**LOG PARSER**

**Detailed Level Design**

**DRAFT – Version 0.1**



|  |  |
| --- | --- |
| Version Number | 0.1 |
| Status | Draft |
| Document Date | 2024 |
| Owner |  |

**Contents**

1. INTRODUCTION

1.1 CONVENTIONS USED WITHIN THIS DOCUMENT

1.2 BACKGROUND

1.3 INTENDED AUDIENCE

1. APPROACH
2. EXECUTABLE SCRIPT
3. EXECUTION
4. SAMPLE OUTPUT SCREENS
5. SAMPLE FILES

**1 Introduction**

This Document will explain how the code is build and the flow of execution process associated to Log Parser

## **Conventions used within this document**

All the code present in this document is represented with the text style “CALIBRI”.

All the actual file names and table names are represented as *‘inclined with blue color’.*

## **Background**

This document will help in understanding the basic concepts of Python.

* 1. **Intended Audience**
* Trainees and Trainers
* Personnel responsible for the documentation, who may incorporate relevant aspects in to the user documentation.

|  |  |
| --- | --- |
| **Package** | **Version** |
| **python** | **3.6 +** |
| **re** | **2023.8.8** |
| **csv** | **0.0.1** |
| **collections** | **0.1.6** |
| **Pip** | **23.2.1** |

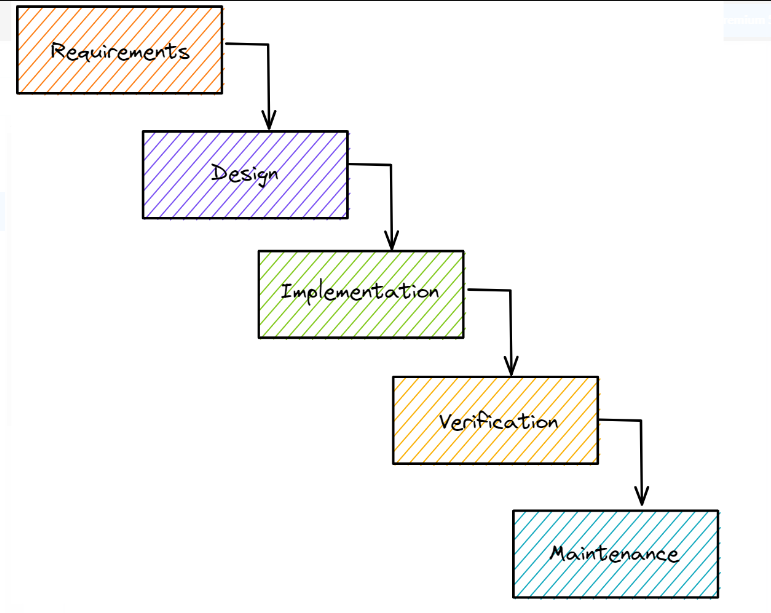
**SDLC: Software Development Life Cycle**

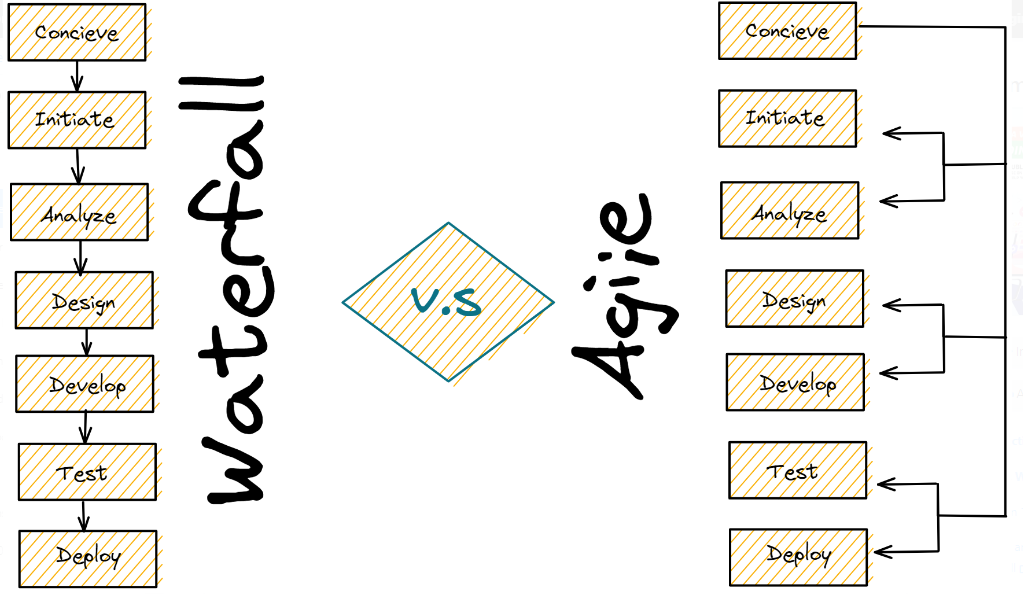
The **Waterfall methodology** was named after its sequential phases arranged in a downward fashion (similar to actual waterfalls), representing the various steps of software development from one end to the other. It was first mentioned in 1970 by the computer scientist Winston Walker Royce, and it originally contemplated five distinct phases:

