fullName:__________________________andrewID:________________________ recitationLetter:_____

## 15-112 N23

# Quiz3 version A (40 min)

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. Read problem statements carefully and if you are unsure of how to interpret a problem, take your best guess.
- You may not use any concepts (including builtin functions) which we have not covered in the notes in week 3.
- 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 rounded(n) are both supplied for you. You must write all other helper functions you wish to use, unless we specify otherwise.

## Multiple Choice [10pts total]

**MC1. [1pt each]** Given a list `L = [1,2,3,4],` mark each of the following as mutating, non-mutating, a syntax error, or a runtime error. Select the best answer and for each (fill in one circle per question):

```
L = [1,2,3,4]
L += 5
```

- ◯ mutating
- ◯ non-mutating
- ◯ It will crash with a syntax error
- ◯ It will crash with a runtime error

```
L = [1,2,3,4]
L = L + [5,6]
```

- ◯ mutating
- ◯ non-mutating
- ◯ It will crash with a syntax error
- ◯ It will crash with a runtime error

```
L = [1,2,3,4]
L = L + [n for n in range(10) if n else n**2]
```

- ◯ mutating
- ◯ non-mutating
- ◯ It will crash with a syntax error
- ◯ It will crash with a runtime error

```
L = [1,2,3,4]
L.append(5,6)
```

- ○ mutating
- ○ non-mutating
- ○ It will crash with a syntax error
- ○ It will crash with a runtime error

```
L = [1,2,3,4]
L.remove(4)
```

- ○ mutating
- ○ non-mutating
- ○ It will crash with a syntax error
- ○ It will crash with a runtime error

```
L = [1,2,3,4]
L.insert(6,5)
```

- ○ mutating
- ○ non-mutating
- ○ It will crash with a syntax error
- ○ It will crash with a runtime error

```
L = [1,2,3,4]
L = L * 5
```

- ○ mutating
- ○ non-mutating
- ○ It will crash with a syntax error
- ○ It will crash with a runtime error

```
L = [1,2,3,4]
L = sort(L)
```

- ○ mutating
- ○ non-mutating
- ○ It will crash with a syntax error
- ○ It will crash with a runtime error

**MC2. [2pt]** What is the output of the following code? Select the best answer (fill in one circle).

```
L = [5,6,7,8]
M = L + [L.pop()]
print(M)
```

- ○ `[5,6,7]`
- ○ `[5,6,7,8]`
- ○ `[5,6,7,8,8]`
- ○ It will crash with a syntax error
- ○ It will crash with a runtime error

## CT1: Code Tracing [12pts]

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

```
import copy

def ct1(a):
    b = a
    (a, c) = (b + [b[0]], copy.copy(b))
    b.reverse()
    a[0] = b[0]
    d = a.extend(b[::2])
    a.remove(4)
    a.pop(4)
    for (s,L) in (("a:",a), ("b:",b), ("c:",c), ("d:",d)):
        print(s,L)

a = [x**2 for x in range(5)]
print("start:", a)
ct1(a)
print("end:", a)
```

## CT2: Code Tracing [12pts]

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

```
import copy

def ct2(a):
    (b,c,d) = (a, copy.copy(a), copy.deepcopy(a))
    print ((a == b), (a == c), (a == d))
    print ((a is b), (a is c), (a is d))
    print ((a[0] is b[0]), (a[0] is c[0]), (a[0] is d[0]))
    a[0] += [5]
    b[1] = [5, 6]
    c[1][0] += 10
    d[0] = c[1]
    for L in [a,b,c,d]:
        print(L)

a = [[1],[2,3,4]]
ct2(a)
```

## Free Response 1: countRotation(L, M) [32 points]

A rotation of a list L is constructed by shifting elements from one end of the list to the other. For example, `[2, 3, 4, 5, 6]` rotated once to the left is `[3, 4, 5, 6, 2]`. If you continue rotating this list to the left, you get `[4, 5, 6, 2, 3]`, `[5, 6, 2, 3, 4]`, and so on.

With that in mind, write the function countRotation(L, M) that takes two lists L and M. If M is a rotation of L, it returns the number of shifts to the left it takes to rotate L to get M. The function should check if rotating L to the left some number of times will produce M. If M is not a rotation of L it should return None.

