RIVER VALLEY HIGH SCHOOL
General Certificate of Education Advanced Level
Higher 2
Preliminary Examination

COMPUTING **9569/02**

Paper 2 (Lab-based)

3 AUG 2022
3 hours

Additional Materials: Electronic version of:
`Task1_1.txt`
`Task1_2.txt`
`Task1.ipynb`
`Task2.ipynb`
`Task2_server.ipynb`
`Task3.ipynb`
A `data_files` folder contains:
`ExamDuty.csv`
`ExamSession.csv`
`Teacher.csv`
`Venue.csv`
Insert Quick Reference Guide

READ THESE INSTRUCTIONS FIRST

Answer **all** questions.

All tasks must be done in 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 of programming style.

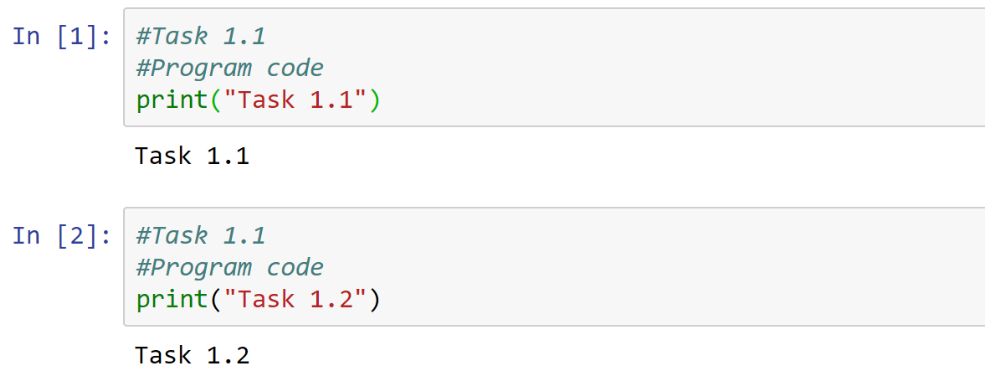
The number of marks is given in brackets [ ] at the end of each question or part question. The total number of marks for this paper is 100.

This document consists of **16** printed pages.

Instruction to candidates:

Your program code and output for each of Task 1 to 3 should be downloaded in a single `.ipynb` file.
For example, your program code and output for Task 1 should be downloaded as `TASK1_<your name>__<index number>.ipynb`

1 The task is to read a single character form the keyboard, check that it is alphabetic and display the character in different number system, for example:



```
In [1]: #Task 1.1
        #Program code
        print("Task 1.1")

        Task 1.1

In [2]: #Task 1.1
        #Program code
        print("Task 1.2")

        Task 1.2
```

Task 1

`Task1_1.txt` contains 50 lines of letter strings. Each line consists of unique letters.

Task 1.1

Your first task is to write the procedure `task1_1()` code to:

- Open and read the content of `Task1_1.txt`.
- For each letter string (each line in `Task1_1.txt`):
    - insert all the letters in the letter string in the same order into a Binary Search Tree (with the left sub-tree contains the alphabetically smaller letters).
    - display (print) the line number (starts with 1), the number of leaves of the BST, the original letter string, followed by the pre-order, in-order and the post-order traversal of the BST

The outcomes of the first 5 lines and the last line of **Task1.1** is as follow:

```
1 3 SUNJBPYVIR SNJBIPRUYV BIJNPRSUVY IBJRPNVYUS
2 5 FZPWJKVNOYIBC FBCZPJIKNOWVY BCFIJKNOPVWYZ CBIONKJVYWPZF
3 3 YJPKVGS YJGPKVS GJKPSVY GKSVPJY
4 6 CXVYHWIFRBMZS CBXVHFIRMSWYZ BCFHIMRSVWXYZ BFMSRIHWVZYXC
5 3 BPMLATCIEXVH BAPMLCIEHTXV ABCEHILMPTVX AHEICLMVXTPB
.
.
.
50 5 ZAUOFIGJCDNBWVEMK ZAUOFCBDEIGJNMKWV ABCDEFGIJKMNOUVWZ BEDCGKMNJ
IFOVWUAZ
```

[20]

Task 1.2

`Task1_2.txt` contains the log of an automated marking script. The script checks if the given letter string (each line in the file) produces the specific order of tree traversal as stated on the same line. It is marked as "`Correct`" if the order of tree traversal is correct and "`Incorrect`" otherwise. However, some of the lines in the file are marked wrongly.