**Requirements, design, implementation, verification, and maintenance.**

More specifically, the sequence of events in Waterfall looks something like this:

1. Gather and document requirements
2. Design
3. Code and unit test
4. Perform system testing
5. Perform user acceptance testing (UAT)
6. Fix any issues
7. Deliver the finished product





Agile software development is based on an **iterative**, incremental approach. Agile offers a more free and fluid approach with the ability to perform changes and iterations as they are needed.

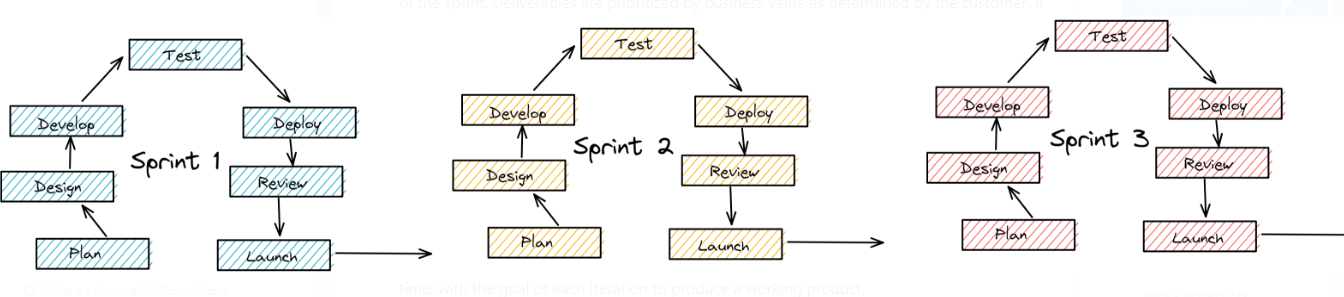
Requirements can change at almost any phase of the project and so, not as much planning is required before beginning a project.

Instead of creating tasks and schedules, all time is split into phases called “**sprints**.”

Each sprint has a defined duration (usually 2 weeks) with a list of deliverables, planned at the start of the sprint.

Deliverables are prioritized by business value as determined by the customer.

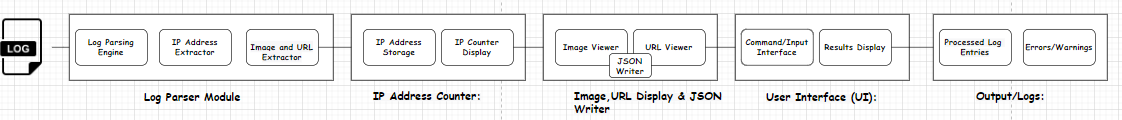
If all planned work for the sprint cannot be completed, work is reprioritized and the information is used for future sprint planning.



* Faster software development life cycle
* Predictable schedule in sprints
* Customer-focused approach, resulting in increased customer satisfaction
* Flexible in accepting changes
* Empowers teams to manage projects
* Promotes efficient communications
* Ideal for projects with non-fixed funding

**2 Approach**

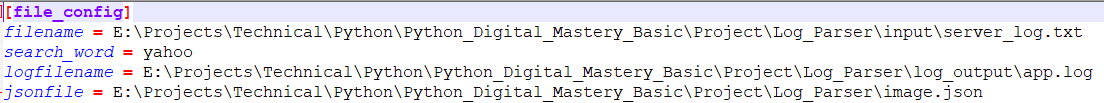
As the objective of this process is to understand the coding knowledge on python.



