Name:_____ **Class:**_____

JURONG PIONEER JUNIOR COLLEGE

2021 JC2 Preliminary Examination

COMPUTING
Higher 2

9569/02
1 September 2021

Paper 2 (Lab-based)

3 hours

Additional materials:
Electronic version of `marathon.CSV` data file
Electronic versions of `Battleship_Client.py` & `Task2_Client_SampleOutput.JPG` files
Electronic version of `members.TXT` data file
Electronic version of `JPgym.SQL` data file
Electronic version of `items.JSON` data file
Insert Quick Reference Guide

READ THESE INSTRUCTIONS FIRST

Answer **all** the 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.

The number of marks is given in brackets [ ] at the end of each task.

The total number of marks for this paper is 100.

This document consists of **9** printed pages.

1 Your program code and output for Task 1 should be saved in a single `.ipynb` file.

Name your Jupyter Notebook as

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

The file `marathon.CSV` contains the full list of athletes who took part in the 42.195km marathon race. The first line in the file is the heading for the records. Each subsequent line is a record of a runner in the form:

`<name of athlete>,<country code>,<timing in h:mm:ss>`

For example, `Abdi ABDIRAHMAN,USA,2:18:27`

Several athletes did not complete the race and their timing is recorded as `'DNF'` to indicate 'did not finish'.

For example: `Alemu BEKELE,BRN,DNF`

Task 1.1

Write program code to find out the number of athletes who did not finish the race and output the following three statements:

```
Number of DNF: x
Total number of athletes: y
Percentage of athletes who finished race: z
```

`x` is the number of athletes who did not finish the race,

`y` is the total number of athletes who participated in the marathon race,

and `z` is the percentage (rounded to 1 decimal place) of athletes who finished the race. [7]

Task 1.2

Write a function `insertionSort` that takes an unsorted list as a parameter, sorts the list using the insertion sort algorithm, and returns the sorted list. [6]

Task 1.3

By making use of the `insertionSort` function from Task 1.2, or otherwise, find out the top 20 athletes and list them in order of rank under the heading (Rank, Country, Name, Timing). [5]

Sample Output:

```
Rank    Country    Name          Timing
1       ABC        Harry TAN     2:08:38
2       XYZ        Andy LEE      2:09:58
3       ...        ...           ...
```

2 Name your Python file as

`TASK2_<your name>_<class>_<index number>.py`

You will build a simplified, one-player version of the classic board game Battleship. Refer to `Task2_Client_SampleOutput.JPG` for the sample output.

In this version of the game:

- The **server program** initialises a grid measuring 4 metres by 5 metres.
- The grid is to be represented on the screen by a rectangular grid.
- Each square metre of the grid is represented by an x-coordinate and a y-coordinate.
- The top left square metre of the grid display has x = 0 and y = 0.
- Use `"O"` to represent an unoccupied space.
- There will be a single ship hidden in a random location.
- The ship only occupies one square metre of the grid.
- The **client program** allows the player to input guesses to sink the ship.
- After each guess, `"X"` represents the incorrect position guessed while `"S"` represents the sunken ship.
- The ship will not appear if it has not been sunk.
- The game is terminated by the server program when **three** guesses are used or the player has guessed the correct position.

Task 2.1

Write the code for the following functions and procedures for the **server program**. [8]

Subroutine Header	Description
`InitialiseGrid(): ARRAY[0:3, 0:4] OF CHAR`	Initialises a 4 by 5 two-dimensional array and returns the array. Use `"O"` to represent blank.
`DisplayGrid(arr: ARRAY[0:3, 0:4] OF CHAR)`	Sends the encoded grid to the client.
`ValidateRow(row: INTEGER): BOOLEAN`	Returns `True` if the row is valid and `False` otherwise.
`ValidateCol(col: INTEGER): BOOLEAN`	Returns `True` if the column is valid and `False` otherwise.
`CheckResult(row,col:INTEGER): BOOLEAN`	Checks if the guess is correct. If the guess is correct, uses `"S"` to represent the sunken ship. Otherwise, uses `"X"` to represent incorrect guess. Returns `True` for the correct guess and returns `False` for the wrong guess.

Task 2.2

The following client program is given in `Battleship_Client.py`.

```
import socket
client_socket = socket.socket()

address = input('Enter IPv4 address of server: ')
port = int(input('Enter port number of server: '))
client_socket.connect((address, port))

while True:
    data = client_socket.recv(1024)
    if b"Enter" in data:
        choice = input(data.decode())
        client_socket.sendall(choice.encode())
        print()
    else:
        print(data.decode())
        if b"GAME OVER" in data or b"YOU WON" in data:
            break

client_socket.close()
```

Write the corresponding **server program** that: [16]

- Instantiates the server socket.
- Binds the socket to localhost and port number 6789.
- Listens for incoming request, accepts incoming request and establishes connection with the client.
- Generates a random position for the hidden ship.
- Sends a “Welcome to Battleship!” message to the client.
- Uses the subroutines coded in Task 2.1 to play the game.
- Sends `"YOU WON!"` to the client for the correct guess and ends the game.
- Sends `"GAME OVER…"` to the client if three guesses are used.
- Closes the sockets.

3 Your program code and output for Task 3 should be saved in a single `.ipynb` file.

Name your Jupyter Notebook as

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

JP Fitness Club is a gym that keeps details of its members. You are tasked to help the club manage the members' details and store them in a SQL database.

There are two types of membership: normal and annual. Each member has a unique membership number, first name, surname, contact number and last visit date recorded.

A normal member deposits a selected amount into their account. Each time the member visits the gym, the entrance fee is deducted from the amount held in his account. The member may top up the account any time.

