

## MiniLang Scanner Report

### 1. Introduction

MiniLang is a small, new programming language designed to demonstrate key programming concepts. The MiniLang scanner is developed to tokenize MiniLang source code according to its language specifications. This report presents the design, implementation details, and test cases of the MiniLang scanner.

### 2. Design

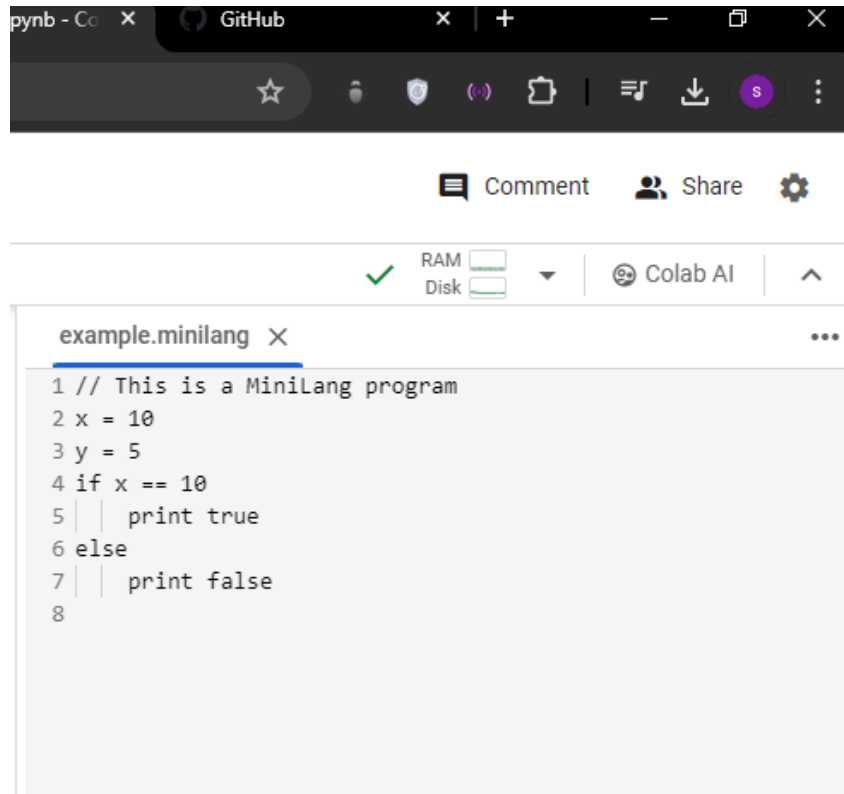
The scanner is designed to recognize tokens defined in MiniLang's specifications using a finite state machine approach. It tokenizes the input source code into tokens, each containing a token type and lexeme.

### 3. Implementation Details

- Token Types: Defined token types include INTEGER, BOOLEAN, operators (+, -, \*, /, =, ==, !=), keywords (if, else, print), literals (true, false), identifiers, comments, and EOF.
- Regular Expressions: Token patterns are defined using regular expressions to match the lexemes in the input source code.
- Tokenization: The input source code is tokenized using the defined regular expressions. Tokens are identified iteratively, and any invalid characters result in lexical error handling.
- File Handling: The scanner reads MiniLang source code from a file and tokenizes it. Error handling is implemented to report lexical errors encountered during tokenization.

### 4. Test Cases

- Basic Test Case: Test the scanner with basic MiniLang source code containing variable assignments, if-else conditions, and print statements.



The screenshot shows a web browser window with a Google Colab notebook. The browser's address bar shows 'pynb - Co' and 'GitHub'. The notebook interface includes a top bar with 'Comment', 'Share', and 'Colab AI' buttons. Below the top bar, there are RAM and Disk usage indicators. The notebook content is titled 'example.minilang' and contains the following code:

```
1 // This is a Minilang program
2 x = 10
3 y = 5
4 if x == 10
5 | | print true
6 else
7 | | print false
8
```

- Edge Cases: Test the scanner with edge cases such as empty input, single-character input, and input with invalid characters to ensure robustness.
- Comments Handling: Test the scanner's ability to handle single-line comments.
- Error Handling: Test the scanner's error handling by providing input with invalid characters and ensuring proper error reporting.



Untitled2.ipynb

File Edit View Insert Runtime Tools Help [All changes saved](#)



+ Code + Text



0s



```
('DIVIDE', '/')
('DIVIDE', '/')
('IDENTIFIER', 'This')
('IDENTIFIER', 'is')
('IDENTIFIER', 'a')
('IDENTIFIER', 'MiniLang')
('IDENTIFIER', 'program')
('IDENTIFIER', 'x')
('ASSIGN', '=')
('INTEGER', '10')
('IDENTIFIER', 'y')
('ASSIGN', '=')
('INTEGER', '5')
('IF', 'if')
('IDENTIFIER', 'x')
('ASSIGN', '=')
('ASSIGN', '=')
('INTEGER', '10')
('PRINT', 'print')
('BOOLEAN', 'true')
('ELSE', 'else')
('PRINT', 'print')
('BOOLEAN', 'false')
```

