fullName:____________________________andrewID:____________________________ recitationLetter:______

15-112 F23

Quiz9 (25 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. If you are unsure 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 weeks 1-9 / units 1-7.
- 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.

CT1: Code Tracing [8pts]

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

```
def ct1(n):
    if (n < 10):
        return n
    else:
        return 10*ct1(n//10) + ct1(n//100)

print(ct1(2351))
```

CT2: Code Tracing [8pts]

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

```
def g(L):
    if L == [ ]:
        return ['yes']
    elif sum(L) >= L[0]:
        return ['no']
    else:
        return [L[-1]] + g(L[1:-1]) + [L[0]]

def ct2(L):
    return g(L), g(list(reversed(L)))

print(ct2([7,-4,-3, 2]))
```

Free Response 1: Recursive hasTwoNines(n) [40pts]

Without using 'for' or 'while' loops, strings, or lists, write the recursive function `hasTwoNines(n)` that takes a possibly-negative integer n and returns True if it contains the digit 9 exactly twice, and False otherwise.

You may use helper and/or wrapper functions if you wish. See the test cases for more details.

```
def testHasTwoNines():
    assert(hasTwoNines( 139495) == True)
    assert(hasTwoNines(-139495) == True)  # negatives work!
    assert(hasTwoNines( 99) == True)
    assert(hasTwoNines( 59299) == False) # too many 9's
    assert(hasTwoNines(-394) == False) # too few 9's
    assert(hasTwoNines( 39401234) == False) # too few 9's
    assert(hasTwoNines( 0) == False) # too few 9's
```

Begin your FR1 answer here or on the next page:

Begin or continue your FR1 answer here

Free Response 2: Recursive indexMap(L) [40pts]

Without using 'for' or 'while' loops or builtin list or string methods that run in O(N) (like `.find`, or `.count`), write the function `indexMap(L)` that takes a 1D list L and returns a dictionary that maps each value in L to a set of the indexes in L where that value occurs. L is guaranteed to only contain immutable objects.

For example, `indexMap([5, 6, 5])` returns `{ 5:{0,2}, 6:{1} }` because 5 occurs at index 0 and 2, and 6 appears at index 1.

You may use helper and/or wrapper functions if you wish. See the test cases for more details.

```
def testIndexMap():
    print('Testing indexMap...')
    assert(indexMap([5, 6, 5]) == { 5:{0,2}, 6:{1} })
    assert(indexMap([9, 6, 3, 6, 9]) == { 3:{2}, 6:{1,3}, 9:{0,4} })
    assert(indexMap([3, 2, 1, 3, 2, 3]) == { 1:{2}, 2:{1, 4}, 3:{0,3,5} })
    assert(indexMap([5, 'cat', 5, 'a', (1,5), 'a']) == { 5 : {0,2},
                                                          'cat' : {1},
                                                          'a':{3, 5},
                                                          (1,5):{4}
                                                         })
    assert(indexMap([]) == {})
```

Begin your FR2 answer here or on the next page:

Begin or continue your FR2 answer here

Continue your FR2 answer here