#### Queue Implementation Using Linked List (queue\_linkedlist\_implementation.py)

class Node:

def \_\_init\_\_(self, data):

self.data = data

self.next = None

class Queue:

def \_\_init\_\_(self):

self.front = None

self.rear = None

def enqueue(self, data):

new\_node = Node(data)

if self.rear is None:

self.front = self.rear = new\_node

return

self.rear.next = new\_node

self.rear = new\_node

def dequeue(self):

if self.is\_empty():

print("Queue underflow!")

return None

temp = self.front

self.front = self.front.next

if self.front is None:

self.rear = None

temp.next = None

return temp.data

def peek(self):

if self.is\_empty():

print("Queue is empty!")

return None

return self.front.data

def is\_empty(self):

return self.front is None

def display(self):

if self.is\_empty():

print("Queue is empty!")

return

current = self.front

while current:

print(current.data, end=" -> ")

current = current.next

print("None")

#### Queue Implementation Using Array (queue\_array\_implementation.py)

class Queue:

def \_\_init\_\_(self, capacity):

self.capacity = capacity

self.queue = []

self.front = 0

self.rear = 0

def enqueue(self, data):

if self.is\_full():

print("Queue overflow!")

return

self.queue.append(data)

self.rear += 1

def dequeue(self):

if self.is\_empty():

print("Queue underflow!")

return None

dequeued\_data = self.queue[self.front]

self.front += 1

return dequeued\_data

def peek(self):

if self.is\_empty():

print("Queue is empty!")

return None

return self.queue[self.front]

def is\_empty(self):

return self.front == self.rear

def is\_full(self):

return self.rear == self.capacity

def display(self):

if self.is\_empty():

print("Queue is empty!")

return

print("Queue elements:", self.queue[self.front:self.rear])

#### Problem 1 for Queue (queue\_problem1.py): Basic Queue Operations

from queue\_linkedlist\_implementation import Queue

if \_\_name\_\_ == "\_\_main\_\_":

queue = Queue()

# Enqueue some elements

queue.enqueue(10)

queue.enqueue(20)

queue.enqueue(30)

print("Queue after enqueues:")

queue.display()

dequeued\_element = queue.dequeue()

print(f"Dequeued element: {dequeued\_element}")

print("Queue after dequeue:")

queue.display()

front\_element = queue.peek()

print(f"Front element: {front\_element}")

#### Example Problem 2: Queue using Two Stacks (queue\_problem2.py)

class QueueUsingTwoStacks:

def \_\_init\_\_(self):

self.stack1 = []

self.stack2 = []

def enqueue(self, data):

self.stack1.append(data)

def dequeue(self):

if not self.stack2:

if not self.stack1:

print("Queue underflow!")

return None

while self.stack1:

self.stack2.append(self.stack1.pop())

return self.stack2.pop() if self.stack2 else None

def peek(self):

if not self.stack2:

if not self.stack1:

print("Queue is empty!")

return None

while self.stack1:

self.stack2.append(self.stack1.pop())

return self.stack2[-1] if self.stack2 else None

def display(self):

if not self.stack2:

if not self.stack1:

print("Queue is empty!")

return

while self.stack1:

self.stack2.append(self.stack1.pop())

print("Queue elements:", self.stack2)

if \_\_name\_\_ == "\_\_main\_\_":

queue = QueueUsingTwoStacks()

queue.enqueue(10)

queue.enqueue(20)

queue.enqueue(30)

print("Queue using two stacks:")

queue.display()

dequeued\_element = queue.dequeue()

print(f"Dequeued element: {dequeued\_element}")

print("Queue after dequeue:")

queue.display()

front\_element = queue.peek()

print(f"Front element: {front\_element}")