Name:		Index Number:		Class:	

DUNMAN HIGH SCHOOL
Preliminary Examination
Year 6

COMPUTING (Higher 2)

9569/02

Paper 2 (Lab-based)

19 August 2024

3 hours

Additional Materials:	Insert
	Electronic version of `CLIENT.py` file
	Electronic version of `SERVER.py` file
	Electronic version of `Audio.txt` file
	Electronic version of `Websites.txt` file
	Electronic version of `SchoolInfo.csv` file
	Electronic version of `Latecoming.csv` file
	Electronic version of `Task4_datetime.py` file

READ THESE INSTRUCTIONS FIRST

Write your name, index number and class on the work you hand in.
Write in dark blue or black pen on both sides of the paper.
You may use an HB pencil for any diagrams or graphs.
Do not use staples, paper clips, glue or correction fluid.

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.

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.

Instructions to candidates:
Your program code and output for each of Task 1 to 4 (**except Tasks 1.4, 4.3, 4.4 & 4.5**) 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`

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

1 You started a new music streaming platform called Spotified. In order to ensure a smooth and seamless experience for your users, your server compresses the music data to reduce its size before transmitting the compressed data to the user's app. The user's app then decompresses the received data back into its original form to play the music.

- **Step 1:** Server reads the data of the audio file. [Note: Each byte is represented by 2 hexadecimal digits and is stored as an element in the list.]
  ['61', '61', '62', '62', '62', '62', '62', **'63'**, **'64'**, **'85'**, **'6F'**, **'2A'**, **'6F'**, '65', '65', '65', '65', '65', '65', '65', '65', '65', '6A', '6B', '72', '74', '74', '74', **'83'**, **'85'**, **'86'**, **'87'**, **'88'**, **'83'**, **'85'**, **'86'**, **'87'**, **'88'**, **'83'**, **'85'**, **'86'**, **'87'**, **'88'**, '92', '95', '9B', '9E', '9E', '9E', '9E', '9E', '9E', '9E', '9D', '9E', '9E', '9E', '9E', '9E', **'63'**, **'64'**, **'85'**, **'6F'**, **'2A'**, **'6F'**, **'63'**, **'64'**, **'85'**, **'6F'**, **'2A'**, **'6F'**]

- **Step 2:** Server searches for the given bytes pattern and replaces it with a single byte representation. E.g. replacing the bytes pattern `636485` to `C0`, `6F2A6F` to `C1` and `8385868788` to `C2`, resulting in the following compressed data:
  ['61', '61', '62', '62', '62', '62', '62', **'C0'**, **'C1'**, '65', '65', '65', '65', '65', '65', '65', '65', '65', '6A', '6B', '72', '74', '74', '74', **'C2'**, **'C2'**, **'C2'**, '92', '95', '9B', '9E', '9E', '9E', '9E', '9E', '9E', '9E', '9D', '9E', '9E', '9E', '9E', '9E', **'C0'**, **'C1'**, **'C0'**, **'C1'**]

- **Step 3:** Server to send the compressed data and the key in string format to the user's app (client). [Note: the key stores all the bytes pattern and its byte representation applied to the original data.]

  ```
  Compressed Data sent to user:
  61616262626262C0C165656565656565656565656A6B72747474C2C2C292959B
  9E9E9E9E9E9E9E9D9E9E9E9E9EC0C1C0C1
  Key sent to user:
  C0,636485,C1,6F2A6F,C2,8385868788
  ```

- **Step 4:** The user's app (client) decompresses the data received from the server with the key provided back to its original form.

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 [ ]: #Task 1.1
        Program code

