SharkNinja Coding Task: Number Analyzer

Aditya Kulkarni

# The Approach:

The approach taken in the provided code revolves around creating a flexible and extensible number analysis tool, the NumberAnalyzer class, which processes numbers within a specified range and categorizes them based on rules defined in a configuration file. The following steps summarize the approach:

1. **Initialization and Input Validation**: The constructor (\_\_init\_\_) ensures the start and end values are integers, with start being less than or equal to end. It also verifies the existence of the configuration file and its directory.
2. **Configuration Parsing**: The \_parse\_config method reads a JSON configuration file containing rules for categorizing numbers. Rules can include built-in checks (e.g., prime, even, odd) or custom rules defined as Python functions or lambda expressions. Assumptions made here include:
   * The configuration file is correctly formatted as JSON.
   * Any custom rules provided are syntactically valid Python expressions or functions.
3. **Rule Mapping**: Each rule in the configuration is mapped to a corresponding function. Built-in rules like "prime," "even," and "odd" are handled by predefined methods, while custom rules are dynamically evaluated or executed using eval or exec. This assumes that users provide safe and valid Python code for custom rules.
4. **Number Analysis**: The \_get\_results method iterates through the specified range of numbers, applying each rule to categorize numbers. Results are stored in two formats:
   * A list (self.results) containing categories for each number as strings.
   * A dictionary (self.results\_debug) mapping numbers to their categories for debugging purposes.
5. **Error Handling**: Custom exceptions (NumberAnalyzerException) are raised when invalid inputs, configuration issues, or syntax errors in rules occur.
6. **Output**: The print\_results method provides options to display results in a detailed (debug) or simplified format.

This approach emphasizes modularity, extensibility, and error handling while assuming that users provide valid inputs and configurations.

Details about the code can be found in README.pdf.

# Test analysis:

All test cases for the NumberAnalyzer class passed successfully, confirming the robustness and correctness of its implementation. The tests validated various aspects of the class, including initialization, configuration parsing, rule application, and error handling. Key highlights include:

**Configuration-Based Tests**

* Valid Initialization: Ensures that valid configuration files are loaded correctly.
* Invalid Configurations:
* Missing configuration directory or file.
* Syntax errors in rules or JSON structure.
* Custom Configurations: Verifies that configurations at custom paths or with full method definitions work as expected.

**Parameter Validation Tests**

* Ensures proper handling of invalid input parameters such as:
* Start greater than end.
* Non-integer start or end values.

**Functional Tests**

* Verifies that individual rules (prime, even, odd) work as expected.
* Ensures correct behavior for large ranges, negative ranges, and edge cases like single-number ranges.

**Output-Based Tests**

* Validates that results are printed in both detailed and simplified formats.

These results indicate that the NumberAnalyzer is well-designed to handle diverse scenarios while maintaining accuracy and reliability.

Sample Output:

A computer screen with text and images

Description automatically generated

The report for all testcases can be found in: NumberAnalyzer /report.htmlA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated