Dynamic Syntax Highlighter for Python, Java, and JSX/TSX Code using Regular Expressions

Made Debby Almadea Putri - 13521153
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13521153@std.stei.itb.ac.id:

***Abstract*—Code readability and comprehension are significantly impacting productivity. Syntax highlighting, a technique used in Integrated Development Environments (IDEs), enhances code readability by visually distinguishing different elements of source code. This paper presents the development and implementation of a web-based syntax highlighter using Next.js and regular expressions. Testing and analysis demonstrate the reliability of the syntax highlighter, supporting multiple programming languages such as Python, Java, and JSX/TSX. By leveraging regular expressions, the highlighter accurately identifies code elements and enhances their visual representation**

Keywords—Pattern Matching, Regular Expression, Syntax Highlighting

I. Introduction

In a world of software development, code readability and comprehension plays a vital role in increasing productivity among developers. As the software and the developer team gets bigger, the time spend in reading, understanding, and modifying the code will also increase significantly. Thus, a tools that helps enhancing code readability is essential. The modern Integrated Development Environment (IDEs) recognize this problem and forms one of the key functionality for IDEs, syntax highlighting.

Syntax highlighting is a technique used by text editor to distinguish between different elements of the source code using color and/or typefaces. It typically highlights keywords, variables, strings, comments, and function calls [1]. This allows developer to quickly identified parts of the code, reducing the work needed to understand complex structure.

Reference [2] conducted a research on the effect of richer visualization on code comprehension. This research demonstrates the impact of richer code visualization in reducing the comprehension time of code feature. Contrary to developer's subjective perception, it is also observed that richer code visualization do not result in visual overload. One example that we commonly see is the difference between commented blocks and uncommented blocks. Commented blocks are more dimmed compared to uncommented blocks, thus the developers can distinguish between commented and uncommented blocks in just one look.

While IDEs provide built-in syntax highlighting capabilities, understanding the underlying mechanism and showcasing its functionality is a key objective of this paper. To achieve this, this paper presents a web-based syntax highlighter developed using Next.js, a powerful JavaScript framework, and utilize the capabilities of regular expression to implement language detection and syntax highlighting algorithm.

The primary focus of this paper is to explain the inner workings of our syntax highlighter, which utilizes regular expression for language detection and efficient code highlighting. By showcasing its implementation details, this paper aim to provides reader with insights into how syntax highlighting operates.

II. Basic Theory

A. *Regular Expression*

Regular expression (often shortened to regex) is an algebraic description of regular language, a class of languages that can be recognized by finite automata, that offer a declarative way to express the accepted strings [3]. The concept of regular expression was introduced by mathematician Stephen Kleene in the 1950s as a way to describe regular languages.

Regular expression is a powerful tools in searching through text, especially if there is a specific pattern and corpus of texts to search within. A regular function search will search through the corpus and returns a first match or every match, if there are more than one [4]. In describing a pattern, regular expression use a combination of a sequence of characters and metacharacters.

B. *Regular Expression in JavaScript*

Regular expression in JavaScript can be constructed in two ways

1. Expression literal, which consists of a pattern enclosed between slashes. This method is preferable if the regular expression remains constant through its runtime [5]
2. Regular expression object by calling the constructor of the object. This method is preferable if the regular expression will be changing in runtime

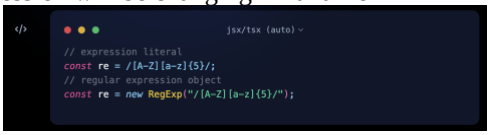

Fig. 1 Regular expression construction in JavaScript (Source: Personal Library)

This paper will use the expression literal in constructing a regular expression because the regex will remain constant. In writing a regular expression pattern, one can use a simple pattern consist of characters to find a direct match in a text. For example, the pattern /abc/ will only match if there is an exact sequence of 'abc' in a text. The other one is to use special characters to find match that require more than direct match.

