Assignment 7:

## Q1

The feature responsible for generating regular expression (Regex) objects in Python is the re module. The re module provides functions and methods for working with regular expressions.

## Q2

Raw strings are often used in Regex objects to handle backslashes (\) effectively. In Python, backslashes are used as escape characters to represent special characters or character sequences (e.g., \n for newline, \t for tab). However, in regular expressions, backslashes have their own special meaning, and using them as escape characters in string literals can lead to unintended behaviour.

To avoid potential issues and to ensure that backslashes are treated as literal characters within regular expressions, raw strings are commonly used.

## Q3

The search() method from the re module in Python returns a match object if it finds a match for the specified pattern within the given string. If no match is found, the method returns None.

## Q4

To get the actual strings that match the pattern from a Match object in Python, we can use the group() method. The group() method returns the substring of the input string that was matched by the regular expression pattern.

## Q5

In the regular expression pattern r'(\d\d\d)-(\d\d\d-\d\d\d\d)', the group zero (group(0)) covers the entire match, including all the captured groups.

Group 1 (group(1)) corresponds to the first capturing group (\d\d\d), which matches three consecutive digits.

Group 2 (group(2)) corresponds to the second capturing group (\d\d\d-\d\d\d\d), which matches a group of three digits followed by a hyphen and then four digits.

## Q6

To match literal parentheses and periods in a regex pattern, we need to escape them using a backslash (\).

To tell a regex that we want to match literal parentheses, we can use \( to match an opening parenthesis and \) to match a closing parenthesis.

## Q7

The findall() method of the re module in Python returns a list of all non-overlapping matches of a pattern in a string. The structure of the returned list depends on the presence of capturing groups (parentheses) in the regular expression pattern.

## Q8

In standard regular expressions, the | character is used as the alternation or OR operator. The | operator is used to create a logical OR condition.

## Q9

The | operator is used to create a logical OR condition in regular expressions. It matches either the pattern on the left side of the | or the pattern on the right side. It allows we to specify multiple alternative patterns, and it matches any one of the patterns.

## Q10

The + and \* characters in regular expressions are both quantifiers, which means that they indicate how many times the preceding expression should be matched.

The "+" character means "one or more" and is used to match one or more occurrences of the preceding pattern.

The \* character indicates that the preceding expression can be matched zero or more times.

## Q11

The difference between {4} and {4,5} in regular expressions is the number of times the preceding expression must be matched. The curly braces are used to represent quantifiers in regular expressions. The quantifier {4} indicates that the preceding expression must be matched exactly 4 times.

The quantifier {4,5} indicates that the preceding expression must be matched at least 4 times but not more than 5 times.

## Q12

In regular expressions, the shorthand character classes \d, \w, and \s are special patterns that match specific sets of characters:

\d represents any digit character (0-9). It matches a single digit.

\w represents any word character. It matches alphanumeric characters (a-z, A-Z, 0-9) and underscores (\_).

\s represents any whitespace character. It matches spaces, tabs, newlines, and other whitespace characters.

## Q13

In regular expressions, the shorthand character classes \D, \W, and \S are negations of the corresponding shorthand character classes \d, \w, and \s. They represent the inverse sets of characters:

\D represents any non-digit character. It matches any character that is not a digit (0-9).

\W represents any non-word character. It matches any character that is not an alphanumeric character or underscore.

\S represents any non-whitespace character. It matches any character that is not a space, tab, newline, or other whitespace character.

## Q14

The main difference between them is the preference for matching shorter or longer sequences. If there are multiple occurrences of the pattern, the non-greedy .\*? will match the shortest possible substring, while the greedy .\* will match the longest possible substring.

## Q15

To match both numbers and lowercase letters in a regular expression using a character class, we can use the following syntax:

[0-9a-z]

[0-9] matches any digit from 0 to 9.

[a-z] matches any lowercase letter from a to z.

## Q16

To make a regular expression case insensitive in Python, we can use the re.IGNORECASE flag as an argument to the matching function. The IGNORECASE flag is passed to the re.compile() function when we are compiling the regular expression.

## Q17

In regular expressions, the . (dot) character normally matches any character except for a newline character (\n). It matches any single character in the input string.

However, if the re.DOTALL flag is passed as the second argument to the re.compile() function then the . character will match any character including newline characters (\n). It will match across multiple lines in the input string.

## Q18

If numReg = re.compile(r'\d+'), then numRegex.sub('X', '11 drummers, 10 pipers, five rings, 4 hen') will return the string 'X drummers, X pipers, five rings, X hen'.

## Q19

Passing re.VERBOSE as the second argument to re.compile() allows we to add comments and whitespace to the regular expression pattern for better readability without affecting its functionality.

## Q20

(r'^\d{1,3}(,\d{3})\*$')

^ matches the start of the string.

\d{1,3} matches 1 to 3 digits.

(,\d{3})\* matches a comma followed by exactly 3 digits, and this group can repeat zero or more times.

$ matches the end of the string.

## Q21

(r'^[A-Z][a-zA-Z]\* Watanabe$')

^ matches the start of the string.

[A-Z] matches an uppercase letter (the first letter of the first name).

[a-zA-Z]\* matches zero or more lowercase or uppercase letters (the rest of the first name).

matches a space between the first name and last name.

Watanabe matches the last name.

$ matches the end of the string.

## Q22

(r'^(Alice|Bob|Carol) (eats|pets|throws) (apples|cats|baseballs)\.$', re.IGNORECASE)

^ matches the start of the string.

(Alice|Bob|Carol) matches one of the given names: Alice, Bob, or Carol.

(eats|pets|throws) matches one of the given actions: eats, pets, or throws.

(apples|cats|baseballs) matches one of the given objects: apples, cats, or baseballs.

\. matches the period at the end of the sentence.

$ matches the end of the string.

re.IGNORECASE enables case-insensitive matching.