- This lab will review basic python concepts, classes, and memory map images.
- It is assumed that you have reviewed **chapters 1 and 2 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 the lab, during office hours, and on Piazza.

Vitamins (70 minutes)

---

1. For each section below, write the correct output shown after the Python code is run. Explain your answer by **drawing the memory image** for the execution of these lines of code. That is, you should draw the variables as they are organized in the call stack, and the data they each point to. (20 minutes)

a.

```
lst = [1, 2, 3]
lst2 = lst
lst.append(4)
lst2.append(5)

print(lst)

_____________________________________
print(lst2)

_____________________________________
```

b.

```
s = "aBc"
s = s.upper()
t = s
t = t.lower()

print(s)

_____________________________________
print(t)

_____________________________________
```

c.

```
s = "abc"
def func(s):
      s = s.upper()
      print("Inside func s =", s)

func(s)

_____________________________________
print(s)

_____________________________________
```

d.

```
lst = [1, 2, 3]
def func(lst):
      lst.append(4)
      lst = [5, 6, 7, 8]
      print("Inside func lst =", lst)

func(lst)

_____________________________________
print(lst)

_____________________________________
```

2. For each of the following, print the result of the list object created using python's list comprehension syntax (10 minutes):

```
[i//i for i in range(-3, 4) if i != 0]

_____________________________________________

['Only Evens'[i] for i in range(10) if i % 2 != 0]

_____________________________________________

[((-i)**3) for i in range(-2, 5)]
```

---

3. For each section below, write the correct output shown after the Python code is run. Explain your answer by **drawing the memory image** for the execution of these lines of code. That is, you should draw the variables as they are organized in the call stack, and the data they each point to. (30 minutes)

   a.

```
import copy
lst = [1, 2, [3, 4]]
lst_copy = copy.copy(lst)
lst[0] = 10
lst_copy[2][0] = 30


print(lst)

_____________________________________
print(lst_copy)

_____________________________________
```

b.

```
import copy
lst = [1, [2, "abc"], [3, [4]], 7]
lst_deepcopy = copy.deepcopy(lst)
lst[0] = 10
lst[1][1] = "ABC"
lst_deepcopy[2][1][0] = 40

print(lst)

_____________________________________________
print(lst_deepcopy)

_____________________________________________
```

c.

```
lst = [1, [2, 3], ["a", "b"] ]
lst_slice = lst[:]
lst_assign = lst
lst.append("c")
for i in range(1, 3):
      lst_slice[i][0] *= 2

print(lst)

_____________________________________
print(lst_slice)

_____________________________________________

print(lst_assign)

_____________________________________________
```

4. Given the generator function, write the output: (10 minutes)

```
def sum_to(n): #also known as triangle numbers

   for i in range(1, n+1):
       total = i * (i + 1)//2
       yield total

for i in sum_to(10):
    print(i, end = ', ')
```

_____

Coding

In this section, it is strongly recommended that you solve the problem on paper before writing code. For the **OPTIONAL** sections, we recommend you do these after lab for practice.

1. For this question, you will define a class to represent a polynomial. For this class, you will use a list as a data member to represent the coefficients. The index of each coefficient in the list will be its corresponding power of x. (45 minutes)

   For example, the coefficient list of the polynomial $p(x) = 2x^4 - 9x^3 + 7x + 3$ is `[3, 7, 0, -9, 2]`.

   index 0, coeff = 3 → $3x^0$. index 1, coeff = 7 → $7x^1$. index 2, coeff = 0 → $0x^2$
   index 3, coeff = -9 → $-9x^3$. index 4, coeff = 2 → $2x^4$

   Notice that $0x^2$ is included and that the coefficients in the list are in reversed order.

   Your class should include the following:

   a. A *constructor* that takes a list as a parameter, and initiates a polynomial with coefficients as given in the list. If no list is given at construction, your polynomial should be $p(x) = 0$. **Name the list member variable self.data.**

      Note: You may assume that the last element in the list (representing the coefficient of the highest power), is not 0.

   b. `__add__` operator. The operator should take another polynomial object, and create a new polynomial object representing the sum of the two polynomials. The operator should **not** modify the polynomial calling the method or the other polynomial object. Adding polynomials simply means adding their coefficients, but note that different polynomials might have different highest powers.