TABLE I. REGEX CHARACTER CLASSES

Characters	***Meaning***
[xyz] [a-c]	Any characters in the enclosed tag
[^xyz] [^a-c]	Anything besides the characters in the enclosed tag
.	Any single character except line terminators
\d	Any digit from 0 to 9
\D	Any characters besides digit
\w	Any alphanumeric character
\W	Any characters besides alphanumeric character
\s	Any single white space characters
\S	Any characters beside single white space characters
x|y	Either x or y

(Source: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_expressions/Cheatsheet)

TABLE II. REGEX ASSERTIONS

Characters	***Meaning***
^	The beginning of an input
$	The end of an input
\b	Word boundary, a character does not have another word-character before or after it
\B	Non-word boundary character
x(?=y)	Matches x only if x is followed by y
x(?!y)	Matches x only if x is not followed by y
x(?<=y)	Matches x only if x is preceded by y
x(?<!y)	Matches x only if x is not preceded by y

(Source: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_expressions/Cheatsheet)

TABLE III. GROUPS AND BACKREFERENCES

Characters	***Meaning***
(x)	Matches x and remembers the match
(?<Name>x)	Matches x and stores it on the groups property under the specified name
(?:x)	Matches x but does not remembers the match

(Source: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_expressions/Cheatsheet)

TABLE IV. REGEX QUANTIFIERS

Characters	***Meaning***
x*	Matches the preceding token zero or more times
x+	Matches the preceding token one or more times
x?	Matches the preceding token zero or one time
x{n}	Matches the preceding token n times
x{n,}	Matches the preceding token n or more times
x{n,m}	Matches the preceding token n to m times
x*? x+? x?? etc	Similar to the one before, except it will stop as soon as it finds a match

(Source: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_expressions/Cheatsheet)

C. Python Syntax

Python is an interpreted object-oriented high-level programming language released on 1990s by Guido van Rossum. Unlike other programming language that needs explicit curly braces or semicolon, python emphasizes the use of white space and indentation to define code blocks.

Python include set of control-flow structures, such as if-else-elif, for-while loop, and exception handling. In defining a method, it starts with keyword "def" and the name of the function along with its parameters followed by a colon.

Python is a dynamically-typed language. This means the variables can hold any types and its type can be change through the runtime.

As an object oriented programming language, python also provides object oriented approach. Defining a class begins with keyword "class" and the name of the class. Unlike java or C++, to reference the instance of class, python use keyword "self". The class constructor is defined inside "__init__" method.

D. Java Syntax

Java is an object oriented programming language developed by James Gosling in the mid 1990s. Java's syntax is derived from C and C++, making it familiar to the developers from those backgrounds.

Java's codes are structured into classes and objects that interacts with each other. Java is a strong-typed language, meaning variable's type should be explicitly declared that encourages type safety. Every code blocks is defined within a curly braces and each line is ended by a semicolon.

Java include set of control-flow structures, such as if-else-else if, for-while loop, and exception handling. In defining a method, it starts with modifier e.g. public, private, and protected if needed then the method return type followed by its name and parameters.

To define a class, it starts with the class modifier then the class name. The class constructor is defined inside a method without any return type with the name equals to the class name.

E. JSX/TSX Syntax

JavaScript XML (JSX) and TypeScript XML (TSX) are JavaScript syntax extensions to write component-based user interfaces. JSX and TSX are used by React, a JavaScript frontend web-framework. JSX and TSX are grouped into one because of its similar syntax. TypeScript is an extensions of JavaScript that adds types annotation and static typing to the usual JavaScript making developers detect errors early in compile time.

JSX and TSX allows developers to write HTML code inside the JavaScript or TypeScript file. Thus in the making of syntax highlighter for JSX and TSX code, HTML syntax also needs to be considered.

Because it's an extension of JavaScript, the syntax also follows JavaScript's syntax. JavaScript provides dynamic behavior of a website and developed by Brendan Eich in the mid 1990s. JavaScript's syntax is derived from the Java and C programming language. However, unlike Java and C, JavaScript is a dynamically-typed language, meaning its variable can hold any types and its type can be change through the runtime.

The syntax of JavaScript also includes control-flow such as if-else-else if, for and while loop, and exception handling. There are two ways of defining a function or method in JavaScript, with function keyword or an arrow function. JavaScript also supports object-oriented programming. An object in JavaScript can be created in two ways, calling the class constructor or by using anonymous objects.

