Expected solution steps:

- Implement a class called *Command* (or any other name) that includes functions for each of the following:
    - Grep
    - Categorize
    - Mv_last

  The project has a description of each one of the mentioned commands. For the grep command, you may also use <*directory*> instead of the <filename>.
  Factory design pattern shows how to use inheritance from a main class in python. It will be helpful in this class.

- Implement a class called *Parse* (or any other name) that will **import** previous class and its functions. This class will **read a text** file that has commands chosen from the three commands (grep, categorize, mv_last) **only**. An example of how this text file can look like is also mentioned at project description (script.txt).
  Start by using a fixed path for the script and output (log), after that you switch to option parser.

- Some commands/steps need configuration values (**like code constants**). These values will be specified into a JSON file. This file can be read as a dictionary in python, check: https://www.w3schools.com/python/python_json.asp
  This step will be the beginning of the Parse class – you load the values from JSON and get them into your code. Once you have the values, you can use them within your code.

- To run the code with arguments, we need to use what we call an option parser in python to add "**-s**" and "**-o**". Check the following to check how to add options to a python run command:
    - https://wiki.python.org/moin/OptParse
    - https://docs.python.org/3/library/optparse.html

- When you run the code, the text file will be read and **each line will be executed** and a result will be pushed into a python dictionary (you are free to use the output you want for each command – True/False for example). **If we have more commands than max_commands in JSON → execution stops.** Your dictionary may look like something like this – or any other format you choose:
  ```
  {
    "Line-1": "True",
    "Line-2": "False",
   }
  ```

- When execution is done, the code will do the following:
    - paste data in the format of logger statements from the above-mentioned dictionary to a new log file. Each run will create a new log file (output). Write a small function to create your new log, the function will take the dictionary and paste results to a file using logging statements:

    - https://realpython.com/python-logging/
    - The output can take the form of normal log or csv. JSON value "csv" decides. One way to create csv from python output: https://stackoverflow.com/questions/37912556/converting-a-log-file-into-a-csv-file-using-python
- If log files are more than the max log files number (value in JSON), you need to delete the **oldest one**.
- PASSED value in JSON decides if you need a separate directory for PASSED scripts or not. In case its True, create a separate log folder for PASSED scripts.