Module 4

Lists, Stacks, and Queues

▼ What is a singly linked list?

- **a list with only one pointer between two successive nodes**
- it can only be traversed in a single direction, that is, you can go from the first node in the list to the last node, but you cannot move from the last node to the first node
- Example:
    - Start with a constructor that holds a reference to the very first node in the list.
    - Since this list is initially empty, set this reference to None.

```
class Node:
   def __init__(self, dataval=None):
      self.dataval = dataval
      self.nextval = None

class SLinkedList:
   def __init__(self):
      self.headval = None

list1 = SLinkedList()
list1.headval = Node("Mon")
e2 = Node("Tue")
e3 = Node("Wed")

# Link first Node to second node
list1.headval.nextval = e2

# Link second Node to third node
e2.nextval = e3
```

▼ What is a doubly linked list?

A list wherein each node has two pointers: a pointer to the next node and a pointer to the previous node.

```
class Node(object):
    def __init__(self, data=None, next=None, prev=None):
       self.data = data
       self.next = next
       self.prev = prev

class DoublyLinkedList(object):
   def __init__(self):
       self.head = None
       self.tail = None
       self.count = 0
```

▼ What is a stack?

- a stack is **a linear data structure that stores items in a Last-In/First-Out (LIFO) manner**
- a data structure that is often likened to a stack of plates
- a stack is a last in, first out (LIFO) structure
- push() - When an element is added to the top of the stack, it is **pushed** onto the stack.
- pop() - When an element is taken off the top of the stack, it is **popped** off the stack.
- peek() - Another operation, which is sometimes used, is **peek**, which makes it possible to see the element on the stack without popping it off.

```
class Node:
    def __init__(self, data=None):
        self.data = data
        self.next = None

class Stack:
    def __init__(self):
        self.top = None
        self.size = 0
```

```
def push(self, data):
   node = Node(data)
   if self.top:
       node.next = self.top
       self.top = node
   else:
       self.top = node
   self.size += 1

def pop(self):
  if self.top:
      data = self.top.data
      self.size -= 1
      if self.top.next:
          self.top = self.top.next
      else:
          self.top = None
      return data
  else:
      return None

def peek(self):
    if self.top
        return self.top.data
    else:
        return None
```

▼ What is a queue?

- **a linear data structure that stores items in a First-In/First Out(FIFO) manner**
- think of a line of people
- anytime an element is **enqueued**, the length or size of the queue increases by one
  - The enqueue operation or method uses the insert method of the list class to insert items (or data) **at the front of the list (the end of the queue)**
- conversely, **dequeuing** items reduce the number of elements in the queue by one
  - The dequeue operation is used to remove items from the queue.
  - **dequeues the last item from the list (first in line)**

- e.g. this operation captures the point where we serve the customer who joined the queue first and also waited the longest

```
# queue implementation using list

# Initializing a queue
class ListQueue:
    def __init__(self):
        self.items = []
        self.size = 0

    def dequeue(self):
      data = self.items.pop() # pop removes the last item from the list
      self.size -= 1
      return data

    def enqueue(self, data):
          self.items.insert(0, data)
          self.size += 1
```