# RAFFLES INSTITUTION
# 2023 YEAR 6 PRELIMINARY EXAM

CANDIDATE NAME

CLASS 23

## COMPUTING 9569/02

Paper 2 (Lab-based) **3 hours**

Additional materials: Electronic version of `RECIPES.txt` data file
Electronic version of `CLIENT_TASK2.py` file
Electronic version of `GAMERS.txt` file
Electronic version of `FITNESS.db` file
Insert Quick Reference Guide

**READ THESE INSTRUCTIONS FIRST**

Answer **all** 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.

Note that up to **6** marks out of 100 will be awarded for the use of common coding standards for programming style.

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

| FOR EXAMINER'S USE | | | | |
|---|---|---|---|---|
| Task 1 | Task 2 | Task 3 | Task 4 | TOTAL |
| | | | | 100 |

This document consists of **12** printed pages.





**[Turn over**

**Instructions to candidates:**

Your program code and output for each of Task 1 to 3 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>_<index number>.ipynb`

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

**1** Name your Jupyter Notebook as:

`TASK1_<your name>_<index number>.ipynb`

You are tasked with analysing a collection of bakery recipes. The recipe data is stored in a file named `RECIPES.TXT`.

For each recipe, the name of the bakery item will be listed first, followed by information about each ingredient for that item separated by commas:

```
<ingredient name>,<quantity required>,<unit of measurement>
```

For example, the first recipe in the file is for the Cheesecake item:

```
Cheesecake
cream cheese,450,g
sugar,150,g
eggs,3,pieces
sour cream,120,ml
vanilla extract,1,tsp
flour,30,g
```

The recipes are separated by blank lines in the file `RECIPES.TXT`.

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 program code for a function named `task1_1` that takes the filename as a parameter and finds the most common ingredient and returns the name of the most common ingredient in the recipes, along with the number of occurrences. [5]

Run your program and show the output. [1]

**Task 1.2**

Write program code for an insertion sort function named `task1_2` that takes a list of strings, sorts it in ascending alphabetical order, and returns the sorted list. [4]

**Task 1.3**

Write program code for a function named `task1_3` that reads the data from the file and calculates the total quantity of each ingredient required across all recipes. If the unit of measurement for an ingredient is the same in multiple recipes, sum up the quantities. Sort the names of ingredients alphabetically using the insertion sort function from Task 1.2 and display the sorted list of ingredients, along with their total quantities and units. [6]

Run your program and show the output. [1]

Save your Jupyter Notebook for Task 1.

**2** Name your Jupyter Notebook as:

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

A chatterbot is a computer program that responds to characters input and produces a response for each line of input as follows:

- The first time that it reads in a line containing the string "`hello`", it should generate as a response the output line "`Hi, how are you?`".
- The second and subsequent times that it reads in a line containing the string "`hello`", it should generate as a response the output line "`Hello again, welcome back!`".
- When it reads in a line containing the string "`thanks`" or the string "`thank you`", it should generate as a response the output line "`You are most welcome.`"
- When it reads in a line beginning with a string of the form "`I <xYz> you`", where `<xYz>` is any string, it should generate as a response an output line of the form (follow the same casing) "`You <xYz> me? I really <xYz> you too.`".
- When it reads in a line not containing any of the strings described above, it should generate as a response the output line "`Sorry, I do not understand...`".

The chatterbot should check each line for the specified strings in the order listed above and generate only one response per input line.

For example, the input line "`I thank you`" should generate as a response ***only*** the output line "`You are most welcome.`", and it should ***not*** generate as a response the output line "`You thank me? I really thank you too.`".

The chatterbot should not be case sensitive. In other words, it must recognise not only "`hello`" but also "`Hello`", "`HELLO`", "`hElLo`", etc.

The task is to write a chatterbot program to work as a server-client application using sockets. The chatterbot server will listen for incoming connections from clients, and clients will send text-based messages to the server. The server will respond to the client based on the input.

