

RIVER VALLEY HIGH SCHOOL
General Certificate of Education Advanced Level
Higher 2
J2 Prelim

COMPUTING

Paper 2 (Lab-based)

9569/02
15 Aug 2025
3 hours

Additional Materials: Electronic version of the following data file
in the folder named “student resources”

- `fishes.txt`
- `useful_code.txt`
- `task3_template.ipynb`
- `candidates.csv`
- `groups.csv`
- `candidate_group.csv`
- `votes.csv`
- `logo.png`

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.

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

This document consists of 16 printed pages.

1 Fish

Name your Jupyter Notebook as

Task_1_<class>_<index number>_<name>.ipynb

For each of the Task 1.1 to 1.3, 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 1.1
Program code`

Output:

Task 1.1

Implement the class `Fish` based on the information described below.

[2]

Class <code>Fish</code>	
Attributes	
<code>name</code>	<string> It stores the name of the fish. <i>You can assume that the names used to create <code>Fish</code> instances are always unique.</i>
<code>size</code>	<integer> It stores the size of the fish.
Functions	
Getters	The getter functions for all the attributes of the class <code>Fish</code> .
<code>__str__()</code>	This function returns a string that shows all its attributes. You should allocate fixed 20 and 4 spaces for the name of the fish and size respectively. For example: "Name:Giant Trevally Size: 33"
<code>consume(fish)</code>	This function takes a <code>Fish</code> instance <code>fish</code> as input and compares its size with <code>self.size</code> . If <code>fish</code> has a larger size than <code>self.size</code> , it returns a new <code>Fish</code> instance with its name equal that of <code>fish</code> . Otherwise, it returns a new <code>Fish</code> instance with its name equal that of <code>self</code> . The size of this new <code>Fish</code> instance created in both cases are equal to the sum of the size of <code>fish</code> and <code>self</code> . For example: >>> f1 = Fish("Giant Trevally", 33) >>> f2 = Fish("Pollock", 59) >>> f3 = f1.consume(f2) >>> str(f3) "Name:Pollock Size: 92"

Task 1.2

Implement the class `Node` and class `SortedLinkedListForFish` using the **free space list** based on the information described below.

Class <code>Node</code>	
Attributes	
<code>data</code>	It stores the data in the node. For this task, data is either <code>None</code> or a <code>Fish</code> instance.
<code>next_ptr</code>	<integer> It stores the array index that leads to the next node.
Functions	
<code>__init__(data)</code>	This constructor initiates the <code>Node</code> attributes as follow: - Assign the argument <code>data</code> to attribute <code>data</code> where <code>data</code> is either <code>None</code> or a <code>Fish</code> instance. - Assign <code>-1</code> to <code>next_ptr</code>

Class <code>SortedLinkedListForFish</code>	
Attributes	
<code>start</code>	<integer> It stores an index to the <code>nodes</code> array which represents the first node of the sorted linked list. <code>-1</code> means the sorted linked list is empty.
<code>next_free</code>	<integer> It stores an index to the <code>nodes</code> array which represents the position of the first-free-to-use node instance. <code>-1</code> means there is no free node available.
<code>count</code>	<integer> It stores the total number of <code>Fish</code> instances stored by the linked list.
<code>nodes</code>	It stores all used and unused <code>Node</code> instances which are linked up by the <code>start</code> and <code>next_free</code> pointers respectively.
Functions	
<code>__init__()</code>	This constructor initiates the sorted linked list attributes as follow: - Assign <code>-1</code> to <code>start</code> - Assign <code>0</code> to <code>next_free</code> - Assign <code>0</code> to <code>count</code> - Append 50 <code>Node</code> instances to <code>nodes</code>. You should use <code>Node(None)</code> to create these <code>Node</code> instances. - Link up these 50 <code>Node</code> instances under the <code>next_free</code> pointer.
<code>insert(fish)</code>	This procedure inserts <code>fish</code> which is a <code>Fish</code> instance into the logical sorted linked list (as its node's data) based on the size of <code>fish</code> in ascending order. Upon successful insertion, increase attribute <code>count</code> by 1.

