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?

The greedy match will try to match as many repetitions of the quantified pattern as possible. The non-greedy match will try to match as few repetitions of the quantified pattern as possible.

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 greedy match will try to match as many repetitions of the quantified pattern as possible. The non-greedy match will try to match as few repetitions of the quantified pattern as possible.

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?

The group will make minimal difference for the thing

program'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.

Lookahead is used as an assertion in Python regular expressions to determine success or failure whether the pattern is ahead i.e to the right of the parser's current position. They don't match anything. Hence, they are called as zero-width assertions.

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

Positive lookahead: (?= «pattern») matches if pattern matches what comes after the current location in the input string. Negative lookahead: (?! «pattern») matches if pattern does not match what comes after the current location in the input string.

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

Groups are used in Python in order to reference regular expression matches.By default, groups, without names, are referenced according to numerical order starting with 1

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

The regex\_replace function performs this task by returning the transformed string. It has the following syntax

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?

Re.findall Return all non-overlapping matches of pattern in string, as a list of strings. The string is scanned left-to-right, and matches are returned in the order found.

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

No