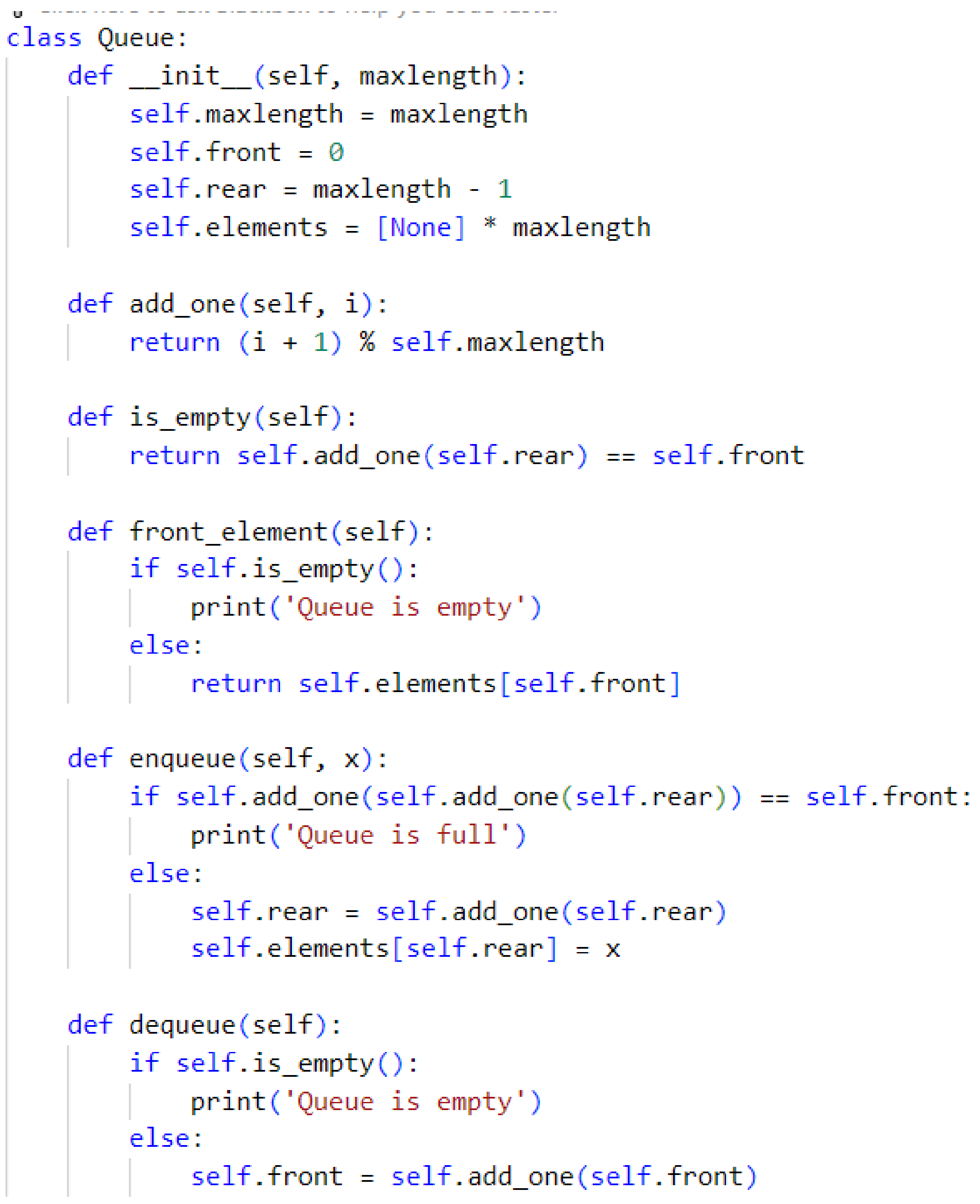
