CS 115 Lab 3: Map Filter Reduce

Science is often a game of trying to extract one piece of information from a sea of data. Knowing how to process measurements is an important skill for analysis.

In this lab, you will implement several functions which take aggregate data and output different kinds of averages. You will do this using the higher-order functions **map**, **reduce**, and **filter** to implement them.

In the later tasks, you will also need to filter out invalid measurements.

Task 1: Mean

The mean of a set of numbers is their average value. More specifically, it is the sum of the set, divided by the number of values in that set.

Your goal in Task 1 is to use `reduce` to implement a `mean` function, which takes a tuple `data` as its parameter and returns the mean of its values. Note that doing this using the built-in `sum` and `len` functions is not too difficult.

However, for this problem, **you are not allowed to use** `sum`! So you have to find the sum of the values all on your own.

To do this, recall that the sum of values is given by

$$sum((a_0, a_1, \ldots, a_n)) = a_0 + a_1 + a_2 + \cdots + a_n.$$

Consider how you can use the tools available to you to solve this problem.

Hint: It might be helpful to review how `reduce` works. (If you need a refresher on `reduce`, see the end of this PDF.)

CA Check-In

After you complete the above task, talk to the CA in your section.

1. Show them your definition from Task 1.
2. Explain how reduce works, using Task 1 as an example

Task 2: Noisy Root Mean Square

The *Root Mean Square* of a set of numbers is another way of averaging them which gives higher value to larger numbers. It's most often used with errors, where we only care about the absolute value of the error, not the sign.

To find the root mean square, you square all the values, find their mean, and then take the square root.

For example, the root mean square of `(1,3,3,5,6)` is `4`. That is because

$$(1^2 + 3^2 + 3^2 + 5^2 + 6^2)/5 = (1 + 9 + 9 + 25 + 36)/5 = 80/5 = 16.$$

Then, taking the square root of the resulting 16 gives us `4`.

Noisy inputs: Sometimes, a machine also takes invalid measurements. This particular machine will output a `None` for invalid measurements. **For this task and task 3, your function should filter out measurements that are `None` and not include them in your calculation.** You can check if a value is `None` with the condition `is None`.

For example, if we have the list `my_list = [1,2,None,"a"]`, then `my_list[2] is None` would be `True`.

Hint: It might be helpful to review how `filter` works. (If you need a refresher on `filter`, see page 3 of this PDF.)

Your goal in Task 2 is use `map`, `filter` and `reduce` to implement the function `rms`, which takes a tuple `data` as its parameter and calculates the root mean square of its values.

It might also be handy to recall that $\sqrt{x} = x^{0.5}$, and that we can thus use `x ** 0.5` to take a square root. Consider how you might reuse functions you've written in the previous task to avoid having to rewrite a lot of code.

Hint: It might be helpful to review how `map` works. (If you need a refresher on `map`, see the bottom of page 2 of this PDF.)

Task 3: Noisy Harmonic Mean

The Harmonic Mean is the last average you have to implement in this lab. The harmonic mean of a set of numbers is the reciprocal of the mean of the reciprocals of the values.

In more simple terms, you invert every number in the set, take the mean, and then flip the number back. So the harmonic mean of `(1,3,3)` would be given by

$$\frac{1}{\left(\frac{\frac{1}{1}+\frac{1}{3}+\frac{1}{3}}{3}\right)} = \frac{3}{1+\frac{1}{3}+\frac{1}{3}} = \frac{9}{5}.$$

Your goal in Task 3 isuse `map`, `filter` and `reduce` to implement the function `hm`, which calculates the harmonic mean of its input tuple, `data`. Much like in Task 2, `data` may occasionally include at least one `None`, which we will consider an invalid input, meaning it should not be included in your calculation. (See **Noisy inputs** in Task 2 for more information).

For this problem, you can reuse a lot of your earlier work. You also can look back to the root mean square for inspiration on how to structure your approach for this problem.

Bonus Task: Length

This is an extra challenge, and is worth no points. However, if you are looking for something to test your skills, it is worth considering.

The `len` function returns the length of a given tuple. Your job is to implement a `my_len` function which does the same thing, using a `reduce`.

Hint: You will want to have an initial value, which should be the result you get when you call the function on an empty tuple.

Rubric

Category	**Description**	**Points**
Name	For stating your name on every file for submission.	1
Pledge	For writing the full Stevens Honor Pledge on every file for submission.	3
Collaboration	For listing your collaborators.	1
Check-In	CA Check-In during the lab.	15
Task 1: Mean	Correct implementation of `mean` using `reduce` (or another **functional** method)	20
Task 2: Root Mean Square (map)	Correct use of `map` in RMS.	15
Task 2: Root Mean Square (filter)	Correct use of `filter` in RMS.	15
Task 3: Harmonic Mean (map)	Correct implementation of Harmonic Mean using `map`	15
Task 3: Harmonic Mean (filter)	Correct implementation of Harmonic Mean using `map`	15
Total		100

Submission Instructions

For this lab, you must submit the following files, with exactly the following names:

- `lab3.py`
- `lab3_report.txt`

Remember to put your name and the Stevens Honor Pledge on every file you submit! Failure to put the pledge can result in marks being deducted, with repeat penalties potentially getting stricter between assignments.

Submissions must be handled through Gradescope (accessible through the homework page on Canvas). Grades will be released through Gradescope once the homework is graded.

We expect students to submit their labs to Gradescope before they leave lab so CAs can help with any submission issues.

