- This lab will focus on classes, objects, and operator overloading.
- 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 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.
- Your TAs are available to answer questions in lab, during office hours, and on Piazza

---

Vitamins (maximum 1 hour)

---

1. Write the correct output and explain your answer by drawing a memory map image:

```
lst1 = [10, [20, [30]], ["a", "b"] ]
lst2 = lst1
lst3 = lst2[:]
lst4 = lst3.copy()

lst1[1][1] = "thirty"
lst2[1][0] *= 2
lst3[2].clear()
lst4.append("c")

print(lst1)

_____________________________________

print(lst2)

_____________________________________

print(lst3)

_____________________________________

print(lst4)

_____________________________________
```

2. For each of the following, state the list defined using python comprehension syntax:

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

_____

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

_____

3. Use list comprehension to generate the following lists:

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

_____

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

_____

4. Given the generator function, write the output:

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

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

_____

5. Write the output for the following lines of code given the `Student` class.

```
class Student:
    def __init__(self, name = "student", age = 18):
        self.name = name
        self.age = age
        self.courses = []

    def add_course(self, course):
        self.courses.append(course)

    def remove_course(self, course):
        if course in self.courses:
            self.courses.remove(course)
            print("Removed Course:", course)
        else:
            print("Course Not Found:", course)

    def __str__(self): #str representation needed for print( )
        info =  "Name: " + self.name
        info += "\nAge: " + str(self.age)
        info += "\nCourses: " + " , ".join(self.courses)
        return info + "\n"

peter = Student(16)
print(peter.name, peter.age)
```

_____

```
peter = Student("Peter Parker")
print(peter.name, peter.age)
```

_____

```
peter = Student(age = 16)
print(peter.name, peter.age)
```

```
peter.name = "Peter Parker"
print(peter)
```

```
peter.add_course("Algebra")
peter.add_course("Chemistry")
print(peter)
```

```
peter.add_course("Physics")
peter.remove_course("Spanish")
```

```
flash = Student("Flash Thompson")
flash.courses = peter.courses
flash.add_course("Economics")
peter.remove_course("Chemistry")
print(peter.courses)
print(flash.courses)
```

```
peter.name, flash.name = flash.name, peter.name
print(peter.name, peter.age)
print(flash.name, flash.age)
```

Coding

In this section, it is strongly recommended that you solve the problem on paper before writing code.

1. For this question, you will define a class to represent a polynomial. For the polynomial class, you will use a list as a data member to to represent the coefficients. The first element in the list represents the constant (or the coefficient of x^0), with each of the next elements being the coefficient of the next power in your polynomial.

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

   Notice that $0x^2$ is included in this list 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 $p(x) = 0$.

      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. 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
print(poly1(1)) #3
print(poly2(1)) #10
print(poly3(1)) #13 (result of 3 + 10;poly1(1) + poly2(1))


print(poly1(2)) #-23
print(poly2(2)) #234
print(poly3(2)) #211 (result of -23 + 23;poly1(2) + poly2(2))
```

OPTIONAL

d. `__str__` 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):
"""
:
```

"""

EXTRA

Use list comprehension to generate the following lists by iterating through some range. For the first one, you are given `my_string = 'Hello',` which you must use.
You are also not allowed any libraries or modules.

```
['H', 'e', 'l', 'l', 'o', 'H', 'e', 'l', 'l', 'o']
```

_____

```
[16.0, 8.0, 4.0, 2.0, 1.0, 2.0, 4.0, 8.0, 16.0]
```

_____

Write the correct output and explain your answer by drawing a memory map image:

```
lst1 = [10, [10], ["would", ["draw"]], "again" ]

lst2 = lst1
lst3 = [lst2] * 3

lst1.insert(1, "/")
lst3[2][-1] += "!"

print(lst1)
```

_____

```
print(lst2)
```

_____

```
print(lst3)
```

_____

Define a class Complex to represent complex numbers. Complex numbers take the form a + bi where a and b are real numbers (float) and i is the imaginary unit $\sqrt{-1}$.
More on Complex numbers: https://en.wikipedia.org/wiki/Complex_number

First, define the constructor below:

```
class Complex:
    def __init__(self, a, b):
```

Then implement the following methods by overloading the operators. For example, by defining the `__add__` operator, you will be able to use the `+` operator to add two complex numbers. With the `+`, `-`, and `*` operators, a new `Complex` object is created while the values of the original complex objects are not changed.

a. This add operator will add two complex numbers and create a new complex number object with the result.
   ```
   def __add__(self, other):
   ```

b. This sub operator will find the difference of two complex numbers and create a new complex number object with the result.
   ```
   def __sub__(self, other):
   ```

c. This mul operator will multiply two complex numbers and create a new complex number object with the result. Use the FOIL method.
   ```
   def __mul__(self, other):
   ```

d. The str operator allows you to convert the Complex object to a str. With a str representation, we can use the print function with the Complex object.
   ```
   def __str__(self, other):
   ```

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

```
#TEST CODE

#constructor, output
cplx1 = Complex(5, 2)
print(cplx1)    #5 + 2i

cplx2 = Complex(3, 3)
print(cplx2)    #3 + 3i

#addition
print(cplx1 + cplx2)  #8 + 5i

#subtraction
print(cplx1 - cplx2)  #2 - 1i

#multiplication
print(cplx1 * cplx2)  #9 + 21i
```