PROG1926 – Programming Concepts 1

Assignment 1 – Concert Reservation System

You will create a program that functions as a concert reservation system. When the program starts, it will load a file containing a previously saved concert hall layout. You will demonstrate your ability to create a looping menu system, display and manipulate data using a 2D list, perform file input and output operations, and handle exceptions
appropriately.

Overall Requirements

- Name your file: `<firstname>_<lastname>_a1.py`. For example, `john_doe_a1.py`, for a student with the name `John Doe`.
- Ensure you are using proper programming conventions. For example, proper variable naming, indenting, header and body comments, function comments etc.
- Use ONLY code constructs demonstrated in class. If you create code different from what was shown in class, then it needs to be fully commented to show your understanding of how it works. Not doing so will affect the overall mark.
- Only use `return` if you are returning a value from a method, (not to exit a method), and only once in the method.
- This is a group assignment; every group must be composed of 2 students. Requesting to create groups more than 2 members will be denied.
- Each member of a group should work on their own copy, feel free to discuss/share code with your group member.
- Do not use any code created by another group in whole or in part.
- When completed, choose one member's work and submit. One submission will be treated as the group submission.

Exception Handling

- Proper error handling and appropriate error messages should be displayed. The catch blocks should handle specific exceptions, not just general exceptions.
- All inputs must be valid to be accepted, as per the assignment instructions below.

Program Requirements

- This program must be created using a 2D list (a List of Lists). Each `row` of the 2D list will represent all the seats (columns) available in one row of the concert hall.
- To start, you need to create an empty list, for example: `concert_hall = []`

Feature 1: Load reservation data from file

- When the program starts it will ask the user to input the name of the file containing a saved layout.
  - This file is given be given to you along with this assignment spec, called `x.txt`.
- Using loops, read this information and generate rows.
- Once one row is generated add it to the `concert_hall`. In this way, the 2D list `concert_hall` will be constructed by a number of rows, containing the data given through the text file.

- Show a message indicating the operation is successful.
- Handle exceptions if load operation fails, and show appropriate message.
- Display the layout. (see example below)

> ⚠️ If the user provides invalid file name, show appropriate error message and exit the program.

> ⚠️ Your instructor may use a different file to test your program.

Save file structure

- The file contains every information in a separate line.

