Documentation for Real-Time Grammar-Based Syntax Highlighter with GUI project

1)Language and Grammar Choice

- Language: Python-like syntax (simplified for demonstration).
- Grammar Rules:

  expression → term (OPERATOR term)*

  term → NUMBER | IDENTIFIER | '(' expression ')'

  - Justification:

Covers arithmetic/logical expressions.

Extensible for statements (e.g., if, for).

2. Lexical Analysis Details

- Approach: Regular expression-based scanner.
- Token Types:

```
    token_specs = [
         ('NUMBER', r'\d+(\.\d+)?'),
      ('OPERATOR', r'[+\-*/=<>!&|^%]'),
        # ... (other tokens)
]
```

  - Key Features:
    1. Position tracking for error reporting.
    2. Fallback UNKNOWN token for unclassified characters.

3. Syntax Analysis Process

- Algorithm: Recursive descent parsing.
- Steps:
  1. Tokenize input.
  2. Validate structure via parse_expression() and parse_term().
  3. Report mismatched parentheses or operators.
- Error Handling: Raises exceptions on invalid syntax.

4. Parsing Methodology

- Type: Top-down, predictive.
- Functions:

1. parse_expression(): Handles operator chaining.

2. parse_term(): Validates literals/variables/parentheses.

- Limitations: No operator precedence or statement support.

5. Highlighting Scheme

- Color Map:

```
{
  'KEYWORD': '#569CD6',    # Blue
  'STRING': '#CE9178',     # Orange
  'COMMENT': '#6A9955',    # Green
  # ... (other colors)
}
```

- Implementation:
  1. Tkinter text tags applied on keystroke.
  2. Full re-scanning for simplicity.

6. GUI Implementation

- Framework: Tkinter.
- Components:
  1. ScrolledText widget for code editing.
  2. Event binding (<KeyRelease>) for real-time updates.
- Features:
  1. Dynamic syntax highlighting.
  2. Basic error indication (red background).