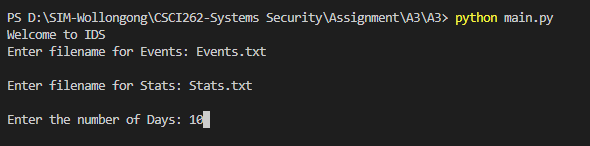
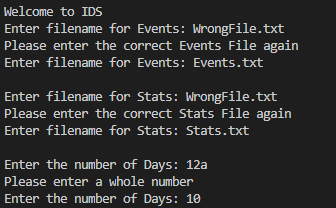
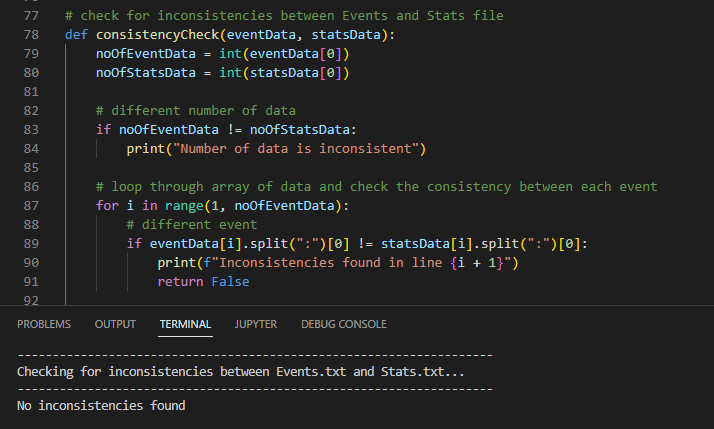
**Initial Input**

To make it clearer, I separated the user input into separated lines.

Some validations with regards to the three inputs above have been implemented.  
For both Events and Stats file, the program checks whether the file entered exists within the system’s directory. If not, the program will prompt the user to enter another file.  
For the number of days, the days entered must be an integer and must be more than 0.  


After reading the two files, the program will store the data in memory, in a list, in the following format:



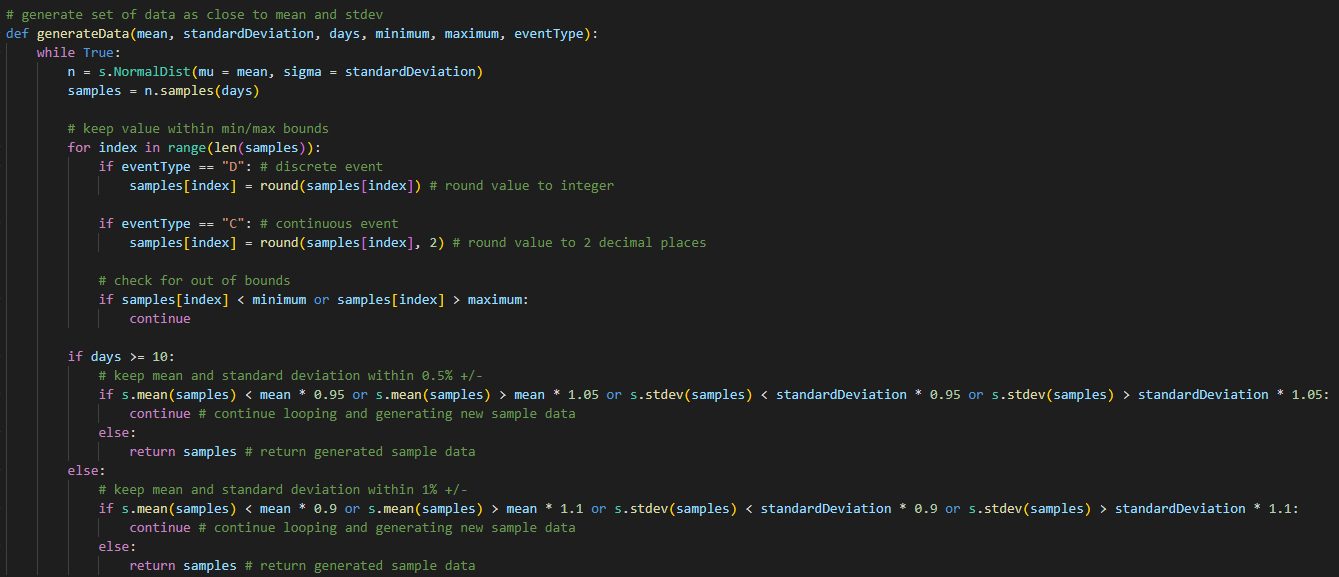
A simple initial consistency check between the number of events in each file is performed afterwards.  


If the files pass this check, the program will go on the process the data extracted from the events and stats file, while performing some validation along the way for example, if the event type is discrete, the min/max values must be an integer.

