**Topic: Regular expressions**

**Course: Formal Languages & Finite Automata**

**Author: Obrijan Filip**

**Theory:**

Regular Expressions (regex) are powerful tools used for pattern matching in strings. They consist of sequences of characters that define search patterns, making them useful for text processing, validation, and data extraction.

**Common Use Cases**

* **Validation**: Ensuring input follows a specific format (e.g., emails, phone numbers, passwords).
* **Searching**: Finding occurrences of a word or pattern in a document.
* **Replacing**: Modifying text dynamically based on patterns.
* **Splitting and Tokenizing**: Breaking text into meaningful parts.

**Basic Components of Regular Expressions**

1. **Literals**: Direct character matches (e.g., abc matches "abc").
2. **Character Classes**: Define sets of characters to match (e.g., [a-z] matches any lowercase letter).
3. **Quantifiers**: Define repetition rules:
   * \* (zero or more times)
   * + (one or more times)
   * ? (zero or one time)
   * {n,m} (between n and m times)
4. **Alternation** (|): Acts like a logical OR (e.g., cat|dog matches "cat" or "dog").
5. **Grouping** (()): Groups expressions and captures matched text.
6. **Anchors**: Define start (^) and end ($) of strings.

**Example**

For the regex pattern a(b|c)d\*:

* It matches "ab", "ac", "abd", "acd", "abdd", etc.
* "aeb" would not match, as "e" is not in the pattern.

Regular expressions are widely used in programming languages such as Python, JavaScript, and Java, and are essential tools for text manipulation.

**Objectives:**

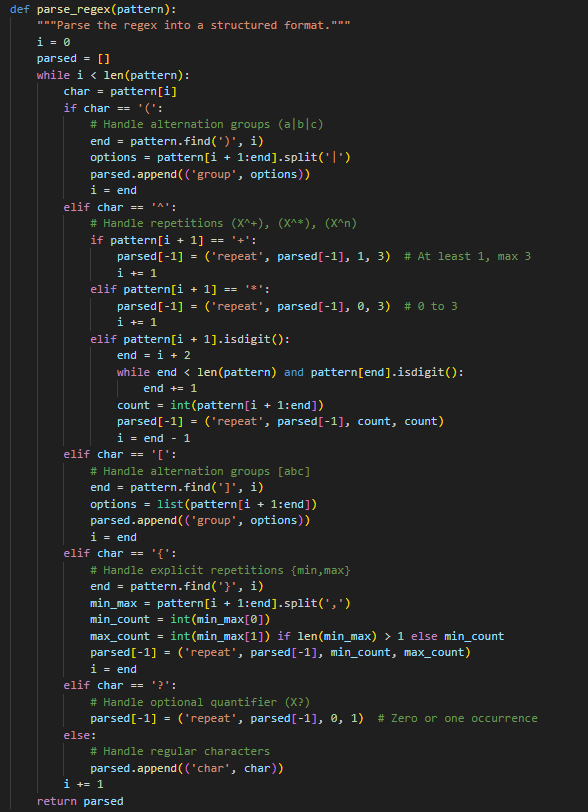
1. Write and cover what regular expressions are, what they are used for;
2. Below you will find 3 complex regular expressions per each variant. Take a variant depending on your number in the list of students and do the following:

a. Write a code that will generate valid combinations of symbols conform given regular expressions (examples will be shown). Be careful that idea is to interpret the given regular expressions dinamycally, not to hardcode the way it will generate valid strings. You give a set of regexes as input and get valid word as an output

b. In case you have an example, where symbol may be written undefined number of times, take a limit of 5 times (to evade generation of extremely long combinations);

c. **Bonus point**: write a function that will show sequence of processing regular expression (like, what you do first, second and so on)

**Implementation description:**

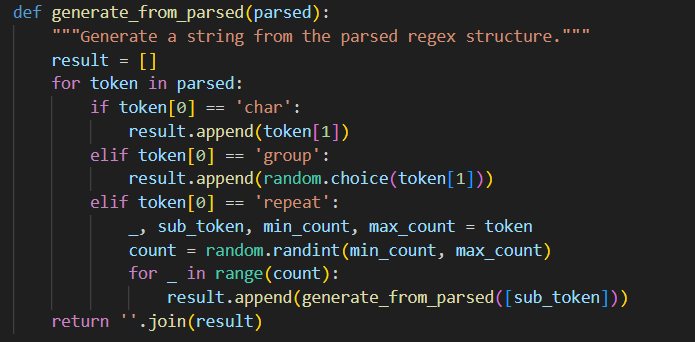


What It Does:

This function takes a regex pattern as input and converts it into a structured format (a list of tuples). It does this by iterating over the pattern and identifying different regex components like character groups, repetitions, and optional elements.

