with the following fields:

- `id` (Integer, Primary Key)
- `question_text` (Text)
- `option_a` (Text)
- `option_b` (Text),
- `option_c` (Text),
- `option_d` (Text)
- `correct_option` (Text, storing 'A', 'B', 'C', or 'D')

Create another table `scores` with the following fields:

- `username` (Text, Primary Key)
- `score` (Integer)

The `scores` table will be populated later in Task 2.4.

Write a program to read quiz questions from a provided text file, `questions.csv` and insert this data into the `questions` table in your `quiz.db` database. [4]

Task 2.2

Declare a Python class named `Question` that encapsulates the data for a single quiz question.

It should have these attributes:

- `question_text`,
- `option_a`,
- `option_b`,
- `option_c`,
- `option_d`, and
- `correct_option`.

Include a constructor to initialise these attributes.

Create a method, `to_str()`, to return a single quiz question **as a string** and when printed, will be in this format:

```
What is 2 + 2?
A. 3
B. 4
C. 5
D. 6
Your answer (A/B/C/D):
```

Create a method, `check_answer(userchoice)`, that when given a `userchoice`, returns `True`, if user guessed the correct option for the question. If `userchoice` is incorrect or not one of the four valid options (A/B/C/D), will return `False`. [3]

Task 2.3

Write a function `generate_5_questions()` that will:

- connect to the database, `quiz.db`,
- generate 5 unique questions
- returns a list of 5 `Question` objects.

Test your function. Use the `to_str()` method to print the contents of the 5 `Question` objects. [4]

Task 2.4

Declare a Python class `QuizSession` that is used by the server to keep track of each new quiz attempt made by a user.

The `QuizSession` class contains the following attributes:

- `username`
- `score` (default to 0 as each quiz is a fresh attempt)

Implement the `updateScoresDB()` method:

- If this `username` does not exist in the `scores` table, insert a new `username` and his/her `score`, and returns a message: "Your score has been updated on quiz.db"
- If this `username` exists, it would mean that the player has already played the quiz before.
 - If the score attained in this `QuizSession` is higher than a previously attained score, update score and return a message: "Congratulations, you have achieved a new high score",
 - Else, return the message: "You did not outperform your previous score."

You can assume that usernames are unique in the `scores` table. [5]

Task 2.5

Write a function, `getLeaderBoard()` which queries the `scores` table and retrieves the Top 3 players by `score`, and returns a formatted string, breaking ties by sorting `username` alphabetically. Your function should **return a formatted string** in this format when printed:

```
Top 3 players
=====
Kelly: 5pts
Malcolm: 5pts
Joy: 4pts
```

[4]

Task 2.6

Referring to the code in `client.ipynb`, build an iterative socket server code that:

- Accept a new client connection
- Request for username from the client
- Create a new `QuizSession` object for the username
- Sends 5 questions (1 at a time) to the client and receives answers.
- Validates answers and updates score in the `QuizSession` object
- Report final score to client
- Updates the `scores` table in `quiz.db` using `updateScoreDB()` method
- Sends client the leaderboard information, using `getLeaderboard()` function.
- Close connection with client

You are advised to make use of your code completed in Task 2.2 to 2.5 to complete the above tasks, when possible.

Run both server program and client programs in separate Jupyter notebooks, and save the output of the client-server interaction below the code cells. [10]

- 3 The school has been experiencing an increasing number of lost items. You are tasked to create a **Lost and Found** web application and store the data about lost items in a MongoDB database. Your app will allow the Student Affairs staff to submit a new lost item report, view existing lost items in the database, and perform a search on lost items database.

Task 3.1

Write a program to:

- create a Mongo client to connect to database `lostfoundDB`
- use the data in `lost_items.json` to populate the collection `reports`

Save your program code as

`Task3_1_<name>_<index number>.py`

in a folder named `Task3_1_<name>_<index number>`

[3]

Task 3.2

Write a Python program and the necessary files to create a web application. When either the route `"/reports"` or `"/"` is visited, the web application will return a HTML page to display a report table of all the lost items found in the database, under appropriate table headings, sorting the records by date in ascending order.

Save your program code as

`Task3_2_<name>_<index number>.py`

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

`Task3_2_<name>_<index number>`

Run the web application and save the returned reports page as

`Task3_2_<name>_<index number>.html`

[5]

Task 3.3

Implement the following functionality to allow for new lost items to be reported.

