Lab 3: Functions and unit testing

Labs are graded for participation rather than correctness. Keep all your lab code in your course GitHub repo to receive credit for your work. We'll be looking to see that you have at least partially completed all problems in each lab.

If you finish the lab assignment early, you may get started on the homework.

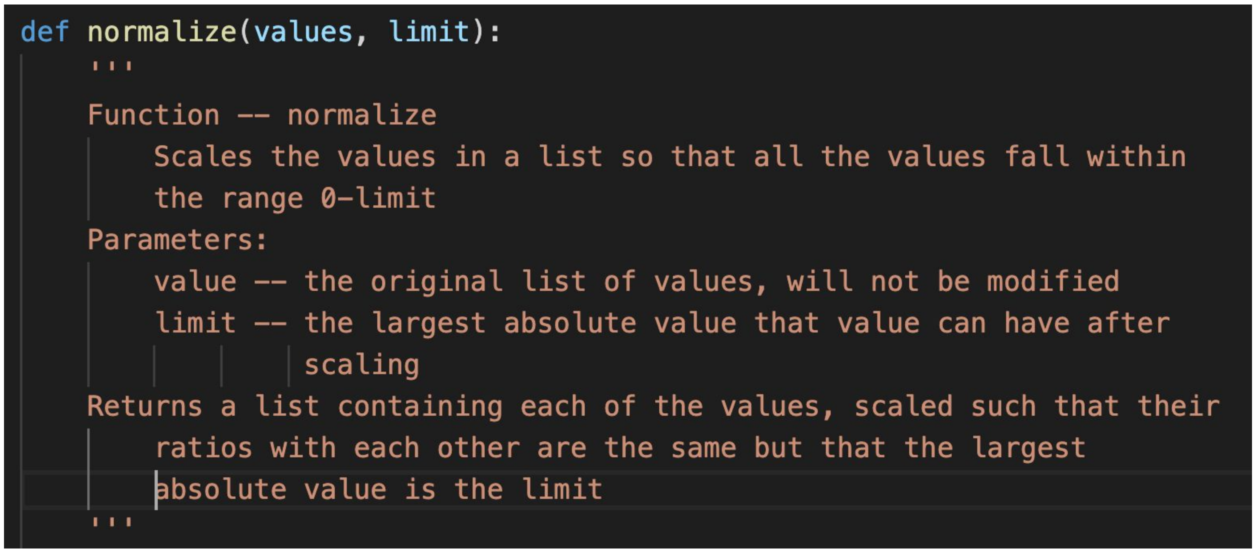
Getting started

In your local repo*, create a folder for Lab 3. Create new .py files for this week's lab problems and save them in the Lab 3 folder.

Documentation and format guidelines

The guidelines from previous labs still apply.

Functions (except `main()`) must be documented
The first line inside a function definition should be a *docstring*—a comment delimited by three single-quotes that explains how to use the function. The first line of the docstring should contain the word "Function" and the function's name. Indented under this heading is the summary—a short description of the function's purpose. After the summary, the docstring should explain the meaning of each parameter by name and the return value.



```
def normalize(values, limit):
    '''
    Function -- normalize
        Scales the values in a list so that all the values fall within
        the range 0-limit
    Parameters:
        value -- the original list of values, will not be modified
        limit -- the largest absolute value that value can have after
                 scaling
    Returns a list containing each of the values, scaled such that their
        ratios with each other are the same but that the largest
        absolute value is the limit
    '''
```

If you modify a function after writing the docstring, don't forget to update the docstring!

Define small functions that do only one thing
Smaller functions that do only one thing are easier to design, code, test, document, read, understand, and use than larger ones. When a function is getting too large (one rule of thumb is > 10 lines of code), find a way to factor a meaningful part of it out into a separate helper function.

Replace repeated non trivial expressions with equivalent functions
Do not duplicate code. It is often the case that we have code that needs to be called from multiple places. Rather than copying code from one place in your file and pasting it in another, move the code into its own function and call it from both places.

Problem 1: One function, one job

Now that we're a few weeks into the semester, your programs are starting to get more complex so it's time to think about writing clean, easy to read, and easy to extend programs. A key best practice in programming is that each function should have only one job. Take this opportunity to review some code you have written in a previous lab (e.g. lab 1, problem 1) and refactor it to follow the "one job per function" rule. Write functions to encapsulate any code that is not currently inside a function and break down larger functions into smaller, single-purpose functions. This means you'll typically have multiple functions per problem.

We've been using `main()` since the beginning of the semester but let's take a moment to think about why it's different from the other functions we write. `main()` is a special, user-defined function that is a required part of a program in many programming languages. It is not required in Python but it is strongly encouraged. Think of `main()` as the entry point to your program. The other functions that you write perform isolated calculations or operations and `main()` puts them together to achieve some larger goal. It should contain all the steps that your program needs to carry out, such as prompting for input, calling other functions to perform calculations or tasks, and printing feedback for the user. `main()` does not take any parameters and is a void function—it doesn't return anything. It is also the only exception to the unit testing rule—you do not need to write unit tests for `main()`.

Avoid prompting the user for input or printing to console from inside of any function other than `main()`. Refactor your code as needed to conform to this guideline using the following tips:

1. Instead of prompting for input inside a function, prompt for input inside `main()`, save the user's input as a variable, then pass that variable to your function as a parameter.
2. Instead of printing to the console inside a function, return the data to be printed, call your function from `main()`, save its output as a variable, then print the variable.
3. Are you printing to the console multiple times in a function? Split that function into multiple smaller functions—at least one for each `print()`.

Problem 2: Unit testing with Pytest

Unit testing is now required for every function that you write, with the exception of `main()`. If you have not already installed Pytest AND tried it out, please see the Testing with Pytest handout to get up to speed.

Write unit tests for all of the functions you wrote in problem 1 (except `main()`). Make sure all your tests pass!

Problem 3: Days until Friday

Design and write a program that will tell the user how many days there are until Friday. Your `main()` function should do the following:

1. Prompt the user to enter their name
2. Print a greeting, e.g. “Hello, <name>”
3. Prompt the user to enter the current day (“M”, “Tu”, “W”, “Th”, “F”, “Sa, “Su”)
4. Print a message stating the number of days until Friday.

You should have at least one additional function that performs the calculation of the number of days until Friday. Ideally, you should have multiple helper functions. For example, our solution has 3 functions, not including `main()`.

Write unit tests for all your functions (except `main()`). Make sure all your tests pass!