- This lab will cover Stacks.
- 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!
- If you finish early, you may leave early after showing the TA your work. Ory you may stay and 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 Edstem.

Vitamins (40 minutes)

---

1. What is the output of the following code? (5 minutes)

```
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: (10 minutes)

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

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

4. Fill out the prefix, infix, postfix table below: (20 minutes)

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)		

Coding (75 minutes)

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** files under Content/Labs on Brightspace

Note: import the class like so → `from ArrayStack 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. 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.

3. Write an **iterative function** that will sort a stack of unsorted integers. You are **only allowed to use another stack** and no other additional data structure for your solution. Do not use any helper functions or change the function signature.

   A sorted stack is determined if its values are in ascending order from top to bottom. Smallest on top, largest on bottom. (30 minutes)

   The **runtime** of the function should be **quadratic**.

```
def stack_sort(s):
    """
    : input_str type: ArrayStack
    : return type: None
    """

    #use this to help with sorting

    helper_stack = ArrayStack( )
```