**Documentation of logParser.py**

Waleed Nasr

4/11/2022

TABLE OF CONTENTS

[EXECUTIVE SUMMARY 2](#_Toc100133737)

[BUSINESS SCENARIO 3](#_Toc100133738)

[PROCEDURES 4](#_Toc100133739)

[RESULTS 5](#_Toc100133740)

[CONCLUSIONS AND RECOMMENDATIONS 6](#_Toc100133741)

EXECUTIVE SUMMARY

An intern developer testing specialist has been asked to write a program that can both parse and consolidate information in the log file. The solution is explored to be able to handle this data as well as any other discrepancies that may exist in this log file, or future log files that we use your program to parse. The main sections of this report include the procedures, which explore the process, and conclusions sections, which explores suggestions.

# BUSINESS SCENARIO

Company x wants to develop a script that can parse through logs to find any irregularities such as DOS attacks. The script should be capable/adhere of doing the following:

* Handle messy, raw log files
* Ability to easily reused for other types of logs
* Outputs data for future use
* Analyze DOS attacks
* Must be written in any modern language (possibilities: Python, C, C#)

The required tools for the project are Visual Studio Code, GitHub, Regex. Please check appendix A for out put file samples. The following noted where taken during the meeting:

The log file consists of individual messages that are either being sent to the target or being received from the target. It is a requirement that the target responds within a certain timeframe in order to pass our tests. In other words, if an instrumentation attempt isn’t immediately followed by a response, a Denial of Service (DoS) failure is present. The way the DoS time is calculated is by taking the difference between the time that the original request was sent and when the target responded. Some examples from the log are shown below for verification.

Example:

Test case #5 – DoS time is 0:00:20.414

Test case #415 – DoS time is 0:00:20.405

Test case #1395 – DoS time is 0:00:06:070

# PROCEDURES

This section will cover in detail how logParser.py operates. Each section will discover a function in the script. Look at figure 1 for the general layout of logParser.py.

