➔ **Log Level Counter**
Write a script that reads the provided log file and counts the number of occurrences of each log level (INFO, DEBUG, WARNING, ERROR) using file I/O and dictionaries. Avoid using regular expressions for this task.
Challenge: Ensure your code gracefully handles lines that do not follow the expected format.

➔ **Error Log Extractor**
Using basic string manipulation (without regex), write a script that filters and prints all log entries containing the word "ERROR".
Challenge: Verify that only well-formed error lines are extracted while ignoring misformatted lines.

➔ **Log Parser to Dictionary**
Develop a function that reads each line of the log file and parses it into a dictionary with keys: `timestamp`, `log_level`, `module` (if available), and `message`. Return a list of such dictionaries.
Challenge: Handle lines that deviate from the standard format by either skipping them or recording an error message.

➔ **Group Logs by Module**
Using the output from Question 3, write a function that groups log entries by their module. For each module, output the count of log entries.
Challenge: Use dictionary comprehensions or iterative methods to build the grouped result.

➔ **Format Validator with Exception Handling**
Create a script that reads the log file and identifies lines that do not match the expected log format. For each misformatted line, log a warning with its line number while continuing to process the rest of the file.
Challenge: Ensure that your solution uses try-except blocks to catch and handle exceptions without crashing.

➔ **JSON Log Converter**
Write a script that converts the log file into a JSON file. Each log entry should be a JSON object containing the keys: `timestamp`, `log_level`, `module`, and `message`.
Challenge: Ensure that the JSON output correctly represents all valid log entries while ignoring or flagging misformatted lines.

➔ **Command-Line Log Filter**
Develop a command-line tool using Python's `argparse` module that accepts two arguments: a log level (e.g., "ERROR") and a module name. The script should filter the log file to print only those entries matching both criteria.
Challenge: Validate the command-line inputs and provide helpful error messages for invalid or missing arguments.

➔ **Regular Expression Challenge (IP Address Extractor)**
Write a Python function that uses a regular expression to extract all unique IPv4 addresses from the log file. Some log messages include IP addresses (e.g., "Ping to server

192.168.1.100 succeeded"). Return a list of unique IP addresses found.
Challenge: Ensure your regex correctly matches typical IPv4 formats and does not capture invalid patterns.