Your task is to write the function `task1_2()` code to find those lines that are wrongly marked and returns these line numbers in an integer list.

For example, in `task1_2.txt`:

```
1,preorder,SUNJBPYVIR,BIJNPRSUVY,Correct
2,inorder,FZPWJKVNOYIBC,BCFIJKNOPVWYZ,Correct
.
.
8,postorder,DHKIRBFAYLZETNO,ABDEFHIKLNORTYZ,Incorrect
.
.
```

Line 1 is wrongly marked. This is because the pre-order traversal of the BST that is created by `SUNJBPYVIR` is `SNJBIPRUYV` and not `BIJNPRSUVY`, but line 1 is marked as `Correct`. So, line 1 is wrongly marked and the line number needs to be returned.

Line 2 is correctly marked. This is because `BCFIJKNOPVWYZ` is the in-order traversal of the BST that is created by `FZPWJKVNOYIBC`.

Line 8 is correctly marked. This is because the post-order traversal of the BST that is created by `DHKIRBFAYLZETNO` is not `ABDEFHIKLNORTYZ`, so it is marked as `Incorrect`.

Part of the solution of `Task1_2()` is as follows:

```
>>> Task1_2()
[1, 4, ..., 49]
```

[5]

Task 2

The server code in `Task2_server.ipynb` contains the program to conduct a Mastermind game. You task is to implement the client's `menu()` procedure which allows the client to establish a connection to the server and communicate based on the message protocol described below. The server's ip address and the port number used are `127.0.0.1` and `12345` respectively. You are to use IPv4 and TCP for the socket connection.

You do not need to know how to play Mastermind to complete this task. You just need to follow the message protocol given. In this implementation, number represents the color of the pin. There are altogether 6 colors, so there are numbered as 1 to 6.

Task 2.1

Write the function `validate_guesscode(guesscode)` to validate the `guesscode` string. A valid `guesscode` string contains 4-digit characters and each digit range from 1 to 6. The function should return `True` if `guesscode` is valid and `False` otherwise.

For example:

Valid guess code:
```
"1122"
"2222"
"6551"
"1256"
```

Invalid guess code:
```
"222"           (must be 4 digits)
"2237"          (7 is invalid. Digits must be from 1 to 6)
"0421"          (0 is invalid. Digits must be from 1 to 6)
```

```
>>> validate_guesscode("1122")
True
>>> validate_guesscode("2237")
False
```

[5]

Task 2.2

Write program code to implement the client's `menu()` procedure which allows the client to establish a connection to the server and communicate based on the message protocol given.

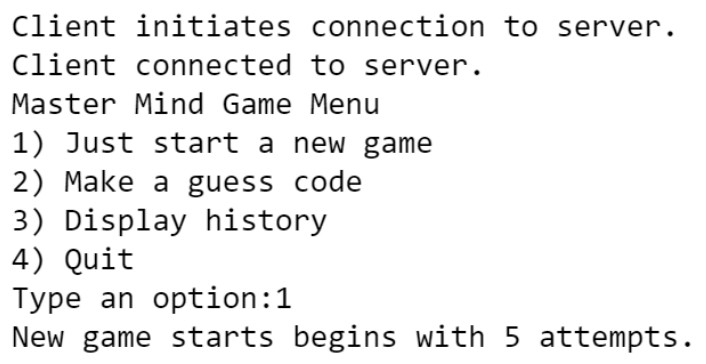
[20]

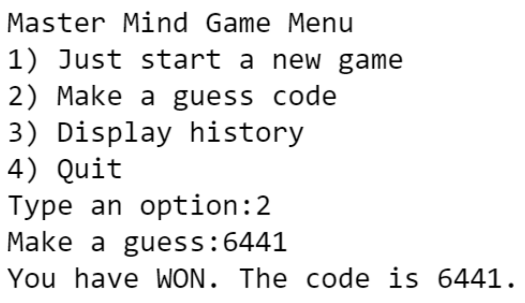
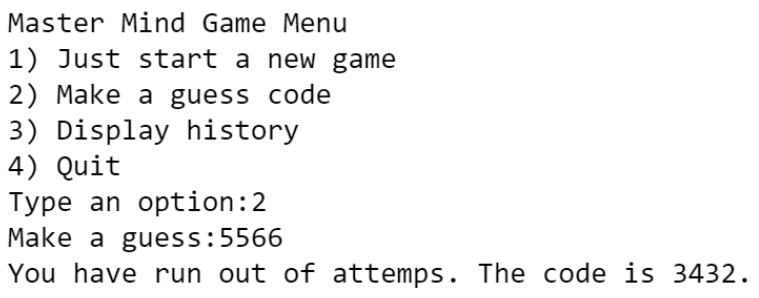
The Mastermind client menu options are as follow. Each option will create a message of its own.

```
Master Mind Game Menu
1) Just start a new game
2) Make a guess code
3) Display history
4) Quit
```

The following table defines the Mastermind game protocol for the client.

