**Week 1**

**1.**

import random

def shuffleArrays():

#an empty array where numbers will be inserted from the user input

array = []

maxArray = int(input("How many numbers would you like to input into an array? "))

#this is going to ask a user for another number to add to an array as long as the length of the

#array is less than the maxArray which is set by the user.

#adds the number into the array

while len(array) < maxArray :

arrayq = input("Please input integers that you want to be shuffled :")

array.append(arrayq)

#Fisher-Yates algorithm, if the length of an array is larger than 1, the following statements will run

if len(array) > 1:

#take one away due to zero based array

index = len(array) - 1

#as long as the index variable is more than 0 the while loop will run

while index > 0:

#selects a random number between 0 and the index value and assigns it to a ranInt variable name

#swaps array[index] and array[randInt] around, randInt variable in this case acts as an index in the list

#decreases value of index variable by 1 everytime the while loop is run to make sure its not an infinite loop

ranInt = random.randint(0, index)

array[index], array[ranInt] = array[ranInt], array[index]

index -= 1

print("The shuffled array is :" ,array)

**2.**

def trailingZeros():

number = int(input("Give me a number: "))

trailing = 0

#A for loop checking every number between 5 and the number user inputted + 1

for i in range(5, number):

factorial = int(i)

#an infinite while loop to make sure the calculations below is made for each number

while factorial:

#checking if a number is a factor of 5

if factorial % 5 == 0:

#if the number is a factor of 5, add 1 to trailing variable

trailing += 1

#if the quotient is also a factor of 5, it does the same calculcation, e.g. 25 % 5 = 0 therefore adds one to trailing, then does

#25/5 = 5, since 5%5 = 0, the while loop runs again

factorial = factorial / 5

#if a number is not a factor of 5, the while loop breaks and goes back to the for loop

else:

break

print("There are ",trailing, "zeros")

**3.**

PERFSQ(number):

IF number > 0:  
 square < - number\*\*1/2  
 square < - square – ( square % 1)  
 square < - square \* square  
 RETURN(square)  
ELSE:  
 PRINT(“Error ! Try again with a positive integer”)

def perfSq():

number = int(input("Please enter a number"))

if number >= 0:

#sqaure roots a number insterted by a user

square = number\*\*(1/2)

#takes away square rooted number and takes away the decimal from the square rooted number

square = square - (square % 1)

#sqaures the number to find the closest perfect square

square = square \* square

print(square)

else:

print("Error ! Try again with a positive integer")

**4.**

**import** random  
  
**def shuffleArrays**():  
  
 array = [] (1)  
 maxArray = int(input("How many numbers would you like to input into an array? ")) (1)  
  
 **while** len(array) < maxArray : (n)  
 arrayq = input("Please input integers that you want to be shuffled :") (n)  
 array.append(arrayq) (n)  
  
 **if** len(array) > 1: (1)  
 index = len(array) - 1 (1)  
 **while** index > 0: (n)  
 ranInt = random.randint(0, index) (n)  
 array[index], array[ranInt] = array[ranInt], array[index] (n)  
 index -= 1 (n)  
 print("The shuffled array is :" ,array) (1)

O(n) 7n + 5

**def trailingZeros**():  
 number = int(input("Give me a number: ")) (1)  
  
 trailing = 0 (1)  
 **for** i **in** range(5, number+1): (n)  
 factorial = int(i) (n)  
 **while** factorial: (n\*n)  
 **if** factorial % 5 == 0: (n\*n)  
 trailing += 1 (n\*n)  
 factorial = factorial / 5 (n\*n)  
 **else**: (n\*n)  
 **break (**n\*n)print("There are ",trailing, "zeros") (1)  
  
  
trailingZeros()

O(n2) 6n2 + 2n + 3

**5.**

TO DO

**6.**

REVERSING-WORDS-IN-A-SENTENCE

A < - USER INPUT  
 FOR EACH character i in A

B < - SPLIT EACH WORD AND PUT IT IN A LIST  
 C < - REVERSE THE LIST

ENDFOR

FOR EACH element j in REVERSED LIST c

RETURN j

ENDFOR

**def reversingWordsInaSentence**():  
 a = str(input("Enter a sentence that you want to be reversed :")) (1)  
 **for** i **in** a: (n)  
 b = a.split(" ") (n)  
 c = b[::-1] (n)  
 **for** j **in** c: (n)  
 print(j, end= " ") (n)  
  