        Output:
```

Task 1.1
The text file `Audio.txt` stores the bytes for a song. Each byte is represented by 2 hexadecimal digits and each byte is written on a new line in the text file. `61` is the first byte in the song.

Write the function `readfile(filename)` that takes a file name as a parameter, reads in the data and returns the data as a list of bytes. [2]

Note: for **Task 1.2** and **Task 1.3**, you are **not allowed** to use any built-in functions or any libraries/modules to **match and replace** the `bytes_pattern` and `byte_rep`.

Task 1.2
Write the function `compress (bytes_list, bytes_pattern, byte_rep)` that takes in a list of bytes, searches for the bytes pattern in the list and replace the affected bytes with the byte representation. The function should return the compressed list of bytes. [6]

Call your `readfile` function with `Audio.txt` as the parameter and test your `compress` function with the returned list of bytes together with the bytes pattern `636485` and byte representation `C0`. [1]

Task 1.3
Write the function `decompress (compressed_list, byte_rep , bytes_pattern)` that takes in the compressed list of bytes, searches for the byte representation in the list and replace the byte representation with the bytes pattern. The function should return the decompressed list of bytes. [4]

Test your `decompress` function with the compressed list of bytes from `Task 1.2` together with the byte representation `C0` and bytes pattern `636485.` [1]

Task 1.4
Using socket programming, complete both the server program used by Spotified to send the compressed music data & key, and the client program used by the users to receive the compressed music data & key and decompress the data back to its original form. [9]

- Copy your functions `readfile(filename)` and `compress (bytes_list, bytes_pattern, byte_rep)` into the server program.
- Copy your function `decompress (compressed_list, byte_rep , bytes_pattern)` into the client program.
- Complete the client program to:
    - receive the compressed data and key from the server program.
    - decompress the compressed data received.
    - display the decompressed data as a list of bytes.

You are provided with the server and client template programs, `SERVER.py` and `CLIENT.py` respectively. Complete both programs and rename as

```
CLIENT_<your name>_<centre number>_<index number>.py
SERVER_<your name>_<centre number>_<index number>.py
```

Study the sample program output in the next two pages to determine your code design, output format and socket protocol. User inputs are underlined.

Sample server program:

```
-------------------
SERVER OPEN
-------------------

Waiting for client request
---------------------------
original data: ['61', '61', '62', '62', '62', '62', '62', '63',
'64', '85', '6F', '2A', '6F', '65', '65', '65', '65', '65', '65',
'65', '65', '65', '6A', '6B', '72', '74', '74', '74', '83', '85',
'86', '87', '88', '83', '85', '86', '87', '88', '83', '85', '86',
'87', '88', '92', '95', '9B', '9E', '9E', '9E', '9E', '9E', '9E',
'9E', '9D', '9E', '9E', '9E', '9E', '9E', '63', '64', '85', '6F',
'2A', '6F', '63', '64', '85', '6F', '2A', '6F']

Enter the bytes pattern to be replaced: 636485
Enter the value to replace the above bytes pattern: C0
Do you want to continue compressing? [Y/N]: Y

Enter the bytes pattern to be replaced: 6F2A6F
Enter the value to replace the above bytes pattern: C1
Do you want to continue compressing? [Y/N]: Y

Enter the bytes pattern to be replaced: 8385868788
Enter the value to replace the above bytes pattern: C2
Do you want to continue compressing? [Y/N]: N

compressed_data: ['61', '61', '62', '62', '62', '62', '62', 'C0',
'C1', '65', '65', '65', '65', '65', '65', '65', '65', '65', '6A',
'6B', '72', '74', '74', '74', 'C2', 'C2', 'C2', '92', '95', '9B',
'9E', '9E', '9E', '9E', '9E', '9E', '9E', '9D', '9E', '9E', '9E',
'9E', '9E', 'C0', 'C1', 'C0', 'C1']
Compressed Data (String):
61616262626262C0C16565656565656565656A6B72747474C2C2C292959B9E9
E9E9E9E9E9E9D9E9E9E9E9EC0C1C0C1
Compressed Data (String) sent to client successfully

Key: C0,636485,C1,6F2A6F,C2,8385868788
Key sent to client successfully

-------------------
SERVER CLOSED
-------------------
```

Sample client program:

```
-------------------
CLIENT OPEN
-------------------
Raw compressed data received:
61616262626262C0C16565656565656565656A6B72747474C2C2C292959B9E9
E9E9E9E9E9E9D9E9E9E9E9EC0C1C0C1
Key received: C0,636485,C1,6F2A6F,C2,8385868788