Message	Description

`START`	The `START\n` message is sent from the client to the server when client chooses option 1).
	
	The server will reply to the client in a `START` message with the number of attempts the client has before the game starts. The format of the message is as follow:
	
	`START,<number of attempts left>\n`
	
	For example, the client program should display the following when option 1 is chosen and `START,5\n` is received.
	
	
	`Client initiates connection to server.`
	`Client connected to server.`
	`Master Mind Game Menu`
	`1) Just start a new game`
	`2) Make a guess code`
	`3) Display history`
	`4) Quit`
	`Type an option:1`
	`New game starts begins with 5 attempts.`

`GUESS`	The `GUESS` message is sent from the client to the server when client chooses option 2). The format of the message is as follow:
	
	`GUESS,<guesscode>\n`
	
	For example, if the client input `1122` as the guess code, the message sent to the server is as follow:
	`GUESS,1122\n`
	
	When the server received a `GUESS` message, there 3 scenarios.
	
	<u>Case 1</u>
	If the guess code is correct, the server replies with a `WON\n` message. Upon receiving the `WON\n` message, the Client program quits
	
	
	`Master Mind Game Menu`
	`1) Just start a new game`
	`2) Make a guess code`
	`3) Display history`
	`4) Quit`
	`Type an option:2`
	`Make a guess:6441`
	`You have WON. The code is 6441.`
	
	<u>Case 2</u>
	If the guess code is incorrect and there is no more attempt left, the server replies with a `LOSE,<answer code>\n` message. Upon receiving the `LOSE\n` message, the Client program quits
	
	E.g. `LOSE,3432\n`
	
	
	`Master Mind Game Menu`
	`1) Just start a new game`
	`2) Make a guess code`
	`3) Display history`
	`4) Quit`
	`Type an option:2`
	`Make a guess:5566`
	`You have run out of attemps. The code is 3432.`
	
	<u>Case 3</u>
	If it is not case 1 and 2, the server replies a `GUESS` message with the guess result. The guess result is a string that is made of “B”, “W” or “-”.
	
	The format of the message the server sent back to client is as follow:

	`GUESS,<result>\n`
	
	For example, if the guess code `1122` has only 1 colour been guessed correctly at the correct position, “B” is returned by server. Then, the message that is sent from the server to the client is as follow:
	`GUESS,B\n`
	
	The client program should display the following when option 2 is chosen, `1122` is input and `GUESS,B\n` is received.
	
	`Master Mind Game Menu`
	`1) Just start a new game`
	`2) Make a guess code`
	`3) Display history`
	`4) Quit`
	`Type an option:2`
	`Make a guess:1122`
	`Guess Result: B`
	
	This is another possible result of case 3 where option 2 is chosen, `5436` is input and `GUESS,BWW\n` is received.
	
	`Master Mind Game Menu`
	`1) Just start a new game`
	`2) Make a guess code`
	`3) Display history`
	`4) Quit`
	`Type an option:2`
	`Make a guess:5436`
	`Guess Result: BWW`

`HISTORY`	The `HISTORY\n` message is sent from the client to the server when the client chooses option 3). The server will reply to the client in a `HISTORY` message with all previous guess codes and their results in the message. The format of the message is as follow: `HISTORY,<no_attempts_left>,<1st guess code>,<1st result>,<2nd guess code>,<2nd result>,…,<last guess code>,<last result>\n` For example, the client program should display the following when option 3 is chosen and `HISTORY,3,1122,B,2233,BW,2525,----\n` is received. Take note that the remaining attempts is displayed too.

	`Master Mind Game Menu`
	`1) Just start a new game`
	`2) Make a guess code`
	`3) Display history`
	`4) Quit`
	`Type an option:3`
	`You have 3 attempts left.`
	`1 1122 B`
	`2 2233 BW`
	`3 2525 ----`

`QUIT`	The `QUIT\n` message is sent from the client to the server when client chooses option 4). Both the client and the server program quit.
	
