fullName:_____ andrewID:_____ section:___

15-112 S25
Quiz5 version B

Read these instructions carefully before starting:

1. Quiz versions are color-coded. You must have a different version (color) of this quiz than the students sitting to your left and right.
2. Stop writing and submit the entire quiz when instructed by the proctor.
    - Do not unstaple any pages.
    - You must submit the entire quiz with all pages intact.
3. Do not discuss the quiz with anyone else until after 5pm.
    - This applies to everyone, including students in either lecture.
4. Do not use your own scrap paper.
    - You should not need scrap paper, there is plenty of room for you on the quiz.
    - However, if you absolutely must use scrap paper, raise your hand and we will provide some. Then, you must write your andrew id clearly on the scrap paper, and hand in the scrap paper with your paper quiz. We will not grade anything on your scrap paper.
5. You may not ask questions during the quiz.
    - The one exception is for English-language clarifications.
    - If you are unsure how to interpret a problem, just take your best guess.
6. Do not use any concepts (including built-in functions) not covered in the notes through week 5 or beyond unit 4.
    - Do not use dictionaries, sets, or recursion.
7. Do not hardcode your solutions.
    - We may test your code using additional test cases.
    - Hardcoding will receive zero points.
8. 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.
9. Good luck!

Code Tracing (CT) [30 pts, 10 pts each]
For each CT, indicate what the code prints
Place your answer (and nothing else) in the box below the code.

Note: If floats occur in these CTs, they will have no more than one digit after the decimal point.

CT1:

```
def ct1(L):
    M = [ ]
    for i in range(len(L)):
        if i in L:
            j = L.index(i)
            M.append(sum(L[i:j]))
    return M
print(ct1([3,6,1,0,2]))
```

CT2:

```
def ct2(L):
    M = [ 2*v for v in L if v%2 == 0]
    return '='.join([str(v) for v in M])
Z = ct2(list(range(8)))
print(Z)
print(Z[::-1])
```

CT3:

```
def ct3(L):
    M = L
    N = copy.copy(L)
    M += [4]
    L.append(M.pop())
    L = L + [2]
    N.insert(0, M.pop(0))
    print(L)
    print(M)
    print(N)
L = [0]
ct3(L)
print(L)
```

You may use this page as scratch area for the CTs. However, as usual you may not unstaple it or any other page, and we will only grade what you write in the answer boxes.

Free Response / FR1: Mutating remove auto-indexes, mrai(L) [35 pts]

Background: For a list L and non-negative index `i`, we will say that `L[i]` is an "auto-index" if `L[i] == i`. For example, in the list `L = [4, 1, 0, 3]`, we see that `L[1] == 1` and `L[3] == 3`, so `L[1]` and `L[3]` are both auto indexes.

With this in mind, write the function `mrai(L)` (which stands for Mutating Remove Auto-Indexes) that takes a possibly-empty list of integers L and mutates L so that all the auto-indexes in L are removed. The function should return the number of removed values.

Here are some test cases:

```
L = [4, 1, 0, 3]
assert(mrai(L) == 2)
assert(L == [4, 0])

L = [1, 4, 3, 0]
assert(mrai(L) == 0)
assert(L == [1, 4, 3, 0])

L = [0, 1, 2, 3, 4, 5]
assert(mrai(L) == 6)
assert(L == [ ])

L = [ ]
assert(mrai(L) == 0)
assert(L == [ ])
```

Note that we remove auto-indexes based on their initial indices in L.
We do **not** remove integers that become auto-indexes later. For example:

```
L = [0, 0, 1]
   assert(mrai(L) == 1) # Remove only the 0 at L[0]
L = [0, 1] # These ints were not originally auto-indexes
```

Begin your FR2 answer on the next page.

Write your answer to FR1 here:

Free Response / FR2: longestCommonSubstring, lcs(L) [35 pts]

Write the function `lcs(L)` (which stands for Longest Common Substring) that takes a possibly-empty list L of strings, and returns the longest-possible string s that is a substring of every string in L, or None if L is empty. You can resolve ties any way you wish.

Here are some test cases for you:

```
assert(lcs(['abcde', 'abcd', 'bcde', 'bcd']) == 'bcd')
assert(lcs(['abcde', 'abc', 'bcde']) == 'bc')
assert(lcs(['de', 'abcde', 'cde']) == 'de')
assert(lcs(['abcde', 'abc', 'cde']) == 'c')
assert(lcs(['de', 'abc', 'cde']) == '')
assert(lcs([ ]) == None)
```

Hint: If L is not empty, the longest common substring must appear in `L[0]` by definition.

Begin your FR3 answer here or on the next page.

Begin your answer to F2 here:

You may continue your answer to FR2 here:

Bonus questions will be given on Thursday