<code>delete(fish_name)</code>	This procedure deletes the node from the logical linked list where its attribute <code>data</code> contains a <code>Fish</code> instance that has attribute <code>name</code> equal to <code>fish_name</code> . Upon successful deletion, decrease attribute <code>count</code> by 1.						
<code>display()</code>	This function displays all the data in sorted linked list headed by <code>start</code>. You should make use of <code>str(data)</code> to display the information of the fish instance. For example, <pre>>>> fll = SortedLinkedListForFish() >>> fll.insert(Fish("Tuna", 9)) >>> fll.insert(Fish("Tiger Shark", 40)) >>> fll.insert(Fish("Giant Trevally", 33)) >>> fll.insert(Fish("Skipjack Tuna", 73)) >>> fll.delete("Giant Trevally") >>> fll.display()</pre> <table> <tr> <td>Name:Tuna</td> <td>Size: 9</td> </tr> <tr> <td>Name:Tiger Shark</td> <td>Size: 40</td> </tr> <tr> <td>Name:Skipjack Tuna</td> <td>Size: 73</td> </tr> </table>	Name:Tuna	Size: 9	Name:Tiger Shark	Size: 40	Name:Skipjack Tuna	Size: 73
Name:Tuna	Size: 9						
Name:Tiger Shark	Size: 40						
Name:Skipjack Tuna	Size: 73						

Task 1.1

Implement the class `Fish`.

[2]

Task 1.2

Implement the class `SortedLinkedListForFish`. Use the test case provided for task 1.2 in the file *"useful_code.txt"* to test your implementation.

[17]

Task 1.3

Class SortedLinkedListForFish	
Additional Functions	
<code>pick2RandomFish()</code>	This function returns a tuple that contains two random fish instances that can be found in the sorted linked list. You must implement in such a way that all fish instances in the linked list have equal chance of being picked.
<code>preyTime()</code>	This function first uses <code>pick2RandomFish()</code> and let one of the 2 fish instances picked to consume the other. The two random fish instances picked are removed from the linked list and the new fish instance is inserted to the linked list.

Your task is to:

- Implement the two additional functions in the Class `SortedLinkedListForFish`.
- Write a function `task1_3()` to:
 - read all the fishes in "*fishes.txt*" and insert them into a sorted linked list.
 - repeatedly call `preyTime()` until there is only 1 fish instance left in the sorted linked list.
 - display the information of this last fish using `str(fish)`
- Call `task1_3()` 10 times.

[11]

2 Word Game

Name your Jupyter Notebook as

Task_2_<class>_<index number>_<name>.ipynb

For each of the Task 2.1 to 2.5, 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]:

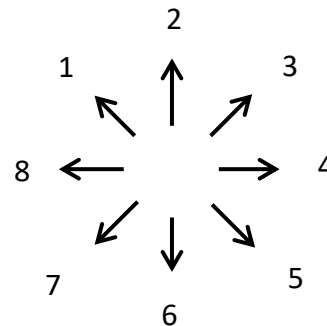
```
#Task 2.1
Program code
```

Output:

In this word game, the player is to identify English words in a 4x4 matrix by connecting adjacent tiles that each contains a letter.

For example, using the 4x4 board below, a player can identify the word “SHORT” by a **move** that is represented by a list of integers: [2, 1, 4, 1, 4, 2].

0	A	E	T	S
1	T	O	R	Y
2	I	S	H	E
3	O	M	E	A
	0	1	2	3



Explanation

The first two integers 2 and 1 indicate the starting tile with the letter S at row 2 and column 1.

From the 3rd integers onwards, they indicate the directions in which the next letter tile is to be visited. The directions are represented by integers 1 to 8. Please refer to the above diagram on the right.

As the next integer in the list is 4, this means that the tile containing the next letter is on the right. By following the directions indicated by the numbers 4, 1, 4 and 2 in the list, the letters H, O, R and T are visited in the same order.

