Noeta - Design - Lexical Specification

1. Introduction

This document defines the lexical structure of the Noeta language. The lexical analyzer (also known as a scanner or tokenizer) is the first phase of the Noeta compiler. It reads the raw Noeta script as a stream of characters and groups them into a sequence of tokens. Each token represents a basic, indivisible unit of the language, such as a keyword, an identifier, a literal value, an operator, or a punctuation mark.

This specification will help in building the scanner for Noeta, whether manually or using a scanner generator tool like Lex/Flex.

2. Token Types

The following are the categories of tokens recognized by the Noeta language.

2.1. Keywords

Keywords are reserved words that have special meaning in the Noeta language and cannot be used as identifiers.

- **Description**: Predefined words that control the structure and operations of Noeta statements.
- **Pattern**: Each keyword is a specific sequence of case-sensitive letters.
- **Examples**:
    - `load`
    - `select`
    - `filter`
    - `sort`
    - `by`
    - `as`
    - `with`
    - `on`
    - `join`
    - `groupby`
    - `agg`
    - `n`
    - `random`
    - `dropna`
    - `columns`
    - `fillna`
    - `value`
    - `mutate`
    - `apply`
    - `function` (as part of `apply` and `rolling`)

    - describe
    - summary
    - outliers
    - method
    - quantile
    - column (as part of quantile, binning, rolling)
    - q
    - normalize
    - binning
    - bins
    - rolling
    - window
    - hypothesis
    - vs
    - test
    - boxplot
    - heatmap
    - pairplot
    - timeseries
    - x (as part of timeseries)
    - y (as part of timeseries)
    - pie
    - values (as part of pie)
    - labels (as part of pie)
    - save
    - to
    - format
    - export_plot
    - filename
    - width
    - height
    - info
    - desc (for sorting order)

2.2. Identifiers (IDENTIFIER)

Identifiers are names used to denote datasets (aliases), column names, and other user-defined entities like method names (e.g., "iqr", "zscore" for outliers or normalize) or function names (e.g., "mean", "sum" for rolling or groupby agg).

- **Description**: User-defined names. They must start with a letter or an underscore, followed by any number of letters, digits, or underscores. Identifiers are case-sensitive. Keywords cannot be used as identifiers.
- **Pattern (Regular Expression)**: [a-zA-Z_][a-zA-Z0-9_]*

- **Examples**:
    - `my_data`
    - `Sales_Data`
    - `column1`
    - `_temp_dataset`
    - `total_amount`
    - `user_id`
    - `iqr` (when used as a method name for `outliers`)
    - `zscore` (when used as a method name for `normalize`)
    - `mean` (when used as a function name for `rolling` or `groupby`)

2.3. Literals

Literals represent fixed values in the Noeta script.

2.3.1. String Literals (`STRING_LITERAL`)

String literals are sequences of characters enclosed in double quotes. They are used for file paths, expressions within `mutate` or `apply` statements, and specific values in `fillna`.

- **Description**: A sequence of characters delimited by double quotes (`"`). Special characters within the string (like a double quote itself or a newline) would typically require escape sequences (e.g., `\"`, `\n`) if such complexity is supported by the scanner. For this specification, we assume simple strings without internal escaped quotes for the basic pattern.
- **Pattern (Regular Expression)**: `\"[^"]*\"`
    - *(Note: This pattern matches a double quote, followed by zero or more characters that are not a double quote, followed by a closing double quote. It does not inherently support escaped quotes like `\"` within the string. A more complex regex like `\"(\\.|[^"\\])*\"` could handle simple escapes like `\\` and `\".`)*
- **Examples**:
    - `"data/my_file.csv"`
    - `"output_report.png"`
    - `"columnA + columnB"` (used in `mutate`)
    - `"x / 100.0"` (used in `apply`)
    - `"Not Applicable"` (used in `fillna`)

2.3.2. Numeric Literals (`NUMERIC_LITERAL`)

Numeric literals represent numbers, which can be integers or floating-point values.

- **Description**: A sequence of digits, optionally including a decimal point for floating-point numbers.
- **Pattern (Regular Expression)**: `[0-9]+(\.[0-9]+)?`

    - *(Note: This pattern matches one or more digits, optionally followed by a decimal point and one or more digits. It doesn't explicitly handle scientific notation like `1e5` or negative numbers; the unary minus operator is typically handled at the parser level.)*
- **Examples**:
    - `100`
    - `0.75`
    - `3.14159`
    - `10` (for `sample n: 10`)
    - `5` (for `binning bins: 5`)
    - `800` (for `export_plot width: 800`)

2.4. Operators (`OPERATOR`)

Operators are special symbols used primarily in filter conditions to perform comparisons.

- **Description**: Symbols representing comparison operations.
- **Pattern (Regular Expression for individual operators or combined)**:
    - `==` (Equal to)
    - `!=` (Not equal to)
    - `<` (Less than)
    - `>` (Greater than)
    - `<=` (Less than or equal to)
    - `>=` (Greater than or equal to)
    - Combined: `(==|!=|<=|>=|<|>)`
- **Examples**:
    - `==`
    - `>`
    - `<=`

2.5. Punctuation (`PUNCTUATION`)

Punctuation characters are used to structure statements and lists.

- **Description**: Special characters that separate or group parts of the Noeta syntax.
- **Pattern (Individual characters)**:
    - `{` (Left curly brace)
    - `}` (Right curly brace)
    - `[` (Left square bracket)
    - `]` (Right square bracket)
    - `:` (Colon)
    - `,` (Comma)
- **Examples**:

    - `:` (e.g., in `by:`, `agg:`, `on:`, `n:`, `value:`, `columns:`, `function:`, `method:`, `q:`, `bins:`, `window:`, `vs:`, `test:`, `x:`, `y:`, `values:`, `labels:`, `to:`, `format:`, `filename:`, `width:`, `height:`)
    - `{` and `}` (e.g., in `select data {col1, col2} as new_data`)
    - `[` and `]` (e.g., in `filter data [condition] as new_data`)
    - `,` (e.g., in `{col1, col2, col3}`)

2.6. Comments (`COMMENT`)

Comments are used to add explanatory notes to the Noeta script. They are ignored by the compiler.

- **Description**: Single-line comments start with a `#` symbol and extend to the end of the line.
- **Pattern (Regular Expression)**: `#.*`
- **Examples**:
    - `# This is a comment explaining the next line`
    - `load "data.csv" as my_data # Load the primary dataset`

2.7. Whitespace (`WHITESPACE`)

Whitespace characters (spaces, tabs, newlines) are used to separate tokens and improve readability. They are generally ignored by the parser, except when they serve to delimit tokens.

- **Description**: Spaces, tabs, and newline characters.
- **Pattern (Regular Expression)**: `[\s\t\n\r]+` (matches one or more whitespace, tab, newline, or carriage return characters)
- **Action**: Typically skipped/ignored by the lexical analyzer.
- **Examples**:
    - (space)
    - `\t` (tab)
    - `\n` (newline)

3. Error Handling

If the lexical analyzer encounters a character or sequence of characters that does not match any of the defined token patterns, it should report a lexical error. This typically involves creating an `ERROR` token containing the problematic character(s) and possibly its location (line number, column number) to aid the user in debugging their script.

- **Token Type**: `ERROR`
- **Description**: Represents a portion of the input that could not be recognized as a valid token.
- **Example**: If the input contains `load "file.csv" @s my_data`, the `@` symbol might be flagged as an error token if it's not part of any valid token definition.

This lexical specification provides the necessary details for tokenizing Noeta scripts, forming the foundation for the subsequent parsing phase.