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

COMPUTING

9569/02

Paper 2 (Lab-based)

17 August 2020
3 hours

Additional Materials: Electronic version of:
"data-gov-sg-dataset-listing.csv"
"TASK_2.ipynb"
"TASK_3_server.ipynb"
"candidates.csv"
"students.csv"
"votes.csv"
"test_avarta.png"

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 **12** 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 `TASK_1_<your name>_<centre number>_<index number>.ipynb`

1 The task is to read the content of the csv file *"data-gov-sg-dataset-listing.csv"* and insert the information as documents into a MongoDB database so that specific information can be retrieved by queries.

The csv file *"data-gov-sg-dataset-listing.csv"* is extracted from Data.gov.sg to provide viewers a list of available datasets provided by different organizations in Singapore.

The first row of the csv file are the names of the 14 fields that describes the dataset. They are:

- _id
- organization
- dataset_id
- dataset_name
- date_created
- last_updated
- description
- frequency
- coverage_start
- coverage_end
- resource_id
- resource_name
- resource_description
- resource_format

Task 1.1

Write program code to create a mongo client to local host and create the database "GovTech" with one collection "Datasets". [2]

Task 1.2

Write program code to read all the information in *"data-gov-sg-dataset-listing.csv"* and insert them as documents in collection "Datasets". [8]

Task 1.3

Write program code to find all the `_id` and `dataset_name` of datasets that belong to `"Infocomm Media Development Authority"`. [3]

Task 1.4

Write program code to return the number of datasets that have `resource_format` `"KML"` or `"GeoJSON"` or `"SHP"`. [3]

Task 1.5

Write program code to find all `_id` of datasets that have `"Adhoc"` `frequency` but no `coverage_start` date. [4]

2 The task is to implement a direct chaining hash table using a linked list data structures with free slot concept.

Task 2.1

Write program code to implement the class `linkedlist` by completing the following class functions. Each logical node of the linked list consists of a key, a value and a pointer. The linked list is eventually used to store the key-value pairs of the hash table. [9]

<table>
<tr><th>Class functions</th><th>Description</th></tr>
<tr><td>

`__init__(size)`

</td><td>

The `__init__` function takes in an integer `size` and sets the maximum number of key-value pairs it can store to `size`.

It initializes the **arrays** (`keys`, `values` and `ptrs)` and **variables** (`head` and `nextFree)` as follow:

```
>>> linkedlist(5)
```

Variable	
head	-1
nextFree	0

Array index	**0**	**1**	**2**	**3**	**4**
keys	-1	-1	-1	-1	-1
values	None	None	None	None	None
ptrs	1	2	3	4	-1

</td></tr>
<tr><td>

`insert (key, value)`

</td><td>

The `insert` function takes in an integer `key` and a string `value` and inserts them to the head of the linkedlist.

For example:

```
>>> llst = linkedlist(5)
>>> llst.insert(11, "apple")
```

Variable	
head	0
nextFree	1

Array index	**0**	**1**	**2**	**3**	**4**
keys	11	-1	-1	-1	-1
values	"apple"	None	None	None	None
ptrs	-1	2	3	4	-1

</td></tr>
</table>

<table>
<tr><td></td><td>

```
>>> llst = linkedlist(5)
>>> llst.insert(34, "egg")
```

Variable	
head	1
nextFree	2

Array index	0	1	2	3	4
keys	11	34	-1	-1	-1
values	"apple"	"egg"	None	None	None
ptrs	-1	0	3	4	-1

</td></tr>
<tr><td>search_by_key (key)</td><td>The search_by_key function takes in an integer key and returns the corresponding value if the key is found, otherwise it returns an empty string. For example:

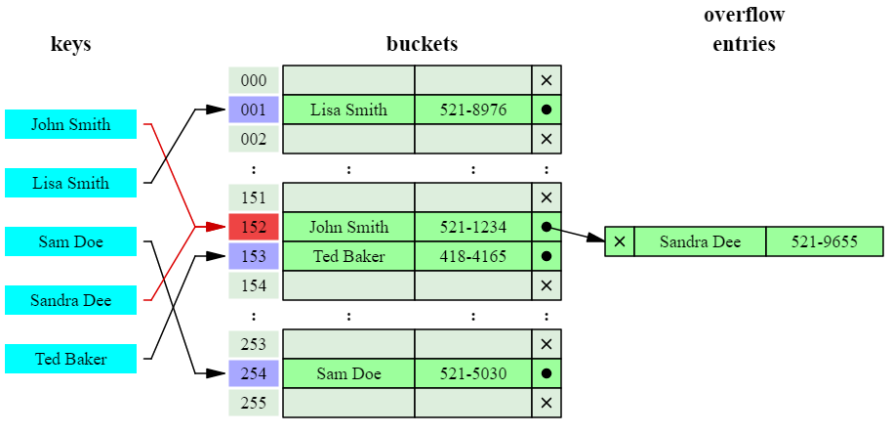
```
>>> llst = linkedlist(5)
>>> llst.insert(11, "apple")
>>> llst.insert(22, "banana")
>>> llst.search_by_key (11)
"apple"
>>> llst.search_by_key (22)
"banana"
>>> llst.search_by_key (9)
""
```

Note: You **must** make use of the ptrs in the process of searching for the key.</td></tr>
</table>

Task 2.2

Write program code to implement the class DirectChainingHashTable using the class linkedlist implemented in **task 2.1**. Your code should pass the two free test cases given to you in the .ipynb file for task 2. [6]

Hint: In direct chaining hash table, records with the same key are chained in a singly-linked list at the same location indicated by the hash table. Below is an illustration.



