Week 3 Live Coding Solutions

Week 3 - Live Coding Problem 1

A **queue** is a data structure in which whatever comes first will go out first. It follows the FIFO (First-In-First-Out) policy. In Queue, the insertion is done from one end known as the rear end of the queue, whereas the deletion is done from another end known as the front end of the queue.

For a given class **Queue**, create two methods:

- **Enqueue(data):** that accepts an integer `data` and inserts it into the queue in the last position.
- **Dequeue():** if queue is empty, returns `None`. Otherwise, delete one element of the queue from the first position and returns deleted element value.

Note- Use given linked list structure for implementation.

```python
class Node:
    def __init__(self, data):
        self.data = data # store data
        self.next = None #point to the next node
class Queue:
    def __init__(self):
        self.head = None # point to the first node of the list
        self.last = None # point to the last node of the list
```

Sample input

```
[2,4,6,8,10] #elements for insert in queue
3 # remove 3 elements from queue
```

Output

```
2 # first removed element from queue
4 # second removed element from queue
6 # third removed element from queue
8, 10 #remaining elements in queue
```

Solution

Prefix Code(Visible)

```python
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
class Queue:
    def __init__(self):
        self.head = None
        self.last = None
```

Solution code

```python

    def Enqueue(self,data):
        newnode = Node(data)
        if self.head == None:
            self.head = newnode
            self.last = newnode
        else:
            self.last.next = newnode
            self.last = newnode

    def Dequeue(self):
        if(self.head != None):
            item = self.head.data
            if self.head == self.last:
                self.head=None
                self.last=None
            else:
                self.head = self.head.next
            return item

```

Suffix Code(hidden)

```python
    def traverse(self):
        temp = self.head
        if temp != None:
            while temp != None:
                if temp.next != None:
                    print(temp.data, end=',')
                else:
                    print(temp.data)
                temp = temp.next
        else:
            print('None')

ins = eval(input())
delt=int(input())
A = Queue()
for i in ins:
  A.Enqueue(i)
for j in range(delt):
  print(A.Dequeue())
A.traverse()
```

Public Test case

Input 1

```
[1,2,3,4,5,6,7]
3
```

Output

```
1
2
3
4,5,6,7
```

Input 2

```
[1,2,3,4,5,6]
2
```

Output

```
1
2
3,4,5,6
```

Input 3

```
[10,20,30,40,50]
5
```

Output

```
10
20
30
40
50
None
```

Private Test case

Input 1

```
[10,20,30]
4
```

Output

```
10
20
30
None
None
```

Input 2

```
[]
1
```

Output

```
None
None
```

Input 3

```
[5,5,5,5,5,5,5]
5
```

Output

```
5
5
5
5
5,5,5
```

Input 4

```
[2,3,6,8,9,10,45,67,99,50]
5
```

Output

```
2
3
6
8
9
10,45,67,99,50
```

Week 3 - Live Coding Problem 2

A **doubly linked list** is a linked data structure that consists of a set of sequentially linked records called nodes. Each node contains three fields: two link fields (references to the previous and to the next node in the sequence of nodes) and one data field. So, it can be traversed in both directions.

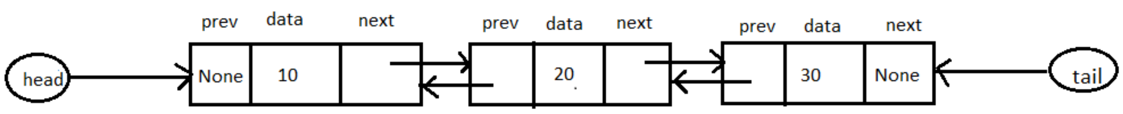
For the given class **doubly_linked_list**, create one methods:

- **insert_at_pos(data,pos):** that accepts an integer `data` and inserts it into the list at given `pos` position where `1 < pos <= length(list)`

Note- Use given linked list structure to implementation.

```python
class Node:
    def __init__(self, data):
        self.data = data # Stores data
        self.next = None # Contains the reference of next node
        self.prev = None # Contains the reference of previous node
class doubly_linked_list:
    def __init__(self):
        self.head = None # Contains the reference of first node of the list
        self.last = None # Contains the reference of the last node of the list
```

Example



Sample Input

```
[1,3,5,7,9] # Elements for insert in list one by one, start from 1
20 # data
2 # position
```

Output

```
1,20,3,5,7,9 # Traversed list from head to last
9,7,5,3,20,1 # Traversed list from last to head
```

Solution

Prefix Code(Visible)

```python
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
        self.prev = None
class doubly_linked_list:
    def __init__(self):
        self.head = None
        self.last = None

    def insert_end(self,data):
        newnode = Node(data)
        newnode.prev = self.last
        if self.head == None:
            self.head = newnode
            self.last = newnode
        else:
            self.last.next = newnode
            self.last = newnode
```

