Week 2 Assignment: Better Numeric Data Processing

Overview

In this assignment, you will improve the numeric data processing script developed in Week 1 to be more flexible and sophisticated.

Preparation

This assignment uses the same data file as Week 1; copy it into the directory being used for this assignment.

Also, copy your program from Week 1 **compute_stats.py** and rename it **compute_stats2.py** in the folder for this assignment.

Note: The steps below do not need to be done in order, but do represent incremental changes that can be tested after each step.

Step 1: Modularize

Convert your program to a module. To do this, put all of the existing code into a function (for consistency, please use "main"):

```
def main():
```

At the bottom of the file, put in the necessary code to call the main function if the code is not being imported.

Now that it can be imported as a module without executing all the code, break out the core functionality of this program into a separate function that takes a list of values (already sorted) and returns the computed values as a tuple. E.g.:

```
def compute_stats(values):
    …
    return (o_min, o_max, o_avg, o_median)
```

Step 2: Unit Testing

Create a new program, **test_compute_stats2.py**, that contains the unit tests for the **compute_stats** function described above. This code should use the **unittest** module as described in lecture, import the **compute_stats2** module, and make calls to the **compute_stats2.compute_stats** function to test it. The core functionality should be tested with

a list containing an even number of elements and an odd number. Corner cases such as an empty list (should return None for all the values) and a list with a single element should also be tested.

Step 3: More Flexible Input Processing

To make this program more flexible and easy to use, it should be changed to accept the whole data file (not just a single column), and the data should not need to be sorted.

Change **compute_stats2.py** so that it takes a single command line argument that is an integer specifying which whitespace separated column it should process. For the user, column numbers start at 1.

Since the program is going to be processing the whole file rather than just a single column of numbers, we will use the **csv** module to parse it. Read the **csv.reader** documentation to figure out how to specify spaces as the delimiter in the file and how to skip over multiple delimiters in a row (if you don't do this, each individual space will be considered a delimiter, which is not going to work well).

Finally, change your program so that it does not require the input values to be sorted already (i.e., it should sort the values itself before passing to the **compute_stats** function).

After making these improvements, the script could be called as such:

```
python3 compute_stats2.py 9 < Data.txt
```

Step 4: Optional File Name

Change **compute_stats2.py** so that it takes an optional second command-line argument that specifies a file to use instead of reading from standard input. The existing functionality should work, but after this change the program can take a file name and read from that. E.g.:

```
python3 compute_stats2.py 9 Data.txt
```

Upload

Please combine **compute_stats2.py** and **test_compute_stats2.py** into a single ZIP file.