- This lab will review basic python concepts, classes, memory map images and analysis of algorithms
- 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 layout 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

---

1. 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 __repr__(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")
```

```
tom = Student("Tom Holland")
tom.courses = peter.courses
```

```
tom.add_course("Economics")
peter.remove_course("Chemistry")
print(peter.courses)
print(tom.courses)
```

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

---

2. State True or False. [OPTIONAL PART] Use the **definitions of** O **and** Θ in order to show the following:
   a. $n^2 + 5n - 2$ is $O(n^2)$
   b. $\frac{n^2-1}{n+1}$ is $O(n)$
   c. $\sqrt{5n^2 - 3n + 2}$ is $\Theta(n)$

3. State **True** or **False** and explain why for the following :
   a. $8n^2(\sqrt{n})$ is $O(n^3)$
   b. $8n^2(\sqrt{n})$ is $\Theta(n^3)$
   c. $16\ log(n^2) + 2$ is $O(log(n))$

4. For each of the following code snippets, find $f(n)$ for which the algorithm's time complexity is $\Theta(f(n))$ in its **worst case** run and explain why.

```
a) def func(lst):
       for i in range(len(lst)):
           if (len(lst) % 2 == 0):
               return
```

b)
```
def func(lst):
    for i in range(len(lst)):
        if (lst[i] % 2 == 0):
            print("i =", i)
        else:
            return
```

c)
```
def func(lst):
    for i in range(len(lst)):
        for j in range(len(lst)):
            if (i+j) in lst:
                print("i+j = ", i+j)
```

d)
```
def func(n):
    for i in range(int(n**(0.5))):
        for j in range(n):
            if (i*j) > n*n:
                print("i*j = ", i*j)
```

e)
```
def func(n):
    for i in range(n//2):
        for j in range(n):
            print("i+j = ", i+j)
```

Coding

In this section, it is strongly recommended that you solve the problem on paper before writing code. This will be good practice for when you write code by hand on the exams.

1. Given a list of values (`int, float, str, …`), write a function that reverses its order in-place. You are not allowed to create a new list. Your solution must run in $\Theta(n)$, where n is the length of the list .

```
def reverse_list(lst):
    """
    : lst type: list[]
    : return type: None
    """
```

2. Given a **sorted** list of positive integers with zeros mixed in, write a function to move all zeros to the end of the list while maintaining the order of the non-zero numbers. For example, given the list [0, 1, 0, 3, 13, 0], the function will modify the list to become [1, 3, 13, 0, 0, 0]. Your solution must be in-place and run in $\Theta(n)$, where n is the length of the list.

```
def move_zeros(nums):
    """
    : nums type: list[int]
    : return type: None
    """
```

3. Complete the following :

    a. The function below takes in a **sorted** list with n numbers, all taken from the range 0 to n, with one of the numbers removed. Also, none of the numbers in the list is repeated. The function searches through the list and returns the missing number.

    For instance, lst = [0, 1, 2, 3, 4, 5, 6, 8] is a list of 8 numbers, from the range 0 to 8, with the number 7 missing. Therefore, the function below will return 7.

    Analyze the worst case run-time of the function:

```
def find_missing(lst):
```

```
        for num in range(len(lst) + 1):
            if num not in lst:
                return num
```

b. Rewrite the function so that it finds the missing number with a better run-time: **Hint:** the list is sorted. Also, make sure to consider the edge cases.

```
def find_missing(lst):
    """
    : nums type: list[int] (sorted)
    : return type: int
    """
```

c. Suppose the given list is **not sorted** but still contains all the numbers from 0 to n with one missing.

For instance, lst = [8, 6, 0, 4, 3, 5, 1, 2] is a list of numbers from the range 0 to 8, with the number 7 missing. Therefore, the function below will return 7.

How would you solve this problem? Do not use the idea in step a, or sort the list and reuse your solution in step b.

```
def find_missing(lst):
    """
    : nums type: list[int] (unsorted)
    : return type: int
    """
```

4. 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 repr operator allows you to convert the Complex object to a str object and display it as output by calling print( ).

```
def __repr__(self, other):
```

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

```
#TEST CODE

'''
def __add__(self, other):
cplx1 + cplx2
In this example, self refers to cplx1 since it is the first argument
and other would refer cplx2 since it is the second argument.
'''


#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


#original objects remain unchanged
print(cplx1)     #5 + 2i
print(cplx2)     #3 + 3i
```

5.

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.

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

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. 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):
"""
:
"""
```

Optional

---

1. Implement the following function

```
def add_binary(bin_num1, bin_num2):
    """
    bin_num1 - type: str
    bin_num2 - type: str
    return value - type: str
    """
```

This function is given bin_num1 and bin_num2 which are two binary numbers represented as strings. When called, it should return their sum (also represented as a binary string). Do not use any python bit manipulation functions such as `bin( )`.

ex) `add_binary("11", "1")` should return `"100"`.