YISHUN INNOVA JUNIOR COLLEGE
JC 2 BLOCK EXAM 2
**Higher 2**

| | | | |
|---|---|---|---|
| CANDIDATE NAME | | | |
| CG | | INDEX NO | |

# COMPUTING **9569/01**

Paper 2 (Lab-based)

**17, 18 Jun 2020**
**3 hours**
**100 Marks**

Additional Materials: Removable storage device with the following files:

- `MRT.TXT`
- `CCAInfo.csv`
- `Civics.csv`
- `Student.csv`
- `StudentCCA.csv`
- `TASK3_2.txt`
- `TASK3_3.txt`
- `Q2 Web App.zip`
- `Q4 Socket Programming.zip`

**READ THESE INSTRUCTIONS FIRST**

Write your name, index number and class clearly on the cover page.

Write in dark blue or black pen on the writing paper provided.
You may use an HB pencil for any diagrams, graphs, tables or rough working.

Do not use staples, paper clips, glue or correction fluid.

Approved calculators are allowed.

Answer **all** questions.

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

**1** The file `MRT.TXT` contains the names of 116 MRT stations in Singapore.

The task is to read the names from the file, store it in a suitable data structure, sort the names and perform a binary search operation.

**Task 1.1**

Write program code to:

- read the names from the file and store them in a suitable data structure
- sort the names into ascending order using **quick sort**
- write the sorted names in a new `SORTED_MRT.TXT` file where the next name is on new line

[14]

**Task 1.2**

Write program code to:

- read the names from the `SORTED_MRT.TXT` file and store them in a suitable data structure
- display the names
- search for a name using **binary search** and return True if the name is in the file, and False otherwise

[9]

**Task 1.3**

Design test data for your program written in **Task 1.2, provide evidence of testing that includes:**

- search for a name that is contained in the file
- search for a name that is not contained in the file

[2]

Save and submit your program codes as `Question_1_<your name>.ipynb`

**2** A school uses a Database Management System (DBMS) application to keep track of the students, their Civics Classes and the CCAs they join.

The fully normalised table descriptions are as follows:

```
Student (MatricNo, Name, Gender, CivicsClass)

Civics (Class, Tutor, HomeRoom)

StudentCCA (MatricNo, CCAName)

CCAInfo (Name, TeacherIC)
```

`Field1` and `Field2` represent the **primary** and **foreign keys** respectively.

**Task 2.1**

Write a Python program to create the database `school.db` with the four tables.

Write a Python program to insert all the information from the data files, `Student.csv`, `Civics.csv`, `StudentCCA.csv` and `CCAInfo.csv`, into the database `school.db`.

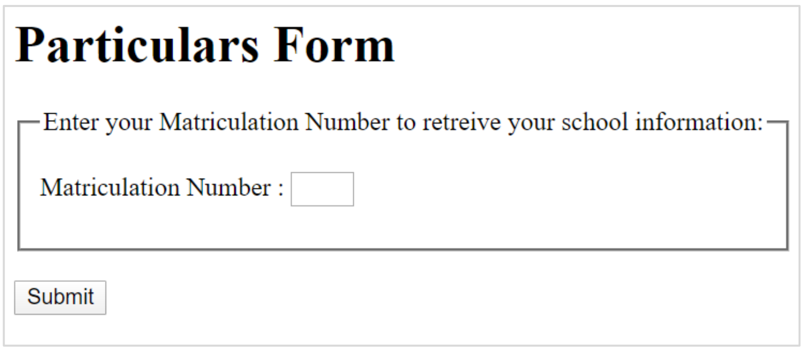
Save and submit your program code as `Task2_1_<your name>.py`

Rename and submit the populated database as `school_<your name>.db`.

[6]

**Task 2.2**

Create a HTML file called `index.html` to display the following Particulars Form for users to submit their request to retrieve their particulars from the back-end server.