The syntax that differentiate between JSX/TSX and Java, beside the HTML syntax is in its variable declaration. While it does not need type declaration, JavaScript provided a variable scoping declaration. It includes keyword var, let, and const. Variable that defined with var and let keyword can be updated but cannot be re-declared. While variable with var keyword exists within a function, let keyword only exists within a scope bounded by curly braces. On the other hand, variable with const keyword, much like its name, cannot be updated nor re-declared.

III. IMPLEMENTATION

In building syntax highlighter web-based app involves two crucial steps, each contributing to the overall functionality and effectiveness of the highlighter. The first step focused on developing programming language detection mechanism using regular expression. The second step focused on the creation of regular expressions to match and categorize individual code elements.

A. Programming Language Detection

Before categorizing individual code elements to highlight the syntax, detecting the programming language is an important thing to do. Each language, in this case python, java, and jsx/tsx, have different keywords and patterns so the categorization for each programming language is different.

1. Python Source Code Pattern

Python can easily be differentiated with java and jsx/tsx because of its different ways to define a code blocks. Python keywords are also very different with java and jsx/tsx, with using def for function and elif, which will be else if in java and jsx/tsx. Each of this function or control-flow definition is followed by a colon to mark the beginning of a code blocks. Based on this facts, the regular expression for python source code pattern can be constructed as follows

The regex above will match the definition of function block, class block, if-elif control block, and for-while loop control block. The regex is inside a non-capturing group for performance reason.

def\s+\w+\s*\(.*\)\s*\: will match an exact sequence of "def", the function keyword, followed by one or more white space followed by a word, in this case a function name. \(.*\) will match the parameters of the function enclosed by parantheses. \: will match the colon that is mandatory to start a function block.

class\s+\w+\s*\(?.*\)?\s*\: will match the class block. The regex is similar to the function's regex except the parantheses is optional because parantheses in a class definition only needed if the class is inheriting another class.

if\s+.+\:\s*.+ will match the if control-flow block. elif and else keyword is not present in this regex because it can only exists if "if" block exists. (?:for|while)\s+.+\:(?!.*\)) will match the for and while loop control-flow block. The (?!.*\)) prevents the regex to match the for-while loop block of java because inside the for-while loop arguments can include colon for iteration.

2. Java Source Code Pattern

One way to differentiate Java source code from Python and JavaScript is through class and method definition. In class and method definition usually includes access modifier e.g. public, private, and protected. However, it is not mandatory.

The regex above will match the definition of class and method block in Java. The regex is divided into two to make it more readable.

(?:public|private|protected)?(?:class|interface)\s+(\w+)\s* will match the access modifier, if present, and the class or interface keyword followed by its name. (?:<[^>]+>)? will detect a generic class. (?:\s+extends\s+\w+(?:\s*<[^>]+>)?)? will match extends syntax in Java. (?:\s+implements\s+[\w\s,]+)?\s*\{ will match the implements syntax in Java. The curly brackets will start the class blocks.

(?:public|private|protected)?\s*(\w+)\s*\([^)]*\)\s* will match the access modifier, name, and parameters of a method.

(?:throws\s+\w+(?:\s*,\s*\w+)*)?\s*\{ will match the throws keyword in a method. The curly brackets will start the method blocks.

3. JSX/TSX Source Code Pattern

Differentiating JSX and TSX source code can be tricky because its blocks are enclosed by curly braces, similar to Java. One obvious way is to look at the HTML syntax in the code which neither java and python have. However, there is another way to detect JSX/TSX source code by using the variable scoping inherited from the JavaScript syntax.

The regex above will match the variable and function definition in JSX/TSX. It will also match the HTML parts of JSX/TSX.

(?:const|function|var|let)\s+\w+ will match the definition of variable with const, var, and let keyword and the definition of function with function keyword.

There are two ways to write an HTML tag: by using an opening and closing tag or by using a self-closing tag. For the opening and closing tag, the pattern <[A-Za-z]*\s*[^>]*> matches the opening tag of an HTML element. The pattern [^<]* matches the content inside the HTML tag.

