

fullName: \_\_\_\_\_ andrewID: \_\_\_\_\_ section: \_\_\_\_\_

**15-112 F25 Practice Quiz3**  
**Time: 30 Minutes**

This is the Practice Quiz3, part of Prep6. In order to receive credit, you must simulate taking this as if it was an actual quiz, and you must write your name on this paper and hand this back in during lecture on 9/30 or prior to then online using [this form](#). This Practice Quiz is graded based on completion rather than correctness, but if we do not receive it at the prescribed time, or if you have not demonstrated apparent effort, you will receive a zero on the assessment.

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

Do not use sets, dictionaries, recursion, or anything else disallowed in the original problem.

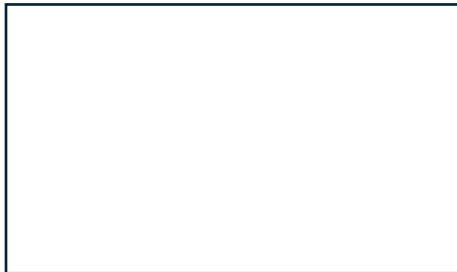
Do not open this or look inside (even briefly) before you are ready to begin. Do not spend more than 30 minutes on this assessment.

## Code Tracing

### CT1:

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

```
def ct1(L, M):  
    N = L  
    L += [1]  
    M += [2]  
    N = N + [3]  
    print(L)  
    print(M)  
    print(N)  
L = [4]  
ct1(L, L+[5])  
print(L)
```



**CT2:**

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

```
def ct2(L):
    M = sorted(L)
    N = list(reversed(M))
    n = L.pop(0) * 10
    L.remove(1)
    L.sort()
    L.append(n)
    L.extend(L[:1])
    return (L.index(3), M.index(3), N.index(3))
L = [1,2,3,2,1]
M = ct2(L)
print(L, M, type(M) == list)
```

**CT3:**

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

```
def ct3(s):
    L = [int(v) for v in s.split(',')]
    M = [ L > L[:-1],
          sum(L) - max(L),
          L[-1:]*L[-1]
        ]
    N = [str(v) for v in M]
    return '-'.join(N)
print(ct3('1,23,4'))
```

## Fill in the Blank (FitB)

### FitB1: prepend

You are given the function `prepend(L, M)` that solves the following problem, only with two parts removed. You need to fill in the blanks with the missing code.

Note: when you fill in blanks, you must not add any newlines and you must not use any semicolons.

Also note: there is an excellent chance that quiz3 will have one or more FitB problems on it.

With that, here is the writeup for `prepend`:

Write the function `prepend(L, M)` which mutates `L` (without mutating `M`) by adding all the values of `M` onto the front of `L`, and returns `None`.

For example:

```
L = [1, 2]
M = [3, 4]
assert(prepend(L, M) == None)
assert((L == [3,4,1,2]) and (M == [3,4]))
```

Here is the FitB solution (fill in the two missing blanks):

```
def prepend(L, M):
    for v in _____(M): # <- BLANK #1

        L. _____(0, v) # <- BLANK #2
```

## FitB2: mutatingSwapOdds

You are given the function mutatingSwapOdds (L) that solves the following problem, only with two parts removed. You need to fill in the blanks with the missing code.

Note: when you fill in blanks, you must not add any newlines and you must not use any semicolons.

Also note: there is an excellent chance that quiz3 will have one or more FitB problems on it.

With that, here is the writeup for mutatingSwapOdds:

Write the function mutatingSwapOdds(L) that takes a list L of ints and swaps each consecutive odd value in L. For example:

```
L = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

The first odd is 1 and the next odd is 3, so these are swapped. After that, the next odd is 5 and the next odd after that is 7, so these are swapped. After that, the next odd is 9, but there is no odd after that, so the 9 remains in the same position. The evens all remain in the same position.

Thus:

```
L = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
assert(mutatingSwapOdds(L) == None)
assert(L == [3, 2, 1, 4, 7, 6, 5, 8, 9, 10])
```

Here is the FitB solution (fill in the two missing blanks):

```
def mutatingSwapOdds(L):
    prevOddIndex = None
    for i in range(len(L)):
        if L[i] % 2 == 1:
            if prevOddIndex == None:
                _____ # <- BLANK #1
            else:
                _____ # <- BLANK #2

            prevOddIndex = None
```

## FR: lookAndSay + inverseLookAndSay

Note: this is taken directly from CS Academy 4.3.3.

Background: When "reading off" the values in a list using the look-and-say method, we describe how many consecutive occurrences of each value there are in the list, in order.

For example, when reading off [3, 3, 8, -10, -10, -10, 3] using the look-and-say method, in order from left-to-right the list contains two 3's, then one 8, then three -10's, then one 3. That is:

Count	Value
2	3
1	8
3	-10
1	3

We can rewrite this as a list of tuples, like so:

`[(2, 3), (1, 8), (3, -10), (1, 3)]`.

This is the look-and-say representation of the original list.

With this in mind, write the function `lookAndSay(L)`, which takes a list of integers `L`, and returns the list of (count, value) pairs constructed by reading `L` using the look-and-say method. Thus,

```
lookAndSay([3, 3, 8, -10, -10, -10, 3]) ==  
[(2, 3), (1, 8), (3, -10), (1, 3)]
```

Also write the function `inverseLookAndSay(L)` which does the inverse of the previous function. It takes a list of tuples representing the (count, value) pairs, and returns the original list. For example,

```
inverseLookAndSay([(2, 3), (1, 8), (3, -10), (1, 3)]) ==  
[3, 3, 8, -10, -10, -10, 3]
```

**Write your solution on the following pages.**