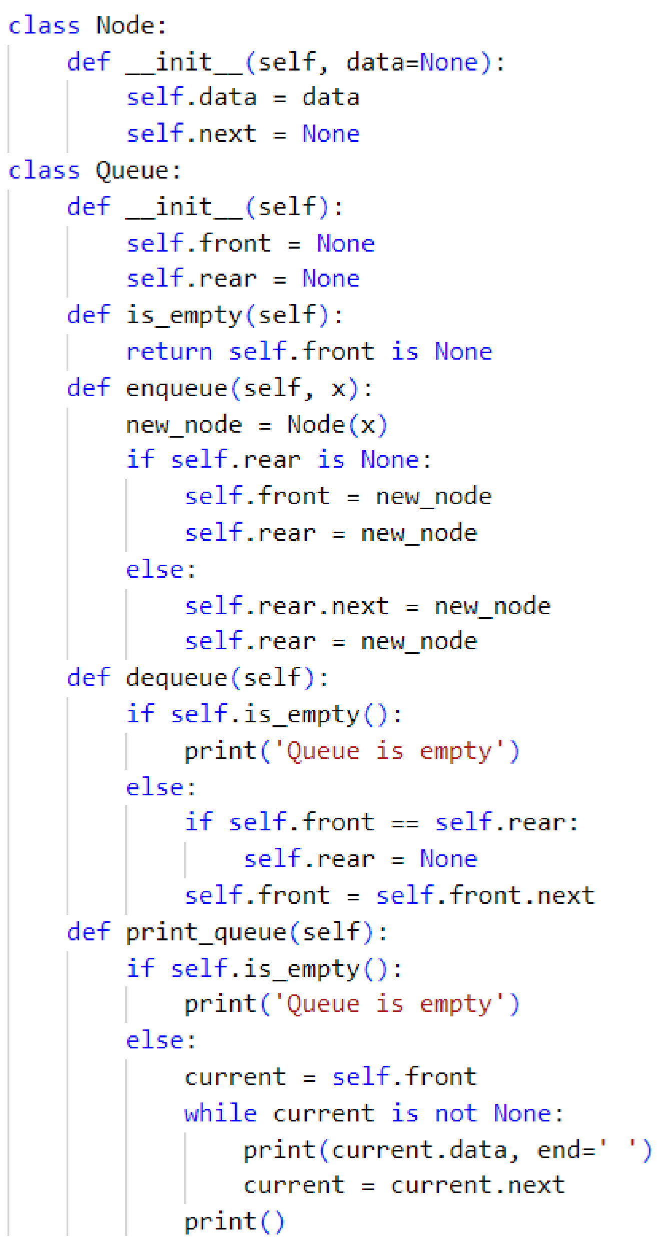
Lab 10

