**Background**

Complete the following exercises. Your answers must be submitted via Canvas. Answers may be typed in this document, handwritten and scanned, or completed in a separate code file. **REGARDLESS OF THE SUBMITTED FORMAT, PLEASE ONLY SUBMIT A SINGLE FILE, WHATEVER THE FORMAT**.

Please keep in mind what this assignment is testing (your ability to use the basic features of Python). As such, your solutions should not be importing libraries and should be implementing the underlying algorithm for each question.

Each question is worth 5 points. There are 30 points available. This assignment is out of 26 points.

**Important Note**: If you are permitted to use a Python list object, the question will refer to a **sequence**, **list,** or **dynamic array**. If the question refers to a **static array**, you **cannot** assume that you can use Python list methods such as append or index slicing (array[2:4] or array[1::2]). You may construct a new array using the literal constructor (array = []), index it using standard indexing syntax (array[0] or array[0]=5), or find out how long it is using the len function (len(array)).

**Questions**

1. The DynamicArray class which we went over does not support negative indicies in the \_\_getitem\_\_ method. Update that method to allow it to take negative indicies, matching Python’s list semantics. This should be done by adjusting the values of the index, **not** simply exposing the underlying Python **sequence** object.

def \_\_getitem\_\_(self, k):

"""Return element at index k."""

if not 0 <= k < self.\_n:

raise IndexError('invalid index')

elif abs(k) > self.\_n:

raise IndexError('invalid index')

return self.\_A[k]

1. Write an algorithm for a function called list\_search which takes 2 parameters: a **sequence** of elements and a value to search for. The function should search the **sequence** for the value and return all indicies of instances of the value in the list.

def list\_search(sequence, search):

a = sequence

b = []

i = 0

while (i < len(a)):

if (a[i] == search):

b += [i]

i+=1

return b

Examples: list\_search([1, 'A', 3, 5, 7, 'A'], 'A') returns [1, 5].

list\_search([1, 'A', 3, 5, 7, 'A'], 3) returns [2].

list\_search([1, 'A', 3, 5, 7, 'A'], 'zebra') returns [].  
**NOTE: This may be done in pseudo-code or the language of your choice**

1. Explain the changes to our CaesarCipher that would need to be made to make it effective for a different alphabet. This explanation should be detailed enough as to operate as effective instructions for a developer who is implementing such a change.

To change CaesarCipher for another alphabet, the instances where 26 is used for calculations(lines 5-9) would need to be changed to however long the alphabet being implemented is. The way that the function transform is implemented would need to be changed. On line 23 if the first letter of the alphabet is different then the A in ord(“A”) would need to be changed to that letter.

1. Write an algorithm for a function called add3D which takes 2 parameters: two different, non-jagged 3D **sequences**. You may assume that the arrays are the same size and contain compatible types. The function should create a new **sequence** of the appropriate size, add the two elements at the corresponding spots together, store them in the new 3D **sequence**, and return the new 3D **sequence**.  
   **NOTE: This may be done in pseudo-code or the language of your choice**

**PSEUDO-CODE**

while i < height of the array:

while m < len(row):

while p < amount of variables in elemental array:

array+= [a2[i][m][p]+a1[i][m][p]]

p+=1

array2d += [array]

m+=1

array3d += [array2d]

i+=1

return array3d

1. Write an algorithm which has a pre-populated **static array** called array\_input of array type and separates it, every other character, into two separate **static arrays** called array1 and array2.

i=0

array1 = []

array2 = []

while (i<len(array\_input)):

if (i%2==0):

array1 += [array\_input[i]]

else:

array2 += [array\_input[i]]

i+=1

Example: array\_input: [1, 2, 3, 4, 5, 6, 7]; array1: [1, 3, 5, 7] and array2: [2, 4, 6]  
**NOTE: This may be done in pseudo-code or the language of your choice**

1. Write an algorithm for a function called removeAll which takes 3 parameters: an array that is a **static array**, a count of elements in the array, and a value. As with the remove method we discussed in class, elements past the count of elements are stored as None. This function should remove all occurrences of value and then shift the remaining data down. The last populated element in the **static array** should then be set to None. The function then returns the count of “valid” (i.e. non-removed) data elements left. This function should do the removal “by hand” and **SHOULD NOT** use the remove method.

Hint: Consider what how you update your current position in an **static array** when you don’t remove an element versus when you do remove an element.

**NOTE: This may be done in pseudo-code or the language of your choice**

def removeAll(array, count, value):

a = []

i=0

firstTime=True

p=count

while (len(array)>p):

array[p] = None

p+=1

while (i<count):

if (array[i] == value):

n= i

m= n+1

while (m<count):

array[n] = array[m]

m+=1

n+=1

if firstTime==True:

array[count-1] = None

firstTime==False

else:

i+=1

p=0

while p<count:

if array[p] != None:

a += [array[p]]

p+=1

return a