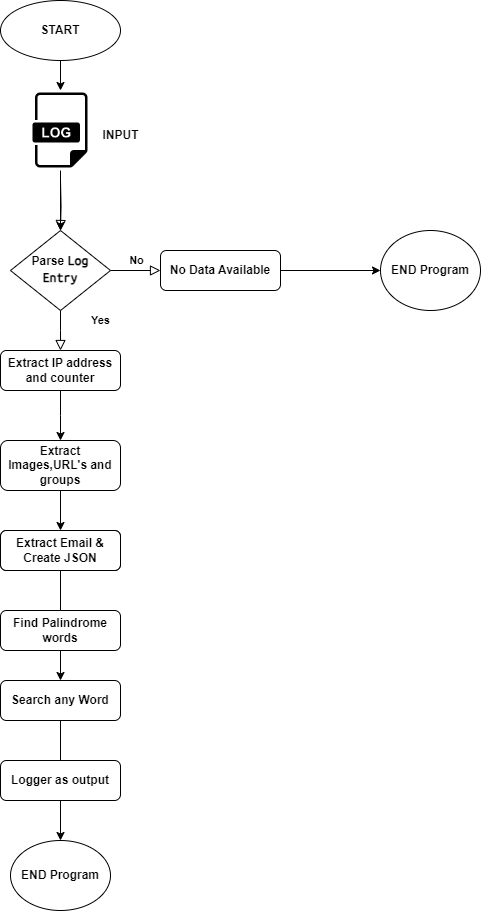
**Log Parser Block Diagram:**

1. **Input Log File:** The log parser starts by taking an input log file, which contains various log entries, each representing an event or activity with associated data like IP addresses, URLs, and images.
2. **Log Parser Module:**
   * **Log Parsing Engine:** This module reads and parses the log file line by line, extracting relevant information such as IP addresses, URLs, and images. It uses various parsing techniques to identify and separate different components of each log entry.
   * **IP Address Extractor:** Identifies and extracts IP addresses from each log entry. These IP addresses are then passed to the IP Address Counter.
   * **Image and URL Extractor:** Detects and extracts images and URLs from the log entries. These extracted images and URLs are forwarded to their respective display components.
3. **IP Address Counter:**
   * **IP Address Storage:** Stores the unique IP addresses encountered in the log file along with their corresponding counts.
   * **IP Counter Display:** Displays the list of unique IP addresses along with their occurrence counts.
4. **Image and URL Display:**
   * **Image Viewer:** Displays the extracted images from the log entries, providing a visual representation of the images found in the logs.
   * **URL Viewer:** Displays the extracted URLs, allowing you to see the web addresses that were recorded in the log entries.
5. **User Interface (UI):**
   * **Command/Input Interface:** Provides a way for users to input the log file's location and start the parsing process.
   * **Results Display:** Displays the output of the log parser, including the list of IP addresses with their counts, extracted images, and URLs and Palindrome words.
   * **Search mechanism:** To find the given search word.
   * **JSON:** JSON output with Images.
6. **Output/Logs:**
   * **Processed Log Entries:** Displays the processed log entries with relevant information extracted for each entry.
   * **Errors/Warnings:** Displays any errors or warnings encountered during the parsing process.

**Configurations:**



**Detail Design Flow:**

****

**Read log as an input**: Using file operations the input file has been processed line by line.

**Parse the input log:** Using regular expressions line by line of input file has been parsed, fetched, and filtered the required data.

**Extracting the ip addresses, image, URL, email etc...:**

Using regular expressions and list and string methods.

**Counter of ip addresses, image, URL, email etc…:**

**:** Using regular expressions and list and string methods.

Ex: { ‘ip1’: 1,’ ip2’: 3} {‘jpg’: 5, ‘png’, 3}

**Grouping: showing the count of ip, image, URL, email etc...:**

**:** Using regular expressions and list and string methods.

Ex: {

"**GIF**": "wpaper.gif, 5star2000.gif, 5star.gif, wpaper.gif,

"**JPG**": "a2hlogo.jpg, a3hlogo.jpg}

**JSON Writer**: Using JSON module we create below grouping files

**Image grouping** --> Image.json

**IP grouping** --> Ip.json

**Email grouping** --> email.json

7. Searching mechanism: Using for loop and membership operator we find

Normal words and using string, split method(s) we find palindromes words.

8. Display the output in console/log.



9. Rename or back up the log parser generated log file using os.rename module.

**3 Executable Scripts**

The main script (*log\_parser.py*) which will be executed by the end user by passing parameters as input to the script.

