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 and non-greedy syntax in regular expressions refer to how matching quantifiers behave.

Greedy: Greedy matching tries to match as much as possible while still allowing the overall pattern to succeed. It matches the longest possible string that satisfies the pattern. It is denoted by \* or +..

Non-greedy: Non-greedy matching tries to match as little as possible while still allowing the overall pattern to succeed. It matches the shortest possible string that satisfies the pattern. It is denoted by \*? 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?

The difference between greedy and non-greedy matching becomes significant when there are multiple potential matches within the input string.

Multiple occurrences

Ambiguous structure

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 of a string where you are only looking for one match and not performing any replacement, the use of a non-capturing group (also known as a non-tagged group) generally does not make a practical difference in terms of the overall outcome of the match.

Non-capturing groups, denoted by (?:...), are used to group subpatterns together without capturing the matched content for later retrieval or reference. They are primarily used for logical grouping, alternation, or applying quantifiers to a group as a whole

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

One scenario in which using a non-capturing group ((?:...)) can have a significant impact on a program's outcomes is when you are using the group for alternation within a larger regular expression pattern

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.

One situation in which the non-consumable nature of a look-ahead condition in a regular expression can make a difference in the results of a program is when you want to assert the presence or absence of certain characters without including them in the final match

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

Positive look-ahead ((?=...)):A positive look-ahead asserts that a specific pattern must be present after the current position in the string, without including it in the final match. It ensures that the pattern inside the look-ahead matches, but it does not consume the characters it examines. It is denoted by (?=...)

Negative look-ahead ((?!...):A negative look-ahead asserts that a specific pattern must not be present after the current position in the string, without including it in the final match. It ensures that the pattern inside the negative look-ahead does not match at the current position. It is denoted by (?!...)

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

Improved Readability

Self-Documenting Code

Avoiding Fragility

Flexible Extraction

Better Error Handling

Enhanced Code Reusability

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

To identify repeated items within a target string, named groups alone may not be suitable since they are typically used to capture and extract specific patterns. However, you can use backreferences in conjunction with named groups to identify repeated items within a target string.

Backreferences allow you to refer back to previously captured groups within the regular expression pattern. By combining backreferences with named groups, you can identify repeated occurrences of the same pattern.

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?

When parsing a string, the Scanner interface, available in certain programming languages or libraries, provides additional functionality compared to the re.findall() feature of regular expressions. One thing that the Scanner interface offers is the ability to tokenize the input string, splitting it into individual tokens based on specific patterns or delimiters

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

No, a Scanner object does not have to be named "scanner". You can choose any valid identifier as the variable name for a Scanner object, as long as it follows the naming rules of the programming language you are using