RIVER VALLEY HIGH SCHOOL
General Certificate of Education Advanced Level
Higher 2
Preliminary Examination

COMPUTING

9569/02

Paper 2 (Lab-based)

15 AUG 2023
3 hours

Additional Materials: Electronic version of:
`Animals_lst.txt`
`Task2a.txt`
`Task2b.txt`
`Teachers.csv`
`GUARDIAN.csv`
`RESULT.csv`
`STUDENT.csv`
Insert Quick Reference Guide

READ THESE INSTRUCTIONS FIRST

Answer **all** questions.

All tasks must be done in 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.

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

Instruction 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 downloaded 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>_<centre number>_<index number>.ipynb`

The task is to implement the class Animal and its inherited classes using OOP based on the information described below.

Class `Animal`	
Attributes	
`name`	`<string>` It stores the name of the animal
`mass`	`<integer>` It stores the mass of the animal in grams.
`volume`	`<integer>` It stores the volume of the animal in cm^3
`age`	`<float>` It stores the age of the animal in year to 1 decimal place.
Functions	
getters	The getter functions for all the attributes of the class `Animal`
`isSick()`	This function returns `True` if the density of the animal is less than 0.8 and `False` otherwise.
`__str__()`	This function returns a string that shows all its attributes. For example: `"Name: D01   Age: 0.5   Mass: 1007  volume: 1009"`

Class `WarmBloodAnimal` (inherits class `Animal`)	
Additional attributes	
`bodyTemp`	`<float>` It stores the body temperature of the animal to 1 decimal place.

Class `ColdBloodAnimal` (inherits class `Animal`)	
Additional attributes	
`isPoisonous`	`<Boolean>` `True` means the animal is poisonous and `False` otherwise.
`getIsPoisonous`	The getter function for `isPoisonous`

Class `Dog` (inherits class `WarmBloodAnimal`)	
Attributes	
`hasOwner`	<Boolean> `True` means the **Dog** has owner and `False` otherwise
Functions	
`hasFever()`	This function returns `True` if `bodyTemp` is higher than 39.0 and `False` otherwise.
`isSick()`	This function behaves the same way as that of its parent class. On top of it, it returns `True` if the **Dog** has fever.
`isYoung()`	This function returns `True` if the age of the **Dog** is less than 1.0 and `False` otherwise
`__str__()`	This function returns a string that shows its class and **all** its attributes and indicates if it is sick/well and if it is young/adult.

Class `Goose` (inherits class `WarmBloodAnimal`)	
Functions	
`hasFever()`	This function returns `True` if `bodyTemp` is higher than 43.0 and `False` otherwise.
`isSick()`	This function behaves the same way as that of its parent class. On top of it, it also returns `True` if the **Goose** has fever.
`isYoung()`	This function returns `True` if the age of the **Goose** is less than 0.5 and `False` otherwise
`__str__()`	This function returns a string that shows its class and **all** its attributes and indicates if it is sick/well and if it is young/adult.

Class `Frog` (inherits class `ColdBloodAnimal`)	
Functions	
`isYoung()`	This function returns `True` if the age of the **Frog** is less than 0.3 and `False` otherwise.
`__str__()`	This function returns a string that shows its class and **all** its attributes and indicates if it is sick/well and if it is young/adult.

Class `Snake` (inherits class `ColdBloodAnimal`)	
Attributes	
`length`	`<float>` It stores the length of the snake in m and to 1 dp.
Functions	
`isSick()`	This function first calculates the normal mass of a **Snake** based on its length. If the mass of the **Snake** is less than 0.8 of the calculated mass, it returns `True`. Otherwise, it returns `False`. The formulae to calculate the mass of the snake in g based on its length in m is 0.00035 x (snake's length x 100)^3.
`isYoung()`	This function returns `True` if the age of the **Snake** is less than 0.5 and `False` otherwise
`__str__()`	This function returns a string that shows its class and **all** its attributes and indicates if it is sick/well and if it is young/adult.

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 to declare all the classes described above.

[18]

Task 1.2

Write program code to:

- create 8 instances of the following:
    - 1 Young Well Dog that is not owned with name "D01"
    - 1 Adult Sick Dog that is not owned with name "D02"
    - 1 Young Well Goose with name "G01"
    - 1 Adult Sick Goose with name "G02"
    - 1 Young Well non-poisonous Frog with name "F01"
    - 1 Adult Sick poisonous Frog with name “F02”
    - 1 Young 1m long Well non-poisonous Snake with name “S01”
    - 1 Adult 4m long Sick poisonous Snake with name “S02"
- Add all the 8 instances in a list. Iterate it and output all of them using the `__str__()` method.

[5]

Task 1.3

Write a function, `task1_3()` to:

- read the file *“Animals_lst.txt”*
- process its content and create the appropriate instances for each animal
- add all the instances in a Python list and return it.

