Name:		Index Number:		Class:	

DUNMAN HIGH SCHOOL
Preliminary Examination
Year 6

COMPUTING (Higher 2) 9569/02

Paper 2 (Lab-based) **14 September 2023**

3 hours

Additional Materials: Insert
Electronic version of `CLIENT.py` file
Electronic version of `SERVER.py` file
Electronic version of `Task2_timing.py` file
Electronic version of `STUDENTS.csv` file
Electronic version of `CANDIDATES.csv` file
Electronic version of `VOTERS.csv` file
Electronic version of `VOTES.csv` file

READ THESE INSTRUCTIONS FIRST

Write your name, index number and class on the work you hand in.
Write in dark blue or black pen on both sides of the paper.
You may use an HB pencil for any diagrams or graphs.
Do not use staples, paper clips, glue or correction fluid.

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 question or part question.
The total number of marks for this paper is 100.

Instructions to candidates:
Your program code and output for each of Task 1 to 4 (**except Tasks 1.3, 4.3, 4.4 & 4.5**) 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>_<centre number>_<index number>.ipynb`

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

1 A spy agency implemented an enhanced Caesar cipher encryption algorithm for its spies to hide their messages sent to headquarters. The encryption works in the following manner:

- Step 1: Convert each character of the message into its ASCII number representation. [Note: use function `ord()` to convert a character into its ASCII number] E.g.,
  `urgent!!!! :117, 114, 103, 101, 110, 116, 33, 33, 33, 33`
- Step 2: Add the ASCII number representation of the first character by the length of the message. For subsequent characters, add the ASCII number representation by 1 less than the previously added number . E.g.,
  `117+10, 114+9, 103+8, 101+7, 110+6, 116+5, 33+4, 33+3, 33+2, 33+1`
  `Result: 127, 123, 111, 108, 116, 121, 37, 36, 35, 34`
- Step 3: When a value goes beyond `'~'` (ASCII value 126), it loops back to `Space` (ASCII value 32). For example, in the above case, the letter 'u' will be encrypted as Space.
  `Result: 32, 123, 111, 108, 116, 121, 37, 36, 35, 34`
- Step 4: Convert each number to its ASCII character. [Note: use function `chr()` to convert a number into its ASCII character]
  `32, 123, 111, 108, 116, 121, 37, 36, 35, 34 -> {olty%$#"`

The following table shows the ASCII values of some of the characters.

ASCII value	Character	ASCII value	Character
32	space	60	<
33	!	61	=
34	"	62	>
35	#	63	?
36	$	64	@
37	%	65	A
38	&	90	Z
39	'	91	[
40	(	92	\
41	)	93	]
42	*	94	^
43	+	95	_
44	,	96	`
45	-	97	a
46	.	122	z
47	/	123	{
48	0	124	|
57	9	125	}
58	:	126	~
59	;		

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 [ ]: #Task 1.1
        Program code

        Output:
```

Task 1.1
Write the function `encrypt(plaintext)` that takes in the message string, encrypt the message as described above and returns the encrypted message. [5]

Test your function `encrypt(plaintext)` by calling it using the following statement:

```
print(encrypt('urgent!!!!!') == ' {olty%$#"')
```

The statement should print `True`. [1]

Task 1.2
Write the function `decrypt(ciphertext)` that takes in the encrypted message string, decrypt the message and returns the original message. [5]

Test your function `decrypt(ciphertext)` by calling it using the following statement:

```
print(decrypt(' {olty%$#"') == 'urgent!!!!!')
```

The statement should print `True`. [1]

Task 1.3
Using socket programming, complete both the client program used by the spies to send messages and the server program used by the headquarters to receive these messages. [13]

- Copy the function `encrypt(plaintext)` into the client program
- Copy the function `decrypt(ciphertext)` into the server program
- Complete the server program to
  - receive the encrypted message from the client program
  - decrypt the message received
  - write the received message into `LOG.csv` in the following format (one message per line)
    `Date/Time, client ip address, client port number, encrypted message, decrypted message`

    [Note: `import datetime` and use `datetime.datetime.now()` to get the current Date/Time]

  - prompt the user(server) whether to continue listening and restart the socket to wait for the next message if yes. Otherwise end the server program.

You are provided with the server and client template programs, `SERVER.py` and `CLIENT.py` respectively. Complete both programs and rename as

```
CLIENT_<your name>_<centre number>_<index number>.py
SERVER_<your name>_<centre number>_<index number>.py
```

Study the following sample program output to determine your code design, output format and socket protocol. User inputs are underlined.

Sample server program:

