Possibilities of implementing a coverage tester:

- Huffman tree to correspond to the lines of code actually being covered by the test cases implemented.
- Lists to store the lines of code that are being covered and those that are not
- Using a queue or a stack to divide lines of code that are being covered and those that aren't

1. summary.py

   collects all the data and displays it in in terms of lines of code that were covered, and those that weren't. Lists the percentage of covered code and also includes a missing category that specifies which lines of code aren't getting tested. Uses a dictionary to store lines that are missing, covered, name of the line, and statements made. More includes using lists to store the lines of code being tested under coverage, and organizes based on strings that are being passed and values that include numeric solutions or calculations based on the program itself. Comments are well spaced out and describe the major roles of big functions, as well as helper functions.

2. data.py

   collects and implements something similar to a Huffman tree that can organize based on the frequency of how many times gets used for testing or coverage within the code, and can distinguish in which location or line the code is located. You can compare the sorted version of how many lines of code are supposed to be tested with how many lines of code are actually being tested with the test cases provided. The file contains almost one thousand lines of code, which could maybe be broken down into more helper function, or

import a separate file with more calculations in place. Some comments are too long as opposed to being in a more proper note formatting.

3. collector.py

   keeps track of how many times each line is tested, and doesn't count or repeat any lines over again that may cause an overestimation of accuracy of coverage. Also keeps track if test cases ran or not, or if valid and outputs information if there is an error. If test cases don't run, coverage doesn't properly display since lines of code remain untested. Uses graphing similar to project 5 where it runs through the input file and fills up on the lines that it lines up to test, then resets itself to allow to gather more data from input file to be stored and tested through.

4. results.py

   displays the results collected from summary, data, and collector on the compiler. Displays it in a graph format as well as strings to show which categories are completed based on a percentile as well as the different categories being tested and searched for. Also displays which lines need to be covered and estimates the overall average of how much is covered based on all the input files submitted into the console.