The format of the information in *“Animals_lst.txt”* is as follow:

- If it is a dog,
  `Dog, name, mass, volume, age, body_temperature, hasOwner`
- If it is a goose,
  `Goose, name, mass, volume, age, body_temperature`
- If it is a frog,
  `Frog, name, mass, volume, age, isPoisonous`
- If it is a snake,
  `Snake, name, mass, volume, age, isPoisonous, length`

[3]

Task 1.4

Write a function, `task1_4(a_list, animal_type)` to:

- iterate `a_list` which contains different animal instances
- collect 4 pieces of information based on `animal_type` (see example below)
- output it in a table in the same format as show below.

For example,

```
>>> a_list = task1_3()
>>> task1_4(a_list, Dog)
-------------------------------
| Dog      | Well     | Sick     |
|-----------------------------|
| Young    | 8        | 4        |
|-----------------------------|
| Adult    | 6        | 2        |
-------------------------------
```

[3]

Test your program with the following test data:

```
a_list = task1_3()
task1_4(a_list, Dog)
task1_4(a_list, Goose)
task1_4(a_list, Frog)
task1_4(a_list, Snake)
```

[1]

Task 1.5

Write a function, `task1_5(a_list, animal_type)` to:

- find all the young animal of `animal_type` that has its volume larger than that of the smallest adult animal of the same animal type
- return the **name** of this group of young animals in a list.

[4]

Task 1.6

Write a function, `task1_6(a_list)` to:

- filter out all the Cold Blood Animal in `a_list`
- perform merge sort on them according to their mass in ascending order
- return only the names of the cold blood animals in a list sorted according to their mass in ascending order.

[4]

Test your program with the following test data:

```
a_list = task1_3()
task1_6(a_list)
```

[1]

2 Name your Jupyter Notebook as:

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

The task is to implement the class `Node` and class `SortedLinkedList` using the **free space** concept.

Class `Node`	
Attributes	
`data`	`<string>` It stores the data in the node.
`next_ptr`	`<integer>` It stores the index of an array which leads to the next node

Class `SortedLinkedList`	
Attributes	
`start`	`<integer>` It stores an index to the `nodes` array which represents the first node of the sorted linked list. `-1` means the sorted link list is empty.
`next_free`	`<integer>` It stores an index to the `nodes` array which represents the position of the first-free-to use node instance. `-1` means there is no free node available.
`nodes`	`<list of 50 Node instances>` It stores all used and unused node instances which is linked up by the `start` and `next_free` pointers respectively.
Functions	
`SortedLinkedList()`	This constructor initiates the sorted linked list attributes as follow: • Assign `start` to `-1` • Assign `next_free` to `0` • Append 50 `Node` instance to `nodes` • Link up all 50 unused `Node` instances under the `next_free` pointer.
`addNode(data)`	This procedure adds `data` which is an alphabetical string into the sorted linked list. The main steps performed are: • Identify an unused node from the linked list headed by the `next_free` pointer. • Use the unused node to store `data` • Detach the same node from the linked list headed by the `next_free` pointer. • Traverse the linked list (headed by `start`) from the head to the position to insert `data`. • Update the pointers so that `data` is now in the sorted link list.

`display_content()`	This procedure outputs the content of the sorted linked list as follow: `The items in the list in alphabetical order: <Item1>, <Item2>, <Item3>, <Item4>`

`findCommon (sorted_list)`	This function takes `sorted_list` which is also an `SortedLinkedList` instance and returns a list that contains all node data that presents in both `self` and `sorted_linked_list`.
	
	For example, if `sll1` is a `SortedLinkedList` instance which contains the following data:
	• Apple
	• Lemon
	• Orange
	• Pear
	and `sll2` is a `SortedLinkedList` instance which contains the following data:
	• Apple
	• Banana
	• Pear
	• Mango
	Then,
	`>>> sll1.findCommonItems(sll2)`
	`[“Apple”, “Pear”]`

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 program code to implement the two classes described above.

[14]

Task 2.2

Write a function, `task2_2()` to:

- create a `SortedLinkedList` instance
- add the following words in the same order in the sorted linked list:
    - "Orange"
    - "Banana"
    - "Grapes"
    - "Apple"
    - "Lemon"
    - "Eggplant"
- output the items in the sorted linked list.

[2]

Task 2.3

Write a function, `task2_3(filename)` to:

- read the file identified by `filename`
- split the content and return them in a list.

[1]

Output the content of the files *"Task2a.txt"* and *"Task2b.txt"* using the following statements.

```
Itemlst1 = Task_2_3("Task2a.txt")
print(itemlst1)
itemlst2 = Task_2_3("Task2b.txt")
print(itemlst2)
```

[1]

Task 2.4

Write a function, `task2_4()` to:

- read the items from *"Task2a.txt"* and *"Task2b.txt"* and add them into two new `SortedLinkedList` instances `sl1` and `sl2` respectively,
- output all the common items in the two sorted linked lists using the class function `findCommon(sorted_list)`

[1]

3 Name your Jupyter Notebook as:

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

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 a function, `task3_1()` to:

- read the file *"Teachers.csv"*
- create a Mongo client to connect to the database
- use the data to populate the collection `teacher` in the database `school`
- do not create any field in the document if the value of the field is not provided. For example, the document with the field `Name Juli Barnhill` does not have a `Role` value, hence the field `Role` should not be in the document.
- data should be converted to appropriate data type before it is inserted into the database collection.