An annual member pays a fixed fee per year, starting from the date of registration. He may then visit the gym any number of times for the whole year without paying the entrance fee.

Three classes have been identified: `Member`, `NormalMember`, `AnnualMember`.

The class `Member` has these attributes and methods defined on it.

Attribute	**Data type**	**Description**
`memberID`	String	8 digit membership number. First four digits represent the year of joining gym and last four digits are used to make the `memberID` unique, e.g. `20210357`.
`first_name`	String	First name of member, at most 15 characters.
`surname`	String	Surname of member, at most 15 characters.
`contact_number`	String	8 digit contact number.
`last_visit`	String	The date when member last visited the gym, in the format `YYYY-MM-DD`. Initialises to today's date.
`memberType`	String	Indicates type of membership, either "normal" or "annual". Initialises to None.
Method	**Return type**	**Description**
`showMember()`	None	Outputs member's membership number, first name, surname, contact number and last visit date.
`isActive()`	Boolean	Indicates whether a member is active or not. Returns True if the last visit date is within 30 days, otherwise returns False.

The class `NormalMember` inherits from `Member` and has these additional attributes and methods defined on it.

Attribute	Data type	Description
`stored_value`	Float	Amount stored in member's account. Display in 2 decimal places. Initialise to $0.00.
Method	**Return type**	**Description**
`showMember()`	None	Output `memberType` in addition to member's membership number, first name, surname, contact number and last visit date.

The class `AnnualMember` also inherits from `Member` and has these additional attributes and methods defined on it.

Attribute	Data type	Description
`annual_fee`	Integer	Annual fee paid by member. Initialises to $500.
`date_register`	String	Date when member joins annual membership, in format `YYYY-MM-DD`. Initialises to today's date.
Method	**Return type**	**Description**
`showMember()`	None	Outputs `memberType` in addition to member's membership number, first name, surname, contact number and last visit date.

Task 3.1

Write program code using object-oriented programming for the classes `Member`, `NormalMember`, and `AnnualMember`. Include all the identifiers stated and other appropriate methods to access and modify the attributes. [15]

Task 3.2

The text file, `members.TXT`, contains data items for a number of members. Each data item is separated by a comma, with each member's data on a new line as follows:

- membership number
- first name
- surname
- contact number
- member type

Write program code to read in the information from the text file, `member.txt`, creating an instance of the appropriate class for each member, storing each instance in the same list.

Run `showMember` method to display each of the members' details. [5]

Task 3.3

The members' details are to be stored in a SQL database.

The file `JPgym.SQL` contains the SQL code to create database `JPgym.db` with the single table, `Member`. The table will have the following fields:

- MemberID – primary key, text
- FirstName – first name of member, text
- Surname – surname of member, text
- ContactNo – contact number of member, text
- LastVisit – date that member last visited gym, text
- MemberType – indicates 'normal' or 'annual' membership, text

Copy and paste this SQL code into your Python program to create the database and table.

Also, write program code in Python to insert all the information from the file into the `JPgym.db` database.

Run your program and check that all information has been inserted using SQLite database software. [6]

Task 3.4

Write a SQL query code in Python to display all members with "normal" membership in ascending order of `FirstName`.

Display only the following fields from the query: `FirstName, Surname, ContactNo`. [3]

4 JP Mobile sells mobile phones and manages its inventory using a NoSQL database.

Information about the mobile phones is stored in the JSON file `items.JSON`.

The following fields are recorded:

- brand of mobile phone,
- model,
- colour(s) available,
- price in dollar,
- quantity in stock.

Task 4.1

Write program code to import the information from the JSON file into a MongoDB database. Save the information under the `phone` collection in the `jp_mobile` database. Ensure that the collection only stores the information from the JSON file.

Save your program code as

`TASK4_<your name>_<class>_<index number>.py` [4]

Task 4.2

The shop decides to include one or more free gifts for new batches of mobile phones it sells.

Write program code for a user to insert information of a mobile phone by getting user input of the following: brand, model, colour, price, quantity, free gift(s).

Your code should allow user to input one or more free gifts.

If a phone's `brand`, `model` and `colour` already exists, add the new quantity to the existing quantity in the database, and replace the existing price with the new price.

Run your program and insert the following 2 documents:

No.	Brand	Model	Colour	Price	Quantity	Free gift(s)
1	orange	22	black	900	11	power bank
2	solo	A33	red	1300	7	• power bank • earbuds

Add your program code to

`TASK4_<your name>_<class>_<index number>.py` [7]

Task 4.3

Write a function `display_all` that will display all the information in the `phone` collection under these fields: `brand, model, colour, price, quantity, free gift(s).`

If no free gift comes with the phone, print a `None` statement.

Include a final statement that shows the total number of documents in the collection.

Run the `display_all` function to show your output.

Add your program code to

`TASK4_<your name>_<class>_<index number>.py` [6]

Task 4.4

The shop uses a web browser to display the database content. The manager wants to filter the mobile phones by `brand` and display the results in a web browser.

Write additional Python code and the necessary files to create a web application that

- receives a `brand` string from a HTML form, then
- creates and returns a HTML document that enables the web browser to display an ordered list of mobile phones sorted by `price`.

For each document, the web page should include the:

- brand as the heading
- model,
- colour,
- price

Save your program as

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

with any additional files / sub-folders as needed in a folder named

`TASK4_4_<your name>_<class>_<index number>`

Run the web application and input brand as '`solo`' in the webpage.

Save the output of the program as

`TASK4_4_<your name>_<class>_<index number>.html` [12]

<END OF PAPER>