fullName:_____

andrewID:_____

recitationLetter:_____

15-112 F22

Quiz4 version A

You **MUST** stop writing and hand in this **entire** quiz when instructed in lecture.

- You may not unstaple any pages.
- Failure to hand in an intact quiz will be considered cheating. Discussing the quiz with anyone in any way, even briefly, is cheating. (You may discuss it only once the quiz has been posted to the course website.)
- You may not use your own scrap paper. If you must use additional scrap paper, raise your hand and we will provide some. You must hand any scrap paper in with your paper quiz, and we will not grade it.
- You may not ask questions during the quiz, except for English-language clarifications. If you are unsure how to interpret a problem, take your best guess.
- You may not use any concepts (including builtin functions) we have not covered in the notes this semester.
- You may not use dictionaries, sets, or recursion.
- We may test your code using additional test cases. Do not hardcode.
- Assume almostEqual(x, y) and roundHalfUp(n) are both supplied for you. You must write all other helper functions you wish to use.
- **Write your answers entirely inside the boxes!**

- **Note: There are three required problems in the following order: FR1, FR2, and CT1. Don't forget CT1!**

Free Response 1: destructive rotateListRight(L, n) [45 points]

Write the function rotateListRight(L, n) which takes a list L and an integer n, and **destructively** modifies the list so that each element is shifted to the right by n indices (including wraparound). As usual for destructive functions, this function should always return None. Examine the test cases carefully!

Note: If you do not know how to write this destructively, you may write it nondestructively instead for half-credit. A nondestructive function must return a new (properly rotated) list and must not mutate L.

```
def testRotateListRight():
    L = ['a','b','c','d']
    rotateListRight(L, 1)
    assert(L == ['d','a','b','c'])
    rotateListRight(L, 0)
    assert(L == ['d','a','b','c'])
    rotateListRight(L, -3)
    assert(L == ['c','d','a','b'])

    L = ['a', 42, True, None, 4.2]
    rotateListRight(L, 3)
    assert(L == [True, None, 4.2, 'a', 42])
    rotateListRight(L, 10)
    assert(L == [True, None, 4.2, 'a', 42])

    L = []
    rotateListRight(L, 10)
    assert(L == [])
testRotateListRight()
```

Begin your answer on the next page

Begin your FR1 answer here

Free Response 2: nondestructive isSinglePath(L) [40 points]

Assume L is a list of non-negative integers. If each integer in the list represents the next index to be visited in the list, a "single-path" list visits each index once before returning to the starting index.

The following list L is an example of a single-path list:

```
L = [3, 2, 0, 4, 1]
```

We can see this by starting at index 0:

```
L[0] == 3    # Visit L[3] next
L[3] == 4    # Visit L[4] next
L[4] == 1    # Visit L[1] next
L[1] == 2    # Visit L[2] next
L[2] == 0    # Visit L[0] next
```

This brings us back to where we started. Each index has been visited.
The following list L is NOT a single-path:

```
L = [1, 0, 3, 2]
```

Starting again at index 0:

```
L[0] == 1    # Visit L[1] next
L[1] == 0    # Visit L[0] next
```

This brings us back to where we started, BUT L[2] was never visited.

The list [5, 0, 2] is NOT a single-path because L[0]==5, which is out of range.

Write the nondestructive function isSinglePath(L) that takes a non-empty list L (guaranteed to contain only non-negative integers) which returns True if L is a single-path list, and False otherwise. Remember, this function must be nondestructive! Here are some example test cases:

```
def testIsSinglePath():
    assert(isSinglePath([0]) == True)
    assert(isSinglePath([1, 0]) == True)
    assert(isSinglePath([1, 2, 3, 4, 0]) == True)
    assert(isSinglePath([3, 2, 0, 4, 1]) == True)
    assert(isSinglePath([5, 0, 1, 2, 3, 4]) == True)
    assert(isSinglePath([5]) == False)
    assert(isSinglePath([1, 1, 2]) == False)
    assert(isSinglePath([1, 2, 3, 4]) == False)
testIsSinglePath()
```

Begin your FR2 answer here

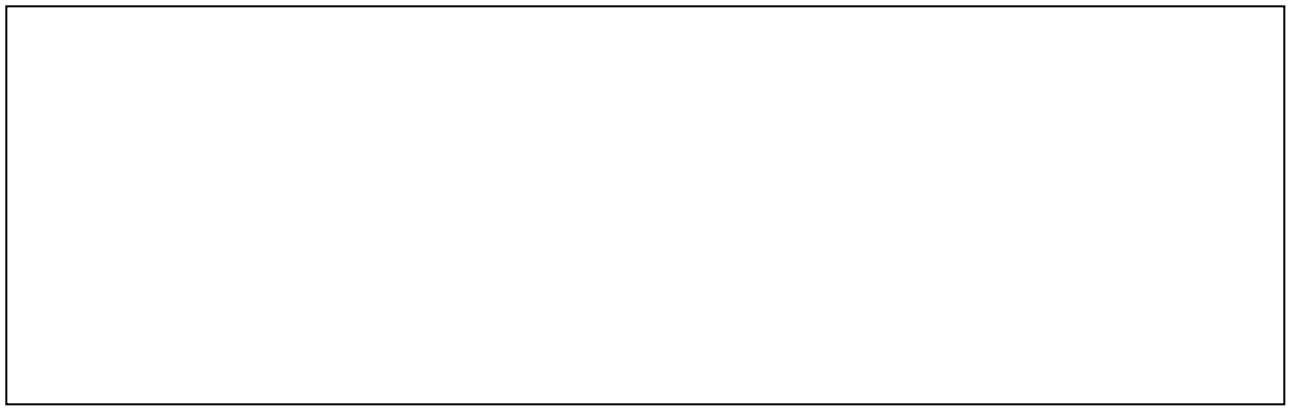
You may continue your FR2 answer here

CT1: Code Tracing [15pts]

Indicate what the following code prints. Place your answers (and nothing else) in the box below.

```
import copy
def ct1(x):
    y = x
    z = copy.copy(x)
    x.append(10)
    y[0] += 3
    x = x[2:]
    print(f'x = {x}')
    x[-1] = x[1] + x[0]
    z.pop(0)
    return x[-1], y[-1], z[-1]

L = [47, 80, 112]
print(f'ct = {ct1(L)}')
print(f'L = {L}')
```



bonusCT: Code Tracing [2pts bonus]

This question is optional. Indicate what the following code prints. Place your answers (and nothing else) in the box below.

```
def bonusCt1(L):
    return sum([int(d) for d in
                str([str(c*2)*2 for c in L*2])
                if d.isdigit()])
print(bonusCt1([4,5,6]))
```