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?

Q2. When exactly does greedy versus non-greedy make a difference? What if you&#39;re looking for a
non-greedy match but the only one available is greedy?

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?

Q4. Describe a scenario in which using a nontagged category would have a significant impact on the
program&#39;s outcomes.

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.

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

A1. Greedy syntax matches as much as possible, while non-greedy syntax matches as little as possible. To transform a greedy pattern into a non-greedy one, you can introduce a question mark after the quantifier. For example, changing `.*` to `.*?`.

A2. Greedy versus non-greedy makes a difference when there are multiple possible matches within a string. If you're looking for a non-greedy match but the only one available is greedy, you may need to modify your regex pattern to ensure a non-greedy match is found.

A3. In a simple match of a string that looks for only one match and does not do any replacement, the use of a non-tagged group is not likely to make any practical difference.

A4. Using a non-tagged category could have a significant impact on the program's outcomes if you need to match a particular set of characters frequently within a regex pattern. For example, using the non-tagged category `\d` to match any digit character could simplify a pattern that needs to match multiple digits.

A5. In a situation where you need to match a pattern that is followed by another pattern but you don't want to include the second pattern in the match, a look-ahead condition that does not consume characters could make a difference in the results of your program.

A6. Positive look-ahead matches a pattern only if it is followed by another pattern, while negative look-ahead matches a pattern only if it is not followed by another pattern.

A7. Referring to groups by name rather than by number in a standard expression can make the expression more readable and easier to understand, especially when dealing with complex patterns that use many groups.

A8. Yes, you can identify repeated items within a target string using named groups. For example, the pattern `(?P<word>\w+)\s+(?P=word)` would match any instance of a word that is repeated later in the string.

A9. When parsing a string, the Scanner interface can automatically skip over delimiters and return different types of data based on the type of pattern being matched, while the re.findall feature simply returns all non-overlapping matches of a pattern.

A10. No, a scanner object does not have to be named "scanner". You can use any valid variable name when creating a scanner object.