fullName: _____ andrewID: _____ section: ____

15-112 S26 Quiz5 Version A
Time: 30 Minutes

You must write your name on this paper and hand this back in immediately after the assessment. If we do not receive it immediately, you will receive a zero on the assessment. Do not unstaple any pages. All pages must be handed in intact.

Do not use your own scrap paper. You should not need it, but if you must absolutely have scrap paper, raise your hand and we will provide some. Write your andrewID clearly on it and hand it in with your quiz. We will not grade anything on scrap paper.

You may not ask questions during the quiz, except for English-language clarification questions. If you are unsure about a problem, take your best guess.

Before and during the quiz, you may not view any other notes, prior work, websites or resources, including any form of AI. You may not use calculators, phones, laptops, or any other devices. You may not communicate with anyone else except for current 112 TAs or faculty during the assessment. All syllabus policies apply.

You may not discuss this test with anyone else, even briefly, in any form, until we have released grades. Failure to abide by these rules may result in an academic integrity violation.

Do not use material we have not yet covered. Thus, do not use sets, dictionaries, OOP, or recursion.

Do not open this or look inside (even briefly) before you are ready to begin. Do not spend more than the specified time noted above on this assessment.

True/False [5 pts, 1 pt each]

Indicate True or False for each of the following. Indicate your answer by filling in the dot. Assume L and M are lists of ints, and T is a tuple of ints.

A) `L += M` is the same as `L.append(M)`.

◯ True

◯ False

B) If `(L is M)` is True, then `(L == M)` must be True.

◯ True

◯ False

C) If `(max(L) < min(M))`, then `(sorted(L)[-1] > sorted(M)[0])` must be True.

◯ True

◯ False

D) Since M contains ints, if M is non-empty then `''.join(M)` will crash.

◯ True

◯ False

E) If `(list(T) == L)`, then `(tuple(L) == T)` must be True.

◯ True

◯ False

Fill in the Blank [5 pts]

Fill in the blank in g(n) so that f(n) == g(n) for all integer values of n:

```
def f(n):
    if n < 0: return None
    L = [ ]
    for v in range(n):
        M = list(range(v))
        L.extend(M)
    return L


def g(n):
    if n < 0: return None
    L = [ ]
    for i in range(n):
        for j in range(i):

            ____________________________________________________________

    return L
```

Code Tracing [30 pts, 7.5 pts each]

For each of the following, indicate what the code prints. Place your answer (and nothing else) in the box beside or below each block of code. If a line of code crashes, just print "crash" (without quotes) and stop the CT at that point.

CT1: [7.5 pts]

```
def ct1(v):
    v *= 2
    v = v * 2
    return v

v = (1,)
print(ct1(v))
print(v)

v = [2]
print(ct1(v))
print(v)
```

CT2: [7.5 pts]

```
def ct2(L):
    M = [n*2 for n in L]
    print(M)
    N = L[1:3]
    N.append(N.sort())
    print(N)
    return '-'.join([str(v) for v in L])

print(ct2(['A', 4, 3, 'B']))
```

CT3: [7.5 pts]

```
import copy

def ct3(L):
    M = L
    N = copy.copy(L)
    N.insert(0, L.pop(-1) ** 2)
    M.extend([v * 10 for v in N[:2]])
    M = [L.pop(1)] + M
    L.append(0)
    print(L)
    print(M)
    return N

print(ct3([1, 2]))
```

CT4: [7.5 pts]

```
def ct4(L):
    print(L.index(L.count(sum(L[1:3]))))
    for i in range(len(L)):
        if L[i] % 2 == 0:
            L.pop(i)
    return len(L)

print(ct4([3,3,1,4,2,3,4]))
```

FR1: adjustMean(L) [30 pts]

Background: the "mean" of a list L of numbers (integers or floats) is simply the average of those values. Also, given such a list, and given a target mean, we can add one more value to the list so that its mean matches the target mean.

For example, if L = [1, 0, 5] and the target mean is 3, we can add the value 6 to the back of the list to get [1, 0, 5, 6], which indeed has a mean of 3.

Further, if the list L already has the target mean, we would not add any value to it. So if L = [1, 0, 5] and the target mean is 2, we do not add any values, since L already has a mean of 2.

Finally, if the value we add is negative, we will add it to the front of the list rather than the back. For example, if L = [1, 0, 5] and the target mean is 1, we would add -2 to the front of L to get [-2, 1, 0, 5], which has a mean of 1.

With this in mind, write the non-mutating function adjustMean(L, targetMean) that takes a list L of numbers and a numeric target mean, and follows the rules described above to return a new list that contains all the original values in L but also perhaps one more value so that the new list has the given target mean.

Note: since results can include floats, we cannot use == to compare L to the result of adjustMean(L). Instead, we will use almostEqualLists which returns True if two lists have the same number of values and all those values are themselves almostEqual.

For example:

```
assert(almostEqualLists(adjustMean([1, 0, 5], 3), [1, 0, 5, 6]))
assert(almostEqualLists(adjustMean([1, 0, 5], 2), [1, 0, 5]))
assert(almostEqualLists(adjustMean([1, 0, 5], 1), [-2, 1, 0, 5]))

# This must also work with floats:
assert(almostEqualLists(adjustMean([0.5], 1.5), [0.5, 2.5]))

# Also, consider empty lists:
assert(almostEqualLists(adjustMean([ ], 42), [42]))
```

```
    # Also, be sure to be non-mutating:
    L = [1]
    assert(almostEqualLists(adjustMean(L, 2), [1,3]))
    assert(L == [1]) # this checks for non-mutating
```

Write your solution here:

FR2: sortInts(L) [30 pts]

Write the mutating function sortInts(L) that takes a list L and mutates L so that all the ints in L occur in sorted order, while leaving the non-ints in L in their original position. As with most mutating functions, this function returns None.

For example:

```
L = [42, 'B', 3, 'A', 1, 0]
assert(sortInts(L) == None)
assert(L == [0, 'B', 1, 'A', 3, 42])
```

Also, note that floats do not get sorted, only ints:

```
L = [4, 99.9, 1]
assert(sortInts(L) == None)
assert(L == [1, 99.9, 4])
```

Write your solution on the next page:

BonusCT [2 pts, 1 pt each]

These CTs are optional, and intended to be very challenging. They are worth very few points. Indicate what the following code prints. Place your answers (and nothing else) in the boxes to the right of each code block. If a line of code crashes, just print "crash" (without quotes) and stop the CT at that point.

bonusCT1: [1pt]

```
def bonusCt1(n):
    M = [int(v*2) for v in str(n*2)*2]
    return sum([(M[i]%M[-i-1]) for i in range(len(M)//2)])
print(bonusCt1(642))
```

bonusCT2: [1pt]

```
import math

def bonusCt2(L):
    def f(L, k): return sum([v // 10**k % 10 for v in L])
    return [f(L, k)
            for k in range(1 + math.floor(math.log10(max(L))))]
print(bonusCt2([1, 234, 5600, 78]))
```