	`Master Mind Game Menu`
	`1) Just start a new game`
	`2) Make a guess code`
	`3) Display history`
	`4) Quit`
	`Type an option:4`
	`Game program ends.`

A full display of a WIN case in the client program for your reference.

```
Client initiates connection to server.
Client connected to server.
Master Mind Game Menu
1) Just start a new game
2) Make a guess code
3) Display history
4) Quit
Type an option:1
New game starts begins with 5 attempts.
```

Option 1

```
Master Mind Game Menu
1) Just start a new game
2) Make a guess code
3) Display history
4) Quit
Type an option:2
Make a guess:3344
Guess Result: BW
```

Option 2

```
Master Mind Game Menu
1) Just start a new game
2) Make a guess code
3) Display history
4) Quit
Type an option:2
Make a guess:2414
You have WON. The code is 2414.
```

Option 2

A full display of a LOSE case in the client program for your reference.

```
Client initiates connection to server.
Client connected to server.
Master Mind Game Menu
1) Just start a new game
2) Make a guess code
3) Display history
4) Quit
Type an option:1
New game starts begins with 5 attempts.
```

Option 1

```
Master Mind Game Menu
1) Just start a new game
2) Make a guess code
3) Display history
4) Quit
Type an option:2
Make a guess:1111
Guess Result: BB
```

Option 2

```
Master Mind Game Menu
1) Just start a new game
2) Make a guess code
3) Display history
4) Quit
Type an option:2
Make a guess:2222
Guess Result: ----
```

Option 2

```
Master Mind Game Menu
1) Just start a new game
2) Make a guess code
3) Display history
4) Quit
Type an option:2
Make a guess:3333
Guess Result: B
```

Option 2

```
Master Mind Game Menu
1) Just start a new game
2) Make a guess code
3) Display history
4) Quit
Type an option:2
Make a guess:4444
Guess Result: ----
```

Option 2

```
Master Mind Game Menu
1) Just start a new game
2) Make a guess code
3) Display history
4) Quit
Type an option:3
You have 1 attempts left.
1 1111 BB
2 2222 ----
3 3333 B
4 4444 ----
```

Option 3

```
Master Mind Game Menu
1) Just start a new game
2) Make a guess code
3) Display history
4) Quit
Type an option:2
Make a guess:4455
You have run out of attemps. The code is 1631.
```

