The **goal of this lab** is to practice writing **code** using Exceptions, Bounded Queues and Circular Queues.

Q1) Implement the **Bounded Queue** and **Circular Queue ADT** as seen in class. The python implementations of these data structures are given **in the lecture slides**. Make sure you **raise exceptions** when:
– peek()/dequeue() if the queue is **empty**.
– enqueue() if the queue is **full** .

After completing the implantations, compare the runtime of **dequeue()** methods in **Bounded** Queue and **Circular** Queue. In **bounded queue**, once you dequeue an item, the remaining items in the list will be shifted left. Therefore, the dequeue in Bounded Queue is O(n). However, in **circular queue**, you just change the start pointer (index) when dequeue. Therefore, the dequeue in a Circular Queue is O(1). Thus, in theory, dequeue in **Circular Queue** should be much faster than the dequeue in a **Bounded Queue**. Write your **code** to test whether this hypothesis is **true** or **false**.

Here are the steps for this experiment:
1. Create a Circular Queue **object** and enqueue 100000 items in it.
2. Create a Bounded Queue **object** and enqueue 100000 items in it
3. Compute the time required to dequeue all items from the Circular Queue.
4. Compute the time required to dequeue all items from the Bounded Queue.

Here is how to use the **time module**:

```
import time
start = time.time()
# The statement(s) that you want to test
end = time.time()
time_interval = end - start
```

5. Finally, print the **dequeue**() runtime for **each queue** as shown in the sample output.

Please note: We need to enqueue many items in the Circular and Bounded Queues in order to get a reasonable run time for both queues. Moreover, keep in mind that time can be **different** from computer to another and the **time module** gives you time in second.

Here is a sample output:

```
For Bounded Queue, the total runtime of dequeing 100000 is:
1.8420054912567139 Seconds.
For Circular Queue, the total runtime of dequeing 100000 is:
0.04388284683227539 Seconds.
```

Q2) Write a program to simulate **two waiting queues at a grocery store**. One queue **line** is for **normal customers** and the other **line** is for **VIP customers**. Normal customers can only wait in the **Normal customers' queue**, and VIP customers can only wait in the **VIP customers' queue**. Your program should **start by asking** the user if they want to **add** a customer to the queue, **serve** the next customer in the queue, or **exit**.

If the user chooses to **add** a customer, the program asks the user to enter the name of the customer followed by **whether or not** the customer is a **VIP**. If the customer is a **VIP** then the customer's name is added to the **VIP queue** else the customer's name is added to the **Normal Queue**.

After EACH **Add** operation, the program prints the **customer's name** and the queue (**Normal** or **VIP**) the customer has been **added to**. The program also prints both queues after EACH **Add** operation.

After the add operation is completed, the user is asked to enter **Add**, **Serve** or **Exit** again. Your program will **continue to prompt** the user repeatedly with these options and perform the task associated to these operations until the user enters **Exit**. Here is a **partial** sample output of the program that shows what is displayed after the **add** operation

```
Add, Serve, or Exit: Add
Enter the name of the person to add: Fred
Is the customer VIP? True
Add Fred to VIP line
Line Normal customers queue: ]]
VIP customers queue: ]Fred]

Add, Serve, or Exit: Add
Enter the name of the person to add: Bob
Is the customer VIP? False
Add Bob to the normal line.
Normal customers queue: ]Bob]
VIP customers queue: ]Fred]
```

If the user chooses to **serve** a customer, the program will check the **VIP** Queue **first**. If the **VIP Queue** is not empty, a customer from **VIP queue** will be served first. If the **VIP** Queue is **empty**, a customer from the **Normal Queue** is served. After the **serve** operation, the program will **print** the name of the customer who has been served and the content of each queue. Here is a **partial** output from the program that shows what is displayed after the **serve** operation:

```
Add, Serve, or Exit: Serve
Fred has been served
Normal customers queue: ]Bob]
VIP customers queue: ]]
```

If the user wants to **add** a customer to a queue (**Normal or VIP**) that is **full**, the program should print one of the following **error messages**:

`Error: Normal customers queue is full`
or
`Error: VIP customers queue is full`

The program, however, should continue to ask the user if they wish to **Add**, **Serve** or **Exit**.
If both **Normal** and **VIP** Queues are empty and if the user requests a **serve** from the queue, then the program should print the following error message:
`Error: Queues are empty`
However, in this case also the program continues to ask the user if they wish to **Add**, **Serve** or **Exit**. If the user enters the word **Exit**, the program should end by printing the word **Quitting**.

Please note the following:

- You may restrict the **capacity** of each queue to **3** for testing simplicity.
- `You must use the code of` **`Bounded Queue`** and **`Circular Queue`** that is given to you **in the lecture slides** for this exercise. Once your program works with **`Bounded Queue`**, simply substitute the class with the **`Circular Queue`** to verify it also works with the new implementation of the ADT.

Here is the **complete sample output** from the program:

```
Add, Serve, or Exit: Add
Enter the name of the person to add: Fred
Is the customer VIP? True
Add Fred to VIP line
Normal customers queue: ]]
VIP customers queue: ]Fred]

Add, Serve, or Exit: Add
Enter the name of the person to add: Barney
Is the customer VIP? False
Add Barney to the normal line.
Normal customers queue: ]Barney]
VIP customers queue: ]Fred]

Add, Serve, or Exit: Add
Enter the name of the person to add: Wilma
Is the customer VIP? False
Add Wilma to the normal line.
Normal customers queue: ]Barney,Wilma]
VIP customers queue: ]Fred]

Add, Serve, or Exit: Add
Enter the name of the person to add: Pebbles
Is the customer VIP? False
```

```
Add Pebbles to the normal line.
Normal customers queue: ]Barney,Wilma,Pebbles]
VIP customers queue: ]Fred]
Add, Serve, or Exit: Add
Enter the name of the person to add: Flint
Is the customer VIP? False
Error: Normal customers queue is full
Normal customers queue: ]Barney,Wilma,Pebbles]
VIP customers queue: ]Fred]

Add, Serve, or Exit: Serve
Fred has been served
Normal customers queue: ]Barney,Wilma,Pebbles]
VIP customers queue: ]]

Add, Serve, or Exit: Serve
Barney has been served
Normal customers queue: ]Wilma,Pebbles]
VIP customers queue:]]

Add, Serve, or Exit: Serve
Wilma has been served
Normal customers queue: ]Pebbles]
VIP customers queue: ]]

Add, Serve, or Exit: Serve
Pebbles has been served
Normal customers queue: ]]
VIP customers queue: ]]

Add, Serve, or Exit: Serve
Error: Queues are empty
Normal customers queue: ]]
VIP customers queue: ]]

Add, Serve, or Exit: Exit
Quitting
```