ASSIGNMENT 16

# Q1

They are concise and efficient.

They are portable across languages.

They can be used to match complex patterns.

They can be used to validate input.

They can be used to extract data from text.

They can be used to manipulate text.

# Q2

"(ab)c+": This pattern matches a sequence that starts with "ab" and is followed by one or more occurrences of "c". It captures the "ab" part as a group. For example, it would match "abc", "abcc", "abccc", and so on.

"a(bc)+": This pattern matches a sequence that starts with "a" and is followed by one or more occurrences of "bc". It captures the "bc" part as a group. For example, it would match "abc", "abcbc", "abcbcbc", and so on.

The unqualified pattern "abc+" matches a sequence that starts with "ab" and is followed by one or more occurrences of "c". It does not capture any specific groups.

# Q3

The line import re is needed to import the re module in Python, which provides functions and classes for working with regular expressions. It is necessary to use the re module to use regular expressions in our Python code.

# Q4

Hyphen (-): The hyphen is used to specify a range of characters inside square brackets. For example, [a-z] matches any lowercase letter from a to z.

Caret (^): The caret has special significance when used as the first character inside square brackets. It negates the character set, matching any character that is not in the specified set. For example, [^0-9] matches any character that is not a digit.

Closing bracket (]): The closing bracket is treated as a literal character if it is the first character inside square brackets.

# Q5

Compiling a regular expression object using re.compile() provides several benefits:

Performance improvement: Compiling a regular expression once and reusing the compiled object can significantly improve the performance of pattern matching operations. The compiled object is optimised for efficient matching.

Readability and maintainability: By compiling a regular expression, we can assign it to a variable with a descriptive name, making our code more readable and self-explanatory. It also allows us to reuse the compiled object in multiple places.

Error handling: Compiling a regular expression object allows us to catch and handle any syntax errors or exceptions that occur during compilation.

# Q6

The match object returned by re.match() and re.search() provides various methods and attributes to work with the matched patterns. Some examples of using the match object include:

Accessing matched text: we can use the group() method to retrieve the matched text. For example, match.group() returns the entire matched string.

Accessing matched groups: If the pattern contains capturing groups, we can use the group() method with an argument to access specific captured groups. For example, match.group(1) returns the text captured by the first group.

Extracting match position: The start() and end() methods give the start and end positions of the matched text.

Extracting match span: The span() method returns a tuple containing the start and end positions of the matched text.

Other methods and attributes: The match object also provides methods like groups() to retrieve all captured groups, and attributes like pos and endpos to access the search range.

# Q7

The vertical bar (|) in a regular expression is used as an alteration or logical OR operator. It allows us to specify alternative patterns. For example, the pattern "a|b" matches either "a" or "b". It can be used to match any one of the alternatives.

Square brackets ([]) are used to define a character set or character class. They allow us to specify a set of characters from which any single character can match. For example, the pattern "[abc]" matches either "a", "b", or "c".

# Q8

The raw-string indicator (r) is necessary to use in regular-expression search patterns because it prevents the Python interpreter from interpreting special characters in the pattern. For example, the regular expression \n will match a newline character if it is not enclosed in a raw string. However, if the regular expression is enclosed in a raw string, such as r"\n", then the Python interpreter will not interpret the \n character as a newline character, and the regular expression will match any character.