Queue Extended

Sure! Here are a couple of homework questions based on the queue data structure tutorial:

Homework Questions

Question 1: Implementing a Circular Queue

A circular queue is a linear data structure that follows the First In First Out (FIFO) principle but connects the end of the queue back to the front, forming a circle. Implement a circular queue in Python with the following methods:

- `enqueue(item)`: Adds an item to the queue. If the queue is full, display a message indicating that the queue is full.
- `dequeue()`: Removes an item from the queue. If the queue is empty, display a message indicating that the queue is empty.
- `peek()`: Returns the front item without removing it. If the queue is empty, display a message indicating that the queue is empty.
- `is_empty()`: Returns `True` if the queue is empty, otherwise returns `False`.
- `is_full()`: Returns `True` if the queue is full, otherwise returns `False`.

Here is the template for the circular queue class. Complete the implementation.

```
class CircularQueue:
    def __init__(self, size):
        self.queue = [None] * size
        self.max_size = size
        self.front = -1
        self.rear = -1

    def enqueue(self, item):
        # Implement the enqueue method
        pass
```

```
    def dequeue(self):
        # Implement the dequeue method
        pass

    def peek(self):
        # Implement the peek method
        pass

    def is_empty(self):
        # Implement the is_empty method
        pass

    def is_full(self):
        # Implement the is_full method
        pass

# Example usage
cq = CircularQueue(3)
cq.enqueue(1)
cq.enqueue(2)
cq.enqueue(3)
print(cq.dequeue())  # Output: 1
cq.enqueue(4)
print(cq.peek())     # Output: 2
```

Question 2: Simulating a Print Queue

Simulate a print queue where documents are added to the queue based on their priority. Each document has a unique ID, name, and priority level. Implement the print queue using the `PriorityQueue` class provided in the tutorial. Add the following functionalities:

- `add_document(doc_id, name, priority)`: Adds a new document to the print queue.
- `print_document()`: Removes and prints the document with the highest priority.

- `view_next_document()`: Displays the next document to be printed without removing it.
- `view_all_documents()`: Displays all documents currently in the queue.

Here is the template for the print queue class. Complete the implementation.

```
import heapq

class PrintQueue:
    def __init__(self):
        self.queue = []
        self.index = 0

    def add_document(self, doc_id, name, priority):
        # Implement the add_document method
        pass

    def print_document(self):
        # Implement the print_document method
        pass

    def view_next_document(self):
        # Implement the view_next_document method
        pass

    def view_all_documents(self):
        # Implement the view_all_documents method
        pass

# Example usage
pq = PrintQueue()
pq.add_document("DOC1", "Document 1", 1)
pq.add_document("DOC2", "Document 2", 3)
pq.add_document("DOC3", "Document 3", 2)
pq.print_document()  # Output: Printing Document 2
pq.view_next_document()  # Output: Next document is Document
```

```
3
pq.view_all_documents()
```

Expected Output for `view_all_documents()`:

```
Document ID: DOC3, Name: Document 3, Priority: 2
Document ID: DOC1, Name: Document 1, Priority: 1
```

Submission

- Submit the completed implementations for both `CircularQueue` and `PrintQueue` classes.
- Provide example usage and test cases demonstrating that all methods work as expected.

These questions will help students deepen their understanding of queue data structures and their practical applications.