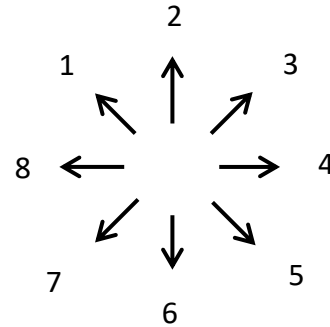
Here are a few examples of other English words in the board.

Move	Word
[2, 1, 7, 4, 4]	SOME
[2, 2, 4, 1, 5]	HERE
[2, 2, 4, 1, 1]	HERE

Task 2.1

Implement the function `valid_dir(board, curr_pos, direction)` to check whether there is an adjacent tile in the given direction from the current tile position `curr_pos`. If there is no tile in the given direction indicated by `direction`, return `False`. Otherwise, return `True`.

0	A	E	T	S
1	T	O	R	Y
2	I	S	H	E
3	O	M	E	A
	0	1	2	3



For example,

```
>>> board =
[["A", "E", "T", "S"],
 ["T", "O", "R", "Y"],
 ["I", "S", "H", "E"],
 ["O", "M", "E", "A"]]
>>> valid_dir(board, (0, 0), 1)
False
>>> valid_dir(board, (0, 0), 2)
False
>>> valid_dir(board, (0, 0), 3)
False
>>> valid_dir(board, (0, 0), 4)
True
>>> valid_dir(board, (0, 0), 5)
True
>>> valid_dir(board, (0, 0), 6)
True
>>> valid_dir(board, (0, 0), 7)
False
>>> valid_dir(board, (0, 0), 8)
False
```

The result above occurs because the tile at position `(0, 0)` has only three adjacent tiles — in directions 4, 5, and 6. Hence they return `True`.

For your own testing, you can use the code provided for **Task 2** in `“use_code.txt”` to initialize `board`. [4]

Task 2.2

Implement the function `result_word(board, move)` to return the English word identified by `move`. Use `valid_dir()` in **Task 2.1** to check if there is any invalid direction in `move`. If `move` cannot successfully identify an English word, return `None`. Otherwise return the English word in string.

For example,

```
>>> result_word(board, [2,1,7,4,4])
SOME
>>> result_word(board, [2,2,4,1,5])
HERE
>>> result_word(board, [2,2,4,1,1])
HERE
>>> result_word(board, [2,2,4,1,1,1])
None
```

Test your function using `valid_moves` and `invalid_moves` that are provided for **Task 2** in the file `"useful_code.txt"`.

[6]

Task 2.3

Each word that is identified by a move successfully is given a score. The score is calculated by adding the number that is assigned to the tile each time the tile is visited in a move.

The number assigned to each tile in the 4 x 4 board is represented by a 2D array as follows.

```
score_board = [[1,2,3,1],[2,2,3,1],[1,2,2,1],[1,4,2,1]]
```

0	1	2	3	1
1	2	2	3	1
2	1	2	2	1
3	1	4	2	1
	0	1	2	3

0	A	E	T	S
1	T	O	R	Y
2	I	S	H	E
3	O	M	E	A
	0	1	2	3

Using the score board above and the move `[2,1,4,1,4,2]`. The score for the word "SHORT" is $2 + 2 + 2 + 3 + 3 = 12$.

Implement the function `result_score(score_board, move)` to return the score of the word identified by `move`. You can assume that `move` is always a valid move.

Test your function using `score_board` and `valid_moves` that are provided for **Task 2** in the file `"useful_code.txt"`.

[4]

Task 2.4

Implement the function `word_scores(board, score_board, moves)` which returns a list of lists that contains all the words that can be identified by `moves` and their respective scores.

You can assume that all the move in `moves` are valid.