  
  
  
  
  
reversingWordsInaSentence()

5n + 1 O(n)

def reversingWordsInaSentence():

a = str(input("Enter a sentence that you want to be reversed :"))

#starts a for loop which goes through every character in the user input

for i in a:

#splits the words in a sentence, detects a new word whenever there is a SPACE in the sentence

b = a.split(" ")

#reverses a list

c = b[::-1]

#a for loop that changes a list into normal looking sentence

for j in c:

#prints the sentence, end= " " basically takes care of the sentence being in one line

print(j, end= " ")

**7.**

TO DO

**8.**

REMOVE-VOWELS(c <- 0)

vowels <- A LIST OF ALL VOWELS

IF c = LENGTH of vowels

RETURN TRUE

ELSE IF vowels[c] in USER INPUT  
 REMOVE ONE VOWEL AT A TIME FROM USER INPUT

RETURN the word after vowel has been removed

REMOVE-VOWELS(c)

ELSE

REMOVE-VOWELS(C + 1)

s < - USER INPUT

s1 < - CHANGE USER INPUT into a list

def removeVowels(c = 0):

vowels = ["a","e","i","o","u"]

if c == len(vowels):

return True

#check every vowel if it's in the word

elif vowels[c] in s1:

#removes a vowel from the word if found

s1.remove(vowels[c])

#prints the word after one vowel has been removed

print(s1)

#calls the function to check if there are multiples of the same vowel

removeVowels(c)

#calls the function if there are no more multiples of the vowels and adds one to the counter to move onto the next vowels in the list

else:

removeVowels(c + 1)

#converts user input to lower case letters

s = input("Type a word : ").lower()

#changes the user input into a list

s1 = list(s)

**9.**

BINARY-SEARCH(array, a, b)

firstIndex <- 0

lastIndex <- LENGTH OF AN ARRAY - 1

itemFound <- FALSE

WHILE (firstIndex <= lastIndex AND NOT itemFound)

mp <- FLOOR[(firstIndex + lastIndex) / 2]

IF array[mp] >= a and array[mp] <=b

itemFound <-TRUE

ELSE

IF a < array[mp] AND b < array[mp]

lastIndex = mp - 1

ELSE

firstIndex = mp + 1

RETURN(itemFound)

def binarySearch(array, a, b):

#first index in an array is always 0

firstIndex = 0

#since array start with 0, to get an actual length of an array we need to take 1 away

lastIndex = (len(array) - 1)

itemFound = False

while firstIndex <= lastIndex and not itemFound:

#finds the mid-point

mp = (firstIndex + lastIndex) // 2

#if statement that checks if midpoint is in the between the numbers that user inputted

if array[mp] >= a and array[mp] <= b:

#if number is in range, variable value is change to TRUE

itemFound = True

else:

#if a and b are smaller than the mid-point, take one away from the mid-point,

if a < array[mp] and b < array[mp]:

#and start looking for a number in the first half of an array

lastIndex = mp - 1

else:

#otherwise add one to the mid-point and start looking for a number in the second half of an array

firstIndex = mp + 1

print(itemFound)

binarySearch([1,5,7,8,9,10,15],-1,0)

**10.**

def sequences(L, a, b):

#the for loops that goes through the whole list

for i in range (0, len(L)):

if i == 0:

#always adds the first element of the list to a

a.append(L[0])

#if value of i is smaller than the value before it

elif L[i] <= (L[i-1]):

#if length of list a is larger than length of list b

if len(a) > len(b):

#list b = a

b = a[:]

#clear all the elements in list a

a[:] = []

#add element i to the list a

a.append(L[i])

else:

pass

#if value of i is larger than the previous value

elif L[i] > (L[i-1]):

#add the value to the list a

a.append(L[i])

#if length of a is larger than b, print list a

if len(a) > len(b):

print("The longest sequence so far is ", a)