Decompressed data: ['61', '61', '62', '62', '62', '62', '62',
'63', '64', '85', '6F', '2A', '6F', '65', '65', '65', '65', '65',
'65', '65', '65', '65', '6A', '6B', '72', '74', '74', '74', '83',
'85', '86', '87', '88', '83', '85', '86', '87', '88', '83', '85',
'86', '87', '88', '92', '95', '9B', '9E', '9E', '9E', '9E', '9E',
'9E', '9E', '9D', '9E', '9E', '9E', '9E', '9E', '63', '64', '85',
'6F', '2A', '6F', '63', '64', '85', '6F', '2A', '6F']
-------------------
CLIENT CLOSED
-------------------
```

Save your Jupyter Notebook and Python files for Task 1.

2 Name your Jupyter Notebook as:

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

You are trying to create your own web browser. Two key features of all web browsers are the back button, which allows users to go back to the previous web page, and the history page, which shows all the visited web pages in time sequence and allows the user to revisit any of the web pages directly.

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 [ ]: #Task 2.1
        Program code

        Output:
```

Task 2.1
You decided to use the Stack Data Structure to create the back button that keeps track of the last 15 web pages visited.

The class `Stack` contains three attributes:
- `stackList`, a 1-dimensional list of web page links (web page link is stored as a string).
- `maxSize`, the maximum number of web page links that can be stored in the stack.
- `topPointer`, the index of the top web page link in the stack, initialised to `-1`.

The class `Stack` has the following methods:
- a constructor that intialises `maxSize` and `topPointer` to appropriate values for an empty stack. It also initialises the correct number of `None` elements in `stackList` based on the `maxSize`.
- `push(webpage_link)` that takes a web page link(string) as a parameter and inserts it to the top of the stack. If the stack is full, the bottom web page link is deleted to make space for the latest web page link to be inserted at the top of the stack.
- `pop()` that returns the top web page link and removes it from the stack. Returns "`History is empty`" if the stack is empty.
- `display()` shows the outputs of all the current web page links in the stack, from the top to the bottom of the stack, in the format shown below, without removing any of the web page links. Below shows a sample display when there are three web page links in the stack. Display "`History is empty`" if the stack is empty.

```
http://ameblo.jp/at/nibh/in/hac/habitasse/platea.jsp<- topPointer
http://lycos.com/ac/est.png
http://tinyurl.com/phasellus/sit/amet/erat/nulla/tempus.jsp
```

Write program code to declare the class `Stack` with all its attributes and methods. [9]

Task 2.2
In a new cell, write program code to: [2]
- Initialise a new `Stack`.
- Read the data in the file `Websites.txt` and push each web page link into the stack.
- Call `display()` method with each insertion of a web page link.
- Call `pop()` 16 times and call `display()` with each `pop()`.

Task 2.3
You decided to use the Linked List Data Structure to create the history page.

Write the `LinkedList` class in Python. Include the following methods:

- `insert(webpage_link)` creates a node that stores the web page link(string) and inserts the node at the beginning (head) of the linked list.
- `delete(webpage_link)` attempts to delete the node that stores the given web page link value. If the web page link is not found, return `None`. Else return the deleted node.
- `revisit(webpage_link)` moves the node that stores the given web page link value to the beginning (head) of the linked list. Display `Page Not Found` if the web page link is not found.
- `display()` shows the outputs of all the current web pages in the linked list, from the first node to the last node in the linked list, in the format shown below, without removing any of the web page links. Below shows a sample display when there are three web page links in the linked list. Display "`History is empty`" if the linked list is empty.

```
http://ameblo.jp/at/nibh/in/hac/habitasse/platea.jsp
http://lycos.com/ac/est.png
http://tinyurl.com/phasellus/sit/amet/erat/nulla/tempus.jsp
```

Write program code to declare the class `LinkedList` with all its attributes and methods.
[9]

Come up with appropriate code to test that your `revisit()` method in Linked List is working correctly. You may reuse the web page links in `Websites.txt` in your tests. [2]

3 Name your Jupyter Notebook as:

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

A school wants to use Object-oriented Programming and Hash Table to store information about teachers and students.

