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 - `onlyOdds(lst)` - 10pts

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]`

#2 - `removeEvens(lst)` - 10pts

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 return `None` instead of the list; we'll 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.

#3 - `recursiveReverse(lst)` - 10pts

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]`.

#4 - `recursiveCount(lst, item)` - 10pts

Write a function `recursiveCount(lst, item)` that takes a list and a value as input and returns a count of the number of times that item occurs in the list. This function must use **recursion** in a meaningful way; a solution that uses a loop or built-in count functions will receive no points.

For example, `recursiveCount([2, 4, 6, 8, 10], 6)` returns `1`,
`recursiveCount([4, 4, 8, 4], 4)` returns `3`, and
`recursiveCount([1, 2, 3, 4], 5)` returns `0`.