For example,

```
>>> board =
[["A", "E", "T", "S"],
 ["T", "O", "R", "Y"],
 ["I", "S", "H", "E"],
 ["O", "M", "E", "A"]]
>>> score_board =
[[1, 2, 3, 1],
 [2, 2, 3, 1],
 [1, 2, 2, 1],
 [1, 4, 2, 1]]
>>> valid_moves =
[[2, 1, 4, 1, 4, 2],
 [2, 1, 7, 4, 4],
 [2, 1, 1, 4, 4, 5],
 [0, 3, 8, 7, 4, 4],
 [1, 2, 8, 6, 5],
 [2, 0, 4]]
>>> word_scores(board, score_board, valid_moves)
[['SHORT', 12], ['SOME', 9], ['STORE', 10], ['STORY', 10],
 ['ROSE', 9], ['IS', 3]]
```

Test your function using `board`, `score_board` and `valid_moves` that are provided for **Task 2** in the file *“useful_code.txt”*.

[3]

Task 2.5

Implement the function `display_word_score_sorted(board, score_board, valid_moves)` that does the following:

- Use the function in **Task 2.4** to receive a list of words and their respective scores.
- Perform bubble sort on the list based on their scores in descending order. For those words that have the same score, sort them in alphabetical order.
- Display them using the following format.

WORD	SCORE
SHORT	12
SHEER	10
SHORE	10
STORE	10
STORY	10
MOST	9
SOME	9
STATE	9
ROT	8
SHOE	8
SHOT	8
HERE	7
TOE	7
ME	6
SHE	6
TEA	6
ATE	5
EAT	5
OR	5
AT	3
HE	3
IS	3

Test your function using `board`, `score_board` and `more_moves` that are provided for **Task 2** in the file *“useful_code.txt”*.

[8]

3 Alien Language

Task 3 Jupyter Notebook template is provided for you. Please rename the Jupyter Notebook as

Task_3_<class>_<index number>_<name>.ipynb

You are a cryptographer decoding messages sent by an alien civilization. The aliens use a special numeric encoding system where:

- Each character in their messages is first converted to its ASCII code.
- Then the ASCII code is converted to a 3 digits base-7 number (since their civilization uses base-7).

Your mission is to decode the secret messages back into human-readable text.

For example, the following alien code is received.

205166226166220203211216231216

The above alien code is made of 10 characters with each character encoded by 3 digits. The table below first divides the code into 10 parts with each part containing 3 digits. Then, it calculates its respective denary number. Finally, convert it back to ascii letter.

Code	205	166	226	166	220	203	211	216	231	216
Denary	103	97	118	97	112	101	106	111	120	111
ASCII	g	a	v	a	p	e	j	o	x	o

The decoded word is therefore gavapejoxo.

Task 3.1

Implement the function `display_alien_words(alien_codes)` which takes a list of alien code stings as input and display their corresponding decoded words in the given format.

For example,

```
>>> alien_codes_3 =
['205166226166220203211216231216',
 '224166214166220166200210',
 '200166220203226216']
>>> display_alien_words(alien_codes_3)
WORD          CODE
gavapejoxo    205166226166220203211216231216
tamapabi      224166214166220166200210
bapevo        200166220203226216
```

Test your function using `alien_codes` that is provided in the template.

[5]

Task 3.2

As a cryptographer decoding alien messages for many years, you have come to understand more alien words. These words are stored in the variable `known_alien_word_list` as a list in the template provided. You are often required to check if a decoded alien word is inside the known alien word list. Storing these words in a list makes this search operation $O(n)$ time complexity.

Implement the class `AlienKnowledgeDataBase` where its search operation can now be done in $O(\lg n)$ time complexity.

Class <code>AlienKnowledgeDataBase</code>	
Attributes	
You can decide on whatever attribute you need for this class	
Functions	
<code>__init__(known_alien_word_list)</code>	The constructor takes in the known alien word list and store the words in a data structure that allows the search operation to be done in $O(\log n)$ time complexity.
<code>search(alien_word)</code>	This function checks if <code>alien_word</code> is one of the words that can be found in <code>known_alien_word_list</code> . It returns <code>True</code> if it is found and <code>False</code> otherwise. The search operation must be in $O(\log n)$ time complexity.

Test your implementation using the `task3_2()` that is provided in the template. [4]

Task 3.3

Implement the class `AlienKnowledgeDataBase2` where its search operation can now be done in $O(1)$ time complexity.