Important notes:

- A list is a rotation of itself.
- The list may contain duplicates of the same item.
- **The function should be non-mutating and you cannot use copy.copy, copy.deepcopy, or L.copy.**

Carefully read the test cases below:

```
def testcountRotation():
    assert(countRotation([2, 3, 4, 5, 6], [2, 3, 4, 5, 6]) == 0)
    assert(countRotation([2, 3, 4, 5, 6], [4, 5, 6, 2, 3]) == 2)
    assert(countRotation([2, 3, 4, 5, 6], [2, 4, 3, 5, 6]) == None)
    assert(countRotation([1, 3, 3, 3, 7], [3, 7, 1, 3, 3]) == 3)
    assert(countRotation([1, 3, 3, 3, 7], [1, 7, 3, 3, 3]) == None)
    assert(countRotation([1], []) == None)
    assert(countRotation([1,2,3],[1,2]) == None)
    assert(countRotation([], []) == 0)
```

**Begin your FR1 answer on the following page**

Begin your FR1 answer here

You may continue your FR1 answer here

## Free Response 2: traverseMap(n, commands) [34 points]

Write the function traverseMap(n, commands), which takes in the dimensions of a map (n x n) and a multiline string of commands (commands).

The function should create a 2D list with n rows and n columns which is initially populated with all zeros. Based on the string of commands, you will then traverse and update the 2D list you created starting at index (0,0). After executing all the given commands, traverseMap should return the final state of the 2D list.

For example, if n = 5 and the commands were as follows:

```
n = 5 # dimensions
commands = '''\
MOVE RIGHT for 3 spaces and PLACE a 1
MOVE DOWN for 2 spaces and PLACE a 2
MOVE LEFT for 1 spaces and PLACE a 4
MOVE DOWN for 2 spaces and PLACE a 5
MOVE RIGHT for 3 spaces and PLACE a 6
'''

Final Map =
[[0, 0, 0, 1, 0],
[0, 0, 0, 0, 0],
[0, 0, 4, 2, 0],
[0, 0, 0, 0, 0],
[6, 0, 5, 0, 0]]
```

The commands will always follow the following format:

'MOVE `<dir>` for `<n>` spaces and PLACE a `<m>`'

where `<dir>` is UP, DOWN, LEFT, or RIGHT
and `<n>` is a positive integer
and `<m>` is a positive integer.

Important notes:

- **The commands may result in going out of bounds. In that case, it should wrap around.** If a row is `[0, 0, 5, 0, 0]` and current space is at index 2, if the command says 'MOVE RIGHT for 3 spaces and PLACE a 6', the result would be `[6, 0, 5, 0, 0]` rather than crashing with an IndexError.
- If there is already a number in a cell and a command says to place a number there, you should place the new number.

Carefully read the test cases on the next page:

```
def testtraverseMap():
    commands = """\
MOVE RIGHT for 3 spaces and PLACE a 1
MOVE DOWN for 2 spaces and PLACE a 2
MOVE LEFT for 1 spaces and PLACE a 4
MOVE DOWN for 2 spaces and PLACE a 5
MOVE RIGHT for 3 spaces and PLACE a 6
"""
    finalMap = [[0, 0, 0, 1, 0],
                [0, 0, 0, 0, 0],
                [0, 0, 4, 2, 0],
                [0, 0, 0, 0, 0],
                [6, 0, 5, 0, 0]]
    assert (traverseMap(5, commands) == finalMap)
    commands = """\
MOVE DOWN for 2 spaces PLACE a 5
MOVE RIGHT for 3 spaces and PLACE a 1
MOVE UP for 1 spaces and PLACE a 4
MOVE LEFT for 4 spaces and PLACE a 7
MOVE RIGHT for 1 spaces and PLACE a 6
MOVE LEFT for 10 spaces and PLACE a 100
"""
    finalMap = [[0, 0, 0, 0],
                [6, 0, 100, 7],
                [5, 0, 0, 1],
                [0, 0, 0, 0]]
    assert (traverseMap(4, commands) == finalMap)
```

**Begin your FR2 answer on the following page**

Begin your FR2 answer here

You may continue your FR2 answer here

You may continue your FR2 answer here

## bonusROC: Reasoning over Code [2pts]

This question is optional, and intentionally quite difficult. Find the argument L for the function that makes it return True. Place your answer (and nothing else) in the box below.

```
def rc(L):
    if ((type(L) != list) or (len(L) != 2) or
        (type(L[0]) != list) or (len(L[0]) != 3)):
        return False
    for row in L:
        if (type(row) != list) or (len(row) != len(L[0])): return False
        for x in row:
            if (type(x) != int) or (x < 1) or (x > 3): return False

    L.append(L[0])
    c1 = [(1,0), (0, 1),]
    c2 = [(-1, 0), (0, -1)]
    for (drow, dcol) in c1:
        if L[drow + 1][dcol + 1] >= L[1][1]: return False
        if L[drow][dcol] <= L[0][0]: return False
    for (drow, dcol) in c2:
        if L[drow + 2][dcol + 2] <= L[2][2]: return False
        if L[drow + 1][dcol + 1] >= L[1][1]: return False
    return True
```

Find the argument L for the function that makes it return True.