

15-110 S26 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. Also complete the programming problems in the starter file check3.py from the course website.

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 - 10pts](#)

[#2 - Aliasing and Mutability - 25pts](#)

[#3 - Base Cases and Recursive Cases - 20pts](#)

Programming Problems

[#1 - sumAnglesAsDegrees\(angles\) - 10pts](#)

[#2 - Destructive and Non-Destructive Functions - 20pts](#)

[#3 - recursiveStringToList\(s\) - 15pts](#)

Written Problems

#1 - Tracing Lists - 10pts

Can attempt after Lists and Methods lecture

Consider the following code:

```
scores = [ 53, 92, 10, 120, 81, 3, 78 ]
initials = [ "KAR", "FRX", "FOO", "SLH", "KAR", "FOO", "KAR" ]

name = "KAR"
result = [ ]

for i in range(len(initials)):
    if initials[i] == name:
        result = result + [ scores[i] ]
```

What will each of the following expressions evaluate to?

Expression	Value
initials[1]	
scores[2:len(scores)-2]	
max(scores)	
result	

#2 - Aliasing and Mutability - 25pts

Can attempt after References and Memory lecture

The following code creates and modifies lists. Determine what each variable holds after the code has run.

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

Variable	Value
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 - 20pts

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 $5+4+3+2+1 = 15$. Start with the general recursive form we discussed:

```
def recursiveSum(n):  
    if n == ??? : # A  
        return ____ # B  
    else:  
        smallerResult = recursiveSum(???) # C  
        return ____ # D
```

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

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

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

D: What do you return in the **recursive case**?

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 current script' on the starter file, which calls the function `testAll()` to run test cases on all programs.

#1 - `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 loop over the angles and 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.

Note: you are not allowed to use the built-in function `sum` for this problem. Use a loop instead!

#2 - Destructive and Non-Destructive Functions - 20pts

Can attempt after References and Memory lecture

First, write a **non-destructive** function `findMultiples(lst, num)` that takes a list of integers and a positive integer and returns a **new** list containing only the elements of `lst` that are also multiples of `num`.

For example, `findMultiples([11, 20, 35, 43, 50, 66], 5)` returns `[20, 35, 50]`, and `findMultiples([17, -77, 34, -95, 88], 11)` returns `[-77, 88]`.

Your `findMultiples` function must **not** modify the original list in any way.

Second, write a **destructive** function `removeNonMultiples(lst, num)` that does the same thing, but destructively. This function takes a list of integers and a positive integer and destructively removes the elements that are **not** multiples of `num` in the provided list.

In other words, at the end of the function call `lst` should contain only the original elements that are multiples of `num`. This function should return `None` instead of the list; we'll test it by checking whether the input list was modified properly.

For example, `removeNonMultiples([1, 2, 3, 4, 5, 6], 3)` returns `None` and modifies the list to be `[3, 6]`, and `removeNonMultiples([4, 5, 70, -3, 10], 2)` returns `None` and modifies the list to be `[4, 70, 10]`.

Hint: this is tricky because `lst` will change as the function runs. You should use an appropriate loop to account for this. See the 'Destructive Looping' portion at the end of the course slides for a big hint!

#3 - recursiveStringToList(lst) - 15pts

Can attempt after Recursion lecture

Write a function `recursiveStringToList(s)` that takes a string as input and returns a list which contains all the characters that were in `s`, but as separate items in the list. This function must use **recursion** in a meaningful way; a solution that uses a loop or the built-in `split` function will receive no points, and the function should not call `list()`.

For example, `recursiveStringToList("hello")` should return `["h", "e", "l", "l", "o"]`.

Hint: start from the framework in the Recursion slides! What's your base case, and how do you make the problem smaller? What should the function return, and how can you combine it with the leftover part?

Another Hint: make sure to keep your types straight! The **parameter** should always be a string, and the **returned value** should always be a list.