PYTHON JSON LIBRARY

PRACTICAL MASTERY TEST

SECTION 1: JSON BASICS (FILES & STRINGS)

1. Load a JSON file named `sales.json` that contains a list of sales records and print the total number of records.
2. Read a JSON string containing employee data and extract all employee names into a Python list.
3. Write a Python dictionary containing daily revenue data to a JSON file using readable indentation.
4. Convert a nested Python dictionary into a compact JSON string (no extra spaces or indentation).
5. Read a JSON file and determine whether the top-level structure is a list or a dictionary.

SECTION 2: WORKING WITH NESTED JSON (CORE SKILL)

6. From a nested JSON structure containing customers → orders → items, extract:
   - customer_id
   - order_id
   - total number of items per order
7. Flatten a nested JSON so that each output record represents a single item.
8. Safely access a deeply nested key that may or may not exist without raising an exception.
9. Extract all unique product IDs from a deeply nested JSON dataset.
10. Count how many times each product appears across all orders.

SECTION 3: DATA CLEANING & VALIDATION

11. Validate whether a file contains valid JSON before loading it.
12. Handle JSON decoding errors gracefully and log the error without stopping execution.
13. Remove records from a JSON list where mandatory keys (`id`, `date`, `value`) are missing.
14. Replace all `null` values in a JSON dataset with default values based on the field name.
15. Detect and remove duplicate JSON objects using a unique key.

SECTION 4: JSON IN ANALYTICS PIPELINES

16. Convert a JSON file containing transaction data into a Pandas DataFrame.
17. Normalize nested JSON into a flat DataFrame suitable for analysis.
18. Filter JSON records where `status` equals `"SUCCESS"` and write them to a new JSON file.
19. Group JSON records by date and calculate daily totals.
20. Merge two JSON files using a common key (similar to a SQL JOIN).

SECTION 5: JSON FROM APIs (AUTOMATION SCENARIOS)

21. Parse a JSON API response and extract only required fields into a new structure.
22. Handle paginated JSON API responses and combine all pages into a single dataset.
23. Detect and standardize inconsistent field names across multiple API responses.
24. Convert timestamp fields in JSON into Python `datetime` objects.
25. Store API JSON responses using timestamped filenames for auditing purposes.

SECTION 6: JSON TRANSFORMATION & EXPORT

26. Transform JSON keys from `camelCase` to `snake_case`.
27. Rename specific keys and remove unwanted keys from a JSON dataset.
28. Convert a JSON file into CSV format using Python.
29. Split a large JSON file into multiple smaller JSON files based on date.
30. Create a summary JSON file containing aggregated metrics.

SECTION 7: PERFORMANCE & LARGE JSON FILES

31. Read a large JSON file using a streaming approach instead of loading it fully into memory.
32. Process a JSON Lines (`.jsonl`) file line by line.
33. Measure the time taken to load a large JSON file and optimize the process.
34. Write JSON output incrementally to avoid memory overflow.
35. Compress JSON output files using gzip.

SECTION 8: CONFIGURATION FILES & LOGGING

36. Load a JSON configuration file and dynamically apply values in a Python script.
37. Update a single configuration value inside a large JSON file.

38. Append execution metadata (run time, status, timestamp) to a JSON log file.
39. Maintain a rolling JSON log that stores only the last N executions.
40. Mask or obfuscate sensitive fields (API keys, emails, tokens) before saving JSON to disk.

SECTION 9: REAL DATA ANALYST SCENARIOS

41. Compare two JSON datasets and identify added, removed, and modified records.
42. Implement a JSON schema-like validator using pure Python logic.
43. Build a reusable function to safely extract nested JSON values.
44. Convert JSON data into SQL-ready INSERT statements.
45. Build an end-to-end JSON ETL pipeline:

- Read JSON
- Clean data
- Transform structure
- Save final output

SECTION 10: CAPSTONE TASKS (MANDATORY)

46. Build a reusable JSON ingestion module that includes:

- Validation
- Error handling
- Logging

47. Create a generic JSON flattener that works with unknown nested structures.
48. Design a JSON-based audit trail for an analytics pipeline.
49. Optimize an existing JSON processing script for better performance and lower memory usage.
50. Build a fully automated script that:

- Reads JSON from an API
- Cleans and transforms the data
- Saves analytics-ready output
- Logs execution metadata
