Q1. Greedy syntax in regular expressions means that the pattern will match the longest possible string that fits the pattern. Non-greedy syntax, on the other hand, will match the shortest possible string that fits the pattern. To transform a greedy pattern into a non-greedy one, you can introduce a `?` after the pattern. For example, `.\*` is a greedy pattern that matches any number of characters, while `.\*?` is a non-greedy pattern that matches the shortest possible string.

Q2. Greedy versus non-greedy makes a difference when you have a string that contains multiple matches for a pattern, and you want to extract each match separately. If you use a greedy pattern, it will match the longest possible string that fits the pattern, potentially including multiple matches. If you use a non-greedy pattern, it will match the shortest possible string that fits the pattern, ensuring that each match is separate. If you're looking for a non-greedy match but the only one available is greedy, you can introduce a `?` after the pattern to make it non-greedy.

Q3. In a simple match of a string, using a nontagged group is not likely to make any practical difference. Nontagged groups are mainly used for capturing parts of a match that you want to extract and use in replacement strings or further processing.

Q4. A scenario in which using a nontagged category would have a significant impact is when you need to extract specific parts of a match and use them in replacement strings or further processing. For example, if you have a string that contains dates in the format `MM/DD/YYYY`, you can use a nontagged group to extract the year and use it in a replacement string that reformats the date as `YYYY-MM-DD`.

Q5. A look-ahead condition that does not consume the characters it examines can make a difference in the results of your program when you need to match a pattern that is preceded or followed by a specific string, but you don't want that string to be part of the match. For example, if you have a string that contains a list of names separated by commas, and you want to extract all the names that are followed by the word "Jr.", you can use a positive look-ahead condition to match the name without including "Jr." in the match.

Q6. In standard expressions, a positive look-ahead condition is used to match a pattern that is followed by another pattern, while a negative look-ahead condition is used to match a pattern that is not followed by another pattern. For example, the pattern `foo(?=bar)` will match "foo" only if it is followed by "bar", while the pattern `foo(?!bar)` will match "foo" only if it is not followed by "bar".

Q7. The benefit of referring to groups by name rather than by number in a standard expression is that it makes the expression more readable and easier to maintain. If you refer to groups by number, you have to keep track of the order in which the groups appear in the expression, which can become confusing if the expression is complex or changes over time. By referring to groups by name, you can give them meaningful names that reflect their purpose, and easily refer to them in replacement strings or further processing.

Q8. Yes, you can identify repeated items within a target string using named groups. For example, the pattern `(?P<word>\w+)\s+(?P=word)` will match any two consecutive words that are identical.

Q9. When parsing a string, the Scanner interface can do several things for you that the re.findall feature does not, such as skipping over delimiters or whitespace, matching tokens of different