Solution code

```python
    def insert_at_pos(self,data,pos):
        newnode = Node(data)
        c = 1
        temp = self.head
        while c < pos-1:
            temp = temp.next
            c += 1
        temp1 = temp.next
        temp.next = newnode
        newnode.next = temp1
        newnode.prev = temp
        temp.next=newnode
        temp1.prev=newnode

```

Suffix code(hidden)

```python
    def traverse(self):
        temp = self.head
        while temp != None:
            if temp.next != None:
                print(temp.data, end=',')
            else:
                print(temp.data)
            temp = temp.next

    def traverse_rev(self):
        temp = self.last
        while temp != None:
            if temp.prev != None:
                print(temp.data, end=',')
            else:
                print(temp.data)
            temp = temp.prev

ins = eval(input())
data = int(input())
pos=int(input())
A = doubly_linked_list()
for i in ins:
  A.insert_end(i)
A.insert_at_pos(data,pos)
A.traverse()
A.traverse_rev()
```

Public test case

Input 1

```
[1,3,5,7,9]
20
2
```

Output

```
1,20,3,5,7,9
9,7,5,3,20,1
```

Input 2

```
[10,20,30,40,50,60]
15
2
```

Output

```
10,15,20,30,40,50,60
60,50,40,30,20,15,10
```

Input 3

```
[10,20,30,40,50,60]
55
6
```

Output

```
10,20,30,40,50,55,60
60,55,50,40,30,20,10
```

Private test case

Input 1

```
[10,20,30,40,30,20,10]
50
5
```

Output

```
10,20,30,40,50,30,20,10
10,20,30,50,40,30,20,10
```

Input 2

```
[4,5,6,7,8,9,10]
30
5
```

Output

```
4,5,6,7,30,8,9,10
10,9,8,30,7,6,5,4
```

Input 3

```
[1,2,3,4,5,6,7,8,9]
10
8
```

Output

```
1,2,3,4,5,6,7,10,8,9
9,8,10,7,6,5,4,3,2,1
```

Input 4

```
[30,50,56,20,65,90]
45
2
```

Output

```
30,45,50,56,20,65,90
90,65,20,56,50,45,30
```

Week 3 - Live Coding Problem 3

Implement a Stack using only two Queues. The implemented Stack should support all the functions of a normal stack (`push`, `pop`, `top`, and `isempty`).

Create the following method for given class `stack`.

- `push(data)` insert element `data` to the top of the stack.
- `pop()` If stack is empty return message `Stack is empty`. Otherwise, remove the element from the top of the stack and return it.
- `top()` If stack is empty return message `Stack is empty`. Otherwise, return the element on the top of the stack.
- `isempty()` Returns `True` if the stack is empty, `False` otherwise.

Sample Input

```
[10,20,30,40,50] #push one by one
2 # pop 2 elemnts
```

Sample Output

```
50 #first popped element
40 #second popped element
30 #top element
False #isempty
```

Solution

Prefix Code(Visible)

```python
class Queue:
    def __init__(self):
        self.items = []
    def enqueue(self, key):
        self.items.insert(0,key)
    def dequeue(self):
        return self.items.pop()

class stack:
    def __init__(self):
        self.q1 = Queue()
        self.q2 = Queue()
        self.size = 0
```

Solution Code

```python
    def isempty(self):
        return (self.size==0)
    def top(self):
        if (not self.isempty()):
            return self.q1.items[-1]
        else:
            return 'Stack is empty'
    def push(self,data):
        self.size += 1
        self.q2.enqueue(data)
        while (self.q1.items != []):
            self.q2.enqueue(self.q1.dequeue())
        self.q = self.q1
        self.q1 = self.q2
        self.q2 = self.q
    def pop(self):
        if (not self.isempty()):
            self.size -= 1
            return self.q1.dequeue()
        else:
            return 'Stack is empty'

```

Suffix Code(Visible)

```
inp = eval(input())
dl = int(input())
A = stack()
for el in inp:
    A.push(el)
for i in range(dl):
    print(A.pop())
print(A.top())
print(A.isempty())
```

Public test case

Input 1

```
[10,20,30,40,50]
2
```

Output

```
50
40
30
False
```

Input 2

```
[1,2,3,4,5,8,10,45,76,88,99,32,19]
5
```

Output

```
19
32
99
88
76
45
False
```

Input 3

```
[]
1
```

Output

```
Stack is empty
Stack is empty
True
```

Private test case

Input 1

```
[1,2,3,4,5]
5
```

Output

```
5
4
3
2
1
Stack is empty
True
```

Input 2

```
[1,2,3,4,5,6,7,8,9]
2
```

Output

```
9
8
7
False
```

Input 3

```
[1,2,3,4,5]
1
```

Output

```
5
4
False
```

Input 4

```
[1,2,3,4,5]
6
```

Output

```
5
4
3
2
1
Stack is empty
Stack is empty
True
```