Q1. Explain the difference between greedy and non-greedy syntax with visual terms in as few words as possible. What is the bare minimum effort required to transform a greedy pattern into a non-greedy one? What characters or characters can you introduce or change?

Greedy syntax matches the longest possible string, while non-greedy matches the shortest. To make a greedy pattern non-greedy, add a ? after the quantifier, like \*? or +?.

Q2. When exactly does greedy versus non-greedy make a difference?  What if you're looking for a non-greedy match but the only one available is greedy?

Greedy vs. non-greedy matters when you want to match the maximum vs. minimum. If only a greedy match is available, you can use a non-greedy approach with .\*? or .\* to simulate.

Q3. In a simple match of a string, which looks only for one match and does not do any replacement, is the use of a nontagged group likely to make any practical difference?

In a simple match without replacement, using a non-tagged group usually doesn't impact the result. It's more relevant when you capture and use groups later.

Q4. Describe a scenario in which using a nontagged category would have a significant impact on the program's outcomes.

Using a non-tagged category might significantly impact outcome if it alters the order of matching, such as matching tags in HTML/XML with more precision.

Q5. Unlike a normal regex pattern, a look-ahead condition does not consume the characters it examines. Describe a situation in which this could make a difference in the results of your programme.

In a situation where you need to find overlapping matches, like extracting overlapping DNA sequences, look-ahead conditions help since they don't consume characters.

Q6. In standard expressions, what is the difference between positive look-ahead and negative look-ahead?

Positive look-ahead asserts a condition must be true ahead, while negative look-ahead asserts a condition must not be true ahead.

Q7. What is the benefit of referring to groups by name rather than by number in a standard expression?

Referring to groups by name improves readability and makes the pattern more self-explanatory.

Q8. Can you identify repeated items within a target string using named groups, as in "The cow jumped over the moon"?

Named groups can identify repeated items with (?P<name>...). In the given sentence, it would allow detecting repetition of words.

Q9. When parsing a string, what is at least one thing that the Scanner interface does for you that the re.findall feature does not?

The Scanner interface can tokenize input and handle more complex parsing tasks, which re.findall alone can't do.

Q10. Does a scanner object have to be named scanner?

No, a scanner object doesn't have to be named "scanner"; you can use any valid variable name.