**Key parts:**

* **Character groups ((a|b|c))**: Identified using ( and ), and split using |.
* **Repetitions (X^+, X^\*, X^n)**: These specify how many times a character or group should repeat.
  + X^+ means at least 1 and at most 3 times.
  + X^\* means between 0 and 3 times.
  + X^n means exactly n times.
* **Character sets ([abc])**: Enclosed in [], these are treated as character options.
* **Explicit repetition ({min,max})**: Specifies a range for how many times a character or group can repeat.
* **Optional quantifier (X?)**: Indicates that a character or group appears 0 or 1 time.

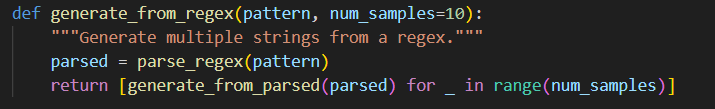


What It Does:

This function takes the structured representation from parse\_regex() and generates a string by randomly selecting characters and repetitions.

**Key parts:**

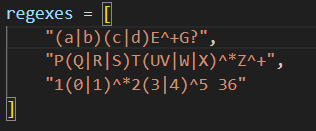
* **Characters (char)**: Directly added to the output.
* **Character groups (group)**: A random choice is made from the available options.
* **Repetitions (repeat)**: A random number is chosen between min\_count and max\_count, and the corresponding characters are repeated.

  
What It Does:

This function generates multiple random strings that match the given regex.

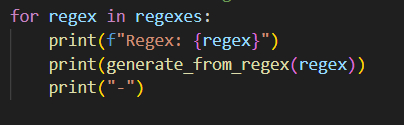
**Steps:**

1. **Parse the regex** using parse\_regex(pattern).
2. **Generate multiple samples** (default: 10) using generate\_from\_parsed(parsed).
3. **Return the list of generated strings**.



What It Does:

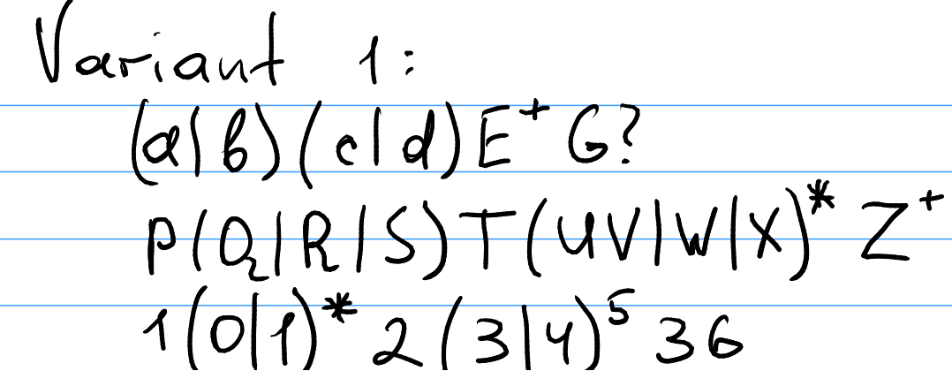
This list contains example regex patterns.



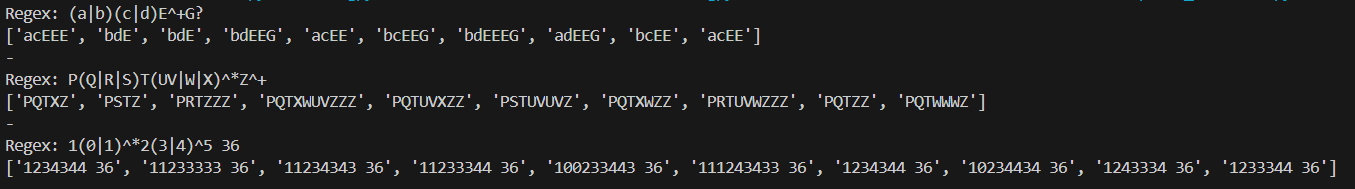
What It Does:

This loops through each regex, generates random matching strings, and prints them.

**Variant 1:**



**Output example:**

****

**Conclusion:**

In this laboratory work, i developed a program that can parse and generate random strings based on simplified regular expressions. The implementation consisted of three main functions:

1. **parse\_regex(pattern)** – This function analyzed the regex pattern and converted it into a structured format, identifying character groups, repetitions, and optional elements.
2. **generate\_from\_parsed(parsed)** – This function used the structured format to generate strings by randomly selecting characters and applying the defined repetition rules.
3. **generate\_from\_regex(pattern, num\_samples=10)** – This function combined the previous steps to produce multiple valid strings based on the input pattern.

The program successfully demonstrated how to interpret and generate random text sequences based on different regex patterns, following defined repetition and grouping rules. It provided insights into text pattern generation and parsing techniques, which are useful in fields such as data validation, text generation, and programming language design.

Further improvements could include expanding regex support by handling more complex constructs like nested groups, advanced quantifiers, and character ranges. Additionally, optimizing performance for large patterns would enhance the program’s efficiency.

This experiment helped reinforce the understanding of regular expressions, parsing techniques, and random string generation, which are valuable concepts in computer science and software development.

**References:**

1.Else Course FAF.LFA21.1

2.[**Regular expressions**](https://en.wikipedia.org/wiki/Regular_expression)