3 The task is to write the **client** code of the game hangman with the help of server code given to you in `"TASK_3_server.ipynb"`. The server ipv4 address and its port used should be `"127.0.0.1"` and `12345`. The client program should also display the following menu as it runs:

```
Menu:
1) Guess a letter
2) Guess a word
3) Quit
```

Option 1 allows client to guess 1 letter a time
Option 2 allows client to guess the full word
Option 3 allows the client to quit the program

Read and follow the hangman protocol attached closely. Some examples of the client terminal testing different menu option are also given to you for your reference.

Note: When you run the server code, you will be prompted to set a hidden word first.

Distribution of marks

- Proper socket connection [2]
- Protocol implementation [14]
- Menu implementation with validation of inputs [4]

Hangman message protocol

Message	Client	Server
START	No message is required to be sent back to the server upon the receive of the **START** message. However, it will print to terminal the following: `Current word guessed: ??????` where the number of "`?`" is the length of the hidden word.	The **START** message is the **FIRST** message sent by the server to the client to inform the client the length of the hidden word. The following format is used. "START,<length_of_hidden_word>\n" For example, "START,6\n"
GUESS	The **GUESS** message is sent to the server to make a guess of a letter of the hidden word. The following format is used. "GUESS,<letter>\n" For example, to guess a "`s`", the following is used. "GUESS,s\n" Note: If the hidden word is not completely guessed, the server returns a **GUESS** message after a **GUESS** message is sent to the server. The **GUESS** message from the server will include the positions of the correctly guessed letter. Upon receive the server's **GUESS** message, the client will print out the partially guessed word. For example, if `s` and `i` are guessed previously, the following should be printed to terminal. `Current word guessed: ?iss?s` Note: If the hidden word is completely guessed, the server returns a **WIN** message instead.	When the server receives a **GUESS** message, it sends a **GUESS** message back to the client to inform the client on the positions of the letter guessed correctly. For example, if the hidden word is "`misses`" and the following GUESS message is received from the client. "GUESS,s\n" The **GUESS** message returns from the server to the client will be: "GUESS,2,3,5\n" If the letter guessed is not found in the hidden word, the following **GUESS** message is sent instead. "GUESS\n" If the hidden word is completely guessed, the server returns a **WIN** message. The format is as follow: "WIN\n"

HWORD	The **HWORD** message is sent to the server to guess the whole hidden word. The following format is used.
	
	`"HWORD,<guess_word>\n"`
	
	For example, if the client wants to guess "kisses", the following is sent.
	`"HWORD,kisses\n"`
	
	Note: The server will either return a **WIN** or **LOSE** message after a HOWRD message is sent to the server. Upon the receive of a **WIN** or **LOSE** message, the client will print the result to terminal, close its socket and quit.
		When the server receives a **HWORD** message, it sends a **WIN** or **LOSE** message based on the <guess_word> and the hidden word.
		
		For example, if the hidden word is misses and the client guessed kisses, the following **LOSE** message is sent.
		
		`"LOSE\n"`
		
		If the hidden word is misses and the client also guessed misses, the following **WIN** message is sent.
		
		`"WIN\n"`
		
		Then, the server program closes all sockets and quits.

WIN/LOSE	No message is required to be sent back to the server upon the receive of the **WIN/LOSE** message. The client just needs to print to terminal the result, closes all sockets and quits the programme.	The **WIN** message is sent from the server to client, to inform the client has won. The format is as follow:
		
		`"WIN\n"`
		
		The **LOSE** message is sent from the server to client, to inform the client has lost. The format is as follow:
		
		`"LOSE\n"`
		
		The server then closes all sockets and quits the programme after a **WIN** or **LOSE** message is sent to client.

QUIT	The **QUIT** message is sent to the server to close the connection. The following format is used. `"QUIT\n"` After the **QUIT** message is sent, the client will close its socket and the whole program quits.	When the server receives a **QUIT** message, the server then closes all sockets and quits. No message is sent back to the client.

All client examples shown uses hidden work: kiss

Note: All letters and word are in lower case.