      For example:

$$(2x^4 - 9x^3 + x^2 + 7x + 3) + (3x^9 + 9x) = 3x^9 + 2x^4 - 9x^3 + x^2 + 16x + 3$$

   c. `__call__` operator, that takes a number and returns the value of the polynomial for that number when evaluated. For instance, calling `p(1)` where `p = Polynomial([3, 7, 0, -9, 2])` should return `3` because

      $2(1)^4 - 9(1)^3 + 7(1) + 3 = 3.$

If your Polynomial class works properly, you should see the following behavior:

```
#TEST CODE

#Constructor
poly1 = Polynomial([3, 7, 0, -9, 2]) #2x^4 - 9x^3 + 7x + 3
poly2 = Polynomial([2, 0, 0, 5, 0, 0, 3]) # 3x^6 + 5x^3 + 2


#add operator
poly3 = poly1 + poly2 # 3x^6 + 2x^4 - 4x^3 + 7x + 5
print(poly3.data) #[5, 7, 0, -4, 2, 0, 3]


#call operator
val1 = poly1(1)
print(val1) #3
val2 = poly2(1)
print(val2) #10
val3 = poly3(1)
print(val3) #13 (same result of 3 + 10; poly1(1) + poly2(1))
```

OPTIONAL

d. *__repr__* operator, that returns a str representation of a polynomial in the format presented above. Instead of superscript, we will represent powers using the caret symbol ^ . You may format it as such: $p(x) = 2x^4 - 9x^3 + 7x + 3$

```
2x^4 + -9x^3 + 0x^2 + 7x^1 + 3x^0
```

To achieve the formatting, you may want to use the join function. https://www.geeksforgeeks.org/join-function-python/

e. *__mul__* operator. The operator should take another polynomial object, and create a new polynomial object representing the multiplication of the two polynomials. To multiply polynomials, multiply all pairs of coefficients from both lists, and group the ones of the same order.

For example:

$$(5x^2 + x) * (2x^8 + 3x^2 + x) = 10x^{10} + 15x^4 + 5x^3 + 2x^9 + 3x^3 + x^2$$
$$= 10x^{10} + 2x^9 + 15x^4 + 8x^3 + x^2$$

You may want to start with a simpler example first to test your code:

$$(x + 1) * (x + 2) = x^2 + 3x + 2$$

f. A *derive* method that mutates the polynomial object to its derivative. You will have to implement the power rule. The modification must be in-place, that means you are not creating a new list with new values.

More on Derivatives: https://www.khanacademy.org/math/ap-calculus-ab/ab-derivative-rules/ab-diff-negative-fraction-powers/a/power-rule-review

For example, for the polynomial $2x^4 - 9x^3 + 7x + 3$, the derive method will modify it to be: $8x^3 - 27x^2 + 7$.

```
def derive(self):
"""
:
```

"""

2. In this question, we will implement the `UnsignedBinaryInteger` class to represent non-negative integers by their binary (*base 2*) representation. Each object will have a data-member of type string, containing the binary representation of the number.

ex) the `UnsignedBinaryInteger` object representing the decimal-number 13, will have '1101', as its string data-member. **Assume that the bin_num_str passed in the constructor does not have excess leading '0' in the front and will always begin with a '1' for positive numbers, and a single '0' for 0.**

Your implementation should account for the edge case where both numbers do not have the same number of digits. (45 minutes)

```
class UnsignedBinaryInteger:

    def __init__(self, bin_num_str):
        self.data = bin_num_str


    def __add__(self, other):
       ''' Creates and returns an UnsignedBinaryInteger object
        that represent the sum of self and other (also of
        type UnsignedBinaryInteger) the result also shouldn't have
       excess leading 0's'''


    def decimal(self):
       ''' returns the decimal value of the binary integer'''


    def __lt__(self, other):
       ''' returns True if self is less than other, or False
        otherwise'''


    def __gt__(self, other):
       ''' returns True if self is greater than other, or False
        otherwise'''


    def __eq__(self, other):
       ''' returns True if self is equal to other, or False
        otherwise'''
```

```
    def is_twos_power(self):
        ''' returns True if self is a power of 2, or False
         otherwise'''


    def largest_twos_power(self):
        ''' returns the largest power of 2 that is less than or
         equal to self'''


    def __repr__(self):
        ''' Creates and returns the string representation
         of self. The string representation starts with 0b,
         followed by a sequence of 0s and 1s'''
```

Note: the 0b is only included in repr when printing the object. Do not add 0b to the string for self.data.