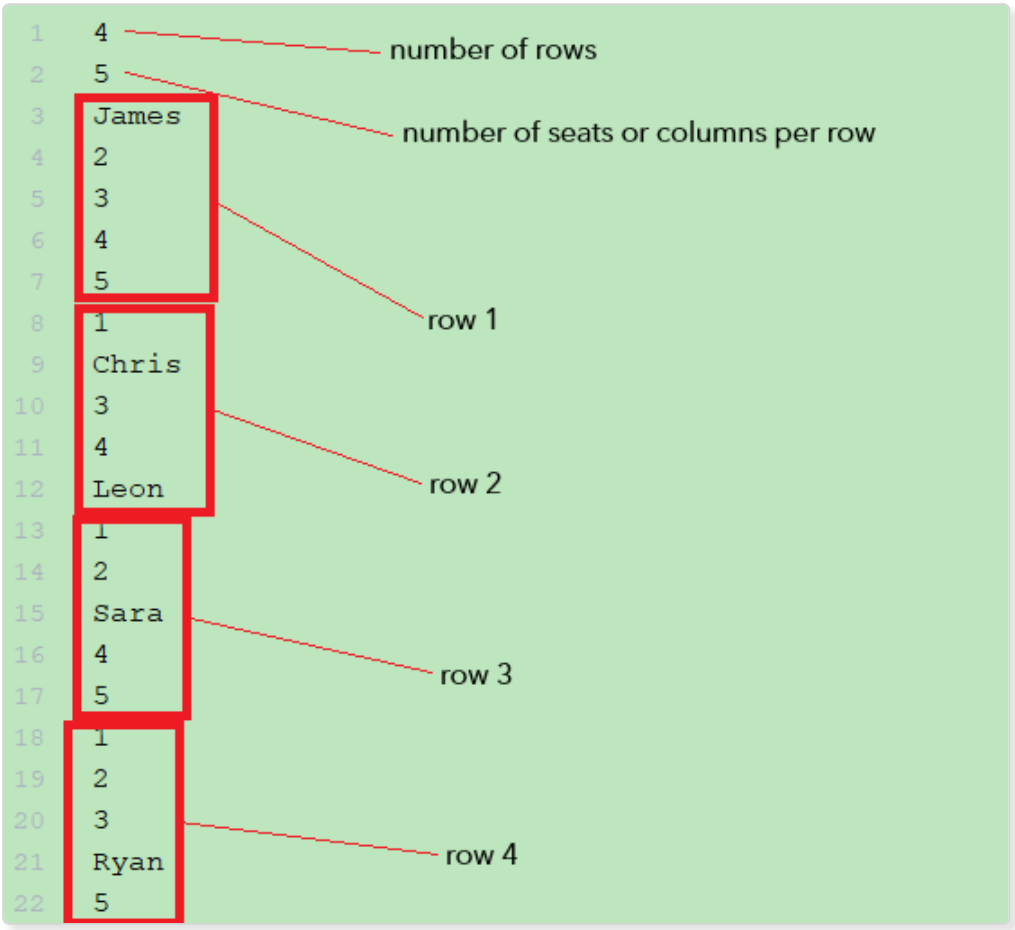

Figure 1: Data organization and file structure

- After generating the 2D list, it will look like:

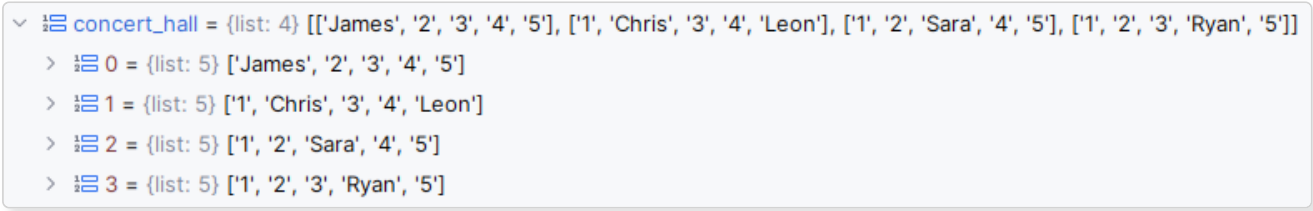

Figure 2: Data structure of `concert_hall` 2D list

- When displaying the seats on screen it may look something like this:

```
Row: 1:  James  2    3    4    5
Row: 2:  1   Chris   3    4    Leon
Row: 3:  1   2    Sara     4    5
Row: 4:  1   2    3    Ryan     5
```

Figure 3: Snapshot of a displayed seating chart

> ⚠️ Note: List elements are `0-index` based. However, this program assumes the user will start counting with `1` when specifying row and column.

> ⚠️ `James`'s seat is at `Row 1, Column 1`, however, in the list, it is stored at `0th row` and `0th column`.

> 💡 If the user inputs `r` as row and `c` as column, the seat's row index and column index will be `r-1` and `c-1`, respectively.

> 💡 You may create your 2D list with `1` extra row and `1` extra column. This way you can ignore the `0th` row and `0th` column, and track information starting at index `1` for both the rows and columns. For example, if you have a concert hall with `3` rows and `4` seats per row the 2D list will look as follows:

		Columns				
		0	**1**	**2**	**3**	**4**
Rows	**0**					
	1		1	2	3	4
	2		1	2	3	4
	3		1	2	3	4

Figure 4: Example of organizing data with empty row and column

Pseudocode to load the file

```
create an empty list - concert_hall
open file
read a line and save data as n_rows
read a line and save data as n_cols

loop n_rows times
    create an empty list for row - my_row
    loop n_cols times
            read a line and append to my_row
        append my_row to concert_hallclose file
```

- Based on the given file, after reading the first `2` lines, it will loop `4` times (`n_rows = 4`) and for each time, it will read `5` lines (`n_cols = 5`). `2 + (5 + 5 + 5 + 5) = 22` lines will be read and

`concert_hall` will be populated with data.