Example A

```
Current word guessed: ????
Menu:
1) Guess a letter
2) Guess a word
3) Quit
Type an option:1
Type a letter: m
Current word guessed: ????
Menu:
1) Guess a letter
2) Guess a word
3) Quit
Type an option:1
Type a letter: k
Current word guessed: k???
Menu:
1) Guess a letter
2) Guess a word
3) Quit
Type an option:1
Type a letter: s
Current word guessed: k?ss
Menu:
1) Guess a letter
2) Guess a word
3) Quit
Type an option:1
Type a letter: i
Current word guessed: k?ss
You win!
```

Example B

```
Current word guessed: ????
Menu:
1) Guess a letter
2) Guess a word
3) Quit
Type an option:2
Type a word: kiss
Current word guessed: ????
You win!
```

Example C

```
Current word guessed: ????
Menu:
1) Guess a letter
2) Guess a word
3) Quit
Type an option:2
Type a word: miss
Current word guessed: ????
You lose!
```

Example D

```
Current word guessed: ????
Menu:
1) Guess a letter
2) Guess a word
3) Quit
Type an option:3
You quit!
```

4 With the new challenges faced due to the pandemic outbreak, the school would like to implement an online voting system, which will aid in the Students' Council Election process and provide meaningful data analysis for the voting results.

In this exercise, you may assume all data are valid and do **not** need to worry about data validation or implementation of restrictions such as the maximum number of candidates a student can vote for.

The following information of each `Student` is stored:
`MatricNo` – unique string in the format of `"RVHS-YYYY-XXX"` where `YYYY` is the year of entry to school and `XXX` is a 3-digit string starting from `"001"`.
`Class` – class of student
`IndexNo` – index number of the student in the class
`Gender` – gender of student, to be stored as a single character, using either `"M"` or `"F"`

The following information of each `Candidate` is stored:
`CandidateNo` – unique autoincrement integer value to identity each candidate
`Name` – name of candidate
`Slogan` – campaign slogan of the candidate
`PortraitLink` – campaign portrait image url link, optional field

The following information of each `Vote` is stored:
`MatricNo` – matric number of the student
`CandidateNo` – number to identify the candidate

The information is to be stored in three tables:
`Student`
`Candidate`
`Vote`

Task 4.1
Create an SQL file called `Task_4_1.sql` to show the SQL code to create the database `voting_mgm.db` with the three tables.

The table `Student` must use `MatricNo` as its primary key, and the table `Candidate` must use `CandidateNo` as its primary key. The table `Vote` should use `MatricNo` and `CandidateNo` as a composite key, while `MatricNo` and `CandidateNo` must refer to `MatricNo` in `Student` and `CandidateNo` in `Candidate` as foreign keys.

Save your SQL code as
`Task_4_1.sql`

[5]

Task 4.2

The files `students.csv`, `candidates.csv` and `votes.csv` contains information about the student, candidate and votes of this year. 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.

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

Save your program code as
`Task_4_2.py` [5]

Task 4.3

The teacher in charge would like to find out all the votes of one candidate, **Ee Pei Chi Neoma**, from the **Secondary 1** level. Query and display a list of data with the following fields as shown in the table, sorted in the **ascending** order according to **Class**, followed by **IndexNo** of the students who voted for this person.

Note: Secondary 1 class names ranged from `"1A"` to `"1M"`.

Class	IndexNo
…	…

Write the SQL code required.

Save this code as
`Task_4_3.sql` [6]

Task 4.4

The school would like you to implement a function to display names of all the candidates a student has voted based on the student's matric number.

Write a Python program and the necessary files to create a web application that:

- Receive the `MatricNo` from a HTML form, then,
- Creates and returns a HTML document that enables the browser to display a table tabulating the all candidates information in 2 columns, `CandidateNo` and `Name`.
- The list of information should be sorted according to the `CandidateNo` in **ascending** order.

Save your program as
`Task_4_4.py`
With additional files or sub-folders as needed in a folder named
`Task_4_4`

Run the web application. Enter the following `MatricNo`:
MatricNo: RVHS-2015289

Then save the output of the program as `Task_4_4.html`.

[12]

Task 4.5

Design a simple web interface for the candidate to upload their portrait images. The form should contain a textfield to enter **CandidateNo** and a file upload option which allows the student to choose a **png** file to be uploaded.

Upon receiving the image uploaded by the student, the programme should save the file into the `"static\portraits\"` folder and rename the image file as `"portrait_xx.png"` where `xx` is the 2-digit candidate number such as `"09"`.

A webpage should be displayed to signal the uploading is successful, and display his/her candidate number, name, slogan and portrait image as a result. The result page should be formatted as a html table, with presentable cell **borders**, **background colors** and **alignments**.

Save your program as
`Task_4_5.py`
With additional files or sub-folders as needed in a folder named
`Task_4_5`

Run the web application, upload the following image for the candidate:

Image: test_avartar.png
Candidate Name: Ee Pei Chi Neoma

Then save the webpage displayed upon successfully uploaded the image as `Task_4_5.html`.

[17]

End of paper