Notes and Reminders

- **Remember to include docstrings!** If your submission does not have docstrings, or has docstrings that are not related to your code, you will not get credit for them.
- **Until we tell you otherwise, you are not allowed to use loops!** We want you to learn to use higher-order functions, so you are not allowed to use loops or recursion to solve these problems.
- **You are not allowed to use the built-in `sum` function for this assignment!** If you wish to find a sum, you have to manually compute it using a higher-order function.
- We have provided a test file for you to test your solutions. By running this test file, you should be able to verify that your files are named properly, and that your code works correctly.
- Test frequently! Try out many different cases to verify that your code runs. Our provided test file may not be comprehensive, and so you should ensure to test your code on many other cases as well.
- Code that doesn't compile (throws a `SyntaxError` on being run) will receive a 0 for the assignment! Make sure you submit code that can be run, even if it does not pass all the test cases.

Review of Map, Filter, and Reduce

This lab expects you to use the higher-order functions `map`, `filter`, and `reduce`. As such, it might be helpful to get a reminder of what these functions do.

First of all, note that these are all higher-order functions. That means that each of them takes in another function as an argument. In fact, all of them have the same pattern: `map`, `filter`, and `reduce` all take in two arguments, the first of which is a function and the latter of which is a tuple (technically an iterable, but we will only ever be using them with tuples). However, the form of these functions, and what the three do with them, varies quite significantly.

`map` expects to be given a function that transforms a single value into a single value. It then applies that function to each element of the tuple, getting a new tuple of the same length (but with new values). For example, the following code will output `(2,4,6,8,10)`, as it doubles every values in the original input tuple.

```
def double(n):
    return 2*n

print(tuple(map(double, (1,2,3,4,5))))
```

`filter` expects to be given a function that takes in a single value and outputs a single `bool`. It then applies this function to each element of the tuple, and keeps only those that return `True`. For example, the following code will output `(1,3,5)`, as those are the inputs on which the passed function outputs `True`.

```
def is_odd(n):
    return n % 2 == 1

print(tuple(filter(is_odd, (1,2,3,4,5))))
```

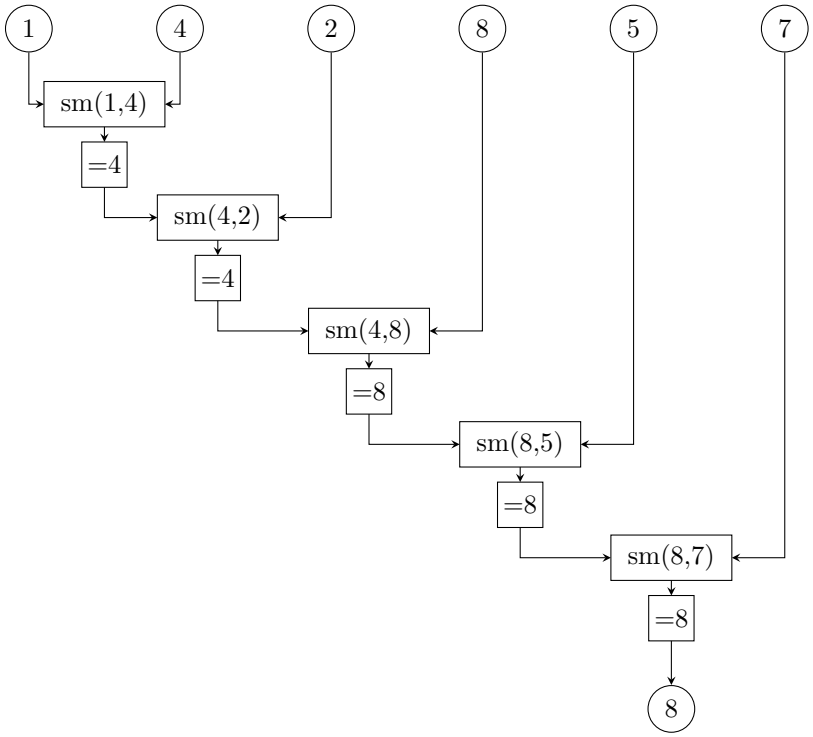
`reduce` expects to be given a function that takes in two values, and which outputs a single value. This function is then used to reduce the entire tuple to a single value, through repeated evaluation of the given function.

For example, consider the following code.

```
def small_max(a, b):
    if a >= b:
        return a
    else:
        return b

reduce(small_max, (1, 4, 2, 8, 5, 7))
```

This will begin by applying `small_max` to the first two elements, `1` and `4`, returning `4`. It then calls `small_max` on that returned `4` and `2`, returning `4`. It then calls `small_max` on that returned value and the next element `8`, and so on, eventually returning `8` overall. To help highlight this visually, the following diagram shows when we call the function, with the output always going down (we use `sm` in place of `small_max` for space reasons).



An equivalent way to think of a call to `reduce` is that it repeatedly updates an accumulator variable. For example, the above call can also be thought of as the following:

```
acc = 1
acc = small_max(acc, 4)
acc = small_max(acc, 2)
acc = small_max(acc, 8)
acc = small_max(acc, 5)
acc = small_max(acc, 7)
return acc
```

More generally, if you do `reduce(f, lst)`, it behaves equivalently to the following:

```
acc = lst[0]
acc = f(acc, lst[1])
```

```
acc = f(acc, lst[2])
acc = f(acc, lst[3])
...
acc = f(acc, lst[-2])
acc = f(acc, lst[-1])
return acc
```

We also have the initializer form of `reduce`. This is not necessary for this lab, but might be helpful for if you do the extra challenge.

Oh, and **don't forget to include `from functools import reduce` if you are using `reduce`!**