- If the file had contained a layout of `6` rows and `8` columns, there would have been `2 + 6 * 8 = 50` lines in the file.

Feature 2: Main Menu

- This menu will repeat until the user decides to exit.
- After completing each feature mentioned below, show this menu.
- All menu options should be created in their own separate functions, the logic of which is explained below.

```
Main Menu
------------------------------
1. Add New Reservation
2. Edit Existing Reservation
3. Cancel Reservation
4. Display Seating Chart
5. Save Reservations to a file
6. Exit
------------------------------
```

Feature 3: Add New Reservation

- Display a chart of the seats in the concert hall.
- Request a row and column number to reserve
  - If either or both row and column are invalid (non-numeric, numeric but no-existent row/column)
  - Show meaningful error message
  - Keep repeating asking for row and column until both information is valid.
- Once, valid row, column received, if the seat is available
  - Ask for the customer's name
    - If the user inputs an empty name
    - Show meaningful error message.
    - Keep repeating asking for name until a valid name (non-empty) received.
  - Once a valid name is received, add the name to the seating chart
  - Display a message to indicate seat has been successfully reserved
- If seat is not available (already reserved)
  - Display a message to indicate this.

Feature 4: Edit Existing Reservation

- Display a chart of the seats in the concert hall.
- Request the row and column number
- If either or both row and column are invalid (non-numeric, numeric but no-existent row/column)
  - Show meaningful error message
  - Keep repeating asking for row and column until both information is valid.

- Once, valid row, column received, if the seat contains a name
  - Ask the user for a new name
  - If the user inputs an empty name
    - Show meaningful error message
    - Keep repeating asking for name until a valid name (non-empty) received.
  - Once a valid name is received, replace the old name with the new name.
  - Display a message to indicate the change was made
  - If the seat is NOT already reserved
    - Display a message to indicate this.

Feature 5: Cancel Existing Reservation

- Display a chart of the seats in the concert hall.
- Ask the user for the row and column number they would like to cancel.
- If either or both row and column are invalid (non-numeric, numeric but no-existent row/column)
  - Show meaningful error message.
  - Keep repeating asking for row and column until both information is valid.
- Once valid row and column found, if no reservation exists at that seat
  - Display message to indicate this.
- If the seat does contain a reservation
  - Display `Confirm reservation cancellation. (y/n)`
  - If the user enters `y` or `Y`
    - Remove the name and make the seat available again by replacing the content with the column number (`column index +1`).
  - If the user enters `n`/`N` or anything other than `y`/`Y`
    - Return to the main menu

Feature 6: Save reservation to a file

- Ask the user to input a file name to save the current state of the reservations.
- If the same name is given, overwrite it.
- If a different name is given create it.
- Follow the same structure that was used to read the reservation layout file to save.
- Show a message indicating the operation is successful.
- Handle exceptions if save operation fails, and show appropriate message.

TIPS

> 💡 Don't forget to test save file generated as `feature 6`; exit the program, re-run and input this file name and see if it displays the layout properly.

> 💡 Consider making a function that will repeat asking until a valid row, column is received.

> 💡 Consider making a function that will repeat asking until a valid name is received.

> 💡 Remember to use proper data types while comparing seat numbers.

> 💡 Consider storing seat numbers for empty seats as string; e.g., instead of storing `3` store `'3'`

> 💡 If confused clarify with your faculty.

Sample Output

Please check the video `Assignment1.mp4` to get an idea about the program flow.