HWA CHONG INSTITUTION
C2 BLOCK TEST 2022

COMPUTING
Higher 2

26 May 2022 **Paper 2 (9569 / 02)** **0815 -- 1015 hrs**

Additional Materials:

Electronic version of `TICKET.json` data file

Electronic version of `LYRICS.txt` data file

Electronic version of `NAMES.txt` data file

Electronic version of `CLIENT.py` file

Insert Quick Reference Guide

READ THESE INSTRUCTIONS FIRST

Answer **all** questions.

The maximum mark for this paper is **60**.

The number of marks is given in brackets [ ] at the end of each question or part question.

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.

This document consists of **8** printed pages.

Instructions to candidates:

Your program code and output for each of Task 1 to 3 should be saved in a single a single `.ipynb` file. For example, your program code and output for Task 1 should be saved as

`TASK1_<your name>_<ct>.ipynb`

1 Name your Jupyter Notebook as `TASK1_<your name>_<ct>.ipynb`

The task is to create a NoSQL database for a travel agency to manage the flights information for tickets from Singapore to other countries. Each flight contains the destination country and city, the price in SGD, the airline company, the duration in hours, and the number of stops (if any).

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:
```

Task 1.1

Write a python program to:

- Create a MongoDB database named `Travel` and a new collection named `Flight`
- Insert the flight documents into the `Flight` collection in the `Travel` database. Use the sample dataset in the file `TICKET.json`.
- Display all flight documents in the `Flight` collection

[3]

Task 1.2

Write program code which makes use of `Flight` collection in `Travel` database to:

- Update `stop` field to 0 for those flights without any stops
- Display all the information for flights by company `S Airlines`
- Display the `city` and `price` for flights under 10 hours and under $ 1500
- Display all the information for the cheapest air ticket

All outputs should have appropriate messages to indicate what you are showing. [7]

Save your Jupyter Notebook for Task 1.

2 Name your Jupyter Notebook as `TASK2_<you name>_<ct>.ipynb`

The task is to generate song lyrics based on an existing song. For this question, you are provided with a text file `LYRICS.txt`, containing the lyrics to a song.

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:
```

Task 2.1

Write a function `task2_1(filename)` that:

- takes a string `filename` which represents the name of a text file
- reads in the content of the text file
- converts all text to lower case
- ignores new lines or punctuations
- returns the content as a list of words (keeping the original ordering of the words)

[4]

Call your function `task2_1` with the file `LYRICS.txt`, printing the length of the returned list using the following statement:

```
print(len(task2_1('LYRICS.txt')))
```

[1]

Task 2.2

Write a function `task2_2(list_of_words)` that:

- takes a list of words
- filters only the **unique** words that come immediately before the word `it`
- sorts the words by length (shortest in front, longest behind)
- sorts same length words by alphabetical order
- returns the sorted list of words [5]

Call your function `task2_2` with the contents of the file `LYRICS.txt`, printing the returned list and its length, using the following statements:

```
result = task2_2(task2_1('LYRICS.txt'))
print(result)
print(len(result))
```
[1]

Task 2.3

A musician asks you to help him generate a line of lyrics in the format

`A it, B it, C it, … it.`

where `A, B, C, …` are distinct words.

Write a function `task2_3(list_of_words, number)` that:

- takes a list of unique words `list_of_words`, and an integer `number`
- randomly picks `number` **distinct** words from the list
- returns a string using the randomly picked words in the format required [4]

Call your function `task2_3` using `result` from **Task 2.2**, printing the returned string with `number` set to the value of `8`, using the following statements:

```
print(task2_3(result, 8))
```
[1]

Save your Jupyter notebook for Task 2.

Reference: Song lyrics from "Technologic" by Daft Punk

3 Name your Jupyter Notebook as `TASK3_<your name>_<ct>.ipynb`

The task is to implement a Binary Search Tree, to store a dataset of maximum 10 items, using Object Oriented Programming.