**Activity Simulation Engine and the Logs**

Activity Simulation

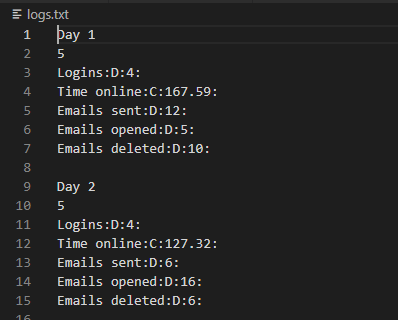
Program will generate data as close as possible to the mean and standard deviation specified in the Stats file. (please zoom in)



I made use of the statistics import from Python to access statistics.NormalDist method which takes in the mean and standard deviation as arguments to generate a set of data based on the number of days specified. Because the lesser the data, the harder it is to generate close approximates, so for days less than 10, I allow a 10% +/- deviation from the specified mean and standard deviations. For days more than 10, I allow a 5% +/- deviation instead.  
While the generated samples fall outside the stated deviation above, the program will continue generating new samples until the parameters have been met.

Logs File

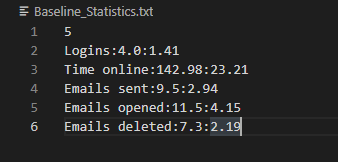
For the logs file, the first thing is the day number, followed by the number of events, and then the events itself. For each event, it includes the name of the event, the event type, and the total number for that specific event. After the last event, I included a new line to separate each day more clearly



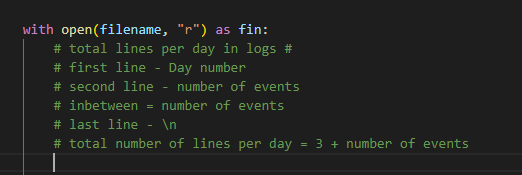
**Analysis Engine**

This part of the program will first read in the log file generated in the previous step, then returns the data extracted from the logs file and then output the data into another text file which I titled as “Baseline\_Statistics.txt”.

The format for the baseline statistics file generated follows the same format for the first Stats file that was read in during the ‘Initial Input’ part of the program.



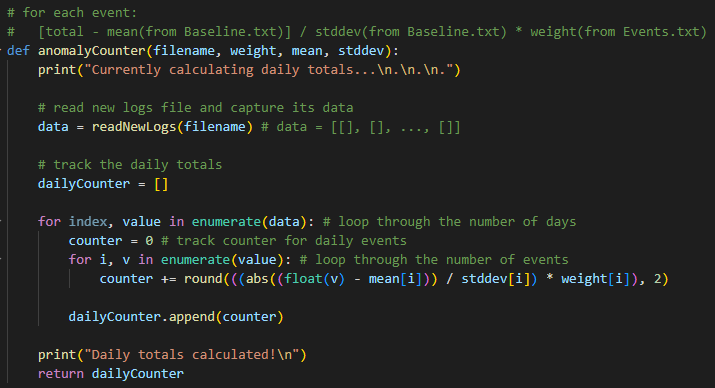
Because the format for the logs file is standardised, I can use the pattern to read and store the data based on the number of events.



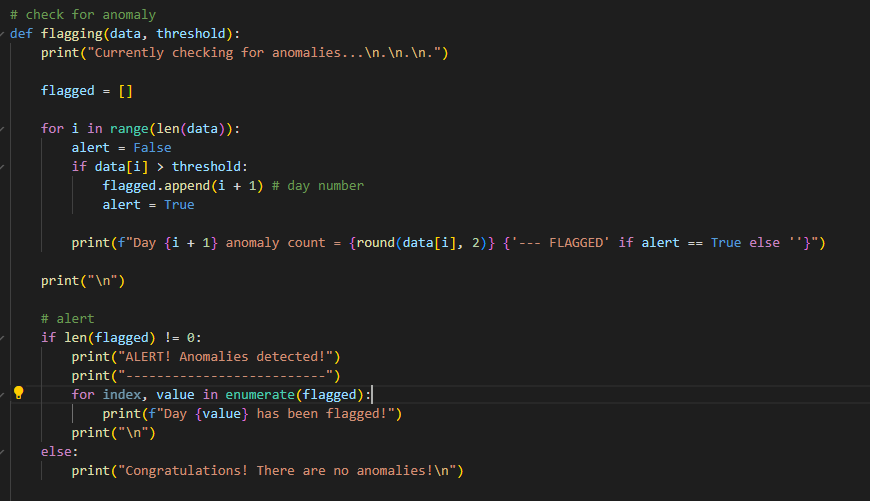
While the program attempts to output the relevant data to the baselines statistics file, it will calculate the mean, and standard deviation of the data extracted from the logs file. The mean and standard deviation will then be stored in memory for future use.

**Alert Engine**

The program will run this function to read the new logs file to get the daily totals and use the values to calculate the anomaly counter.



The program will then run the next function which is to put the anomaly counter up against the threshold to too whether it exceeds the threshold.



If the anomaly counter exceeds the threshold, the program will show the days flagged in the command line below:

