class CircularArray:

def \_\_init\_\_(self, lin, st, sz):

# Initializing Variables

self.start = st

self.size = sz

self.cir = [None]\*len(lin)

# if lin = [10, 20, 30, 40, None]

# then, CircularArray(lin, 2, 4) will generate

# cir = [40, null, 10, 20, 30]

idx = st

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

idx = idx % len(lin)

self.cir[idx] = lin[i]

idx+=1

# To Do.

# Hints: set the values for initialized variables

# Print from index 0 to len(cir) - 1

def printFullLinear(self):

for i in range(0,len(self.cir)):

if i != len(self.cir)-1:

print(self.cir[i],end=',')

else:

print(self.cir[i])

# Print from start index and total size elements

def printForward(self): #Easy

idx = self.start

for i in range(0,self.size):

if idx > len(self.cir)-1:

idx = 0

print(self.cir[idx])

idx += 1

def printBackward(self): #Easy

idx = ((self.start+self.size)-1) % len(self.cir)

for i in range(0,self.size):

if idx < 0:

idx = len(self.cir)-1

print(self.cir[idx])

idx -= 1

# With no null cells

def linearize(self):

self.renew = [None]\*self.size

idx = self.start

for i in range(0,self.size):

if idx > len(self.cir)-1:

idx = 0

self.renew[i] = self.cir[idx]

idx += 1

print(self.renew)

# Do not change the Start index

def resizeStartUnchanged(self, newcapacity): #Medium

self.renewunchng = [None]\*newcapacity

idx = self.start

for i in range(idx,(idx+self.size)):

if idx > len(self.cir)-1:

idx = 0

self.renewunchng[i] = self.cir[idx]

idx += 1

print(self.renewunchng)

# This method will check whether the array is palindrome or not

def palindromeCheck(self):

var = True

st\_idx = self.start

last\_idx = (self.start+self.size-1)%len(self.cir)

for i in range(0,self.size//2):

if self.cir[st\_idx] == self.cir[last\_idx]:

var = True

else:

var = False

if st\_idx > len(self.cir)-1:

st\_idx = 0

else:

st\_idx += 1

if last\_idx < 0:

last\_idx = len(self.cir)-1

else:

last\_idx -=1

if var == True:

print('This array is a palindrome')

else:

print('This array is not a palindrome')

# This method will sort the values by keeping the start unchanged

def sort(self):

st\_idx = self.start

for idx1 in range(0,self.size):

for idx2 in range((idx1+1), self.size):

if (self.cir[(idx1+st\_idx)%len(self.cir)] > self.cir[(idx2+st\_idx)%len(self.cir)]):

temp = self.cir[(idx1+st\_idx)%len(self.cir)]

self.cir[(idx1+st\_idx)%len(self.cir)] = self.cir[(idx2+st\_idx)%len(self.cir)]

self.cir[(idx2+st\_idx)%len(self.cir)] = temp

print(self.cir)

# This method will check the given array across the base array and if they are equivalent interms of values return true, or else return false

def equivalent(self, cir\_arr):

var = True

st\_idx1 = self.start-1

st\_idx2 = cir\_arr.start-1

if self.size == cir\_arr.size:

for i in range(0,self.size):

if st\_idx1 >= len(self.cir)-1:

st\_idx1 = 0

else:

st\_idx1 += 1

if st\_idx2 >= len(cir\_arr.cir)-1:

st\_idx2 = 0

else:

st\_idx2 += 1

if self.cir[st\_idx1] == cir\_arr.cir[st\_idx2]:

var= True

else:

var= False

break

else:

var = False

return var

# the method take another circular array and returns a linear array containing the common elements between the two circular arrays.

def intersection(self, c2):

list1=[]

for idx1 in range(0,len(c2.cir)):

for idx2 in range(0,len(self.cir)):

if c2.cir[idx1] == self.cir[idx2] and c2.cir[idx1] != None:

list1.append(c2.cir[idx1])

return list1

# Tester class. Run this cell after completing methods in the upper cell and

# check the output

lin\_arr1 = [10, 20, 30, 40, None]

print("==========Test 1==========")

c1 = CircularArray(lin\_arr1, 2, 4)

c1.printFullLinear() # This should print: 40, None, 10, 20, 30

c1.printForward() # This should print: 10, 20, 30, 40

c1.printBackward() # This should print: 40, 30, 20, 10

print("==========Test 2==========")

c1.linearize()

c1.printFullLinear() # This should print: 10, 20, 30, 40

print("==========Test 3==========")

lin\_arr2 = [10, 20, 30, 40, 50]

c2 = CircularArray(lin\_arr2, 2, 5)

c2.printFullLinear() # This should print: 40, 50, 10, 20, 30

c2.resizeStartUnchanged(8) # parameter --> new Capacity

c2.printFullLinear() # This should print: None, None, 10, 20, 30, 40, 50, None

print("==========Test 4==========")

lin\_arr3 = [10, 20, 30, 20, 10, None, None]

c3 = CircularArray(lin\_arr3, 3, 5)

c3.printForward() # This should print: 10, 20, 30, 20, 10

c3.palindromeCheck() # This should print: This array is a palindrome

print("==========Test 5==========")

lin\_arr4 = [10, 20, 30, 20, None, None, None]

c4 = CircularArray(lin\_arr4, 3, 4)

c4.printForward() # This should print: 10, 20, 30, 20

c4.palindromeCheck() # This should print: This array is NOT a palindrome

print("==========Test 6==========")

lin\_arr5 = [10, 20, -30, 20, 50, 30, None]

c5 = CircularArray(lin\_arr5, 5, 6)

c5.printForward() # This should print: 10, 20, -30, 20, 50, 30

c5.sort()

c5.printForward() # This should print: -30, 10, 20, 20, 30, 50

print("==========Test 7==========")

lin\_arr6 = [10, 20, -30, 20, 50, 30, None]

c6 = CircularArray(lin\_arr6, 2, 6)

c7 = CircularArray(lin\_arr6, 5, 6)

c6.printForward() # This should print: 10, 20, -30, 20, 50, 30

c7.printForward() # This should print: 10, 20, -30, 20, 50, 30

print(c6.equivalent(c7)) # This should print: True

print("==========Test 8==========")

lin\_arr7 = [10, 20, -30, 20, 50, 30, None, None, None]

c8 = CircularArray(lin\_arr7, 8, 6)

c6.printForward() # This should print: 10, 20, -30, 20, 50, 30

c8.printForward() # This should print: 10, 20, -30, 20, 50, 30

print(c6.equivalent(c8)) # This should print: True

print("==========Test 9==========")

lin\_arr8 = [10, 20, 30, 40, 50, 60, None, None, None]

c9 = CircularArray(lin\_arr8, 8, 6)

c6.printForward() # This should print: 10, 20, -30, 20, 50, 30

c9.printForward() # This should print: 10, 20, 30, 40, 50, 60

print(c6.equivalent(c9)) # This should print: False

print("==========Test 10==========")

lin\_arr9 = [10, 20, 30, 40, 50, None, None, None]

c10 = CircularArray(lin\_arr9, 5, 5)

c10.printFullLinear() # This should print: 40, 50, None, None, None, 10, 20, 30

lin\_arr10 = [5, 40, 15, 25, 10, 20, 5, None, None, None, None, None]

c11 = CircularArray(lin\_arr10, 8, 7)

c11.printFullLinear() # This should print: 10, 20, 5, None, None, None, None, None, 5, 40, 15, 25

output = c10.intersection(c11)

print(output) # This should print: [10, 20, 40]