#if length of list a and b are the same, print both of them

elif len(a) == len(b):

print("The two longest sequences are ", a, b)

#if length of b is larger, print b

else:

print("The longest sequence so far is ", b)

else:

pass

sequences([1,2,3,4,5,8,9,10,1,2,3,4,1,6,7,8,9,10,11,12,12],[],[])

**11.**

def listRemove(self,n):

if n.prev != 0:

n.prev.next = n.next

else:

self.head = n.next

if n.next != 0:

n.next.prev = n.prev

else:

self.tail = n.prev

**12.**

def in\_order(tree):

s = []

while True:

if tree != None:

s.append(tree)

tree = tree.left

else:

if len(s) > 0:

tree = s.pop()

print(tree.value)

tree = tree.right

else:

return False

**13.**

class Graph():

def \_\_init\_\_(self):

self.nodes = []

self.edges = dict()

def addVertex(self,value):

self.nodes.append(value)

if value in self.nodes:

self.edges[value] = []

else:

pass

def printVertices(self):

return self.nodes

def addEdges(self, node1, node2):

if node1 not in self.nodes:

self.nodes.append(node1)

if node2 not in self.nodes:

self.nodes.append(node2)

self.edges[node1].append(node2)

self.edges[node2].append(node1)

if \_\_name\_\_ == '\_\_main\_\_':

g = Graph()

g.addVertex(1)

g.addVertex(5)

g.addVertex(3)

g.addVertex(4)

g.addVertex(10)

g.addVertex(7)

g.addEdges(1,5)

g.addEdges(1,3)

g.addEdges(10,7)

g.addEdges(4,10)

g.addEdges(5,10)

for node in g.edges:

print(node, ":", g.edges[node])

print(g.printVertices())

…

class Stack:

def \_\_init\_\_(self):

self.items = []

def isEmpty(self):

return self.items == []

def push(self, item):

self.items.append(item)

def pop(self):

return self.items.pop()

def peek(self):

return self.items[len(self.items) - 1]

def size(self):

return len(self.items)

class Queue:

def \_\_init\_\_(self):

self.items = []

def isEmpty(self):

return self.items == []

def enqueue(self, item):

self.items.insert(0,item)

def dequeue(self):

return self.items.pop()

def size(self):

return len(self.items)

class Graph():

def \_\_init\_\_(self):

self.nodes = []

self.edges = dict()

def addVertex(self,value):

self.nodes.append(value)

if value in self.nodes:

self.edges[value] = []

else:

pass

def printVertices(self):

return self.nodes

def addEdges(self, node1, node2):

if node1 not in self.nodes:

self.nodes.append(node1)

if node2 not in self.nodes:

self.nodes.append(node2)

self.edges[node1].append(node2)

self.edges[node2].append(node1)

**14.**

def DFS(self, start):

visited = []

stack = Stack()

stack.push(start)

while stack.isEmpty() == False:

u = stack.pop()

if u not in visited:

visited.append(u)

for edge in self.edges[u]:

stack.push(edge)

dfsText = open("dfsTraversalOutput.txt", "w")

dfsText.write("DFS Traversal: %s " % visited)

dfsText.close()

def BFS(self, start):

Q = Queue()

visited = []

Q.enqueue(start)

while Q.isEmpty() == False:

u = Q.dequeue()

if u not in visited:

visited.append(u)

for edge in self.edges[u]:

Q.enqueue(edge)

bfsText = open("bfsTraversalOutput.txt", "w")

bfsText.write("BFS Traversal: %s " % visited)

bfsText.close()

if \_\_name\_\_ == '\_\_main\_\_':

g = Graph()

g.addVertex(1)

g.addVertex(5)

g.addVertex(3)

g.addVertex(4)

g.addVertex(10)

g.addVertex(7)

g.addEdges(1,5)

g.addEdges(1,3)

g.addEdges(10,7)

g.addEdges(4,10)

g.addEdges(5,10)

for node in g.edges:

print(node, ":", g.edges[node])

print(g.printVertices())

print(g.DFS(7))

print(g.BFS(7))

**15.**