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 will try to match multiple repetitions of the pattern as possible.

Non greedy will try to match quite few repetitions of the pattern.

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?

We need to create a Regex object with the re.compile() function.

Then we need to pass the string we want to search into the Regex object’s search() method. This returns a Match object.

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

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 with 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?

It makes the code more organized and readable as compared to number referencing.

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

string = "The cow jumped over the moon"

print("Duplicate characters in a given string: ");

for i in range(0, len(string)):

count = 1;

for j in range(i+1, len(string)):

if(string[i] == string[j] and string[i] != ' '):

count = count + 1;

string = string[:j] + '0' + string[j+1:];

if(count > 1 and string[i] != '0'):

print(string[i]);

Duplicate characters in a given string:

h

e

e

e

o

o

o

m

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?

Scanner interface immediately reverts back the first occurrence but, re.findall completes a full search protocol.

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