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 [ ]: #Task 3.1
        Program code

        Output:
```

Task 3.1
The `Person` class has the following **private** data fields:

- `nric` – stored as a string, for example, T0675069D
- `fullname` – stored as a string, for example, Chang Jia Sheng
- `date_of_birth` - initialized with a string with the format YYYY-MM-DD

The class contains all the appropriate methods to set and access the above private data fields. It also includes one additional method:

- `display( )` – outputs the data fields of the `Person` Object, for example:

```
NRIC: S8775128G
Full Name: Tee Kai Ling
Date of Birth: 1987-02-10
```

Write program code in Python to define the class `Person`. [4]

Task 3.2
The `Student` class and `Staff` class inherits from the `Person` class. The `Student` class has an additional `class` private data field and the `Staff` class has an additional `role` private data field. Both are stored as a string.

Both classes also override their parent class's `display()` method by adding the value of their additional private data field in their respective `display()` method as shown below:

```
Student Information
----------------------
NRIC: T0843266F
Full Name: Xu Zhi Xin
Date of Birth: 2008-06-23
Class: 4D

Staff Information
----------------------
NRIC: S8775128G
Full Name: Tee Kai Ling
Date of Birth: 1987-02-10
Role: Subject Head
```

Write program code in Python to define the classes `Student` and `Staff`. [4]

Task 3.3
The class `HashTable` contains 11 buckets and uses linear probing (seeks the next available slot in the hash table by probing sequentially, skipping **2 buckets at a time**) for its collision resolution. It has two attributes:

- `size`, the number of buckets in the Hash Table.
- `array`, a 1-dimensional list that is used to store either `Student` or `Staff` objects. Each element in the list is initialised to `-1` in the beginning.

The class `HashTable` has the following methods:

- a constructor that intialises `size` and `array` to appropriate values for an empty Hash Table. It also initialises the number `-1` for each element in `array`.
- `hash(nric)` takes the `nric` (string) as a parameter and returns the position in the `array` to store the data (`Student` or `Staff` Object). The hash algorithm is shown in the pseudocode below:

```
h ← 0
FOR i ← 0 TO length(nric) - 1
    val ← 57 * (ASCII value of nric[i])
    h ← (h + val) % size
NEXT i
RETURN h
```

- `insert(Object)` takes a `Staff` or `Student` object as a parameter and inserts it into the `array` based on the calculated position returned by the above `hash` algorithm. If there is a collision, use linear probing to seek the next available slot (Note: to skip 2 buckets at a time). Return `False` if unable to find any available bucket, otherwise return `True`.
- `displayInformation(nric)` takes the `nric` (string) as a parameter and calls `display()` method of the `Staff` or `Student` Object that contains that `nric` value. Print "`Person not found`" if unable to locate any `Staff` or `Student` object with that `nric` value.

Write program code to declare the class `HashTable` with all its attributes and methods.
[12]

Task 3.4
The csv file, `SchoolInfo.csv`, contains information of a number of Staff and Students. Each row is a comma-separated list of data of the following:

- Text identifier (indicating whether the row belongs to a Student or Staff)
- NRIC
- Full Name
- Date of Birth in the form YYYY-MM-DD
- Class (for Students), Role (for Staff)

Write a program to:

- Create an empty `HashTable` instance.
- Read in the information from the csv file, creating an instance of the `Student` class or `Staff` class based on the Text identifier.
- Insert all `Student` and `Staff` instances into the `HashTable` instance. Output "`Insert successful`" if successfully inserted into the `HashTable`. Otherwise, output "`Insert unsuccessful`".
- Prompt user for an input to key in an `nric` value and call the `displayInformation()` method of the `HashTable` instance to display the

information of the `Student` or `Staff` with that `nric`.

- Ask the user if they would like to search again. Repeat the process of prompting user to key in an `nric` value and displaying information of the `Student` or `Staff` instance if User input “Yes”.

Test your program by running the application with a `Student nric`, a `Staff nric` and an `nric` value that is not found in `SchoolInfo.csv`. [5]

Save your Jupyter Notebook for Task 3.

4 Name your Jupyter Notebook as:

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

Your school currently keeps paper records about students who are late for school. The school's Discipline Committee wants to create a suitable database to store the data and to allow them to run reports to find out who are the repeated offenders and how many late-coming offences each class has committed. In addition, each late-coming offence results in a detention on the next weekday. The school also wants to keep track of the status of each detention to ensure that late-coming offenders learn their lesson and be punctual. You helped the school to create a web application to keep track of the late-coming and detention data. The database used for the web application has two tables: a table to store the students' information and a table to store the late-coming and detention information.