```
------------------
SERVER OPEN
------------------

Waiting for message
------------------
Encrypted message: f"+*~$"2uu$ovx~*{mjknzhf/
Decrypted message: Mission details received.
------------------
Do you want to continue listening? [Y/N]: Y

Waiting for message
------------------
Encrypted message: _z$#w|z+mxuwrjxhf"
Decrypted message: Mission completed!
------------------
Do you want to continue listening? [Y/N]: N

------------------
SERVER CLOSED
------------------
```

Sample client programs (in sequence):

CLIENT 1

```
------------------
CLIENT OPEN
------------------

Enter message: Mission details received.
message sent
------------------
CLIENT CLOSED
------------------
```

CLIENT 2

```
------------------
CLIENT OPEN
------------------

Enter message: Mission completed!
message sent
------------------
CLIENT CLOSED
------------------
```

Save your Jupyter Notebook and Python files for Task 1.

2 Name your Jupyter Notebook as:

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

This task is to compare the searching efficiency of Hash Table versus Binary Search on a sorted list.

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 [ ]: #Task 2.1
        Program code

        Output:
```

Task 2.1
Write a function, `task2_1()` to:

- initialise a global 1-dimensional list
- generate 50 random integers between 1 and 1000 (inclusive) [Note: `import random` and use `random.randint()` to generate the random integers]
- store each integer in the list
- output the contents of the list. [2]

Test your program and show the output. [1]

Task 2.2
Implement a Hash Table with 10 buckets that uses chaining with Linked List for its collision resolution.

- The Hash Table, Linked List and Node are implemented using Object-Oriented Programming (OOP)
- Write program code for the 3 classes based on the specifications below
- Create the necessary Hash Table, Linked List and Node objects
- Insert all the values in the global list from Task2.1 into the Hash Table.
- Display the Hash Table. [17]

Sample Final Output for displaying hash table (your values will be different):

```
#Each bucket has its own LinkedList with 0 to many Nodes

index 0: [710, 660, 410, 670]
index 1: [241, 301, 61, 651]
index 2: [192, 372, 532, 22]
index 3: [363, 633, 253, 553]
index 4: [244, 414, 594, 964]
index 5: [465, 75, 295, 15, 795, 525, 725]
index 6: [296, 96, 136, 416, 336, 976, 356, 916, 206]
index 7: [587, 337, 507]
index 8: [28, 888, 288, 528, 378, 308, 348, 628, 208]
index 9: [249, 789]
```

Class: Node		
Identifier	**Data Type**	**Description**
`data`	`Integer`	• The Node's data
`next`	`Node Object`	• The next Node in the Linked List. • Default value is `None`.
`get_data()`	`Function`	• Return the value of the `data` attribute
`get_next()`	`Function`	• Return the next Node object
`set_data(value)`	`Procedure`	• Set the value of the `data` attribute with the given value
`set_next(nextNode)`	`Procedure`	• Set the value of the `next` attribute with the given Node object

Class: LinkedList		
Identifier	**Data Type**	**Description**
`head`	`Node Object`	• The first Node in the Linked List. • Default value is `None`.
`add_to_end(value)`	`Procedure`	• Create a new Node object with the given value • Add the new Node object to the end of the Linked List.
`search(target)`	`Function`	• Search for the target value in the Linked List. • Return `True` if found, otherwise return `False`.
`get_values()`	`Function`	• Return all the Nodes' data in the Linked List as a python list • Return `"Empty Linked List"` if Linked List is empty

Class: HashTable		
Identifier	**Data Type**	**Description**
`size`	`Integer`	• The size of the Hash Table • Set the size to **10**
`array`	`Python List of LinkedList Objects`	• Python List containing **10** Linked List objects
`hash(value)`	`Function`	• Map the given value to the array index using the formula `value%size` • Return the array index

`insert(value)`	`Procedure`	• Insert given value in the correct Linked List object in the `array` attribute
`search(target)`	`Function`	• Search for the target value in the correct Linked List object in the `array` attribute • Return `True` if found, otherwise return `False`.
`display()`	`Procedure`	• Display all the values in the Hash Table (follow the Sample Final Output shown above)

Task 2.3

Write the procedure `task2_3()` to search for every integer present in the global list from Task 2.1 in the Hash Table created in Task 2.2. [2]

Task 2.4

Another method to search for specific values in the global list is to first sort the list using merge sort followed by performing binary search.