a) Construct the queue class and implement the above algorithms in Python



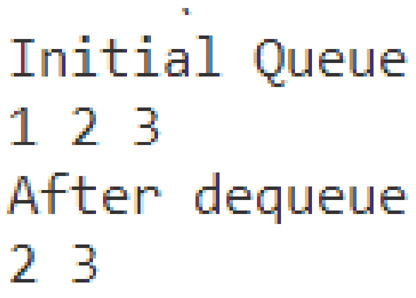
```python
class Queue:
    def __init__(self, maxlength):
        self.maxlength = maxlength
        self.front = 0
        self.rear = maxlength - 1
        self.elements = [None] * maxlength

    def add_one(self, i):
        return (i + 1) % self.maxlength

    def is_empty(self):
        return self.add_one(self.rear) == self.front

    def front_element(self):
        if self.is_empty():
            print('Queue is empty')
        else:
            return self.elements[self.front]

    def enqueue(self, x):
        if self.add_one(self.add_one(self.rear)) == self.front:
            print('Queue is full')
        else:
            self.rear = self.add_one(self.rear)
            self.elements[self.rear] = x

    def dequeue(self):
        if self.is_empty():
            print('Queue is empty')
        else:
            self.front = self.add_one(self.front)
```

d) Implement EMPTY, ENQUEUE, DEQUEUE algorithms in Python using linked list



```python
class Node:
    def __init__(self, data=None):
        self.data = data
        self.next = None
class Queue:
    def __init__(self):
        self.front = None
        self.rear = None
    def is_empty(self):
        return self.front is None
    def enqueue(self, x):
        new_node = Node(x)
        if self.rear is None:
            self.front = new_node
            self.rear = new_node
        else:
            self.rear.next = new_node
            self.rear = new_node
    def dequeue(self):
        if self.is_empty():
            print('Queue is empty')
        else:
            if self.front == self.rear:
                self.rear = None
            self.front = self.front.next
    def print_queue(self):
        if self.is_empty():
            print('Queue is empty')
        else:
            current = self.front
            while current is not None:
                print(current.data, end=' ')
                current = current.next
            print()
```

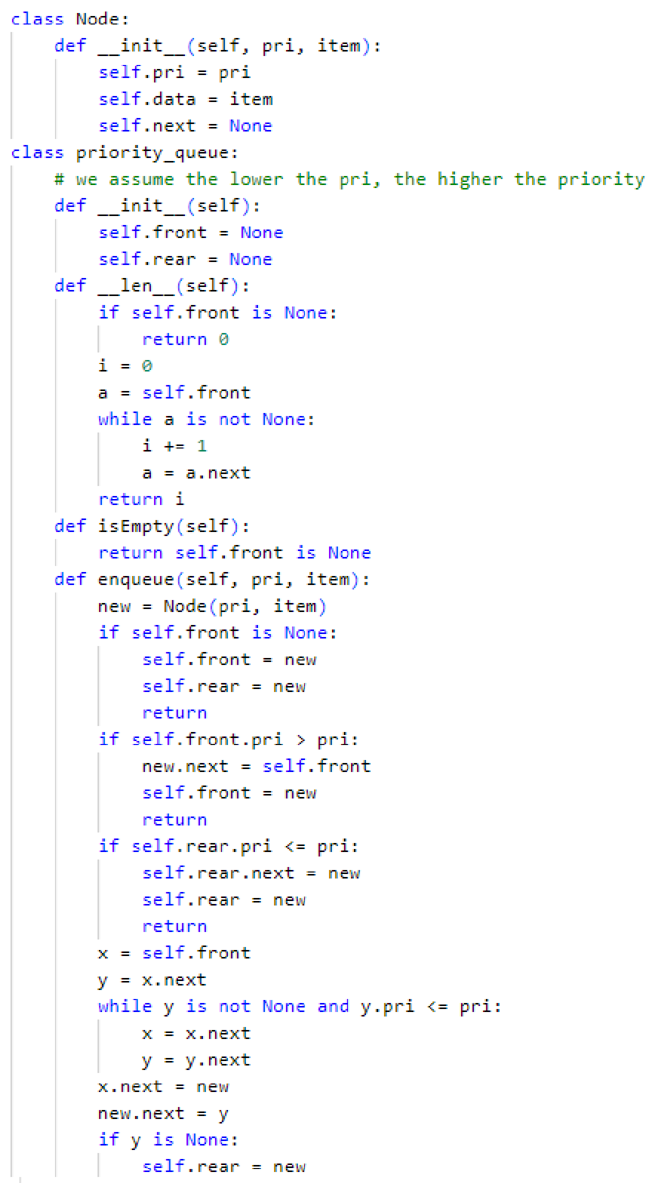
Output:



