

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

---

**COMPUTING**

Paper 2 (Lab-based)

**9569/02**

**15 Aug 2024**

**3 hours**

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

- example.txt
- inputA.txt
- inputB.txt
- rubik.txt
- student.csv
- cubicle.csv
- booking.csv

---

**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.

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 17 printed pages and 1 blank page.

## 1 Study Cubicle Booking RDBS

Name your Jupyter Notebook as

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

To support J2 students in their studies for A Levels, River Valley Hypothetical School is setting up study cubicles for students to book for night study. Students will need to register before being allowed to book. Each cubicle booking is for a night on a selected date for a registered student.

To help manage the booking process, RdeV is engaged to help program the booking system. RdeV decided to use a relational database to store records of students' details, cubicle details, and booking details. The database will have 3 tables to store data on (1) **student**; (2) **cubicle**; and (3) **booking**. All fields cannot be left empty.

### student:

- `id` – unique student identification. for example, `rv0001`.
- `name` – name of student.
- `contact` – contact number of student. Not unique as siblings may share contact numbers.

### cubicle:

- `cubicle_no` – unique cubicle identification. for example, `c01`.
- `maintenance` – status to indicate if a cubicle is under maintenance.  
     1 indicates that cubicle is under maintenance.  
     0 indicates that cubicle is not under maintenance.  
     database will need to check for validity of entry.  
     defaults to 0 on new cubicle entry.

### booking:

- `date` – date for which a cubicle is booked for. In *DDMMYYYY* format.  
     for example, `18112024`.  
     database will not be checking for validity of entry.
- `cubicle_no` – identification of cubicle that was booked.
- `student_id` – identification of student who made the booking.

For each of the task 1.1 to 1.4, 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

Write a Python program that uses SQL code to create database `cu_booking.db` with the 3 tables given. The database is in the same directory as this Jupyter Notebook. Define the primary and foreign keys for each table in your codes. [8]

### Task 1.2

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

Write a Python program that uses SQL code 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 1.3

The status of maintenance needs to be updated if a cubicle is damaged or is repaired.

Write a Python program that uses SQL code to update the `maintenance` status of `c01` to 0. [2]

### Task 1.4

Part of the system is to give feedback to administrators the availability of the cubicles on night of a specific date. A cubicle is available on the date only if it is not under maintenance and not book

Write a Python program that uses SQL code to print out a list of `cubicle_no` that is available on the date `'17112024'`. [4]

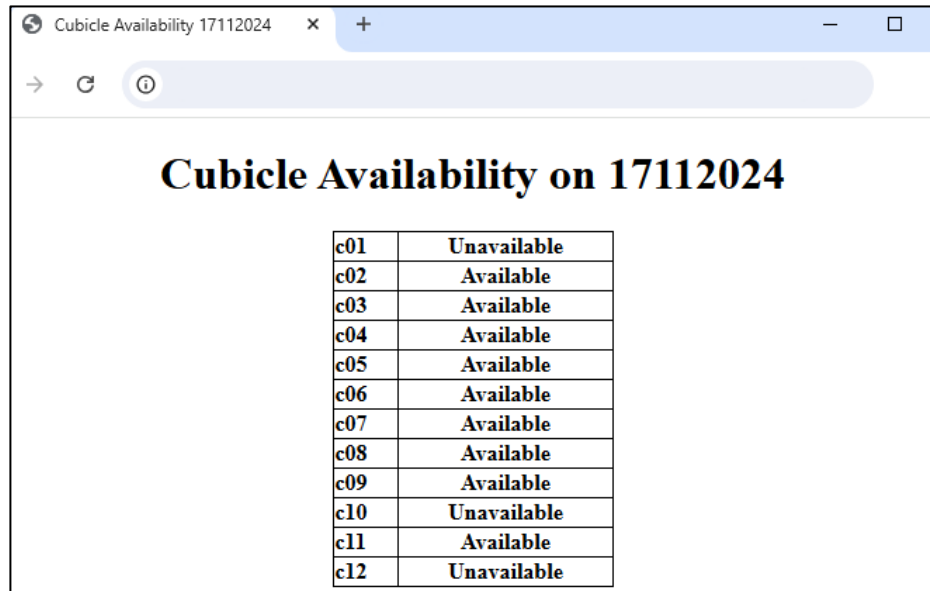
### Task 1.5

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

You are tasked to write part of the web application to display the availability of cubicles on a given date. Your part of the web application will be on an URL sub-path `/available/` from its root path. The URL sub-path will have a `date` parameter to display information on cubicle availability for that date.

