- This lab will cover Stacks and Queues.
- It is assumed that you have reviewed chapters 6 of the textbook. You may want to refer to the text and your lecture notes during the lab as you solve the problems.
- When approaching the problems, think before you code. Doing so is good practice and can help you lay out possible solutions.
- Think of any possible test cases that can potentially cause your solution to fail!
- You must stay for the duration of the lab. If you finish early, you may help other students. If you don't finish by the end of the lab, we recommend you complete it on your own time. Ideally you should not spend more time than suggested for each problem.
- Your TAs are available to answer questions in lab, during office hours, and on Piazza.

Vitamins (25 minutes)

---

1. What is the output of the following code?

```
s = ArrayStack()
i = 2

s.push(1)
s.push(2)
s.push(4)
s.push(8)

i += s.top()
s.push(i)
s.pop()
s.pop()
print(i)
print(s.top())
```

2. Trace the following function with different list inputs. Describe what the function does, and give a meaningful name to the function:

```
def mystery(lst):
    s = ArrayStack()
    for i in range(len(lst)):
        s.push(lst.pop())
    for i in range(len(s)):
        lst.append(s.pop())
```

3. Trace the following function, which takes in a stack of integers. Describe what the function does, and give a meaningful name to the function:

```
def mystery(s):
    if len(s) == 1:
        return s.top()
    else:
        val = s.pop()
        result = mystery(s)

        if val < result:
            result = val
        s.push(val)
        return result
```

4. Fill out the prefix, infix, postfix table below:

Prefix	**Infix**	**Postfix**	**Value**
- * 3 4 10	3 * 4 - 10		2
	(5 * 5) + (10 / 2)	5 5 * 10 2 / +	30
		10 2 - 4 / 8 +	
+ * 6 3 * 8 4			
	(8 * 2) + 4 - (3 + 6)		

5. What is the output of the following code?

```
q = ArrayQueue()
i = 2

q.enqueue(1)
q.enqueue(2)
q.enqueue(4)
q.enqueue(8)

i += q.first()
q.enqueue(i)
```

```
q.dequeue()
q.dequeue()
print(i)
print(q.first())
```

6. Describe what the following function does and give it an appropriate name. Trace the function with a queue of integers.

```
def mystery(q):
    if (q.is_empty()):
        return

    else:
        val = q.dequeue()
        mystery(q)
        if val % 2 != 0:
            q.enqueue(val)
```

7. Trace the following function with different string inputs. Describe what the function does, and give a meaningful name to the function:

```
def mystery(input_str):
   s = ArrayStack()
   q = ArrayQueue()

   for char in input_str:
       s.push(char)
       q.enqueue(char)

   while not s.is_empty():
       if s.pop() != q.dequeue():
           return False

   return True
```

Coding

In this section, it is strongly recommended that you solve the problem on paper before writing code. Note that you should not access the underlying list in the ArrayStack implementation. **Treat it as a black box and only use the len, is_empty, push, top, and pop methods.**

Download the **ArrayStack.py**, **ArrayList.py, & ArrayQueue.py** files under Week 4 on NYU Brightspace

Note: import the class like so → `from ArrayStack import *`
→ `from ArrayQueue import *`

1. Write a **recursive function** that takes in a Stack of integers and returns the sum of all values in the stack. Do not use any helper functions or change the function signature. Note that the stack should be restored to its original state if you pop from the stack.

   ex) s contains [1, -14, 5, 6, -7, 9, 10, -5, -8] from top → bottom.
   stack_sum(s) returns -3

   ```
   def stack_sum(s):
       """
       : s type: ArrayStack
       : return type: int
       """
   ```

   Hint: See how the stack is restored in the code snippet from vitamins question 3.

2. Create an **iterative function** that evaluates a valid prefix string expression. You may only use **one ArrayStack** as an additional data structure to the given setup. Do not use any helper functions or change the function signature.

   In addition, each character is separated by one white space and numbers may have more than one digit. Therefore, we will use the split function to create a new list of each substring of the string separated by a white space. You may assume all numbers will be positive.

   ex) exp_str is " - + * 16 5 * 8 4 20"
   exp_lst = exp.split(" ") → ["-", "+", "*", "16", "5", "*", "8", "4", "20"]
   eval_prefix(exp_str) returns = 92

```
def eval_prefix(exp_str):
    """
    : exp type: str
    : return type: int
    """
    exp_lst = exp_str.split( )
```

   Hint:
   To check if a string contains digits, use `.isdigit( )`.
   To check if a string is an operator, you may want to do `if char in "-+/*"` similarly to how you checked for vowels.

   As you parse the expression lst, think about when you would push/pop the operator or number to/from the stack. Try to trace this execution on paper first before writing your code.