**Particulars Form**

Enter your Matriculation Number to retreive your school information:

Matriculation Number :

Submit

Save the file `index.html` in the **Q2 Web App's templates folder**. [3]

The back-end server uses the following program code, `server.py`, to display the Particulars Form on the clients' browser when they visit the school's website.

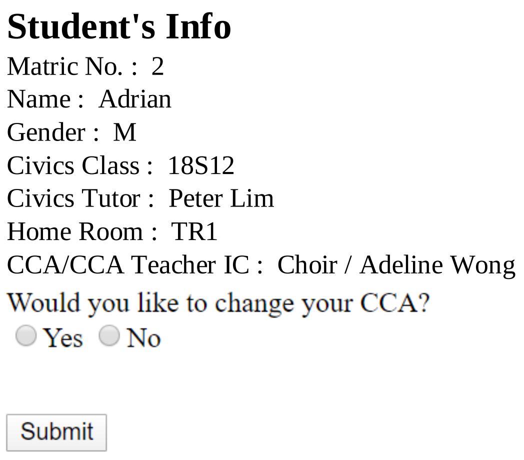
```
from flask import Flask, render_template, request

app = Flask(__name__)

@app.route('/')
def index():
    return render_template("index.html")

app.run(debug=True, port=5000)
```

**Task 2.3**

For the back-end server to receive the input in the Particulars Form, an additional route '`/form`' should be included. Modify the program code to:

- prevent the user from accessing the '`/form`' route directly
- receive the inputs for the matriculation number from the Particulars Form
- reject empty or null inputs
- read from the **given** database `school.db`
- reply by sending a `info.html` page back to the client's browser, displaying clearly the student's information

The display should be similar to the following:



**Student's Info**

Matric No. : 2
Name : Adrian
Gender : M
Civics Class : 18S12
Civics Tutor : Peter Lim
Home Room : TR1
CCA/CCA Teacher IC : Choir / Adeline Wong
Would you like to change your CCA?
◯ Yes ◯ No

Include a radio button in the `info.html` for the user to request to change the CCA.

Save the `info.html` file and the modified `server.py` file in the respective folders within the **Q2 Web App folder**. [7]

If the user chooses not to change the CCA, the given `the_end.html` will be displayed on the client's browser.

If the user chooses to change the CCA, the back-end server uses the following sub-routine within the `server.py` program code to display the list of CCAs available on the clients' browser.

```
import sqlite3
db = sqlite3.connect('school.db')

@app.route('/cca', methods=["POST"])
def cca():
    change_cca = request.form.get("change")
    if change_cca == "N":
        return render_template("the_end.html")
    elif change_cca == "Y":
        db = sqlite3.connect('school.db')
        c = db.cursor()
        c.execute('''SELECT name FROM CCAInfo''')
        data = c.fetchall()
        return render_template("cca.html", data=data)
```

**Task 2.4**

Create a HTML file called `cca.html` to display the CCA Form which

- has the list of the CCAs available in a *drop-down menu*
- allows the user to select **one** CCA to submit to the back-end server

Save the file `cca.html` in the **Q2 Web App's templates folder**. [4]

**Task 2.5**

For the back-end server to process the user's input in the CCA Form, an additional route '`/update`' should be included in the `server.py`. Modify the program code to:

- prevent the user from accessing the '`/update`' route directly
- accept the user's choice of CCA and update his data in the table `StudentCCA` in the **given** database `school.db` using the Matriculation Number
- acknowledge the submission by sending a `success.html` page back to the client's browser, displaying clearly the Matriculation Number and the new CCA

Save the `success.html` file and the modified `server.py` file in the respective folders within the **Q2 Web App folder**. [5]

**3** A program is to be written to implement a food item list using object-oriented programming (OOP) as part of a smart refrigerator application.

The list shows food that are stored in the refrigerator.

Each food is given a category and a description.