Eg `.../available/17112024` will display information on cubicle availability on 17112024.

Your web application will return a HTML page to display the header and the table with the associated cubicle availability information with the URL sub-path and the URL parameter as shown:



The screenshot shows a web browser window with the title 'Cubicle Availability 17112024'. The page content includes a heading 'Cubicle Availability on 17112024' and a table with 12 rows of cubicle availability data.

Cubicle ID	Availability
c01	Unavailable
c02	Available
c03	Available
c04	Available
c05	Available
c06	Available
c07	Available
c08	Available
c09	Available
c10	Unavailable
c11	Available
c12	Unavailable

The HTML must look similar to the one shown above with the following details:

1. HTML title and webpage header (h1 size) are the same.
2. Text within the table is all bold and must be aligned as shown.
3. Table width is 200 pixels with black 1-pixel thick border and aligned to the center of the web page.

Run your web application for date 17112024.

Save your program code as

`Task_1_5_<class>_<index>_<name>.py`

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

`Task_1_5_<class>_<index>_<name>`

Run the web application and save the returned pages as

`Task_1_5_<class>_<index>_<name>_<page name>.html`

[6]





Write a function `task_2_3(char_lst)` that:

- takes a list of ascii characters, `char_lst`
- inserts each character of `char_lst` in a spiral way into a 2D list of size  $n \times n$  where  $n$  is the square root of the length of `char_lst`.
- returns the 2D list.

[6]

For example, `charset` contains the following.

[illegible]

```
>>> task 2 3(char lst)
```

[illegible]

Run the following code if you would like to visualise the image.

```
result = task_2_3(task_2_1("example.txt"))
for row in result:
    for char in row:
        print(char, end = "")
    print()
```

The following image should be displayed.

\_\_\_\_\_

( \ / )  
( . . )  
C ( " ) ( " )

\_\_\_\_\_

## Task 2.4

Write a function `task 2 4(input file, outputfile)` that:

- takes two string values `input_file` and `outputfile`
- processes the content of `input_file` using `task_2_1()` and `task_2_3()`
- writes the result from `task_2_3()` in `outputfile` with each row of the 2D array occupies a line and each character in the line adjacent to each other. [3]

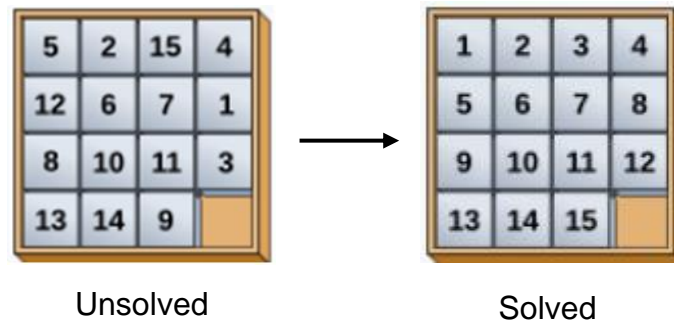
Test your function using the following call:

- `task_2_4("example.txt", "output.txt")`
- `task_2_4("inputA.txt", "outputA.txt")`
- `task_2_4("inputB.txt", "outputB.txt")`

[1]

### 3 Moving Hole Puzzle

A moving hole puzzle, also known as a sliding puzzle, involves moving pieces around a board to achieve a specific configuration. The puzzle typically has one empty space (the "hole") that allows the pieces to be slid into different positions.