   Test your code with the various prefix expressions from the Vitamins q4.

3. Write an **iterative functio**n that flattens a nested list while retaining the left to right ordering of its values using one **ArrayStack** and its defined methods. That is, you should not directly access the underlying array in the implementation. Do not use any helper functions or change the function signature.
(30 minutes)

   In addition, do not create any other data structure other than the ArrayStack.

   ex) lst = [[[[0]]], [1, 2], 3, [4, [5, 6, [7]], 8], 9]
   flatten_list(lst)
   print(lst) → lst = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

```
def flatten_list(lst):
    """
    : lst type: list
    : return type: None
    """
    s = ArrayStack()
```

   Hint: You may want to traverse the list from the end for 2 reasons: pop and append has an amortized cost of O(1) and a stack reverses the collection order because of FIFO.

4. Implement the **ArrayDeque** class, which is an **array based implementation** of a Double-Ended Queue (also called a deque for short).

   A deque differs from a queue in that elements can be inserted to and removed from both the front and the back. (Think of this as a queue and stack combined).

   Like the ArrayQueue and ArrayStack, the standard operations for an ArrayDeque should occur in **O(1) amortized runtime**. **You may want to use and modify the ArrayQueue implementation done in lectures.**

   Your implementation should include the following methods (30 minutes):

   a)
```
def __init__(self):
    '''Initializes an empty Deque using a list as self.data.'''
```

   b)
```
def __len__(self):
```

```
    '''Return the number of elements in the Deque.'''
```

b)
```
def is_empty(self):
    '''Return True if the deque is empty.'''
```

c)
```
def first(self):
    '''Return (but don't remove) the first element in the Deque.
    Or raises an Exception if it is empty'''
```

d)
```
def last(self):
    '''Return (but don't remove) the last element in the Deque.
    Or raises an Exception if it is empty''''''
```

e)
```
def enqueue_first(self, elem):
    '''Add elem to the front of the Deque.'''
```

f)
```
def enqueue_last(self, elem):
    '''Add elem to the back of the Deque.'''
```

g)
```
def dequeue_first(self):
    '''Remove and return the first element from the Deque.
    Or raises an Exception if the Deque is empty'''
```

h)
```
def dequeue_last(self):
    '''Remove and return the last element from the Deque.
    Or raises an Exception if the Deque is empty'''
```

5. The MeanQueue is **mean** because it only enqueues integers and floats and rejects any other data type (bool, str, etc)! However, the nice thing about this queue is that it can provide the sum and average (mean) of all the numbers stored in it in **O(1) run-time**. **You may define additional member variables of O(1) extra space for this ADT.**

   The MeanQueue will use an ArrayQueue as its underlying data member. To test the datatype, you may use the "type(var)" function in python.

```
class MeanQueue:

      def __init__(self):
            self.data = ArrayQueue()

      def __len__(self):
      '''Return the number of elements in the queue'''

      def is_empty(self):
      ''' Return True if queue is empty'''


      def enqueue(self, e):
      ''' Add element e to the front of the queue. If e is not
      an int or float, raise a TypeError '''


      def dequeue(self):
      ''' Remove and return the first element from the queue. If
      the queue is empty, raise an exception'''


      def first(self):
```

```
    ''' Return a reference to the first element of the queue
    without removing it. If the queue is empty, raise an
    exception '''


    def sum(self):
    ''' Returns the sum of all values in the queue'''


    def mean(self):
    ''' Return the mean (average) value in the queue'''
```

6. In this question we will explore an alternative way to implement a *Stack* using just a *Queue* as the main underlying data collection. (40 minutes)

   Write a QueueStack class that implements a *Stack ADT* using an ArrayQueue as its only data member.

   You may only access the ArrayQueue's methods which include:
   `len, is_empty, enqueue, dequeue, and first.`

   Implement two sets of the push & pop/top methods:

   a. Consider an implementation that optimizes **push so that it has a run-time of O(1)** amortized.

   b. Consider an implementation that optimizes **pop and top so that they have a run-time of O(1**) amortized.

   c. Analyze the worst case run-time of your two sets of implementations for push and pop/top methods.

   Your implementation should be like so:

```
class QueueStack:

    def __init__(self):
        self.data = ArrayQueue()

    def __len__(self):
        return len(self.data)
```

```
def is_empty(self):
      return len(self) == 0

def push(self, e):
''' Add element e to the top of the stack '''

def pop(self):
''' Remove and return the top element from the stack. If the stack
is empty, raise an exception'''

def top(self):
''' Return a reference to the top element of the stack without
removing it. If the stack is empty, raise an exception '''
```