- Write a separate function to perform merge sort on the given list and return the sorted list.
- Write a separate function to perform binary search on the given sorted list. The function is to return `True` if the target value is found, otherwise return `False`.
- Write the procedure `task2_4()` to execute merge sort on the global list from Task 2.1 followed by searching for every integer present in the global list using the binary search function. [12]

Task 2.5

The `timeit` library is built into Python and can be used to time simple function and procedure calls. Example code is shown in `Task2_timing.py`.

Using the `timeit` module, display the time taken to execute the procedures `task2_3()` and `task2_4()`. [1]

Save your Jupyter Notebook for Task 2.

3 Name your Jupyter Notebook as:

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

Our school wants to use Python Programming and object-oriented programming to store information about its students.

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 [ ]: #Task 3.1
        Program code

        Output:
```

Task 3.1
The `Student` class has the following **private** data fields:

- `name` – stored as a string
- `gender` – stored as a string, for example, Male, Female
- `date_of_birth` - initialized with a string with the format YYYY-MM-DD

The class contains all the appropriate methods to set and access the above private data fields. It also includes one additional method:

- `get_age( )` – calculates and returns the age of the `Student` by deducting year of birth from the current year [Note: `import datetime` and use `datetime.datetime.now().year` to get the current year]

Write program code in Python to define the class `Student`. [4]

Task 3.2
The `JuniorHigh` class and `SeniorHigh` class inherits from the `Student` class. Both have one additional method:

- `get_zoom_name( )` – returns a string which is used as the identifier for school-based Zoom calls. It is constructed as follows:
  - `[JH]` for Junior High students and `[SH]` for Senior High students
  - followed by the name of the `Student` with all spaces and punctuation removed
  - followed by the `Student` age

For example, Max Lee born on 31st December 2005 ("`2005-12-31`"), would have the zoom name "`[SH]MaxLee18`"

The `SeniorHigh` class also has an additional **private** data field:

- `house` – stored as a string, for example, Drakon

Write program code in Python to define the classes `JuniorHigh` and `SeniorHigh`. [4]

Task 3.3

The csv file, `STUDENTS.csv`, contains information of a number of students. Each row is a comma-separated list of data of the following:

- Name
- Gender
- Date of Birth in the form YYYY-MM-DD
- House (for Senior High Students only)

Write a program to:

- Read in the information from the csv file, creating an instance of the `SeniorHigh` class for students with House and an instance of the `JuniorHigh` class for students without House
- Store all instances into a global student list named `student_list`
- Sort the students by age using the code `student_list.sort(key=lambda x: x.get_age())`
- Show the zoom name of all students in `student_list` [4]

Save your Jupyter Notebook for Task 3.

4 Name your Jupyter Notebook as:

`TASK4_<your name>_<centre number>_<index number>.ipynb`

Your school recently concluded an election for the new President of the Student Council. You helped the school to keep track of the votes and created a web application to showcase the voting outcome. The database used for the web application has three tables: a table to store the candidates' information, a table to store the voters' information and a table to store which candidate each voter voted for.

```
Candidate (CID, Name, Gender, Class)
Voter (VID, Name, Gender, Class)
Vote (VID, CID)
```

`Candidate:`

- `CID` – unique candidate number, for example, 2
- `Name` – the name of the candidate
- `Gender` – the gender of the candidate, for example, Male, Female
- `Class` – the class of the candidate, for example, 6C11

`Voter:`

- `VID` – unique voter number, for example, 15
- `Name` – the name of the voter
- `Gender` – the gender of the voter, for example, Male, Female
- `Class` – the class of the voter, for example, 6C11

`Vote:`

- `VID` – unique voter number, for example, 15
- `CID` – unique candidate number, for example, 2

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 [ ]: #Task 4.1
        Program code
```

`Output:`

Task 4.1

Write a Python program that uses SQL code to create the database `election.db` with the three tables given. Define the primary and foreign keys for each table. [3]

Task 4.2
The files `CANDIDATES.csv`, `VOTERS.csv` and `VOTES.csv` store comma-separated values for `Candidate`, `Voter` and `Vote` tables respectively.