Lastly, the pattern <\/[A-Za-z]*> matches the closing tag of the HTML element. On the other hand, for self-closing tags, the pattern <[A-Za-z]*\s*[^>]*\/> is used.

B. Regular Expression for Common Pattern

In creating syntax highlighter, there are several common category or pattern that are presents in multiple programming language, in this case Python, Java, and JSX/TSX. Those categories are:

1. Operators and Punctuation

Operators such as binary operators (+, -, /, *, ^, %, |, &&), unary operators (!), assignment operators (=, +=, -=), and comparison operators (==, <=, >=, <, >) along with punctuation (., ,, :, ;, ?, @) can be matched using

```
[\+\/\-\*\%\!\=\,\.\:\;\@\\\?\|\^]|&lt;|&gt;|&&
```

< and > is the entity name for less-than (<) and greater-than (>). This encoding is used because browser might mixed up (<) and (>) with tags.

2. Char and String

Char and string literals are enclosed with quotes and can be matched with pattern

```
(?<!\/(\S)*)(["][^"]*["]|['][^']*[']|[`][^`]*[`])
```

The regex `(?<!\/(\S)*)` at the beginning prevents the regex to match the string pattern inside a regex. `(["][^"]*["]|['][^']*[']|[`][^`]*[`])` is a pattern for each type of quotes: double quote (“), single quotes (‘), and (`).

3. Default Parameters

A function or procedure can have default parameters that can be override when calling a function or procedure by using assignments. This can be matched using

```
(?<=\([^\)]*)(?<!=)(\w+)(?==.*\)
```

Passing props in HTML tags are similar to default parameters in function or procedure, thus passing props in HTML tags can be matched using

```
(\w+)(?==(?:\{[^\}]*\}|<[^>]*?"[^"]*"|<[^>]*?'[^']*'
))
```

4. Parentheses

Parentheses, all round brackets (()), curly brackets ({}), and square brackets ([]), can be matched using

```
[\(\)\[\]\{\}]
```

5. Number

Number can be matched using digit character class \d+. This is because the dot in floating point will be captured first by operator regex and thus the floating point will be read as two separated digit and can be captured using \d+ only.

6. Decorator

Decorator are typically identified by an @ symbol followed by an identifier. A pattern like @(\w+) can be used to match decorators.

7. Regex

Regex are typically identified by enclosed // thus a pattern like `\/(\S)*\/[gimuy]*` can be used to match regex. The regex in python are bounded by double quotes thus the string pattern will took care of it.

C. Regular Expression for Programming Languge-specific Pattern

As mentioned in the Programming Language Detection segment, each programming language has specific keywords and syntax. For example, function definition for Python, Java, and JSX/TSX are all different. Thus, a regular expression that match a pattern for specific programming language is necessary.

To make the code readable and avoid redundancy, keywords and patterns in a code are categorized as follows

1. Keywords
   - *basic* will include the control-flow keyword as well as Boolean value and imports keyword
   - *important* will include the definition keyword such as function and class definition as well as the keyword that usually follows it
   - *types* will include the basic type keyword defined in each programming language
   - *specials* will include keyword that needs to standout from other keywords
2. Patterns
   - *className*
   - *comment*

- *functionName*
- *special*, a pattern that can't be categorized into className, comment, and functionName and also needs to standout from other pattern

1. Patterns in Python
   - *className*

     A word is a class name if it's preceded by "class" keyword. To increase the readability, if a word is bounded by words with dot in between, e.g. *word.class.word*, then it is categorized as a class except if it's called by "self" keyword. Thus, the pattern that will match the class name in python is

     ```
     (((?<=class)\s*(\w+)(?=\s*\(?.*\)?:))|((?<
     !self\.)(?<=\.)(\w+)(?=\.)))
     ```

   - *comment*

     Comment in python are classified into two: inline comment and multi-line comment. Inline comment are noted by "#" at the start or bounded by quotes. Multi-line comment are bounded by triple single-quotes (''') or triple double-quotes (""""). However, the quotes are already matched by the string pattern. Thus, the pattern that will match the comment block in python is

     ```
     #(.*)
     ```

   - *functionName*

     A word is a function name if it's followed by round brackets. Thus, the pattern that will match the class name in python is

