15-110 Check3 - Written Portion

Name:

AndrewID:

Complete the following problems in the fillable PDF, or print out the PDF, write your answers by hand, and scan the results.

When you are finished, upload your check3.pdf to **Check3 - Written** on Gradescope, and upload your check3.py file to **Check3 - Programming** on Gradescope. Make sure to check the autograder feedback after you submit!

Written Problems

#1 - Tracing Lists - 5pts

Can attempt after Lists and Methods lecture

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

Can attempt after References and Memory lecture

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

```
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 - 16pts

Can attempt after Recursion lecture

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**?

#4 - Binary Search - 14pts

Can attempt after Recursion II & Search Algorithms lecture

In the following table, write out the recursive calls that our implementation of `binarySearch` from lecture would make while searching the given list for the given item. Make sure to write out the **function call**, not the result. You might not need to use all the rows.

For example, if you were to trace `binarySearch([1, 2, 3, 4, 5], 1)`, you'd get:
Recursive Call 1: `binarySearch([1, 2], 1)`
Recursive Call 2: `binarySearch([1], 1)`

Q1: Search for 5

Original Call	`binarySearch([3, 5, 6, 7, 9, 11, 11, 15, 15, 19], 5)`
Recursive Call 1	
Recursive Call 2	
Recursive Call 3	
Recursive Call 4	
Recursive Call 5	

Q2: Search for 14

Original Call	`binarySearch([3, 5, 6, 7, 9, 11, 11, 15, 15, 19], 14)`
Recursive Call 1	
Recursive Call 2	
Recursive Call 3	
Recursive Call 4	
Recursive Call 5	

Programming Problems

For each of these problems (unless otherwise specified), write the needed code directly in the Python file, in the corresponding function definition.

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

#1 - Hw2 Code Review - 5pts

It isn't always good enough just to write code that works. It's also important to write code that is **clear** and **robust** - easy to understand and ready to handle a variety of inputs.

To help you learn how to write good code, we will have up to three **code reviews** this semester, where you will meet with a course TA to go over the code you wrote for a previous assignment. The TA will point out things you're doing well and areas where your code can be cleaner (even if it works already!).

To receive five points for the Hw2 code review, sign up for and attend a code review session with a TA the weekend of 10/01 - 10/03. We'll release more details about how to sign up for and attend these sessions via Piazza.

#2 - `sumAnglesAsDegrees(angles)` - 10pts

Can attempt after Lists and Methods lecture

When analyzing data, you need to convert the data from one format to another before processing it. For example, you might have a dataset where angles were measured in radians, yet you want to find the sum of the angles in degrees.

Write the function `sumAnglesAsDegrees(angles)` which takes a list of angles in radians (floats) and returns the sum of those angles **in degrees** (an integer). To do this, you will need to change each angle from radians to degrees before adding it to the sum. You can do this with the library function `math.degrees()`. Make sure to round the final result to get an integer answer.

For example, `sumAnglesAsDegrees([math.pi/6, math.pi/4, math.pi])` should convert the radians to approximately `30.0`, `45.0`, and `180.0`, then return `255`.

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

Can attempt after References and Memory lecture

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]`

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

Can attempt after References and Memory lecture

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.

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

Can attempt after Recursion lecture

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]`.