You are provided with the client program code that establishes a connection to the server and sends user input. Your task is to create the server program as follows:

- Write a function `process_input` function to process user input and generate responses based on the input rules described above. The function will take a parameter `input_text` and process the logic and return a respond based on the parameter.
- Write the `main` function for the server program to establish the socket connection, handle communication with client, and close the connection properly.

The server program should process client requests and respond with the appropriate chatterbot responses.

When a client is connected to the server, the server should print this message on its side: "`Chatterbot server is listening.`"

The server should output the conversation with the client on its side, using the labels "`Client`" and "`Chatterbot`" to identify the message source.

For example, the output on server's side should show:

```
Client: Hello there!
Chatterbot: Hi, how are you?
```

The client should also output the conversation using the labels "`You`" and "`Chatterbot`", for example, the output on the client's side should show:

```
You: Hello there!
Chatterbot: Hi, how are you?
```

The server will terminate the program when the client enters "`exit`", with a suitable message.

**Task 2.1**

Create the Python code for the server program. [13]

Name your server program as:

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

Test the program by inputting the following lines in client program: [2]

```
This is a test
Well, hello there!
I am fine, thank you.
Wait, I have to go...
Never mind. Thanks.
HELLO! I have returned.
I missed you so much!
I thank you... hello?
You and I think alike, you know...
exit
```

Save your Jupyter Notebook for Task 2.

Client Program in `CLIENT_TASK2.py`:

```
import socket

def main():
    client_socket = socket.socket()
    client_socket.connect(("127.0.0.1", 12345))

    print("Connected to Chatterbot server.")

    while True:
        user_input = input("You: ")
        client_socket.send(user_input.encode()+b'\n')
        if user_input.lower() == "exit":
            break
        response = b""
        while b'\n' not in response:
            response += client_socket.recv(1024)
        response = response.decode()
        print("Chatterbot:", response)

    print("Goodbye! Connection closed.")
    client_socket.close()

main()
```

**3** Name your Jupyter Notebook as:

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

An online gaming company uses a linked list to store the scores of gamers, together with their names and gamer IDs.

The data is stored in a file named `GAMERS.txt`, with each line containing a record in the following format:

`<gamer ID>,<name of gamer>,<score>`

The linked list and its nodes are implemented using Object-Oriented Programming (OOP).

The class `Node` contains two properties:

- `Data` stores the value of the node.
- `Pointer` references to the next node in the linked list.

During initialisation, the `Data` is set to the parameter value while the `Pointer` is set to null because the node has no next node when created.

The class `LinkedList` contains a `Start` pointer that references the first node of the linked list. It contains the following methods:

- a constructor to set the `Start` pointer to null.
- a recursive method named `insert_last` that takes the data value as parameter and inserts a node into the last position of the linked list.
- a method named `display` to display a specified number of nodes starting from the first node. The parameter is the number of nodes to be displayed.

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:

**Task 3.1**

Write program code to declare the classes `Node` and `LinkedList`. [3]

Write the recursive method named `insert_last` to insert a new node into the last position of the linked list. [5]

Write the method named `display` to display a specified number of nodes from the first node. [4]

Write the main program to:

- declare a new instance of `LinkedList`
- read all the records from the `GAMERS.txt` file and insert them into the linked list. Each record should be contained in a suitable data structure and stored in the `Data` part of a node.
- display the first 8 nodes from the linked list. [5]

Run your program and show the output. [1]

Due to the frequency of accessing the records, the company decides to change the storage from a linked list to a hash table.

The hash table will also be implemented using OOP.

The class `HashTable` contains two properties:

- `Size` to indicate the maximum size of the hash table.
- `Slots` to store the data as a node, including the scores of gamers, their names and gamer IDs.

The class `HashTable` contains the following methods:

- a constructor that initialise the `Size` from the parameter and declare an array of `Slots` base on `Size`.
- a `hash_function` that computes the remainder of dividing the key (gamer ID) by the size of the hash table, and returns the index where the corresponding record will be stored or retrieved.
- a method to insert a record into the hash table. Handle collisions using **separate chaining** with the `LinkedList` class defined in Task 3.1. The record is stored as a `Node` object.
- a method to display all the records from a specified slot of the hash table, with the slot number specified as the parameter. Make use of the display method from the `LinkedList` class.

