HWA CHONG INSTITUTION
C2 TIMED PRACTICE EXERCISE 2023

COMPUTING
Higher 2

25 May 2023 **Paper 2 (9569 / 02)** **1400 -- 1600 hrs**

Additional Materials:

Electronic version of `SARAH.py` file

Electronic version of `DRINK.json` data file

Electronic version of `WORDS.csv` data file

Insert Quick Reference Guide

READ THESE INSTRUCTIONS FIRST

Answer **all** questions.

The maximum mark for this paper is **65**.

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.

Instructions to candidates:

Your program code and output for each of Task 2 to 4 should be saved in a single a single `.ipynb` file. For example, your program code and output for Task 2 should be saved as

`TASK2_<your name>_<ct>.ipynb`

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

This document consists of **7** printed pages.

1 Sarah and Chloe bought a large number of chicken nuggets. They took turns to take 1 to 3 nuggets, and agreed that the person who took the last piece would have to pay for all the nuggets.

The game can be simulated with 2 programs, 1 for Sarah and 1 for Chloe such that:

- Sarah starts by inputting the number of nuggets available and sends this number to Chloe.
- Chloe starts taking the nuggets by inputting a number.
- Sarah and Chloe then alternate in taking nuggets each:
    - Each girl can take exactly 1, 2 or 3 nuggets each turn.
    - Each girl cannot take more than the number of nuggets left.
    - Prompt is asked again if an invalid number is provided.
- After nuggets are taken by either girl:
    - If there are nuggets left, this number is sent over.
    - If no nuggets are left after taking:
        - `'end'` is sent.
        - `'You lose.'` is displayed.
- `'You win.'` is displayed if the opponent takes the last nugget.

The following is an example of what each girl would see.

Sarah	**Chloe**

`How many nuggets? 11`	`There are 11 nuggets left.`
	`How many to take? 3`
	`3 nuggets taken, 8 left.`

`There are 8 nuggets left.`
`How many to take? 4`
`There are 8 nuggets left.`
`How many to take? 2`
`2 nuggets taken, 6 left.`
	`There are 6 nuggets left.`
	`How many to take? 0`
	`There are 6 nuggets left.`
	`How many to take? 2`
	`2 nuggets taken, 4 left.`

`There are 4 nuggets left.`
`How many to take? 2`
`2 nuggets taken, 2 left.`
	`There are 2 nuggets left.`
	`How many to take? 3`
	`There are 2 nuggets left.`
	`How many to take? 1`
	`1 nuggets taken, 1 left.`

`There are 1 nuggets left.`
`How many to take? 1`
`You lose.`
	`You win.`

Complete the program `SARAH.py` and design the program for Chloe. The socket protocol uses `\n` to detect the end of a message.

Save your program codes as `TASK1_SARAH_<your name>_<ct>.py` and `TASK1_CHLOE_<your name>_<ct>.py` [10]

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

The task is to create a NoSQL database for a tea store to manage the orders. Each order contains the date and time, the name of the tea, the sugar level (in percentage), and the price. It may include optional information, e.g. less or no ice, upsize, toppings, membership.

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 Python program to:

- Create a MongoDB database named `Store` and a new collection named `Order.`
- Use the sample dataset in the file `DRINK.json` to insert the order documents into the `Order` collection in the `Store` database.
- Display all the order documents in the `Order` collection. [3]

Task 2.2

Write program code which makes use of the `Order` collection in the `Store` database to display the following information:

- Orders without upsizing.
- Orders without bubble topping.
- Orders with healthier choices (sugar level up to 50%).
- Total number of orders by Jason.
- Average price per order.

All outputs should have appropriate messages to indicate what you are showing. [6]

Task 2.3

Write program code to make the following changes to the `Order` collection:

- Exclude ice for all the orders from Alice on May 21.
- Remove the field of upsizing for all orders.
- Remove all the orders by Jason.

Display all the order documents after the above three changes. [4]

Save your Jupyter Notebook for Task 2.

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

Frequency analysis is a method used in cryptography to study the frequency of letters or groups of letters in a ciphertext. This technique is employed to aid in breaking classical ciphers.