#### Task 3.1

Write a function `create_puzzle(n)` that:

- takes an integer value `n`
- creates a 2D list of `n x n` dimension
- inserts the numbers from 0 to  $n^2-1$  into the 2D list at **random** positions
- identifies the position of number 0 which represents the missing tile of the puzzle
- returns the 2D list, the row and column index of the number 0 (the missing tile) in a list. [4]

For example,

```
>>> puzzle, r, c = create_puzzle(4)
>>> print(puzzle)
[[14, 10, 9, 3], [2, 4, 5, 12], [13, 0, 15, 8], [7, 1, 6, 11]]
>>> print(r, c)
2, 1
>>> puzzle[r][c]
0
```

Test your function using the following:

[1]

```
for i in range(3, 6):
    puzzle, r, c = create_puzzle(i)
    print("Coordinate of space: ({} , {})".format(r, c))
    print("Puzzle: " + str(puzzle))
    print()
```



### Task 3.2

Write a function `display_puzzle(puzzle)` that:

- takes a 2D list which represents the puzzle, `puzzle`
- displays the puzzle with the row and col index as shown in the example below for easy reference. Take note that the `#` character is used to represent the missing tile. [3]

For example,

```
>>> puzzle = [[14, 10, 9, 3], [2, 4, 5, 12], [13, 0, 15, 8],
[7, 1, 6, 11]]
>>> display_puzzle(puzzle)
```

```
      0    1    2    3
0     14   10    9    3
1      2    4    5   12
2     13    #   15    8
3      7    1    6   11
```

### Task 3.3

The available moves that the player can make depends on the position of the missing tile in the puzzle. There are 4 moves altogether.

- 'U' means move the tile above the missing tile down
- 'R' means move the tile on the right of the missing tile to the left
- 'D' means move the tile below the missing tile up
- 'L' means move the tile on the left of the missing tile to the right

[14][10][ 9][ 3]		[14][10][ 9][ 3]
[ 2][ <b>4</b> ][ 5][12]	U move	[ 2][  ][ 5][12]
[13][  ][15][ 8]	----->	[13][ <b>4</b> ][15][ 8]
[ 7][ 1][ 6][11]		[ 7][ 1][ 6][11]

[14][10][ 9][ 3]		[14][10][ 9][ 3]
[ 2][ 4][ 5][12]	R move	[ 2][ 4][ 5][12]
[13][  ][ <b>15</b> ][ 8]	----->	[13][ <b>15</b> ][  ][ 8]
[ 7][ 1][ 6][11]		[ 7][ 1][ 6][11]

[14][10][ 9][ 3]		[14][10][ 9][ 3]
[ 2][ 4][ 5][12]	D move	[ 2][ 4][ 5][12]
[13][ ][15][ 8]	----->	[13][ <b>1</b> ][15][ 8]
[ 7][ <b>1</b> ][ 6][11]		[ 7][ ][ 6][11]

[14][10][ 9][ 3]		[14][10][ 9][ 3]
[ 2][ 4][ 5][12]	L move	[ 2][ 4][ 5][12]
<b>[13]</b> [ ][15][ 8]	----->	[ ][ <b>13</b> ][15][ 8]
[ 7][ 1][ 6][11]		[ 7][ 1][ 6][11]

*Take note that not all positions of the missing tile will have all 4 moves mentioned above. If the missing tile is at the edge of the puzzle, it will only have 3 available moves. If it is at the corner, it will only have 2.*

Write a function `find_available_move(puzzle, r, c)` that:

- takes a 2D list which represent the puzzle, `puzzle`
- takes two integer values `r` and `c` which represent the row and column index of the missing tile.
- returns a list of moves that is available given the missing tile is at `puzzle[r][c]`.
- You can assume the `r` and `c` are always valid. [3]

For example,

```
>>> puzzle = [[14, 10, 9, 3], [2, 4, 5, 12], [13, 0, 15, 8],
[7, 1, 6, 11]]
>>> find_available_move(puzzle, 2, 1)
['U', 'R', 'D', 'L']
```

### Task 3.4

Write a function `make_move(move, puzzle, r, c)` that:

- takes a string value `move` which has either the values of 'U', 'R', 'D' or 'L'
- takes a 2D list which represent the puzzle, `puzzle`
- takes two integer values `r` and `c` which represent the row and column index of the missing tile.
- executes the move in the `puzzle` by swapping the relevant tiles
- you can assume that the move is always valid. [4]