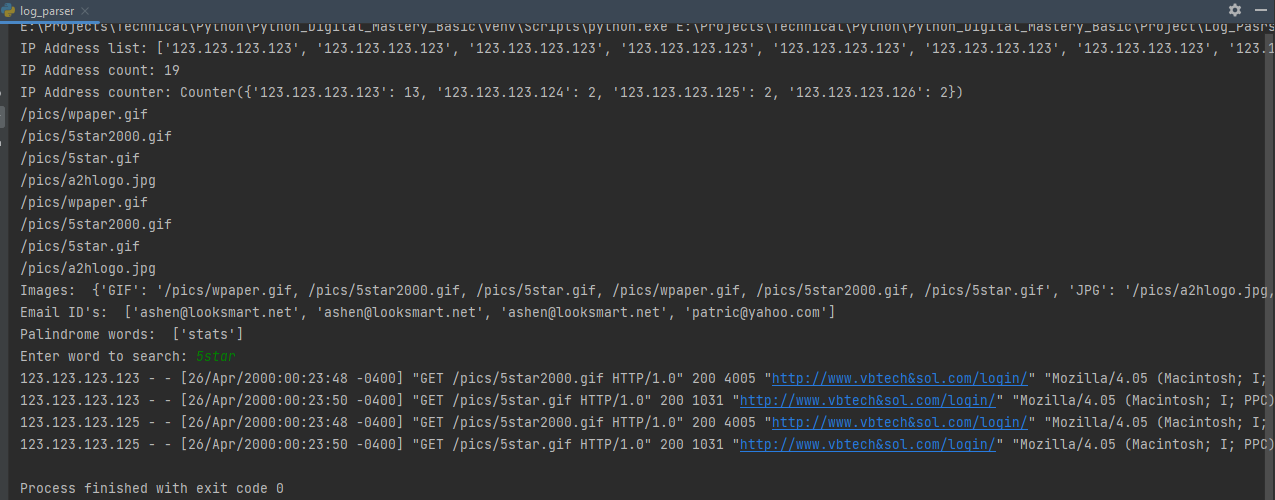
**USAGE:** logparser.py

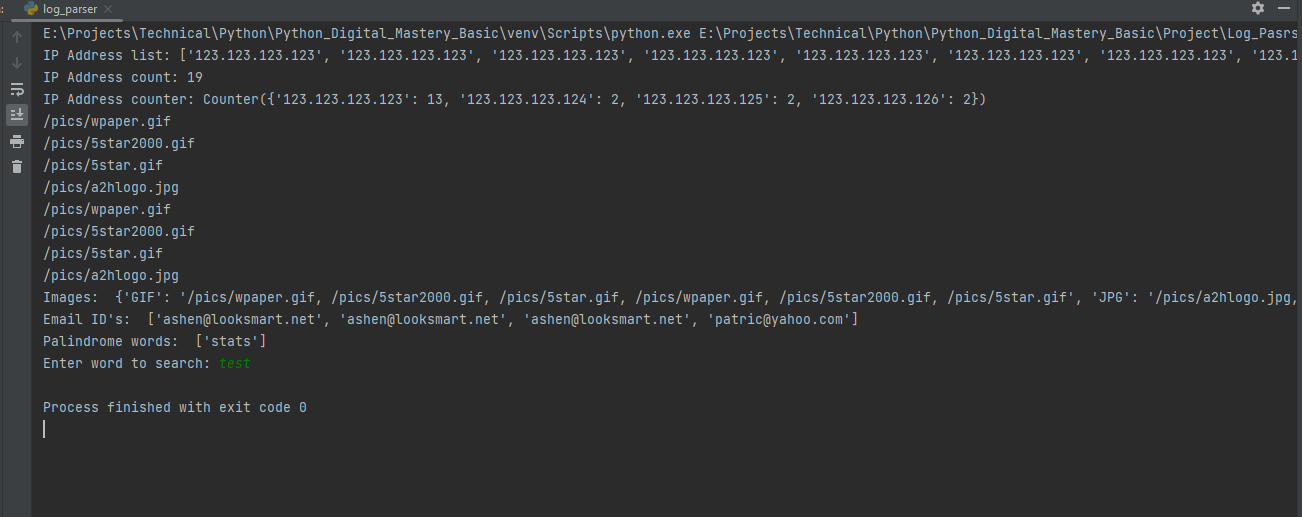
* While we execute the log\_parser.py , we get to understand on how to get all the IP addresses from the log file.
* The total counts of the IPs are shown in the output using return function.
* The total counter of the IPs using return function collections package.
* A function is written to understand whether the string is palindrome using the for loop and return function.
* A function is written to understand on how to check if the string exist in the line using the for loop
* A function is written to search the images and adding them to the dictionary using the if else condition.
* A function is written to search for the E-mail ID's using the find keyword.
* Usage of print statements is shown wherever required.
* A function is written to search for a word in the log file from the output.

**4 Executions**:

* Input: Server\_log file.
* File used : Log\_parser.py
* Command to execute is python log\_parser.py
* The points covered in the executable script are shown in the output one by one.
* Any word from the log file can be searched using this script after we see "Enter word to search:"

**5 Sample Output screen:**





**6. Sample Input log and output Files:**

