Vitamins

1. Trace the following function with different list inputs. Describe what the function does, and give a meaningful name to the function: (5 minutes)

```
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())
```

2. Trace the following function, which takes in a stack of integers. Describe what the function does, and give a meaningful name to the function: (10 minutes)

```
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
```

3. 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)
```

4. Consider the "circular" array implementation of a queue, similar to `ArrayQueue` that we studied in class, where the only difference is that the initial capacity is set to 4

```
class ArrayQueue:
     INITIAL_CAPACITY = 4
     def __init__(self):
          self.data_arr= make_array(ArrayQueue.INITIAL_CAPACITY)
          self.num_of_elems = 0
          self.front_ind = None
     def __len__(self): ...
     def is_empty(self): ...
     def enqueue(self, elem): …
     def dequeue(self): ...
     def first(self): ...
     def resize(self, new_cap): ...
```

Show the values of the data members: `front_ind`, `num_of_elems`, and the contents of each `data_arr[i]` after each of the following operations. If you need to increase the capacity of `data_arr`, add extra slots as described in class.

operation	**front_ind**	**num_of_elems**	**data_arr**
q=ArrayQueue()	None	0	[None, None, None, None]
q.enqueue('A')			
q.enqueue('B)			
q.dequeue()			
q.enqueue('C')			
q.dequeue()			
q.enqueue('D')			
q.enqueue('E')			
q.enqueue('F')			
q.enqueue('G')			
q.enqueue('H')			

Coding

Download the **ArrayStack.py** & **ArrayList.py** files under Content/Labs on Brightspace
Download the **ArrayQueue.py** file under Resources/Lectures on Brightspace.

```
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. (15 minutes)

   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. Implement the `MeanStack` class. The `MeanStack` pushes only integers and floats and rejects any other data type (bool, str, etc). It can also provide the sum and average of all numbers stored in **O(1) run-time**. **You may define additional member variables of O(1) extra space for this ADT.**

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

   ```
   class MeanStack:

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

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

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

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


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


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


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


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

3. Write an **iterative function** 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) when working from the end, and a stack reverses the collection order because of LIFO.

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:
   - `__init__(self):` *`Initializes an empty Deque using an array as self.data`*
   - `__len__(self):` *`Return the number of elements in the Deque`*
   - `is_empty(self):` *`Return True if the deque is empty`*
   - `first(self):` *`Return (but don't remove) the first element in the ArrayDeque. Raise an Exception if it is empty`*
   - `last(self):` *`Return (but don't remove) the last element in the ArrayDeque. Raises an Exception if it is empty`*
   - `enqueue_first(self, elem):` *`Add elem to the front of the ArrayDeque`*
   - `enqueue_last(self, elem):` *`Add elem to the back of the ArrayDeque`*
   - `dequeue_first(self):` *`Remove and return the first element from the Deque. Raise an Exception if the ArrayDeque is empty`*
   - `dequeue_last(self):` *`Remove and return the last element from the Deque. Raise an Exception if the ArrayDeque is empty`*