Example test code provided on the next page.

```
#TEST CODE
b1 = UnsignedBinaryInteger('10011')
b2 = UnsignedBinaryInteger('100')

print("b1 is: ", b1) #0b10011; b1.data is 10011
print("b2 is: ", b2) #0b100;   b2.data is 100

b3 = b1 + b2
print("b3 is: ", b3) #0b10111

print("\nChecking decimal values:\n")
print(b1.decimal()) #19
print(b2.decimal()) #4
print(b3.decimal()) #23

print("\nChecking comparisons:\n")
print(b1 < b2) #False
print(b2 < b1) #True

print(b1 > b2) #True
print(b2 > b1) #False
print(b1 + b2 == b3) #False

print("\nChecking is_twos_power:\n")
print(b1.is_twos_power()) #False
print(b2.is_twos_power()) #True
print(b3.is_twos_power()) #False

print("\nChecking largest_twos_power:\n")
print(b1.largest_twos_power()) #16
print(b2.largest_twos_power()) #4
print(b3.largest_twos_power()) #16
```

OPTIONAL

Define two additional operators for bitwise operations in the `UnsignedBinaryInteger` class. In python, the bitwise or is represented by a single vertical bar, `|`, and the bitwise and is represented by a single and symbol, `&`.

Your implementation should account for the edge case where both numbers do not have the same number of digits.

bitwise OR

```
1010 or 1001 results in 1011

1 or 1 → 1
0 or 0 → 0
1 or 0 → 1
0 or 1 → 1
```

```
    def __or__(self, other):
        ''' Creates and returns a BinaryPositiveInteger object
            that represents the bitwise or result of self and other''
```

bitwise AND

```
1010 and 1001 results in 1000

1 and 1 → 1
0 and 0 → 0
1 and 0 → 0
0 and 1 → 0
```

```
    def __and__(self, other):
        ''' Creates and returns a BinaryPositiveInteger object
            that represents the bitwise and result of self and other''
```

```
#TEST CODE
b1 = UnsignedBinaryInteger('10011')
b2 = UnsignedBinaryInteger('100')
print("\nTesting b1: ", b1, "b2: ", b2)

b3 = b1 | b2
b4 = b1 & b2

print(b1, "|", b2, "=", b3) #0b100
print(b1, "&", b2, "=", b4)



b1 = UnsignedBinaryInteger('1010')
b2 = UnsignedBinaryInteger('1001')
print("\nTesting b1: ", b1, "b2: ", b2)

b3 = b1 | b2
b4 = b1 & b2

print(b1, "|", b2, "=", b3)
print(b1, "&", b2, "=", b4)
```

Output:

```
Testing b1:  0b10011 b2:  0b100
0b10011 | 0b100 = 0b10111
0b10011 & 0b100 = 0b0

Testing b1:  0b1010 b2:  0b1001
0b1010 | 0b1001 = 0b1011
0b1010 & 0b1001 = 0b1000
```

OPTIONAL VITAMINS

5. Use python's list comprehension syntax to generate the following lists: (10 minutes)

   a. `[1, -2, 4, -8, 16, -32, 64, -128]`

   _____

   b. `[1, 11, 111, 1111, 11111, 111111, 1111111]`

   _____

6. Finish the python's list comprehension syntax. The result is a list of characters of the input repeated twice. **Do not use any arithmetic operators or additional libraries.**

   Your answer must use my_str and length. (10 minutes)

```
print([_____________________________________])

my_str = "Python"
→ ["P","y","t","h","o","n","P","y","t","h","o","n"]

my_str = "Java"
→  ["J","a","v","a","J","a","v","a"]
```