Before you begin

- Make a new project for this lab
- Grab `sensor.py` and `amp_filter.py`

Learning Objectives

You should be able to do the following:

- Use functions to decompose a problem in different ways
- Use lists to solve problems
- Create/use a generator function

Notes

Come back to these after you have read through the problems

- You should write tests for all of the functions you write for this lab. Write the tests before you write the code, and put them after the `if __name__ == '__main__':` line. We're going to import the functions from your files for testing and, if you don't put your tests after this line (and indented to the if block), then it might confuse our testing code. Also, this if good programming practice.
- Yes, we (more or less) wrote the sum functions in class last week.
- There are a number of ways to write the recursive functions. Any of them will do.
- There are likely to be built-in functions for a number of the things we're asking you to do for this lab. Please don't use them. If you do, you'll get zero points for that part of the assignment, since we want you to write them from scratch (this time).
- The amplitude filter only affects sensor readings beyond a certain threshold. These sensor readings get clipped to the maximum allowed amplitude.
- When you write the variable-width mean filter, you might find array slicing useful. Also, the filtered arrays will be shorter than the input arrays. For a filter with width 3, the filtered lists will be 2 elements shorter than the original list. For a width of 5, it will be 4 shorter, and so on.
- Make sure you write testing code for all of the functions that you write.

Grading Rubric

- [1 point] - Iterative sum function
- [1 point] - Recursive sum function
- [1 point] - Testing code for iterative and recursive sum functions
- [1 point] - Iterative reverse function
- [1 point] - Recursive reverse function
- [1 point] - Testing code for iterative and reverse functions
- [1 point] - Uses the functions in `amp_filter.py` and `sensor_filter.py`
- [1 point] - Generate two files, iwth data and filtered data
- [1 point] - Mean filter function calculates the correct value for width of 3
- [2 points] - Mean filter function calculates the correct value for user-specified widths

- [1 point] - Mean filter function checks for reasonable errors in parameters
- [2 point] - Code passes PEP8 checks with 10 errors max.
- [2 points] - Code passes TA-reviewed style checks for cleanliness, layout, readability, etc.

Problem 1

- Write a function called `sum_i` that takes a list of numbers as its argument, and returns the sum of all the numbers in the list. You should calculate the sum using a for loop. Do not use the built-in sum function (although that's the right thing to do outside of the context of this lab). Put this function in a file called `sum.py`. We should be able to call this function with `s = sum_i(l)`.
- Write another function in the sum.py file called `sum_r` that calculates the sum recursively. We should be able to call this function with `s = sum_r(l)`.
- Write some test code to test your two sum functions out.

Problem 2

- In a new file `reverse.py`, write two functions, one recursive and the other iterative, that take in a list and return a new list with the elements in reverse order. These functions should be called `reverse_r` and `reverse_i` respectively.
- Note that if you Google for how to reverse something, you will likely find some answers about list slicing like `[::-1]`. For this problem, we want you not to use this.
- Write some test code to test your two reverse functions out.

Input/Output Examples:

- `reverse_i(['h', 'e', 'l', 'l', 'o'])` returns `['o', 'l', 'l', 'e', 'h']`
- `reverse_r(['h', 'e', 'l', 'l', 'o'])` returns `['o', 'l', 'l', 'e', 'h']`
- `reverse_i([1, 2, 3, 4, 5])` returns `[5, 4, 3, 2, 1]`

Problem 3

- Download `sensor.py` and `amp_filter.py` from Canvas. Read the code in these files, and make sure you understand what they're doing. Do not modify these files; all of your code should go in `test_filter.py`.
- Write some code in `test_filter.py` to generate a single CSV file, where the first column has 1000 points of (simulated) sensor data (generated using the function in `sensor.py`), and the second column has the same data after you pass it through the amplitude filter (from `amp_filter.py`).
- Note that there are many modules in Python that work with CSVs. You can use any standard library modules you wish (but no Numpy/Pandas/etc.), but for this assignment, you may also find that writing each line manually is the easiest. Each line should look something like `1.2411,0.8921`, with a comma separating each number (hence the name comma separated values).

Problem 4

- Write a new filter, called `mean_filter` in `filters.py` that replaces each sensor reading with the mean of itself and the readings on either side of itself. For example, if you have these measurements in the middle of the list

  ```
  10 13 13
  ```

  then the middle reading would be replaced by 12 (since (10 + 13 + 13) / 3 = 12). Write code to generate two files, and plot the data in these files, showing the effect of applying your mean filter. We should be able to call your filter using `l2 = mean_filter(l)`.

- Modify your code to have a variable filter width. Instead of a width of 3, add a parameter to the filter function that specifies the filter width. This parameter must be positive and odd. Test this with a variety of widths, and see what it does to the data. We should be able to call your code with `l2 = mean_filter(l, 5)`. We should also be able to call the code with `l2 = mean_filter(l)`, in which case the width should default to 3.
- Note that for both parts 1 and 2, observations where the window "extends off" the list should not be included. E.g. if we have a list of length 8 and our window size is 3, then the resulting list should be length 6.
- Make sure your code throws a `ValueError` if the user passes in a window size which is not positive and odd.

Input/Output Example:
`mean_filter([1, 2, 3, 2, 1, 2, 3, 2, 1, 2, 3, 2, 1])`
returns `[2, 2.333, 2, 1.667, 2, 2.333, 2, 1.667, 2, 2.333, 2]`