The base class will be called `Food` and is designed as follows:

| **Food** |
|---|
| `Name: STRING`<br>`category: STRING`<br>`description: STRING` |
| `constructor (n: STRING, c: STRING, d: STRING)`<br>`set_name (n: STRING)`<br>`set_category (s: STRING)`<br>`set_description (s: STRING)`<br>`get_name (): STRING`<br>`get_category (): STRING`<br>`get_description (): STRING`<br>`summary(): STRING` |

The `summary()` method displays the name, category and description of the food item.

**Task 3.1**

Write program code to define the class `Food`. [5]

Food items should be sorted alphabetically by category. Within each category, food items should be sorted alphabetically by name.

A food item to be added to the list is compared to the items already in the list to determine its correct position in the list. If the list is empty, it is added to the beginning of the list. This comparison will use an additional member method,

```
compare_with (f: Food) : INTEGER
```

This function compares the instance (the item in the list) and the `Food` object passed to it, returning one of three values:

-1 if the instance is before the given `Food`

0 if the two are equal

+1 if the instance is after the given `Food`

**Task 3.2**

There are five food objects defined in the text file `TASK3_2.TXT`.

Write program code to:

- implement the `compare_with()` method
- create an empty list of `Food` objects
- add each of the five objects in the text file `TASK3_2.TXT` to its appropriate place in the list
- print out the list contents using the `summary()` method.

Your program code. Screenshot of the test run. [13]

The food list can have food items with extra information. One such item has a date by which the food should be eaten.

The `DatedFood` class inherits from the `Food` class, extending it to have an `expiry_date`, designed as follows:

| **DatedFood: Food** |
|---|
| expiry_date: DATE |
| constructor (n: STRING, c: STRING, d: STRING, dt: DATE)<br>set_expiry_date (dt: DATE)<br>get_expiry_date (): DATE |

The `DatedFood` class should extend the `compare_with()` method to ensure that food items are ordered by ascending `due_date`, and then by the ordering used by the base `compare_with()` method. The `summary()` method should also be extended to return the `due_date` and the return values of the base `summary()` method.

**Task 3.3**

There are seven objects defined in the text file `TASK3_3.TXT`.

Amend your program code to:

- implement the `DatedFood` class with constructor, `get_expiry_date` and `set_expiry_date`
- implement the extended `compare_with()` method
- implement the extended `summary()` method
- ensure all seven objects in the text file `TASK3_3.TXT` are added to the list
- print out the list contents using the `summary()` method.

Your program code. Screenshot of the test run. [12]

Save and submit your program codes as `Question_3_<your name>.ipynb`

**[Turn over**

**4** In a computer game, a player ("O") moves around a 10 metres by 11 metres grid to collect a prize ("P"). The prize is placed at a random position within the grid.

Starting from the centre of a grid, the player moves left, right, up or down according to a direction entered by the user. The game is turn-based; a user enters the direction, their player move one position in that direction. If the direction entered is invalid, then the player does not move. The grid is displayed after each move. When the player reaches the position of the prize, the message "You got the Prize in **n** steps!", where **n** is the value from the counter.

The program code of the game is provided in `maze.py`, run it and examine how the code execute the game. **Do not amend the program code provided.**

Using the templates provided, write the server and client program codes for this turn-based game. [20]

**`server.py`**

```
# copy the necessary program code from maze.py
# and paste it here :






from socket import socket

my_socket = socket()
my_socket.bind(('127.0.0.1', 12345))
my_socket.listen()

print('Type Ctrl-C or close the shell to terminate the
game.')
new_socket, addr = my_socket.accept()
print('Connected to: ' + str(addr))

while True:
     # Write your server code here:



new_socket.close()
my_socket.close()
```

**client.py**

```
from socket import socket

my_socket = socket()
my_socket.connect(('127.0.0.1', 12345))

while True:
     # Receive data from the Server
     data = my_socket.recv(1024).decode()
     print(data)

     # Write the code to prompt the user
     # to input the move direction:





my_socket.close()
```

Save the `server.py` and `client.py` files in the **Q4 Socket Programming folder**.

BLANK PAGE