- 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.

---

## Submission Instructions

---

- Submit the completed lab on gradescope by Sunday, 26th July 2020, 11:55pm.
- Submit one pdf file with all your answers to the vitamin section in it. And submit a yourNetID_Lab3.py file with all the solutions to the coding questions in it.

---

## Vitamins

---

1. What is the output of the following code? (3 minutes)

a.)

```
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: (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())
```

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: (5 minutes)

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

    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 arrays in the ArrayStack and ArrayQueue implementations. **Treat it as a black box and only use the len, is_empty, push, top, and pop methods.**

Note:
import the class like so → `from [file name] import *`

```
from ArrayStack import *
```

1. a. Implement an **iterative function** that takes in a string of **only parentheses, brackets, and braces** and returns True if it is a balanced expression, meaning that for every `"("`, there is a corresponding `")"` in the correct nested position. Aim for a linear run-time and solve the problem with an ArrayQueue OR ArrayStack.
   Your function should check for the balance of the following characters:

   `"("`, `")"`, `"["`, `"]"`, `"{"`, `"}"`

   ex) is_balanced(`"{{([])}}([])"`) will return True
   is_balanced(`"{{[(])}}"`) will return False (non matching)

   is_balanced(`"([]{{[]}()}"`) will return False (excess character)
   is_balanced(`"(([]{{[]}})"`) will return False (excess character)
   is_balanced(`"([]{{[]}()})"`) will return True

```
def is_balanced(input_str):
    """
    : input_str type: str
    : return type: bool
    """
```

2. 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

   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.'''
   ```

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

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

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

f)
```
def add_first(self, elem):

    '''Add elem to the front of the Deque.'''
```

g)
```
def add_last(self, elem):

    '''Add elem to the back of the Deque.'''
```

h)
```
def delete_first(self):

    '''Remove and return the first element from the Deque.
    Or raises an Exception if the Deque is empty'''
```

i)
```
def delete_last(self):

    '''Remove and return the last element from the Deque.
    Or raises an Exception if the Deque is empty'''
```

**Use the starter file provided under NYUclasses > Resources > Labs > Lab3**

3. Create an **iterative function** that evaluates a valid prefix string expression. You may only use **one stack** 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.