     ```
     (\w+)(?=\((.*))
     ```

2. Patterns in Java
   - *className*

     A word is a class name if it's preceded by "class" keyword. In Java, there is a naming convention for a class name to always begin with upper-case letter, method and variable begin with lower-case letter. Thus the syntax highlighter will follow this naming convention and the pattern that will match the class name in Java is

     ```
     ((\b[A-Z]\w*\b)|(?<=\.)(\w+)(?=\.))
     ```

   - *comment*

     Comment in java are classified into two: inline comment and multi-line comment. Inline comment are noted by "//" at the start. Multi-line comment are bounded by "/*". Thus, the pattern that will match the comment block in Java is

     ```
     (\/\/(.*))|(\/\*([\s\S]*?)\*\/)
     ```

   - *functionName*

     Similar to Python, a word is a function name if it's followed by round brackets. Thus, the pattern that will match the class name in Java is

     ```
     (\w+)(?=\((.*))
     ```

3. Patterns in JSX/TSX
   - *className*

     The naming convention in JSX/TSX is not as strong as the one in Java, thus there are some rules as how a word can be classified as a class name. A word is a class name if it's preceded by "class", "type", "as", and "new" keyword. Additional rule for TSX is if it is used as a type. A variable is used as a type if the variable is inside angel brackets (<>) and after a colon (:). Thus, the pattern that will match the class name in JSX/TSX is

     ```
     (((?<=(<)\/?.*)(\b[A-
     Z]\w*\b))|((?<!>)(\w+)(?=<[^\/])(?!.
     *\())|((?<=(type|as)\s+)(\w+))|((?<=new\s+
     )(\w+))|((?<!case|default\:\s+)(?<=\w+\:\s
     +)(\w+)))
     ```

   - *comment*

     Comment in JSX/TSX are similar to the one in Java. Thus, the pattern that will match the comment blocks in JSX/TSX is

     ```
     (\/\/(.*))|(\/\*([\s\S]*?)\*\/)
     ```

   - *functionName*

     A word is a function name if it's followed by round brackets. However, there are some additional rule in TSX because of type definition in a function. For example when using useState() in TSX usually done with a type definition by using (<>) such as useState<String>(). The function pattern in Java will not match this function name. Another special thing in JSX/TSX is the arrow function. Thus the pattern will need to match the arrow function syntax and the pattern that will match all of it is

     ```
     (((\w+)(?=(<.*>)?\((.*)))|((?<=funct
     ion\s+)(\w+))|((?<=(?:const|let|var)\s+)(\
     w+)(?=\s+\=\s+\())
     ```

   - *Special*

     The special pattern in JSX/TSX is the pattern that match HTML tag name. The pattern that will match the tag name in JSX/TSX is

     ```
     (?<=<\/?)(\w+)
     ```

To further distinguish different elements of source code, imported class, function, or package is also considered as a class. For example, in Python, a good practice in importing classes from other modules or third library is to import only the needed class. In Python GUI development a file often consist of passing message to the GUI object. Without considering imported class, function, or module as a class then it will be treated as basic variable, making it hard to distinguish between local and imported variables.

To solve this, we make a rule that every imported class, function, or module is considered as a class name.

The regex above will match inline imports and grouped imports in Python. Searching for imported class were done before conducting any recoloring. The matched words are stored and treated as a keyword.

Fig. 2 Regex for Programming Language Specific Pattern

(Source: Personal Library)

On Fig. 2, each pattern are preceded and followed by the same pattern because coloring each line require replacement to HTML tag (in this case a span tag), thus each regex will need to ignore the value inside the tag except for the comment regex because it will be the first regex to be tested and replaced.

D. Integrations

The process of integrating all of the regular expression we have analyzed to the color for each code categories is the heart of syntax highlighting.