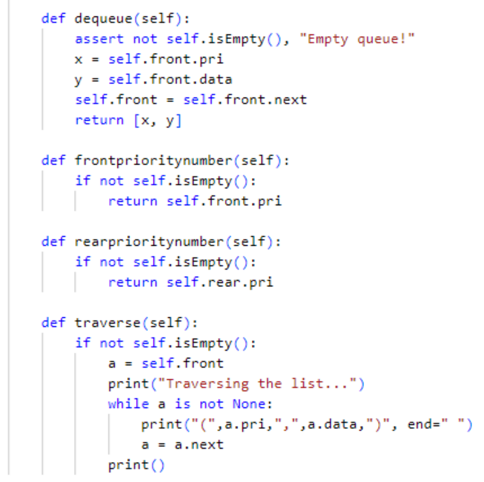
```
Initial Queue
1 2 3
After dequeue
2 3
```

e) Implement priority queue in Python using

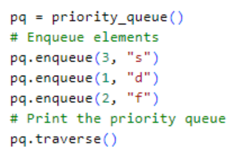
Linked List:



```
class Node:
    def __init__(self, pri, item):
        self.pri = pri
        self.data = item
        self.next = None
class priority_queue:
    # we assume the lower the pri, the higher the priority
    def __init__(self):
        self.front = None
        self.rear = None
    def __len__(self):
        if self.front is None:
            return 0
        i = 0
        a = self.front
        while a is not None:
            i += 1
            a = a.next
        return i
    def isEmpty(self):
        return self.front is None
    def enqueue(self, pri, item):
        new = Node(pri, item)
        if self.front is None:
            self.front = new
            self.rear = new
            return
        if self.front.pri > pri:
            new.next = self.front
            self.front = new
            return
        if self.rear.pri <= pri:
            self.rear.next = new
            self.rear = new
            return
        x = self.front
        y = x.next
        while y is not None and y.pri <= pri:
            x = x.next
            y = y.next
        x.next = new
        new.next = y
        if y is None:
            self.rear = new
```

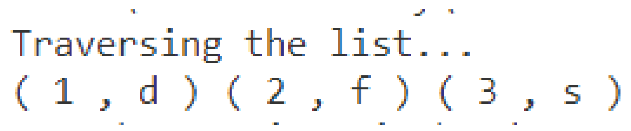


```
    def dequeue(self):
        assert not self.isEmpty(), "Empty queue!"
        x = self.front.pri
        y = self.front.data
        self.front = self.front.next
        return [x, y]

    def frontprioritynumber(self):
        if not self.isEmpty():
            return self.front.pri

    def rearprioritynumber(self):
        if not self.isEmpty():
            return self.rear.pri

    def traverse(self):
        if not self.isEmpty():
            a = self.front
            print("Traversing the list...")
            while a is not None:
                print("(",a.pri,",",a.data,")", end=" ")
                a = a.next
            print()
```