Student Affairs staff will visit the `"/submit"` route to save information of new lost item to the database.

Create a "Submit Lost Item" HTML form that allows the staff to key in details like `item_name`, `description`, `category`, `date_lost` and `location`.

You should do validation of the `date_lost`, and ensure that invalid dates (for example, '2025-02-31' and '2025/12/12') are not accepted, and notify the user that the entered date is invalid.

After submitting the completed form, a new document should be created in the `reports` collection, and the user should be redirected to the `/reports` URL, which will show the new lost item reported, along with existing lost items.

Save your program code as

`Task3_3_<name>_<index number>.py`

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

`Task3_3_<name>_<index number>`

Run the web application, add a new report as below, and save the resultant report page as

`Task3_3_<name>_<index number>.html`

- Item Name: Spectacles
- Description: Pink spectacles left outside the Hall
- Category: Accessories
- Date Lost: 2025-07-31
- Location: Hall

[7]

Task 3.4

Implement the following functionality to allow the staff to search for lost items

- by category, or
- by date, or
- by both category and date.

Student Affairs staff will visit the `"/search"` route to perform a search on lost items in the database.

Create a "Search Lost Item" HTML form that contains two form inputs, "Category" and "Date Lost". If the staff to key in only one of the two fields, the search will only show items that fulfil the search field entered. If both fields are entered, only items that fulfil both criteria will be shown in a html table below the search form. Save your program code as

`Task3_4_<name>_<index number>.py`

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

`Task3_4_<name>_<index number>`

Search for lost items on date: '2025-07-31' and save output as

Task3_4_1_<name>_<index number>.html

Search for lost items with

- date: '2025-07-31' and
- category: "Accessories"

and save the output as

Task3_4_2_<name>_<index number>.html

[7]

- 4 Your school limits assembly announcements to 300 seconds per day. Announcements that cannot fit in 300 seconds are deferred to the next day.

Announcements are read in order of priority: High → Medium → Low. Three queues are used for each of these priorities.

Announcements are stored in two files:

announcement.txt	All new announcements to be read
deferred_announcement.txt	Leftover announcements from previous days

Both of these files follow the same format:

<department>,<priority>,<duration>,<title>

e.g. Discipline,High,90,Uniform Violation Rules

Implement the Announcement Scheduler, using provided code template, scheduler.py, which

- Load announcements from announcement.txt and deferred_announcement.txt
- Schedule announcements up to 300 seconds/day: High → Medium → Low priority (FIFO within priority)
- Display today's schedule, clear announcement.txt, and archive to read_announcements_archived.txt
- Save leftover announcements to deferred_announcement.txt

Rename scheduler.py as

Task4_<name>_<index number>.py

Task 4.1

Complete the `Announcement` class, which stores these attributes:

- `department`
- `priority`
- `duration`
- `title`

Implement the following methods:

- `__str__()`: Return a human-readable string representation of the announcement. Example: `"CCA - Science Fair (90s)"`
- `csv_str()`: Returns a csv-formatted string for writing to file. Example: `"CCA,High,90,Science Fair"`

[3]

Task 4.2

Complete `Queue` Class that stores `Announcement` objects, with suitable constructor and attribute(s).

You should implement these methods:

- `enqueue()`: Adds an announcement to the queue
- `dequeue()`: Remove and return the item at front of the queue, if not empty
- `is_empty()`: Returns `True` if queue is empty, `False`, if otherwise
- `output(file)`: Write each announcement in the queue to provided file, one per line, using `Announcement.csv_str()` method

[6]

Task 4.3

The `AnnouncementScheduler` class serves as the central controller of the entire announcement scheduling process :