[5]

Task 3.2

Write a function, `task3_2()` to:

- query the collection `teacher` in the Mongo database `school`
- output all documents in the collection `teacher`
- If the field of the document does not exist, output `'-'` instead.

[3]

Task 3.3

Write a function, `task3_3()` to:

- query the collection `teacher` in the Mongo database `school`
- output the only the **_id** and **name** of teachers who **EITHER**
  - o do not have both Contact and Role **OR**
  - o have a rating below 1600

[2]

Task 3.4

Write a function, `task3_4()` to:

- update 1 teacher from the Physics department to Computing department in the collection `teacher` in the Mongo database `school`

[2]

4 Name your Jupyter Notebook as

```
Task4_<your name>_<centre number>_<index number>.ipynb
```

In a parallel universe, ABC Secondary School wishes to digitalise the enrolment process for its secondary 1 students. The school will need to create a suitable database to store records of student applicants' details, results, and guardian contacts for the process. The database will have three tables: a table to store data about the students who opt to the school as 1st choice; a table about the results obtained by the students and a table to store guardian email contact of the students. All fields cannot be left empty.

`guardians:`

- `guardian_id` – unique guardian identification. For example, 900001
- `guardian_name` – name of guardian
- `guardian_email` – contact email of guardian

`students:`

- `student_id` – unique student identification. For example, 10001
- `stu_name` – name of student
- `gender` – gender of student. Can only accept male or female
- `primary_sch` – primary school which the student is from
- `guardian_id` – the unique identification reference for guardian. Different students may be taken care by same guardian

`results:`

- `student_id` – unique student identification. For example, 10001
- `subject` – subject name which student has taken.
- `al` – achievement level of the student's subject between level 1 to 8. Can only accept `1` to `8`

For each of the sub-class 4.1 to 4.3, complete the codes in Jupyter Notebook and 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

Write a Python program that uses SQL code to create database ENROLMENT with the three tables given. Define the primary and foreign keys for each table. Include the checks needed to constrain the values.

[7]

Task 4.2

Sample data sets for the different tables are created for testing the database and the programs using it. The csv files *“GUARDIAN.csv”*, *“STUDENT.csv”* and *“RESULT.csv”* store the comma-separated values for each of the tables in the database.

Write a Python program to read in the data from each file and then store each item of the data in the correct place in the database.

[6]

Task 4.3

The contact email of guardian is subjected to frequent changes thus would need to have a program to facilitate updates.

Write a Python program that uses SQL code to update the contact email according to guardian_id. Run the program to update `guardian_id 900005` email contact to `see@see.com`

[2]

Task 4.4

The overall achievement level (AL) of a student is calculated by summing the achievement levels obtained for the 4 subjects by the student.

You can assume that the data sets are complete with all the students submitted having AL of all 4 subjects captured in the database.

Write a Python program and the necessary files to create a web application. The web application will return a HTML page to display a summary table with the following header and its associated information obtained and calculated from the enrolment database query using SQL code.

- `Primary School` – primary school name
- `No. of Students` – number of students that are from the primary school
- `Mean Achievement Level` – average overall achievement level obtained by the students

The table should look below:

Primary School	**No. of Students**	**Mean Achievement Level**
Boon Day Park Primary	4	9.25
…	…	…

The table will display the value rows in descending alphabetical order according to primary school name. Mean achievement level will be in 2 decimal places. The table shown on webpage must have visible border.
Consider the following in the SQL for the query:

- multiply an `INTEGER` value by `1.0` to cast it into `REAL` value
- `ROUND (`*`real number`*`, 2)` to limit SQL `REAL` aggregated values to 2 decimal places.

Save your program code as
`Task_4_4_5_<class>_<index>_<name>.py`
With any additional files / subfolders as needed in a folder named
`Task_4_4_5_<class>_<index>_<name>`
Run the web application and save the returned page as
`Task_4_4_<class>_<index>_<name>.html`

[7]

Task 4.5

Students whose overall AL is lower or equal to a cutoff point will be enrolled into the school.

In the web application on the same web page as Task 4.4, add a text input for entering the cutoff point with an appropriate label and a submit button.

On submitting the cutoff point, the web application will return a page with a table containing student information of students applicants who will enroll into the school. The table will have the following headers and the associated information for each student record – **student name, gender, primary school, overall AL, guardian name and guardian email**

Update the Python program and create the necessary files in the same folder as Task 4.4 such that the web application will return a HTML page to display the student information table obtained and calculated from the enrolment database query using SQL code.

The table will display the value rows in descending alphabetical order according to primary school name then followed by overall AL in ascending order. Table shown on webpage must have visible border.

Run the web application with cutoff value of 11 and save the returned pages as
`Task_4_5_<class>_<index>_<name>_1.html`
`Task_4_5_<class>_<index>_<name>_2.html`

[8]

End of paper