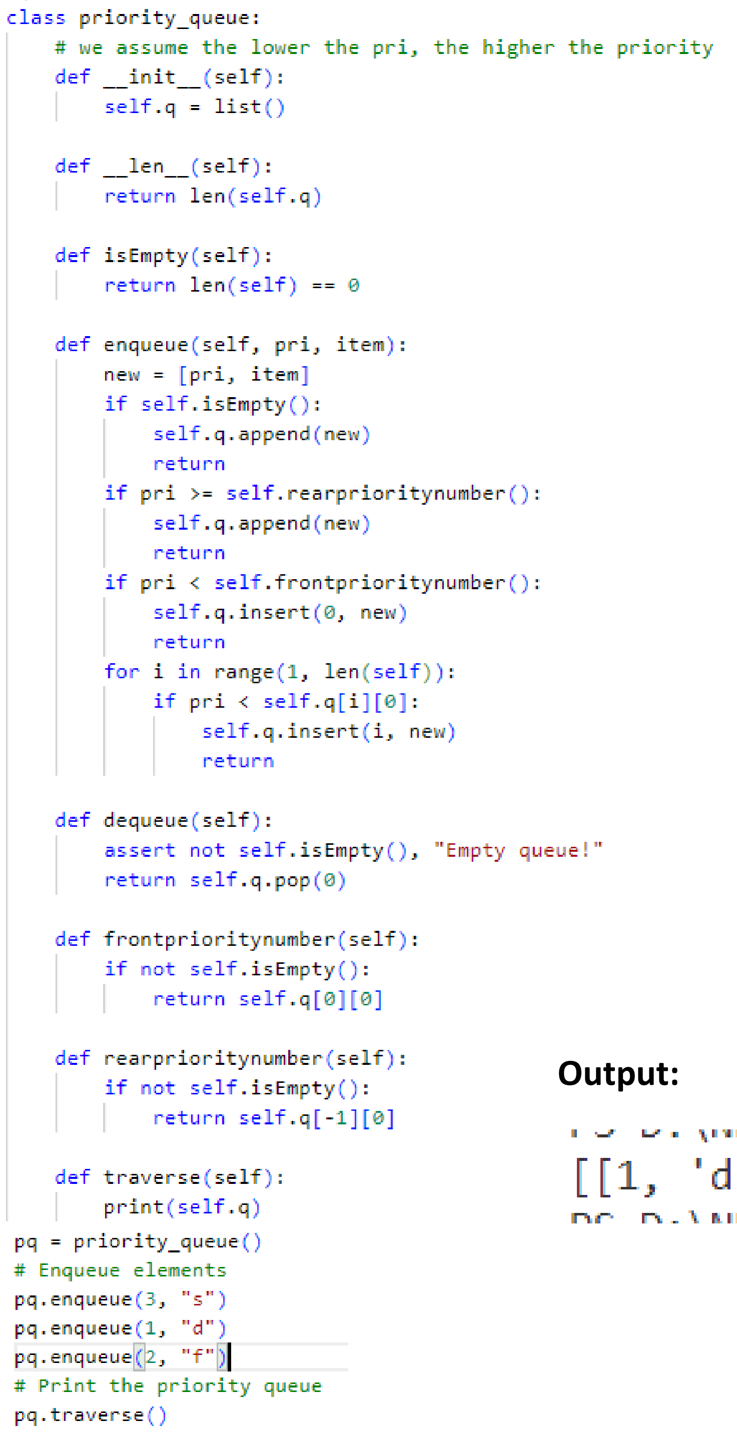
```
pq = priority_queue()
# Enqueue elements
pq.enqueue(3, "s")
pq.enqueue(1, "d")
pq.enqueue(2, "f")
# Print the priority queue
pq.traverse()
```

Output:



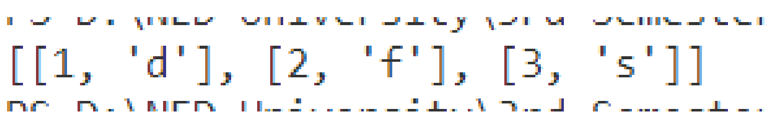
```
Traversing the list...
( 1 , d ) ( 2 , f ) ( 3 , s )
```

Array Implementation:



```python
class priority_queue:
    # we assume the lower the pri, the higher the priority
    def __init__(self):
        self.q = list()

    def __len__(self):
        return len(self.q)

    def isEmpty(self):
        return len(self) == 0

    def enqueue(self, pri, item):
        new = [pri, item]
        if self.isEmpty():
            self.q.append(new)
            return
        if pri >= self.rearprioritynumber():
            self.q.append(new)
            return
        if pri < self.frontprioritynumber():
            self.q.insert(0, new)
            return
        for i in range(1, len(self)):
            if pri < self.q[i][0]:
                self.q.insert(i, new)
                return

    def dequeue(self):
        assert not self.isEmpty(), "Empty queue!"
        return self.q.pop(0)

    def frontprioritynumber(self):
        if not self.isEmpty():
            return self.q[0][0]

    def rearprioritynumber(self):
        if not self.isEmpty():
            return self.q[-1][0]

    def traverse(self):
        print(self.q)
pq = priority_queue()
# Enqueue elements
pq.enqueue(3, "s")
pq.enqueue(1, "d")
pq.enqueue(2, "f")
# Print the priority queue
pq.traverse()
```

Output:



```
[[1, 'd'], [2, 'f'], [3, 's']]
```