**Task 3.2**

Write program code to declare the class `HashTable` and its constructor. [2]

Write the `hash_function` method. [1]

Write the method named `insert_record` to insert a record into the hash table. [3]

Write the method named `display_records` to display all the records from a specified slot of the hash table. [2]

Write a module named `linked_list_to_hash_table` to copy the data from the linked list to the hash table. **DO NOT** copy the data directly from the `GAMERS.txt` file into the hash table. [4]

Write a main program to

- run the module `linked_list_to_hash_table` with a size of 401 for the hash table.
- display all the records in slot number 14. [2]

Test your program and show the output. [1]

Save your Jupyter Notebook for Task 3.

**4** A fitness club stores members' information in a database and manages them to provide statistics related to their fitness progress.

It has a database named '`FITNESS.db`' containing two tables:

- `Member(MemberID, Name, Gender, Age)`: Stores information about club members
- `FitnessRecord (RecordID, MemberID, Weight, Height, WorkoutDate)`: Stores fitness records of members including their weight, height, and the date of the workout.

Each member has a unique member ID, and their name, gender and age are recorded in the `Member` table.

For every visit to the fitness club, members will have their weight and height, with their workout date recorded in the `FitnessRecord` table. Each record has a unique record ID.

**Task 4.1**

Write a Python program and the necessary files to create a web application. The homepage displays a menu with the following options:

```
1. Member Details
2. Fitness Statistics
3. Add Fitness Record
```

Save your program code as

`TASK4_1_<your name>_<index number>.py`

with any additional files/subfolders as needed in a folder name

`TASK4_1_<your name>_<index number>`

Run the web application and save the output of the program as

`TASK4_1_<your name>_<index number>.html` [5]

**Task 4.2**

Write an SQL query that shows

- all members' names, genders, ages, and the latest recorded weight and height, with date. For members with missing fitness records, show only the names, genders and ages.
- sorted by gender, then names in ascending order.

By adding to the program code in Task 4.1, display the results of the query on a web page in a table when user selects option 1 of menu.

Save your SQL code as

`Task4_2_<your name>_<index number>.sql`

and save your program code as

`TASK4_2_<your name>_<index number>.py`

with any additional files/subfolders as needed in a folder name

`TASK4_2_<your name>_<index number>` [7]

Run the web application and save the output of the program as

`TASK4_2_<your name>_<index number>.html` [1]

**Task 4.3**

Write an SQL query that shows

- the total number of male and female members
- average age of male and female members, rounded off to 1 decimal place
- average weight and height of male and female members, rounded off to 1 decimal place, based on the latest workout date.

By adding to the program code in Task 4.2, display the results of the query on a web page in a table when user selects option 2 of menu.

Save your SQL code as

`Task4_3_<your name>_<index number>.sql`

and save your program code as

`TASK4_3_<your name>_<index number>.py`

with any additional files/subfolders as needed in a folder name

`TASK4_3_<your name>_<index number>` [7]

Run the web application and save the output of the program as

`TASK4_3_<your name>_<index number>.html` [1]

**Task 4.4**

By adding to the program code in Task 4.3, create a web page to allow user to input a new fitness record for a member. The record should include the Member ID, Weight, Height, and Workout Date.

When user selects option 3 of menu, this web page will display a form for input.

The information input should be stored in the existing database '`FITNESS.db`'.

Save your program code as

`TASK4_4_<your name>_<index number>.py`

with any additional files/subfolders as needed in a folder name

`TASK4_4_<your name>_<index number>` [7]

Run the web application and input the following new record into the database:

```
Member ID:    M102
Weight:       65 kg
Height:       165 cm
Workout Date: 2023-08-14
```

After adding this new record, select option 1 of menu to display member details.

Save the output of the program as

`TASK4_4_<your name>_<index number>.html` [1]