```
Student (StudentID, Name, Class)
Latecoming (LatecomingID, StudentID, Latecoming_Date, Time_Arrived,
Detention_Date, Detention_Completion_Status)
```

`Student:`

- `StudentID` – unique student identifier, for example, T0690909J
- `Name` – the name of the student
- `Class` – the class of the student, for example, 6C33

`Latecoming:`

- `LatecomingID` – unique late-coming number, for example, 15
- `StudentID` – unique student identifier, for example, T0690909J
- `Latecoming_Date` – the date of the late-coming offence in the form of YYYY-MM-DD
- `Time_Arrived` – the time the student arrived in school, in 24-hour format
- `Detention_Date` – the date of the detention in the form of YYYY-MM-DD
- `Detention_Completion_Status` – the status of the detention, for example, `True` if completed detention, `False` if otherwise

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 [ ]: #Task 4.1
        Program code

        Output:
```

Task 4.1
Create an SQL file called `TASK4_1_<your name>_<centre number>_<index number>.sql` to show the SQL code to create the two tables, `Student` and `Latecoming`, in the database `Discipline.db`. Define the primary and foreign keys for each table. [2]

Task 4.2
The file `Latecoming.csv` stores comma-separated values for both `Student` and `Latecoming` tables.

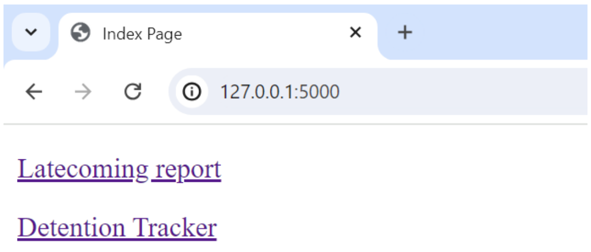
The data in `Latecoming.csv` is given in the following order:

```
        StudentID, Name, Class, Latecoming_Date, Time_Arrived
```

Write a Python program to read in the data from `Latecoming.csv` and store the data in the correct place in the database. Take note that **repeated offenders' student information should not be stored repeatedly** in the `Student` table. (Hint: Before inserting student record into `Student` table, check if the student's data has already been saved in the table before. If no, insert the student information as a new record in `Student` table. Otherwise, do not insert the student information into `Student` table again.)

In addition, the `datatime` library is built into Python and can be used to manipulate dates. See the example code shown in `Task4_datetime.py` and use the datetime library to generate the detention date (the **next weekday** after `Latecoming_Date`). [6]

Task 4.3
Write a Python program and the necessary files to create a web application. The application offers the following menu options:



Latecoming report

Detention Tracker

Save your Python program as:

`Task_4_3_<your name>_<centre number>_<index number>.py`

with any additional files/subfolders in a folder named:

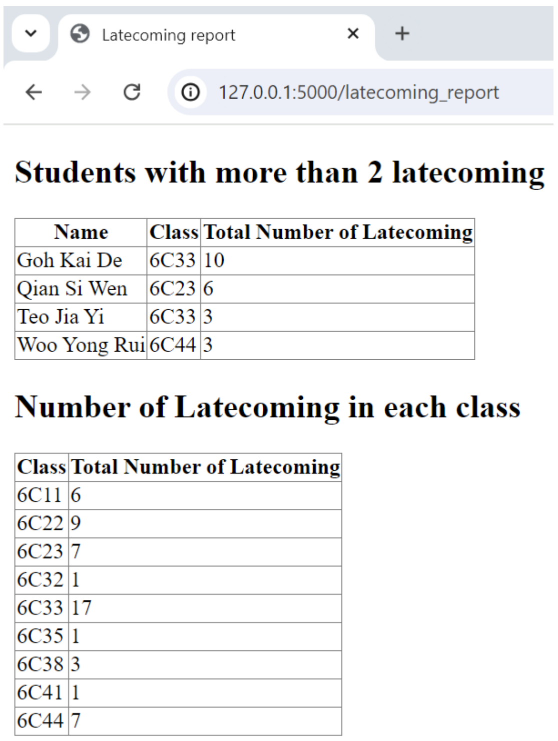
`Task_4_web_<your name>_<centre number>_<index number>`

Run the web application. [4]

Task 4.4
Write two separate SQL queries that shows:

- the `name`, `class` and `total number of latecoming` for students who were late more than 2 times sorted in descending order of the `total number of latecoming`.
- the `total number of latecoming` in each class sorted in ascending order of the `class` name.