AnnouncementScheduler		
Attributes		
Identifier	Data Type	Description
<code>time_limit</code>	<code>int</code>	Default of 300
<code>deferred_filename</code>	<code>str</code>	<code>deferred_announcement.txt</code>
<code>announcement_filename</code>	<code>str</code>	<code>announcement.txt</code>
<code>archive_filename</code>	<code>str</code>	<code>read_announcements_archived.txt</code>
<code>scheduled</code>	<code>list</code>	List of scheduled Announcements for today
<code>total_time</code>	<code>int</code>	Total duration of scheduled announcements.
<code>queues</code>	<code>dict</code>	Three separate queues 'High', 'Medium', 'Low'
Methods		
Identifier	Description	
<code>load_announcements() :</code> <code>List(Announcement)</code>	Loads and combines Announcement from <code>deferred_file</code> and <code>announcement_file</code> . Returns a combined list of <code>Announcement</code> objects.	
<code>categorise(announcements)</code>	Sort announcements into queues by priority	
<code>process() :</code> <code>List(announcements)</code>	Schedule announcements until <code>time_limit</code> reached Update <code>scheduled</code> Update <code>total_time</code> Return list of leftover announcements	
<code>display_schedule()</code>	Print scheduled announcements & <code>total_time</code>	
<code>archive_scheduled()</code>	Append to <code>archive_file</code> with separator	
<code>save_deferred()</code>	Save leftover announcements to <code>deferred_file</code> , using <code>output()</code> method in Task 4.2	
<code>clear_today_file()</code>	Empty <code>announcement_file</code> to allow fresh input tomorrow	
<code>run()</code>	Provided – runs full scheduling process You should not modify this method.	

Implement `AnnouncementScheduler` class.

[14]

Task 4.4

Write a Python program that will allow student councillors to input new announcements into `announcement.txt`. The program will request for information on the department, priority, duration and title, and appends this information into `announcement.txt`.

Name your program as

`Task4_4_<name>_<index number>.py`

[2]

Below shows the result of running the programs on two consecutive days:

Term 1 Day 1

Only `announcement.txt` file is present - with 7 announcements of various priorities.
No `deferred_announcement.txt` file is detected by the `AnnouncementScheduler` program.

`announcement.txt`

```
Discipline,High,90,Uniform Violation Rules
Sports,Medium,120,Track Meet Registration
CCA,Low,60,Guitar Club Meeting
Academics,High,150,Exam Briefing
Wellness,Medium,180,Mental Health Talk
Leadership,Low,45,Student Council Promo
Art,Low,90,Art Exhibition Invite
```



After running `Task4_<name>_<index number>.py`

Scheduler displays scheduled announcement on **console**

```
Today's Announcements:
1. Discipline - Uniform Violation Rules (90s)
2. Academics - Exam Briefing (150s)
3. CCA - Guitar Club Meeting (60s)
Total Time: 300s
```

`deferred_announcement.txt` created with following contents Empty contents of `announcement.txt`

```
Sports,Medium,120,Track Meet Registration
Wellness,Medium,180,Mental Health Talk
Leadership,Low,45,Student Council Promo
Art,Low,90,Art Exhibition Invite
```

`read_announcements_archived.txt` is created - with the following contents

```
Discipline - Uniform Violation Rules (90s)
Academics - Exam Briefing (150s)
CCA - Guitar Club Meeting (60s)
=====
```

At the end of Day 1, the student councillors ran the Task 4.4 program twice to add the following new announcements to the `announcement.txt` file.

Enter a new announcement:
 Department: Principal
 Priority (High/Medium/Low): High
 Duration (seconds): 120
 Title: Welcome message to start of term
 Announcement added successfully.

Enter a new announcement:
 Department: ICT
 Priority (High/Medium/Low): Medium
 Duration (seconds): 100
 Title: Digital Literacy Week Launch
 Announcement added successfully.

Term 1 Day 2

`announcement.txt`

```
Principal,High,120>Welcome message to start of term
ICT,Medium,100,Digital Literacy Week Launch
```

`deferred_announcement.txt`

```
Sports,Medium,120,Track Meet Registration
Wellness,Medium,180,Mental Health Talk
Leadership,Low,45,Student Council Promo
Art,Low,90,Art Exhibition Invite
```

After running

`Task4_<name>_<index number>.py`



Scheduler displays scheduled announcement on **console**

```
Today's Announcements:
1. Principal - Welcome message to start of term (120s)
2. Sports - Track Meet Registration (120s)
3. Leadership - Student Council Promo (45s)
Total Time: 285s
```

`deferred_announcement.txt` created with following contents

```
Wellness,Medium,180,Mental Health Talk
ICT,Medium,100,Digital Literacy Week Launch
Art,Low,90,Art Exhibition Invite
```

Empty contents of `announcement.txt`

`read_announcements_archived.txt` is updated - with the following contents

```
Discipline - Uniform Violation Rules (90s)
Academics - Exam Briefing (150s)
CCA - Guitar Club Meeting (60s)
=====
Principal - Welcome message to start of term (120s)
Sports - Track Meet Registration (120s)
Leadership - Student Council Promo (45s)
```

BLANK PAGE