Option 2

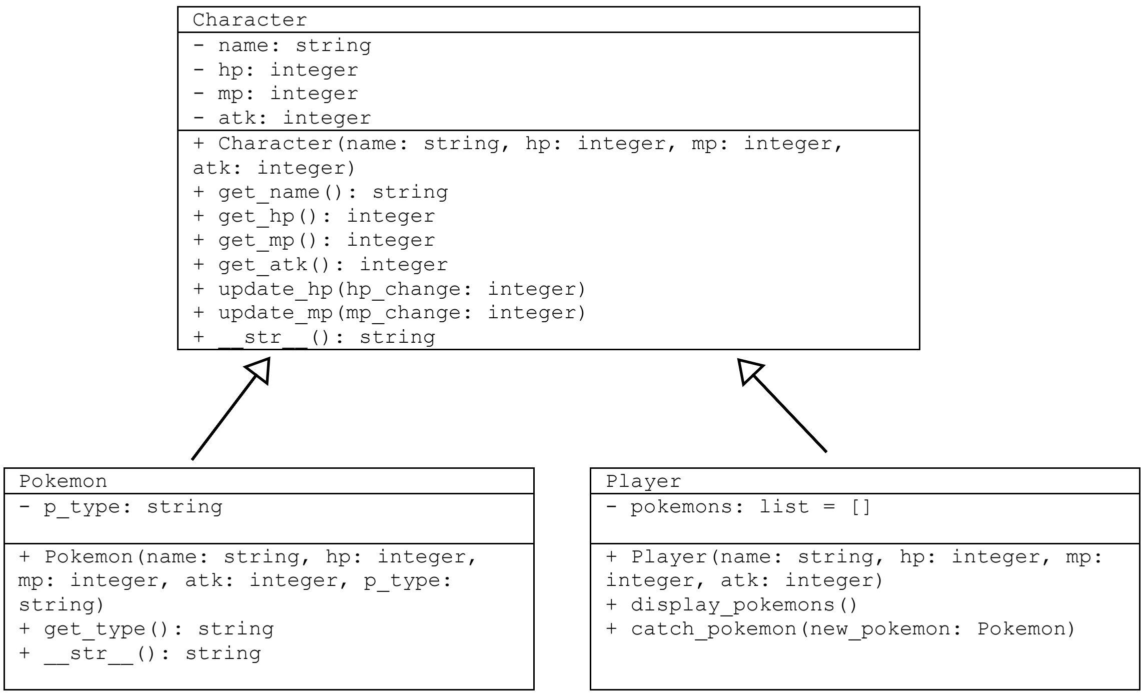
Task 3

You are tasked to create a simple game which allows players to catch pokemons. You are tasked to work on an object-oriented solution to store the information of characters involved, which includes the `player` and the `pokemons`. For all `characters`, their `name`, `hp` (health point), `mp` (mana point), `atk` (attack value) are recorded.

There are mainly two kinds of characters involved:

- `Pokemon`: each pokemon belongs to a specific `p_type`, which can take values such as "`Fire`", "`Water`", "`Electric`" or "`Grass`".
- `Player`: player can catch a `list` of pokemons. However, they could only have up to one pokemon of each type to be stored in this list. If the player wishes to catch a new pokemon with the same type, he will need to release the old one first.

Below is an UML class diagram for your reference.


Character
- name: string
- hp: integer
- mp: integer
- atk: integer
+ Character(name: string, hp: integer, mp: integer, atk: integer)
+ get_name(): string
+ get_hp(): integer
+ get_mp(): integer
+ get_atk(): integer
+ update_hp(hp_change: integer)
+ update_mp(mp_change: integer)
+ __str__(): string
Pokemon
- p_type: string
+ Pokemon(name: string, hp: integer, mp: integer, atk: integer, p_type: string)
+ get_type(): string
+ __str__(): string
Player
- pokemons: list = []
+ Player(name: string, hp: integer, mp: integer, atk: integer)
+ display_pokemons()
+ catch_pokemon(new_pokemon: Pokemon)

Implement the classes based on the following descriptions:

Attributes/Methods	**Description**
`Character Class`	

`- name: string`
`- hp: integer`
`- mp: integer`
`- atk: integer`
	Private attributes of class `Character`.

`+ Character(name: string, hp: integer, mp: integer, atk: integer)`
`+ get_name(): string`
`+ get_hp(): integer`
`+ get_mp(): integer`
`+ get_atk(): integer`
	Constructor of `Character` class and its getter methods.

+ update_hp(hp_change: integer)
+ update_mp(mp_change: integer)
	Methods to update the hp and mp values for the character. hp_change and mp_change can be positive or negative values; they should be added to the current hp and mp values.

+ __str__(): string	String method for the class, to display the necessary information in the following format: name: Ash Ketchum, hp: 100, mp:100, atk: 20.
Pokemon Class	
- p_type: string + Pokemon(name: string, hp: integer, mp: integer, atk: integer, p_type: string) + get_type(): string	Private attribute, constructor and getter method under Pokemon class.
+ __str__(): string	**Polymorphed** string method to display additional information including the p_type of the pokemon. E.g. name: Charmander, hp: 50, mp:60, atk: 18, type: Fire.
Player Class	
- pokemons: list = []	Private attribute to store a list of Pokemon objects. Should be initialized as an empty list.
+ Player(name: string, hp: integer, mp: integer, atk: integer)	Constructor of Player class.
+ display_pokemons()	Method to print output to the python shell. If the player hasn’t caught any pokemons, it should output: You have not caught any pokemons yet. Otherwise, print out a list of pokemons’ information based on the following format: Here are the pokemons in your team: name: Charmander, hp: 50, mp:60, atk: 18, type: Fire. name: Squirtle, hp: 55, mp:60, atk: 15, type: Water.
+ catch_pokemon(new_pokemon: Pokemon)	Check through the current list of pokemons the player has caught. If the new pokemon is of the same type of an existing teammate pokemon, prompt the user with the following message: You already have: name: Charmander, hp: 50, mp:60, atk: 18, type: Fire. Now you meet: name: Ponyta, hp: 60, mp:60, atk: 18, type: Fire. Would you like to replace the pokemon of the same type? [Y/N] If player choose Y, replace the old pokemon with the new one. Then display the following message: You have released Charmander, and caught Ponyta.