The results of the two queries should be shown on a web page in two tables as shown below:



Students with more than 2 latecoming

Name	Class	Total Number of Latecoming
Goh Kai De	6C33	10
Qian Si Wen	6C23	6
Teo Jia Yi	6C33	3
Woo Yong Rui	6C44	3

Number of Latecoming in each class

Class	Total Number of Latecoming
6C11	6
6C22	9
6C23	7
6C32	1
6C33	17
6C35	1
6C38	3
6C41	1
6C44	7

This web page should be accessed from the menu option (Latecoming report) from Task 4.3.

Save your SQL queries as

`Task_4_4_<your name>_<centre number>_<index number>.sql` [4]

Modify the code in your below Python program:

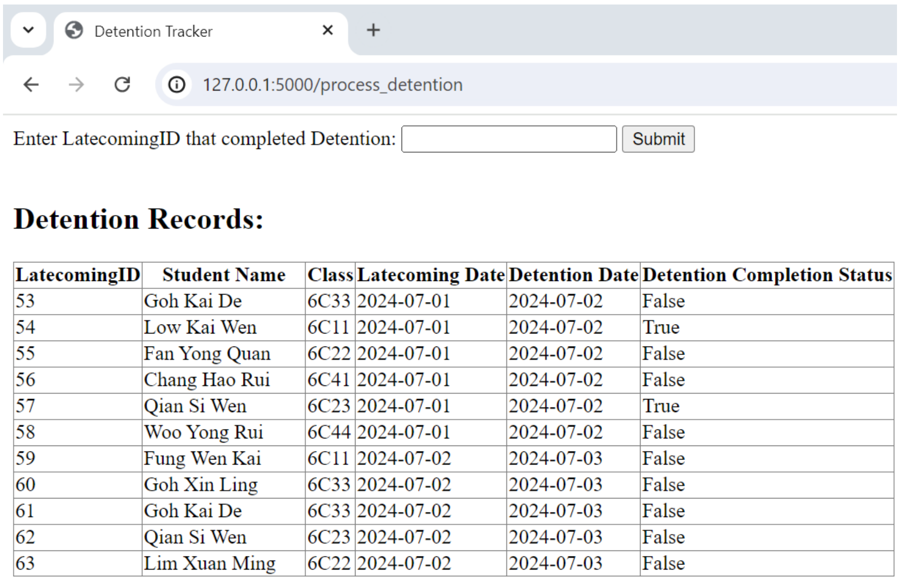
`Task_4_3_<your name>_<centre number>_<index number>.py`

with any additional files/subfolders in the folder named:

`Task_4_web_<your name>_<centre number>_<index number>` [7]

Run the web application.

Task 4.5
Modify your Python program and create the necessary file(s) to create a web page that shows the details of the detention records. There should also be a form with textbox for user to key in and submit the `LatecomingID` of completed detention. Upon submission of the form, the Detention Completion Status for that `LatecomingID` will be changed to `True` in the database and the change also reflected in this web page.



Enter LatecomingID that completed Detention:

Detention Records:

LatecomingID	Student Name	Class	Latecoming Date	Detention Date	Detention Completion Status
53	Goh Kai De	6C33	2024-07-01	2024-07-02	False
54	Low Kai Wen	6C11	2024-07-01	2024-07-02	True
55	Fan Yong Quan	6C22	2024-07-01	2024-07-02	False
56	Chang Hao Rui	6C41	2024-07-01	2024-07-02	False
57	Qian Si Wen	6C23	2024-07-01	2024-07-02	True
58	Woo Yong Rui	6C44	2024-07-01	2024-07-02	False
59	Fung Wen Kai	6C11	2024-07-02	2024-07-03	False
60	Goh Xin Ling	6C33	2024-07-02	2024-07-03	False
61	Goh Kai De	6C33	2024-07-02	2024-07-03	False
62	Qian Si Wen	6C23	2024-07-02	2024-07-03	False
63	Lim Xuan Ming	6C22	2024-07-02	2024-07-03	False

This web page should be accessed from the menu option (Detention Tracker) from Task 4.3.

Save any additional files/subfolders in a folder named:

`Task_4_web_<your name>_<centre number>_<index number>` [6]

Run the web application.

Save the webpage output as:

`Task_4_5_<your name>_<centre number>_<index number>.html` [1]

Save all your files for Task 4.

End of Paper