For example,

```
>>> puzzle = [[14, 10, 9, 3], [2, 4, 5, 12], [13, 0, 15, 8],
[7, 1, 6, 11]]
>>> make_move('D', puzzle, 2, 1)
>>> print(puzzle)
[[14, 10, 9, 3], [2, 4, 5, 12], [13, 1, 15, 8], [7, 0, 6, 11]]
```

### Task 3.5

Write a function `win(puzzle)` that:

- takes a 2D list which represent the puzzle, `puzzle`
- returns `True` if `puzzle` is solved and `False` otherwise.
- take note that square puzzle size  $n \times n$  can vary.

[3]

### Task 3.6

Write a function `game_menu()` that:

- starts the game with a default puzzle using `[[2, 3, 5], [1, 0, 4], [7, 8, 6]]`
- starts the timer when the game is launched
- displays the puzzle and the option menu at the start of the game and after every move made by the player
- implement the following menu options:
  - 1) New game
    - prompts user to choose a puzzle size from 3 to 5 and generates a new puzzle when option 1 is chosen
    - restarts the timer
  - 2) Make one move
    - prompts user to make one valid move by displaying all valid moves based on the position of the missing tile when option 2 is chosen
    - executes the move
  - 3) Make many moves
    - prompts user to make a series of moves (e.g. RULLDRRD) when option 3 is chosen
    - executes the moves
    - reverts the puzzle to the original state if the series of moves contains invalid moves
  - 4) Quit
    - displays "You took <x>s to give up." where x is the time taken for the player to give up
    - quits the game
- checks for winning condition after every move
- displays the total time taken to win the game in second when the game is won
- all data validation must be present

[10]

A sample display of the game is as follow:

Sample run A – demonstrate option 2) and 3)

```

      0    1    2

0     2    3    5
1     1    #    4
2     7    8    6
1) New Game.
2) Make one Move.
3) Make many moves.
4) Quit.
Your option: 2
Please choose your move ['U', 'R', 'D', 'L']: R

      0    1    2

0     2    3    5
1     1    4    #
2     7    8    6
1) New Game.
2) Make one Move.
3) Make many moves.
4) Quit.
Your option: 2
Please choose your move ['U', 'D', 'L']: U

      0    1    2

0     2    3    #
1     1    4    5
2     7    8    6
1) New Game.
2) Make one Move.
3) Make many moves.
4) Quit.
Your option: 3
Your series of moves (no spaces): LLDRRD
You took 15s to win.
```

Sample run B – demonstrate option 1) and 4)

	0	1	2	
0	2	3	5	
1	1	#	4	
2	7	8	6	

1) New Game.  
 2) Make one Move.  
 3) Make many moves.  
 4) Quit.  
 Your option: 1  
 Key in size of puzzle: 5

	0	1	2	3	4
0	2	8	20	15	16
1	14	17	21	1	18
2	3	12	23	7	11
3	13	9	6	10	22
4	4	24	#	19	5

1) New Game.  
 2) Make one Move.  
 3) Make many moves.  
 4) Quit.  
 Your option: 4  
 Game over.  
 You took 2s to give up.

## 4 Rubik's Cube Challenge

A programmer would like to use an OOP approach to store information regarding the contestants of a Rubik's cube challenge.

For class `Contestant`, the data that will be stored include:

- `name` (string)
- `time` (float)

For class `beginner`, the additional data stored include:

- `practice_hours` (integer)

For class `professional`, the additional data stored include:

- `competitions_won` (integer)

In class `Contestant`, the following class methods are to be implemented.

Method	Description
<code>Contestant(name, time)</code>	Constructor of the class <code>Contestant</code>
<code>get_time()</code>	Getter for attribute <code>time</code>
<code>set_time(new_time)</code>	Setter for attribute <code>time</code>
<code>__str__()</code>	<p>The string representation of a <code>Contestant</code> instance.</p> <p>The string returned starts with a "C-", followed by the first 8 characters of the name in uppercase excluding spaces. If the number of characters in the name is fewer than 8, fill up the remaining spaces with "_".</p> <p>Finally, append the <code>time</code> with at most 2 decimal places with a bracket and a "s".</p> <p>For example,</p> <pre>&gt;&gt;&gt; print(str(Contestant("Kan Min Coh", 35.32))) C-KANMINCO(35.32s)  &gt;&gt;&gt; print(str(Contestant("Man Kin", 15.1))) C-MANKIN__(15.1s)</pre>

In class `Beginner`, the following class methods are to be implemented.