	Otherwise, display the following message:
	
	`You choose not to catch Ponyta.`
	
	If the new pokemon belongs to a new type, add it to the pokemon list and print the following message:
	
	`You have caught Pikachu.`

Save your code in the jupyter notebook provided. [12]

Task 4

The school request you to implement an invigilation duty system to aid the exam committee to store and display data related to exam and invigilation arrangements.

Task 4.1

The following information of each `Teacher` is stored:
`TeacherID` – auto increment integer value to keep track of ID of the teacher.
`Name` – name of the teacher.
`Department` – Department of the teacher.
`Contact` – contact of the teacher.

The following information of each `Venue` is stored:
`VenueID` – auto increment integer value to keep track of ID of the venue.
`VenueName` – name of the venue.
`RoomNo` – room number of the venue, based on block, level and room number.

The following information of each `ExamSession` is stored:
`ExamSessionID` – auto increment integer value to keep track of the exam session ID.
`SubjectName` – name of the subject tested.
`PaperNo` – paper number of the exam session.
`VenueID` – venue ID
`Date` – date of the exam session,
`StartTime` – starting time of the exam session.
`EndTime` – end time of the exam session.

The following information of each `ExamDuty` is stored:
`ExamSessionID` – exam session ID.
`TeacherID` – teacher ID.
`Role` – role of this teacher performing for this exam session, can be `'PaperCoordinator'`, `'Invigilator'`, `'Relief'`, `'RestroomDuty'` or `'Reserve'`.

Note:
- All `date` values are stored as a string in the format of `'YYYYMMDD'`.
- All `time` values are stored as a string in the 24hour-format of `'HHMM'`.
- You are required to include check condition for `Role` field.
- You may assume all values given in the data files are valid.
- To make things easier, you may assume each exam session only need 1 venue only.

The information is to be stored in above mentioned tables:
`Teacher`
`Venue`
`ExamSession`
`ExamDuty`

Task 4.1

Create an SQL file called `Task4_1.sql` to show the SQL code to create the database `invigilation.db` with the above tables.

The table `Teacher` must use `TeacherID` as its primary key, the table `Venue` must use `VenueID` as its primary key, the table `ExamSession` must use `ExamSessionID` as its primary key, and the table `ExamDuty` must use `ExamSessionID` and `TeacherID` as its primary key.

The `VenueID` in table `ExamSession` must refer to `VenueID` in `Venue` table as foreign key. `ExamSessionID` and `TeacherID` in table `ExamDuty` must refer to `ExamSessionID` in `ExamSession` table and `TeacherID` in `Teacher` table as foreign keys.

Save your SQL code as
`Task4_1.sql`

[6]

Task 4.2

The following files contains the past data records. The first row of each file contains the header of the respective columns. Each row in the files is a comma-separated list of information.

```
Teacher.csv
Venue.csv
ExameSession.csv
ExamDuty.csv
```

Write a Python program to insert all information from the files into the database `invigilation.db`. Run the program.

Save your program code as
`Task4_2.py` [5]

Task 4.3

You are tasked to implement a function to display a list of exam duties for a teacher by searching the database using his/her name. Query and display a list of data with the following fields as shown in the table, sorted in the **ascending** order according to the `Date` followed by the `StartTime` of the exam duties.

Note:
- `Date` *is a keyword in SQL and you need to apply double quote when using it as an attribute name*