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 3.1

Program Code
```

```
Output:
```

A binary search tree abstract data type (ADT) is to be implemented using object-oriented programming. A linked list is used to maintain all the unused node which do not form part of the tree. The first available node which is used for a new item is indicated by `nextFree`. Items in the unused list are linked using their left pointers.

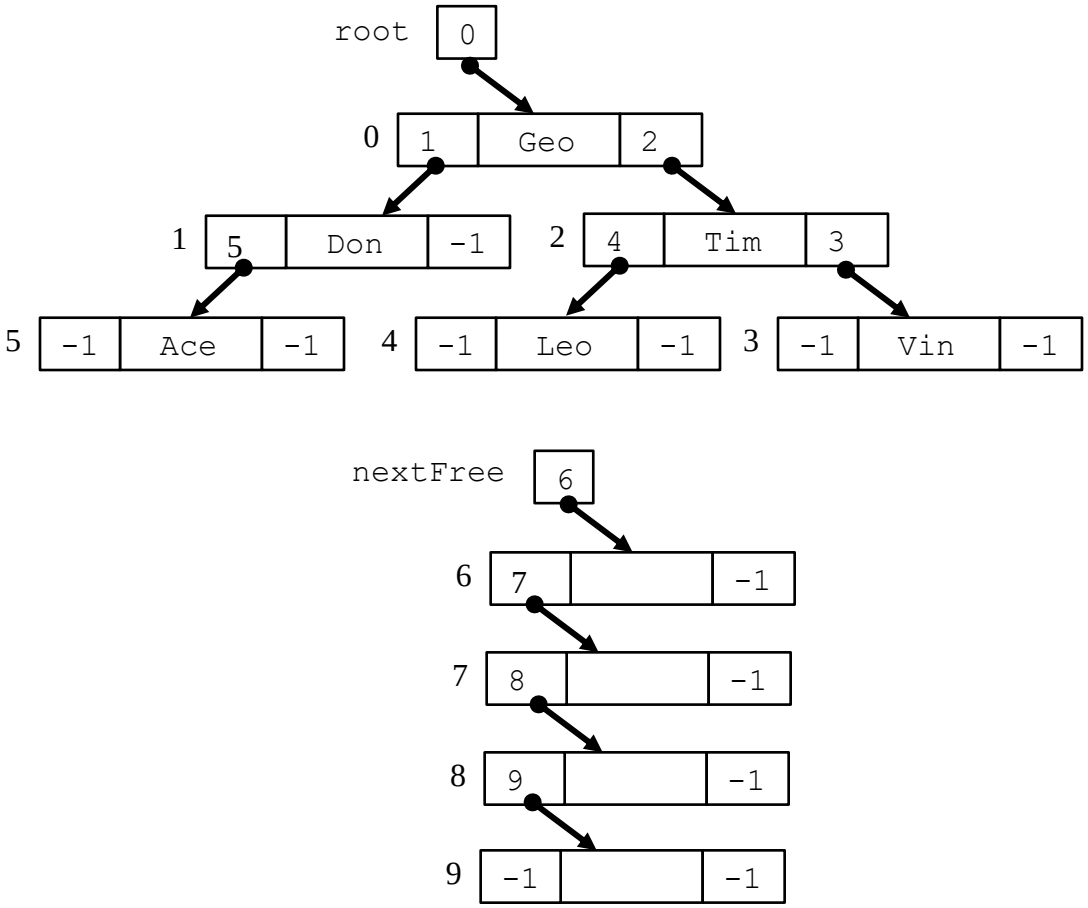
Two classes `Node` and `BSTree` have been identified.

Class: `Node`		
Identifier	**Data Type**	**Description**
`leftPtr`	`INTEGER`	The left pointer for the node
`data`	`STRING`	The node's data
`rightPtr`	`INTEGER`	The right pointer for the node

Class: `BSTree`		
Identifier	**Data Type**	**Description**
`thisTree`	`ARRAY[10] of Node`	An array used to store the 10 nodes.
`root`	`INTEGER`	Index for the root position of the binary tree
`nextFree`	`INTEGER`	Index for the next unused node
`constructor()`	`PROCEDURE`	Set pointers to indicate all nodes are unused and linked. Initialise values for `root` and `nextFree`
`add(newItem)`	`PROCEDURE`	Add `newItem` to the tree using **non-recursive** way
`displayInorder()`	`PROCEDURE`	Display the items in Inorder sequence.
`displayPostOrder()`	`PROCEDURE`	Display the items in PostOrder sequence.
`display()`	`PROCEDURE`	Display the value of `root`, `nextFree` and the contents of `thisTree` in index order.

The diagram shows the `BSTree` with:

- the items `Geo`, `Don`, `Tim`, `Vin`, `Leo` and `Ace` added in that order
- the unused nodes linked together



Task3.1

Write program code for the `Node` and `BSTree` classes. The code should follow the specification given. [17]

Task 3.2

The program is to be tested. Write a main program to:

- create a `BSTree` object
- from the file `NAMES.txt`, read and add all the names to the tree by calling the `add` method
- call `display` method to display current state of pointers and array contents
- call `displayInOrder` method to display the tree
- call `displayPostOrder` method to display the tree [7]

Save your Jupyter Notebook for Task 3.

4 A server program and a client program plays an asymmetric guess-the-number game Nurdle. The server generates a random 4-digit code from 1000 to 9999, and the client tries to guess it within 5 tries. If the client fails after 5 tries, the server sends the correct answer to the client. After each incorrect guess, the server returns a 4-letter message about the accuracy of the guess. The message is made from three letters with the following meanings:

`G`: green for correct digit at the correct position
`Y`: yellow for correct digit at the wrong position
`R`: red for wrong digit

Below is a sample run where the server's code is 9060.

- When the client makes a guess of `8069`, the first digit 8 does not exist in the code and hence gives `R`. The second- and third-digits match and give `G`. The last digit 9 should be in the first position and hence gives `Y`. Therefore, the message is `RGGY`.
- When the client makes a guess of `9066`, the first three digits match. The last digit 6 is in the code but not at the fourth position, and hence gives `Y`. Therefore, the message is `GGGY`.

```
Message Interpretation:
G: correct digit at the correct position
Y: correct digit at the wrong position
R: wrong digit

Enter guess (1000-9999):1234
Your message is: RRRR

Enter guess (1000-9999):5067
Your message is: RGGR

Enter guess (1000-9999):8069
Your message is: RGGY

Enter guess (1000-9999):9066
Your message is: GGGY

Enter guess (1000-9999):9069
You lose the game!
Correct code is: 9060
```

Use the given client program `CLIENT.py` to design the server program. The socket protocol uses `\n` to detect the end of a message and you do not need to handle invalid input by the client.

Save your program code as `TASK4_SERVER_<your name>_<ct>.py` [10]