Topic:

Course: Formal Languages & Finite Automata

Objectives:

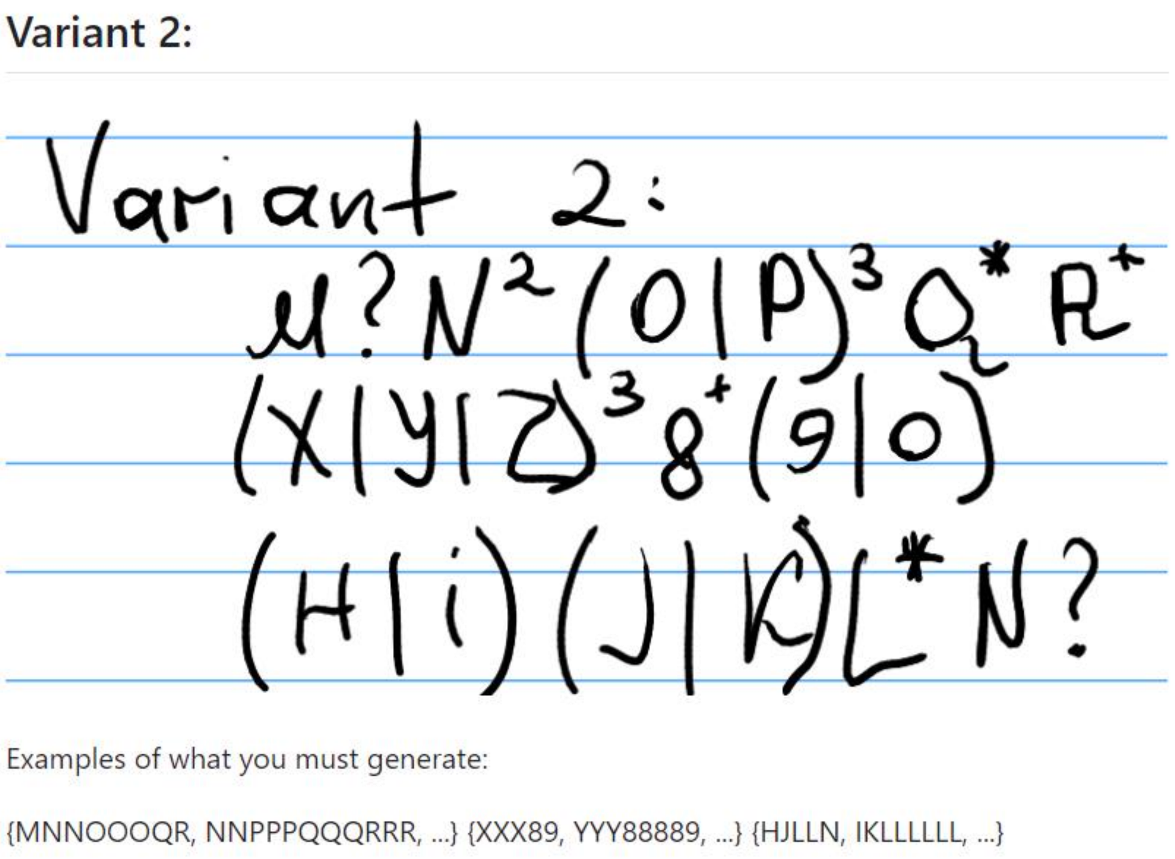
1. Write and cover what regular expressions are, what they are used for;
2. Below you will find 3 complex regular expressions per each variant. Take a variant depending on your number in the list of students and do the following:

   a. Write a code that will generate valid combinations of symbols conform given regular expressions (examples will be shown).

   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)

Write a good report covering all performed actions and faced difficulties.



Variant 2:

Variant 2:

$M?N^2(O|P)^3Q^*R^+$

$(X|Y|Z)^3 8^+(9|0)$

$(H|i)(J|K)L^*N?$

Examples of what you must generate:

{MNNOOOQR, NNPPPQQQRRR, ...} {XXX89, YYY88889, ...} {HJLLN, IKLLLLLL, ...}

Laboratory work:

Theoretical notes:

A regular expression is a sequence of characters that specifies a match pattern in text. It can be defined as a language or string accepted by a finite automata. Regular expressions consist of constants, which denote sets of strings, and operator symbols, which denote operations over these sets.

Regular expressions help us to match, locate and manage text, providing a quick and relatively easy way to manipulate data particularly in large complex programs.

Regex can be used to search, extract, validate, or transform text based on specified patterns. Programmers use regex to perform tasks such as parsing text, validating input, and finding and replacing specific patterns within strings. This pattern can be as basic as searching for a comma (,) in a text or as complex as searching for a valid email address in a text.

Metacharacters are characters with a special meaning in a regular expression. The table below shows some common regex metacharacters and the definition of each metacharacter:

Metacharacter	**Description**
.	Finds any single character
^	Start of string
$	End of string
*	Matches zero or more times
+	Matches one or more times
?	Matches either 0 or 1times
{}	Matches all inside a set amount of times
[]	Specifies a set of characters to match
\	Escapes metacharacters to match them in patterns
|	Specifies either a or b (a|b)
()	Captures all enclosed

3.

Implementation

1. For the expressions provided, there needs to be specified the implementation of the rules it needs to follow that I specified previously in theory. For example, here is the bit of code f what the program will do if it encounters a “*” character in the expression :

```
elif rule[i] == "(" and rule[rule.index(")", i) + 1] == "*": #when we get to this character

            for _ in range(random.randint(0, 5)): #to not repeat more than 5 times

                char = rule[i - 1] #the character previous to the * character

                string += char # will be added to the string n amount of times (btw 0 and 5)

                print(f"Zero or more occurrences. Add {char} to string => {string}")

            i = rule.index(")", i) + 1
```

Here the character previous to the “+” character will be repeated n times (0 to 5) times and added to the string.

For the “?” character, again, as in the rules specified, it specifies that the previous character will be shown 0 or 1 time. Here is how it was implemented:

```
elif rule[i] == "(" and rule[rule.index(")", i) + 1] == "?":
            if random.randint(0, 1):
                char = choice(options(rule[i + 1:rule.index(")", i)]))
                string += char
                print(f"Zero or one occurrence from options: Add {char} to string => {string}")
            i = rule.index(")", i) + 1
```

For these rules, I have the following outputs:

```
Add N to string => N
Fixed occurrences from options. Add P to string => NP
Fixed occurrences from options. Add O to string => NPO
Fixed occurrences from options. Add O to string => NPOO
Add Q to string => NPOOQ
Add R to string => NPOOQR
Final string:  NPOOQR


Fixed occurrences from options. Add Y to string => Y
Fixed occurrences from options. Add Y to string => YY
Fixed occurrences from options. Add X to string => YYX
Add 8 to string => YYX8
Just one occurrence from options. Add 9 to string => YYX89
Final string:  YYX89


Just one occurrence from options. Add H to string => H
Just one occurrence from options. Add J to string => HJ
```

```
Add L to string => HJL
Add N to string => HJLN
Final string:  HJLN
```

Therefore, the rules were followed accordingly.

Conclusion:

This laboratory teaches us the fundamentals of regular expressions and their uses. Here I implemented the code to generate valid combinations of symbols based on these regular expressions. The regular expressions are generated as the rules specified and I had encountered some problems with the repeating of characters but it was all handled. During realization of this project, I understood why they are used in search engines, in search and replace dialogs of word processors and text editors and in lexical analysis.