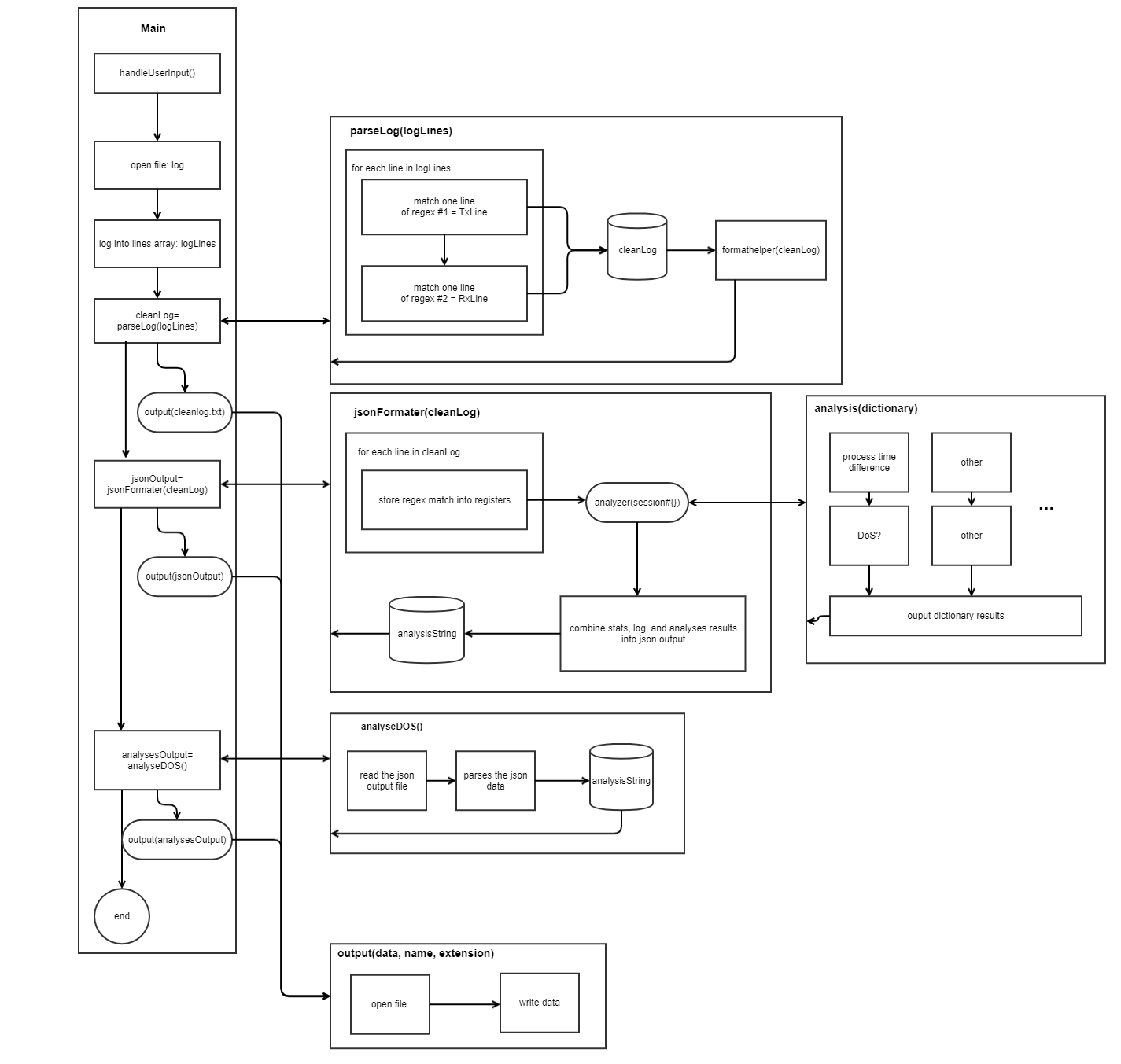


Figure 1. logParser.py

The program is divided into six main functions. The function main() calls each function in the order that is appropriate. Refer to table 1 for a short description of each function.

|  |  |
| --- | --- |
| **Function** | **Description** |
| parseLog | Will parse the raw log file, clean, and return a string with the help from *formatHelper()* |
| jsonFormater | Takes in a string, fetches data using *dataAnalysis(),* and then formatit to .json |
| dataAnalysis | Is responsible for dealing any kind of calculations, returns a dictionary |
| output | Handles all file writes. |
| formattingHelper | Reformates a list into a string |
| handleUserInput | Handles users input, including file selection. |

## Function parseLog(list)

This function takes in the output of the log file as a list. To filter out all of the unnecessary data, Regex will be used. For each line in the file, the function will alternate between the two regex matches (care for spaces):

regexMatchOne = r'^\d+-\d+-\d+ \d+:\d+:\d+\.\d+\s+\D+-\D+.+[0-9].+Tx.+1{6}'

regexMatchTwo = r'^\d+-\d+-\d+ \d+:\d+:\d+\.\d+\s+\D+-\D+.+[0-9].+Rx.+9{6}'

The function will only alternate if there is a successful match. If there is a successful match, the value will be stored in *cleanList.* To help with other Regex matches, the list is turned each stream into one line by using *formattingHelper*. Visit <https://regex101.com/> to have a visual representation of the match.

## Function jsonFormater(string)

This function takes the string input, feeds the filtered data into *dataAnalysis()* and then parses the data into a .json format output. To achieve this, Regex’s register functionality is used. Three capture groups are created using this Regex string:

(\d+-\d+-\d+ \d+:\d+:\d+\.\d+).+ ([0-9]+).+Tx.+1{6} (\d+-\d+-\d+ \d+:\d+:\d+\.\d+).+ [0-9]+.+Rx.+9{6}

The function then loops through each of the results, sends the required data to the function dataAnalysis() and then processes the results into a json formatted string. If future, changes is required, this is the place to change.

## Function dataAnalysis(\*arguments)

This function handles all the calculations, comparisons or any type of data processing. The use of \*arguments allow for a flexible structure. Since the output of the function can be dynamic, the output will be a list.

## Function output()

df

## Function formattingHelper()

Df

## Function handleUserInput()

sd

# RESULTS

# CONCLUSIONS AND RECOMMENDATIONS

# REFRENCES

SOURCES:

# https://pynative.com/python-regex-capturing-groups/

# https://regex101.com/

# https://www3.ntu.edu.sg/home/ehchua/programming/howto/Regexe.html

# https://code.visualstudio.com/docs/languages/json