Method	Description
<code>Beginner(name, time, practice_hours)</code>	Constructor of the class <code>Beginner</code>
<code>__str__()</code>	<p>This is a polymorphed function. The string representation of a <code>Beginner</code> instance.</p> <p>Instead of starting with "C-", the string returned starts with a "B-".</p> <p>For example,</p> <pre>&gt;&gt;&gt; print(str(Beginner("Kan Min Coh", 35.32))) B-KANMINCO(35.32s)  &gt;&gt;&gt; print(str(Beginner("Man Kin", 15.1))) B-MANKIN__(15.1s)</pre>

In class `Professional`, the following class methods are to be implemented.

Method	Description
<code>Professional(name, time, competitions_won)</code>	Constructor of the class <code>Professional</code>
<code>__str__()</code>	<p>This is a polymorphed function. The string representation of a <code>Professional</code> instance.</p> <p>Instead of starting with "C-", the string returned starts with a "P-".</p> <p>For example,</p> <pre>&gt;&gt;&gt; print(str(Professional("Kan Min Coh", 35.32))) P-KANMINCO(35.32s)  &gt;&gt;&gt; print(str(Professional("Man Kin", 15.1))) P-MANKIN__(15.1s)</pre>

**Task 4.1**

Implement the 3 classes as mentioned above.

[6]

Test your function using the following call:

- `print(str(Contestant("ChanChan Chan", 35.01)))`
- `print(str(Beginner("Li uLi uLi u", 55.54, 300)))`
- `print(str(Professional("OhOhOh", 6.34, 60)))`

[1]

**Task 4.2**

Write a function `readfile(filename)` that:

- takes a string value, `filename` which contains data stored in either one of the following formats:
  - `Beginner, <name>, <time>, <practice hour>`
  - `Professional, <name>, <time>, <competition won>`
- create an appropriate instance for each line in the file
- add all instances created in a list and return it

[3]

**Task 4.3**

Implement the class `Node` and class `BST` to store the instances created in Task 4.2. The description of the two classes are as follows.

In class `Node`, the following attributes are to be implemented.

[1]

Attribute	Description
<code>player</code>	This is to store the <code>Beginner</code> or <code>Professional</code> instance.
<code>left</code>	This points to the left sub-tree which is another <code>Node</code> instance
<code>right</code>	This points to the right sub-tree which is another <code>Node</code> instance
Method	Description
<code>Node(new_player)</code>	Constructor of the class <code>Node</code> where <code>new_player</code> is either a <code>Beginner</code> or <code>Professional</code> instance. <code>new_player</code> is assigned as the attribute <code>player</code> in the <code>Node</code> instance.



In class `BST`, the following attribute and methods are to be implemented.

Attribute	Description
<code>root</code>	This is to store the root of the BST which is a <code>Node</code> instance.
Method	Description
<code>BST()</code>	Constructor of the class <code>BST</code>
<code>Insert(player)</code>	Insert <code>player</code> which is either a <code>Beginner</code> or <code>Professional</code> instance into the BST based on its <code>time</code> attribute. Lower time should be inserted to the left side of the tree.
<code>inorder()</code>	Display the string representation of the attribute <code>player</code> of all the nodes in the BST in order.
<code>count_professtional()</code>	This recursive function counts and returns the total number of professionals in the BST.
<code>find_best_beginner()</code>	This recursive function finds the beginner with the lowest time in the BST. The <code>Beginner</code> instance should be returned.

[12]

Test your function using the following:

```
bst = BST()
players = readfile("rubik.txt")
for player in players:
    bst.insert(player)
bst.inorder()
print(bst.count_professional())
print(bst.find_best_beginner())
```

[2]

End of paper