The data in `CANDIDATES.csv` is given in the following order:

```
CID, Name, Gender, Class
```

The data in `VOTERS.csv` is given in the following order:

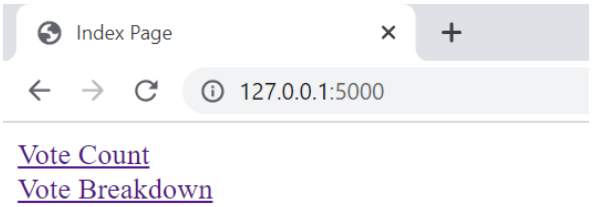
```
VID, Name, Gender, Class
```

The data in `VOTES.csv` is given in the following order:

```
VID, CID
```

Write a Python program to read in the data from each file and store the data in the correct place in the database. [5]

Task 4.3
Write a Python program and the necessary files to create a web application. The application offers the following menu options:



Save your Python program as:

`Task_4_3_<your name>_<centre number>_<index number>.py`

with any additional files/subfolders in a folder named:

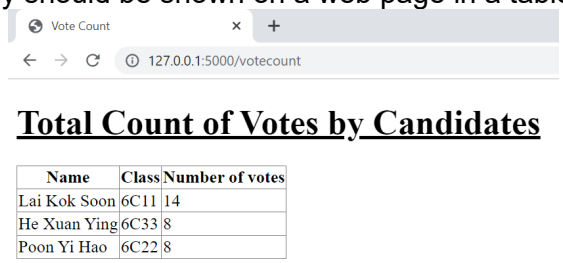
`Task_4_web_<your name>_<centre number>_<index number>`

Run the web application. [4]

Task 4.4
Write an SQL query that shows:

- the `name`, `class` and `total number of votes` for each `candidate`
- the `total number of votes` sorted in descending order

The results of the query should be shown on a web page in a table as shown below:



This web page should be accessed from the menu option (Vote Count) from Task 4.3.

Save your SQL code as

`Task_4_4_<your name>_<centre number>_<index number>.sql` [2]

Modify the code in your below Python program:

`Task_4_3_<your name>_<centre number>_<index number>.py`

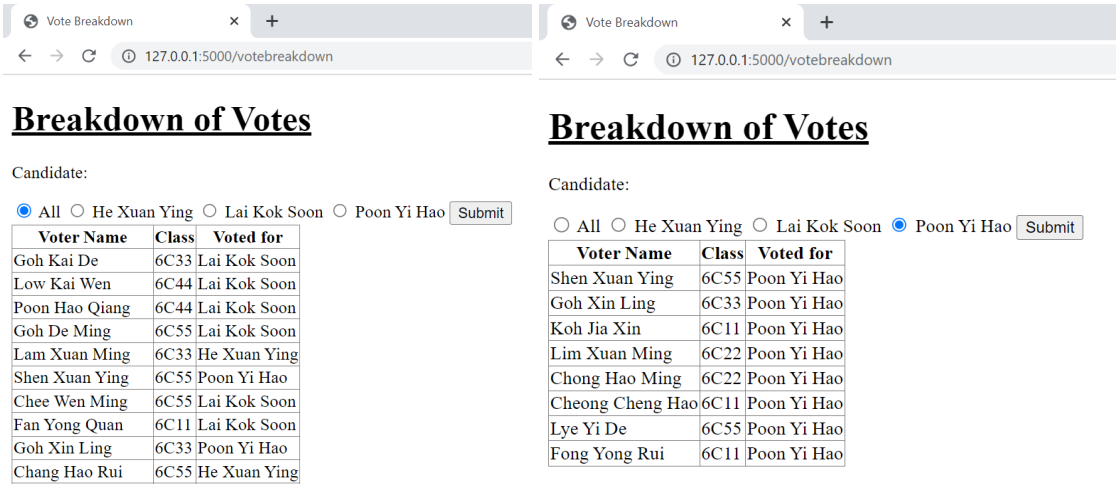
with any additional files/subfolders in the folder named:

`Task_4_web_<your name>_<centre number>_<index number>` [7]

Run the web application.

Task 4.5

Modify your Python program and create the necessary file(s) to create a web page that shows the details of the voters who voted for a particular candidate. There should be a form with radio buttons for user to select one of the candidates. The radio button `All` should be selected by default. [Note: to select the `All` radio button as the default option, add `checked="checked"` as one of the attributes of the input tag for the `All` radio button]



This web page should be accessed from the menu option (Vote Breakdown) from Task 4.3.

Save any additional files/subfolders in a folder named:

`Task_4_web_<your name>_<centre number>_<index number>` [6]

Run the web application and submit the html form with Lai Kok Soon.

Save the webpage output as:

`Task_4_5_<your name>_<centre number>_<index number>.html` [1]

Save all your files for Task 4.

End of Paper