ASSIGNMENT 17

# Q1

Greedy syntax matches as much as possible, while non-greedy (or lazy) syntax matches as little as possible. To transform a greedy pattern into a non-greedy one, we can introduce a question mark (?) after the quantifier. For example, changing .\* (greedy) to .\*? (non-greedy).

# Q2

Greedy versus non-greedy makes a difference when there are multiple possible matches in the input string. Greedy matching will try to match as much as possible, while non-greedy matching will try to match as little as possible. If we are looking for a non-greedy match but the only one available is greedy, we may need to modify our pattern or use additional techniques to achieve the desired result.

# Q3

In a simple match that only looks for one match and does not do any replacement, the use of a non-tagged group (a group without capturing parentheses) is not likely to make a practical difference. Non-tagged groups are mainly used when we need to capture and extract specific parts of the matched text.

# Q4

A scenario where using a non-tagged category would have a significant impact is when we need to extract specific parts of the matched text and perform different operations on each part. Non-tagged groups allow us to match and capture parts without including them in the final matched text. This can be useful when we want to process or analyse different parts separately.

# Q5

Look-ahead conditions in regular expressions allow us to examine characters without consuming them. This can make a difference when we want to match a pattern based on certain conditions without including the condition characters in the final match. For example, using a positive look-ahead (?=...) allows we to find matches that are followed by a specific pattern without including the pattern in the match result.

# Q6

The difference between positive look-ahead and negative look-ahead in regular expressions is as follows:

Positive look-ahead (?=...): Matches a pattern only if it is followed by another pattern. It asserts that the text ahead matches the given pattern without including it in the final match.

Negative look-ahead (?!...): Matches a pattern only if it is not followed by another pattern. It asserts that the text ahead does not match the given pattern.

# Q7

Readability: Group names make the regular expression more descriptive and easier to understand.

Flexibility: Group names can be more meaningful than numbers and allow for easier modification or rearrangement of groups in the expression.

Self-documentation: Group names serve as self-documentation within the regular expression, making it clearer what each group represents.

Maintenance: If the regular expression is modified in the future, using group names avoids potential errors caused by renumbering or miscounting groups.

# Q8

Yes, we can identify repeated items within a target string using named groups. For example, using the regular expression pattern (?P<word>\w+)\s+\1, we can match and capture repeated words in the target string. The \1 backreference refers to the value captured by the named group word, allowing us to find repeated instances of the same word.

import re

string = "The cow jumped over the moon"

pattern = r"(?P<word>\w+)+\1"

match = re.search(pattern, string)

if match:

print(match.group("word")) # cow

# Q9

When parsing a string, the Scanner interface provides additional features compared to the re.findall function. The Scanner interface allows us to tokenize the input string by defining patterns for different tokens and retrieving them one by one. It provides methods like scanner.match(), scanner.search(), and scanner.finditer() to iterate over the tokens and perform more complex parsing tasks.

# Q10

No, a scanner object does not have to be named "scanner." The name of the object can be chosen freely according to the naming conventions and clarity of the code. It is common to use names like "scanner," "tokenizer," or "lexer" for objects that implement the Scanner interface, but it is not a requirement.