The colors for each category is determined by analyzing existing syntax highlighters and coding standards. The objective is to enhance code readability by improving the visual distinction between various elements within the code. One thing that is common in existing syntax highlighters is the dimmed color for commented out block and shades of green for string or char. Table V shows the colors for each code categories

TABLE V. COLORS FOR EACH CATEGORY

Category	***Color***
comment	#6b7280
string	#34d399
regex	#7dd3fc
params (default)	#c084fc
decorator	#f472b6
class name	#facc15
function name	#818cf8
basic keyword	#7dd3fc
important keyword	#ec4899
types keyword	#facc15
specials keyword	#f87171
imported keyword	#facc15
special pattern	#f87171
operator/punctuation	#7dd3fc
parentheses	#f472b6
number	#fb923c

(Source: Personal Library)

Replacing the plain text into colored text utilize regex method in JavaScript called replace. For each regex, if there is a match then the matched text will be replace to a span tag with each has a style according to its designated color. Another important thing while doing regex matching and replacement is deciding the order of replacement. This is because the text that has been replaced cannot be matched by another regex. Based on trial and errors, the order of replacement is shown in Fig. 3

Fig. 3 Regex Matching and Replacement

(Source: Personal Library)

IV. TESTING AND ANALYSIS

1. Python Source Code

Fig. 4 Syntax Highlighting on Python Source Code

(Source: Personal Library)

2. Java Source Code

Fig. 5 Syntax Highlighting on Java Source Code

(Source: Personal Library)

3. JSX/TSX Source Code

Fig. 6 Syntax Highlighting on Java Source Code

(Source: Personal Library)

The test case above is designed to cover all pattern listed in previous section. Regex pattern testing is demonstrated in the Fig. 2. During the testing phase, the test case are a plain text with unknown programming language. The app successfully detect the programming language for Python, Java, and JSX/TSX if the plain text contains a lot of context about the language. The app, however, will set the language to Python (default language) if there is little to no context such as the body only contains simple operation.

The comparison between unhighlighted and highlighted code can be seen on the fig. above. In most cases, the syntax highlighter mostly succeed in mapping the code elements to its designated color. However, there are some cases that resulted in slightly unsatisfiable result. Those case are usually cases that require more context, such as the functionNameRegex is highlighted even though in this context it is a basic variable but because we considered the pattern of type definition in TypeScript, it was detected as a type.

V. Conclusion And Suggestions

. In conclusion, through testing and analysis, we have demonstrated the reliability of the syntax highlighter in improving the visual distinction between code elements. The highlighter supports multiple programming languages, including Python, Java, and JSX/TSX. By utilizing regular expressions, we have achieved language detection and efficient code highlighting.

To further enhance the highlighter, we propose several suggestions for future improvements. First, expanding the language support to include other programming language to reach other developers and codebases. Second, extending the web-app to be customizable to allow developers define their own highlighter. Third, optimizing syntax highlighting either by improving the regular expression or using advanced pattern matching so the categorization can be more precise and can include coloring inside a commented blocks.

Youtube Link

https://youtu.be/HNlhrg1iJBY

Repository Link

https://github.com/debbyalmadea/re-light

Website Link

https://re-light.vercel.app/

Acknowledgment

First and foremost, I would like to express my heartfelt gratitude to God for His blessings, guidance, and unwavering presence throughout this paper. Next, I would like to express my sincere gratitude to Dr. Ir. Rinaldi Munir, M.T, my esteemed professor of Strategi Algoritma, for his invaluable guidance and support throughout the development of this project. His engaging lectures and commitment to excellence have greatly enriched my understanding of algorithm.

References

[1] Omisola, Idowu. (2022). What is Syntax Highlighting? Accessed from https://www.makeuseof.com/syntax-highlighting-what/ on 22 May 2023

[2] Dimitar Asenov, Otmar Hilliges, and Peter Müller. (2016). The Effect of Richer Visualizations on Code Comprehension. In Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems. ACM, 5040–5045. https://doi.org/10.1145/2858036.2858372

[3] John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman. (2006). Introduction to Automata Theory, Languages, and Computation, Prentice Hall

[4] Daniel Jurafsky, James H. Martin. (2023). An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, third edition draft.

[5] MDN Web Docs. Regular Expression. Accessed from https://developer.mozilla.org/docs/Web/JavaScript/Guide/Regular_expressions on 22 May 2023

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 22 Mei 2023

Made Debby Almadea Putri - 13521153