The Oxford English Corpus (OEC) is a text corpus of 21st-century English utilized by the makers of the Oxford English Dictionary and Oxford University Press' language research program. One of the research areas in the OEC is frequency analysis, which involves recording and ranking the frequency of words used in the corpus. The list of the 100 most commonly used words, according to the OEC, is provided in the file `WORDS.csv`.

A binary search tree is used to store the words in order of how frequent they appear, with more common words appearing on the left, and less common words on the right.

For example, if the following data is added to an empty tree:

Word	**Frequency Rank**
`Apple`	2
`Banana`	1
`Coconut`	3

`Apple` becomes the root of the tree. `Banana` has a higher frequency rank than `Apple` and will be inserted on the left of `Apple`, while `Coconut` is inserted on the right.

The tree is implemented using Object-Oriented Programming (OOP).

The class `WordRank` contains three properties:

- `left` points to the left node.
- `right` points to the right node.
- `data` is the data of the node, which is a 2-tuple `(word,rank)` where:
    - `word` is the word stored as a string.
    - `rank` is the rank on the OEC ranking, stored as an integer.

The class `WordRank` contains one method:

- A constructor to set left pointer and right pointer to `None`, and the data to its parameter.

The class `Tree` contains one property:

- `root` points to the root `WordRank` in the tree.

The class `Tree` contains the following methods:

- A constructor to create an empty tree.
- A non-recursive method to take the parameter and store it as a `WordRank` in the correct position in the tree.
- A recursive method to use in-order traversal to output the `rank` and `word` in the tree.
- A recursive method to use pre-order traversal to output the `word` in the tree.

For the task, 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
         Program Code
         output:
```

Task 3

Write program code to declare the classes `WordRank` and `Tree` and their constructors. [3]

Write the non-recursive method to insert a new node into the tree. [6]

Write the main program to:

- Open and read the file `WORDS.csv.`
- Declare a new instance of `Tree` and store each word and its ranking as a `WordRank` in the tree using your method. [6]

Write program code to:

- Declare the method to output the in-order traversal of the binary tree.
- Declare the method to output the pre-order traversal of the binary tree.

Call the in-order and pre-order methods using your tree structure. [5]

Test your program and show the output from each traversal. [2]

Save your Jupyter Notebook for Task 3.

4 Name your Jupyter Notebook as `TASK4_<your name>_<ct>.ipynb`

The task is to converts a numerical value into its equivalent word form, which is typically used in financial or banking applications for cheque writing, invoicing, or other similar tasks.

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 4.1
         Program Code
         output:
```

Task 4.1

For positive integers from 1 to 99, write a function, `up_to_two_digit(num)` that takes an integer `num` and returns its equivalent word form as a string. For example, `print(up_to_two_digit(21))` shall print `twenty one.`

Write a program that

- Generates all the equivalent words for numbers from 1 to 99
- Writes those words, one per line, to a new file, `TASK4_NUMBERS_<your name>_<ct>.txt` [6]

Task 4.2

For positive integers from 1 to 999, write a function `up_to_three_digit(num)` that takes an integer `num` and returns its equivalent word form as a string. You may use the function from Task 4.1. For example, `print(up_to_three_digit(345))` shall print `three hundred forty five.` [4]

Test your function with the following statements:

```
print(up_to_three_digit(60))
print(up_to_three_digit(700))
print(up_to_three_digit(890))
```

Task 4.3

This task works on numerical currency numbers with two decimal places, ranging from `1.00` to `999999999999.99`. Use the functions from the previous tasks to design a function `convert(money)` that:

- Takes a float `money` in the required format and range.
- Displays its number form with comma separators.
- Displays its word form with dollars and cents.

For example, `1.00` in the two forms are `1.00` and `one dollars`.

`1234000567.89` gives the two forms below:

```
Number Form: 1,234,000,567.89
```

```
Word Form: one billion two hundred thirty four million five
hundred sixty seven dollars eighty nine cents
```

Test your function with `1.00` and `1234000567.89`. [10]

Save your Jupyter Notebook for Task 4.