15-110 Check3 - Written Portion

Name:

AndrewID:

#1 - Tracing Lists - 5pts

Trace the code below, then fill in the table with what it prints (one row per line).
You might not need to use all of the rows.

```
myList = [ "a", "b", 1, 2, 3, 2, 4, 6, 10, 20, 30 ]
half = len(myList) // 2
for i in range(half):
    print(myList[i], myList[i + half])
```


#2 - Aliasing and Mutability - 20pts

The following code creates and modifies lists. Determine each list's values after the code has run.

```python
a = [ "apple", "banana", "carrot", "donut" ]
b = a
b.remove("apple")
c = a + [ "eclair" ]
d = c[1:]
d.insert(2, "fig")
```

Variable	List Values
a	
b	
c	
d	

Select all of the pairs of lists that are **aliased** at the end of the code.

☐ **a** and **b**
☐ **a** and **c**
☐ **a** and **d**
☐ **b** and **c**
☐ **b** and **d**
☐ **c** and **d**
☐ None of the lists are aliased

#3 - Base Cases and Recursive Cases - 15pts

Assume you want to write a function `recursiveSum` that takes a positive integer, n, and **recursively** computes the sum from one to n.

For example, the result when calling the function on n=5 is 5+4+3+2+1 = 15.

What condition do you need to check for your **base case**?

What do you return in the **base case**?

What is the recursive call on a smaller problem in the **recursive case**?

How do you use the recursive call's result to solve the whole problem for n in the **recursive case**?

15-110 Check3 - Programming Portion

Each of these problems should be solved in the starter file available on the course website. Submit your code to the Gradescope assignment Check3 - Programming for autograding.

All programming problems may also be checked by running the starter file, which calls the function `testAll()` to run test cases on all programs.

#1 - `interleave(lst1, lst2)` - 15pts

Write a non-destructive function `interleave(lst1, lst2)` which takes two lists and returns a **new** list that contains the elements of the two lists, interleaved in the order they originally appeared. You may assume the lists will be the same length.

For example, `interleave(["a", "b", "c"], [1, 2, 3])` would produce `["a", 1, "b", 2, "c", 3]`.

#2 - `onlyOdds(lst)` - 15pts

Write a **non-destructive** function `onlyOdds(lst)` that takes a list of integers and returns a **new** list containing only the odd elements of `lst`. Note that this should not return the odd indexes- it should return the odd **elements**!

For example, `onlyOdds([1, 2, 3, 4, 5, 6])` returns `[1, 3, 5]`, and `onlyOdds([4, 1, 70, 35, -9])` returns `[1, 35, -9]`

#3 - `removeEvens(lst)` - 15pts

Write a **destructive** function `removeEvens(lst)` that takes a list of integers and destructively removes the even elements of the provided list so that it contains only the original odd elements at the end of the function. This function should not return anything; we'll instead test it by checking whether the input list was modified properly.

For example, `removeEvens([1, 2, 3, 4, 5, 6])` modifies the list to be `[1, 3, 5]`, while `removeEvens([4, 1, 70, 35, -9])` modifies the list to be `[1, 35, -9]`.

Hint: this is tricky because `lst` will change as the function runs. You should use an appropriate loop to account for this - see the course slides! Also, make sure to check for aliasing issues.

#4 - `recursiveReverse(lst)` - 15pts

Write a function `recursiveReverse(lst)` that takes a list as input and returns a **new** list which has the same elements, but in reverse order. This function must use **recursion** in a meaningful way; a solution that uses a loop, built-in reverse functions, or a slice with a negative step will receive no points.

For example, `recursiveReverse([1, 2, 3])` should return `[3, 2, 1]`.