Test your implementation using the `task3_3()` that is provided in the template. [5]

4 RV SC Election

Name your Jupyter Notebook as

Task_1_<class>_<index number>_<name>.ipynb

River Valley Hypothetical School Student Council had decided to use digital voting system for 2025 election. For 2025, the election will be for groups of 3 candidates where the student population will vote for their choice group. From the elected group, the President and 2 Vice Presidents will then be appointed.

To help manage the election process, RdeV is engaged to help with the programming of the database system. RdeV decided to use a relational database to store records of candidates' data, groups' data, and vote data. A separate table will store the candidate-group information as SC had indicated that it is possible for a candidate to be in different groups. The database will have 4 tables to store data on (1) **candidates**; (2) **groups**; (3) **candidate_group** and (4) **votes**. All fields cannot be left empty.

candidates:

- `candidate_id` – candidate identification. Example, `sc2025_01`
- `name` – name of candidate.
- `cls` – class for which the candidate belongs to. Example, `24J13`
- `gender` – gender of the candidate. Can only be `male` or `female`.
Database system will validate the value before accepting the record.

groups:

- `group_id` – group identification. Example, `1`.
- `name` – name of the group.

candidate_group:

- `candidate_id` – identification of candidate. It is possible that for the candidate to be in multiple groups
- `groups` – identification of groups. Each group will have 3 candidates.

votes:

- `voter` – student voters serial number. Example, `2025001`
- `vote` – group identification for which student voters will be casting their vote.

For each of the task 4.1 to 4.5, 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 `sc_election_2025.db` with the 4 tables given. The database is in the same directory as this Jupyter Notebook. Define the primary and foreign keys for each table and set up the necessary constraints in your codes. [6]

Task 4.2

A sample dataset was generated for the different tables for testing the database and the programs using it. The csv files `candidates.csv`, `groups.csv`, `candidate_group.csv` and `votes.csv` store the comma-separated values for each of the tables in the database corresponding to the table names.

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. [4]

Task 4.3

After the voting process, the elected group name and candidate names need to be announced from the `group_id`.

Write a Python program that will print out the group name and the candidate names from a given `group_id`. The print out will follow the format of the following sample print out with the group name on the first line and the candidate names, in no order for the subsequent lines:

```
Student Voice -
Lee Loong Loong
D Dave
Nnarnia Saladin
```

Run the program for `group_id 1`. [4]

Task 4.4

After the voting process, tabulation of the vote count for the different groups is needed to determine the winner of the election.

Write a Python program that will print out the vote count for the different groups in descending order of vote count. The print out will include the group's name and group_id in parenthesis and the vote count following the format of the sample print out:

```
Student Voice          (1) - 69
Student Right         (2) - 62
Total Student         (3) - 56
RV Together           (5) - 55
RV Steady              (4) - 54
```

[4]

Task 4.5

The election results will be implemented as a web application utilising python flask microframework.

You are tasked to write part of the web application to display the 2025 SC election results. Your web application will return a HTML page on the root route to display a table showing the vote count of the different groups by descending order of vote count followed by the details of the winning group.



2025 SC Election Results

Group	Votes
Student Voice (1)	109
Total Student (3)	105
Student Right (2)	103
RV Together (5)	97
RV Steady (4)	83

Winner of 2025 SC Election

Student Voice

1. Lee Loong Loong
2. D Dave
3. Nnamia Saladin

The webpage must look similar to the one shown above with the following for the applied CSS rules:

- 1) The logo width will be 100 px with aspect ratio unchanged
- 2) Title, sub-title and winning group will use h1, h2 and h3 respectively
- 3) The table width and the section to display details of winning group will be 500 px.

Save your program code as

Task_4_5_<class>_<index>_<name>.py

With any additional files / subfolders as needed in a folder named

Task_4_5_<class>_<index>_<name>

Run the web application and save the returned pages in the root directory as

Task_4_5_<class>_<index>_<name>.html

[7]

End of paper