Ministry of Education, Culture, and Research of the Republic of MoldovaTechnical University of Moldova
The Faculty of Computers, Informatics, and Microelectronics

REPORT

Laboratory work no.4
Regular expressions

Executed by: Cornievschi Bogdan
gr. FAF-222

Verified by:
univ. assist. Crețu Dumitru

Chişinău - 2024

Introduction

Regular expressions are a foundational tool in the field of computer science, known for their efficiency and versatility in pattern matching and string manipulation. This report outlines the process of generating strings that match a given regular expression pattern using the Python programming language. Our focus is to demonstrate the practical application of regular expressions in creating strings within defined constraints, thereby showcasing their utility in tasks such as data validation, parsing, and text analysis.

Methodology

A Python script was developed to randomly generate strings that conform to specific regular expressions. These expressions dictate the structure of the strings, encompassing a variety of symbols each with a designated function, such as indicatingthe occurrence of certain characters or the frequency of their repetition. To illustrate the process, a function was also created to explain the sequence of operations performed by the regular expression in a human-readable format.

The regular expressions used for this exercise are complex, involving multiple operations such as grouping, selection, quantification, and specific character inclusion.The symbols within the expressions include parentheses for grouping ( ), the vertical bar for logical OR |, the asterisk for zero or more repetitions *, and the plus sign for one or more repetitions +. Literal characters are also part of the expressions, signifying exact matches required in the resulting strings.

Elements of the regular expressions

(S|T) - This means the expression will match either the character S or T
(U|V) - This means the expression will match either the character U or V
W* - This means the expression will match zero or more occurrences of the characterW
Y+ - This means the expression will match one or more occurrences of the character Y
24 - Will be present in every expression

Output of the programs

The first expression

```
After processing '(S|T)': T
After processing '(U|V)': TV
After processing 'W*': TVW
After processing 'Y+': TVWYY
After processing '24': TVWYY24
Final Sequence: TVWYY24
```

The second expression

```
After processing 'L': L
After processing '(M|N)': LM
After processing 'O^3': LMOOO
After processing 'p*': LMOOOpp
After processing 'Q': LMOOOppQ
After processing '(2|3)': LMOOOppQ2
Final Sequence: LMOOOppQ2
```

The third expression

```
After processing 'R*':
After processing 'S': S
After processing '(T|U|V)': ST
After processing 'W': STW
After processing '(x|y|z)^2': STWxx
Final Sequence: STWxx
```

My Python code defines a function, generate_from_pattern_with_steps, which takes a pattern as input and generates a sequence based on the rules defined within that pattern. The pattern is composed of symbols and operations that dictate how the sequence should be constructed. Let's break down the code to understand how it works:

1. Splitting the Pattern into Parts

The pattern is expected to be a string where different parts are separated by spaces. The function first splits this pattern into its constituent parts using pattern.split(' '). Each part of the pattern could represent an operation like selecting between options, repeating a character, etc.

2. Initialising the Output String

An empty string output is initialised. This string will be built up step by step as the function processes each part of the pattern.

3. Processing Each Part of the Pattern

The function iterates over each part of the split pattern with a for loop. Inside this loop, it checks for special symbols (^, *, +) that indicate specific operations:

^ **for Repetition**: If a ^ is found, the part is split again at ^. The first part is the base character or option group, and the second part is the repeat factor. The base part is then repeated as many times as indicated by the repeat factor.

*** for Optional Repetition**: If a * is found, it means the preceding character can be repeated any number of times from 0 to 5. The character is identified, and a random repeat factor between 0 and 5 is chosen.

+ for Mandatory Repetition: If a **+** is found, the character must appear at least once but can be repeated up to 5 times. Again, a random repeat factor between 1 and 5 is chosen.

4. Handling Option Groups

If the part contains options (denoted by parentheses and separated by |, e.g., (U|V|S)), these are split into an array of options. One of these options is randomly selected.

5. Generating the New Output

For each part processed, the selected option or character, potentially repeated, is added to the output string. After each part is processed, the current state of output is printed, showing how the sequence is built step by step.

6. Final Sequence

Once all parts of the pattern are processed, the final sequence is printed out.

The second version of the code

Then I created the second version of the code that handles the expressions without spaces, the main problem I had during completing the task in the first version of the code, was the problem with the understanding the power symbol ^, to parse that was pretty difficult in the context, therefore, I decided to use {} in order to include how many times will be repeated the certain symbol, or the variants the are proposed in the format (U|B|S){3}.

The output for the second version of the code

```
Enter the regex pattern: (S|T)(U|V)W*Y+24

S
SV
SV
SVYYY
SVYYY2
SVYYY24

Final Result: SVYYY24
```

```
Enter the regex pattern: L(M|N)O{3}p*Q(2|3)

L
LM
LMOOO
LMOOOpp
LMOOOppQ
LMOOOppQ2

Final Result: LMOOOppQ2


Enter the regex pattern: R*S(T|U|V)W(x|y|z){2}

RRRR
RRRRS
RRRRSV
RRRRSVW
RRRRSVWzy

Final Result: RRRRSVWzy
```

Conclusion

Throughout this laboratory exercise, we explored the practical application of regular expressions and the use of Python to generate and analyse strings that match specified regex patterns. By constructing Python scripts that implemented random choices within the bounds of the given expressions, we produced valid strings that adhered to the constraints of literal characters, grouped selections, and quantifiers.

In summary, this laboratory work highlighted the fundamental importance ofregular expressions in software development and data analysis. It showcased the power of computational tools in pattern recognition and the generation of test data,and it reinforced the conceptual knowledge through hands-on coding practice. The exercises completed herein stand as a testament to the integral role that regular expressions play in the broader context of computer science and programming.