SubjectName	PaperNo	Role	Venue	Date	StartTime	EndTime
…	…	…	…	…	…	…

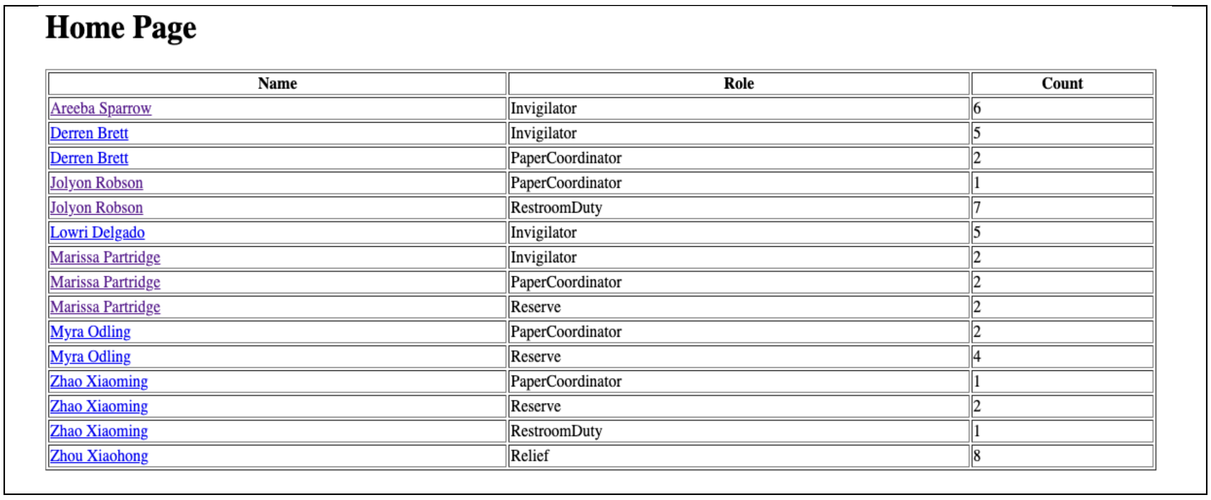
Write the SQL code required.

Save this code as
`Task4_3.sql` [5]

Task 4.4

You are required to implement a function to display a summary of the assigned exam duty roles for the teachers, and count the number of times a teacher is assigned for each role. Query database and display a table sorted by name of teacher and their assigned role on the home page, with the following fields:
`Name, Role, Count`

Here's a sample home page for your reference:



Home Page

Name	Role	Count
Areeba Sparrow	Invigilator	6
Derren Brett	Invigilator	5
Derren Brett	PaperCoordinator	2
Jolyon Robson	PaperCoordinator	1
Jolyon Robson	RestroomDuty	7
Lowri Delgado	Invigilator	5
Marissa Partridge	Invigilator	2
Marissa Partridge	PaperCoordinator	2
Marissa Partridge	Reserve	2
Myra Odling	PaperCoordinator	2
Myra Odling	Reserve	4
Zhao Xiaoming	PaperCoordinator	1
Zhao Xiaoming	Reserve	2
Zhao Xiaoming	RestroomDuty	1
Zhou Xiaohong	Relief	8

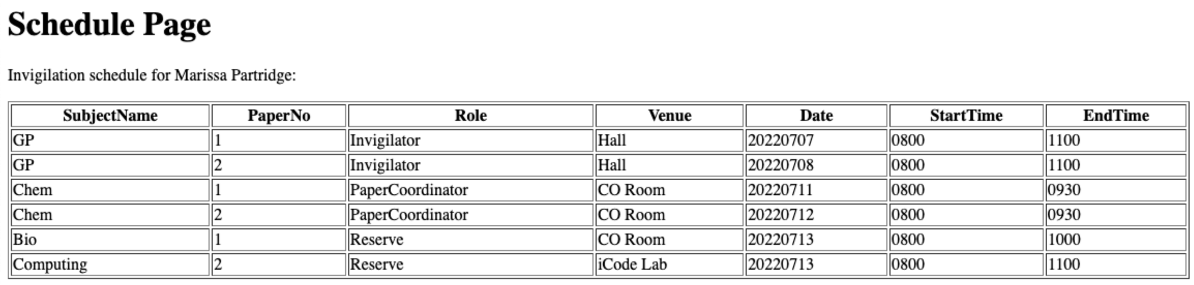
Save all files and folders under the directory `Task4_4`.

Run the web application.
Then save the output of the program as `Task4_4.html`. [10]

Task 4.5

Implement a schedule page to display a list of all exam duties for the teacher based on his/her name. You may choose to implement a hyperlink from the home page based on the teachers' names; or implement a search page for the user to enter the teacher's name.

Here's a sample schedule page output for teacher named `Marissa Partridge`:



Schedule Page

Invigilation schedule for Marissa Partridge:

SubjectName	PaperNo	Role	Venue	Date	StartTime	EndTime
GP	1	Invigilator	Hall	20220707	0800	1100
GP	2	Invigilator	Hall	20220708	0800	1100
Chem	1	PaperCoordinator	CO Room	20220711	0800	0930
Chem	2	PaperCoordinator	CO Room	20220712	0800	0930
Bio	1	Reserve	CO Room	20220713	0800	1000
Computing	2	Reserve	iCode Lab	20220713	0800	1100

Save all files and folders under the directory `Task4_5`.

[6]

- End of Paper -