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?

Repetitions such as \* are *greedy*; when repeating a RE, the matching engine will try to repeat it as many times as possible. If later portions of the pattern don’t match, the matching engine will then back up and try again with fewer repetitions

Non-greedy qualifiers \*?, +?, ??, or {m,n}?, which match as *little* text as possible

So to transform the greedy to non-greedy instead of only \* we can use the \*?. \*+ or ?? such chars for matching the patterns steps wise.

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?

In case we want a specific chars needs to be matched after the pattern in that case we can use the non-greedy. In case of greedy pattern it will match the while string afterwards the it matches a pattern before \*.

In case of only greedy match is available then in that case we can use the {m,n} for defining the repetitions so that we will get the non-greedy pattern.

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 case of the simple match the string will be match and will get the output in form of True and False and that will be used in the group in case if we want to print the patterns which matched.

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

**Did not understood the question that is what exactly is the non-tagged category**

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](https://www.geeksforgeeks.org/python-regex/) to determine success or failure whether the pattern is ahead

import re

  example = re.search(r'geeks(?=[a-z])', "geeksforgeeks")

# display output

print("Pattern:", example.group())

print("Pattern found from index:",

      example.start(), "to",

      example.end())

With this we can check whether the pattern exists or not and based on that we can raise the exception or call the other statements.

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

# Positive lookahead

(?=<lookahead\_regex>)

In the positive lookahead it will specify that what follows the pattern.

In the negative look-ahead it states its shouldn’t following the mentioned string.

# Negative Lookahead

(?!<lookahead\_regex>)

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

We can use the group names as a placeholders and can get the output.

import re

match = re.search('(?P<name>.\*) (?P<phone>.\*)', 'John 123456')

match.group('name')

'John'

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

ll=[]

tstring = "The cow jumped over the moon"

target\_string = tstring.lower()

pattern = re.compile(r"(\b[A-Za-z]+\b)")

for match in pattern.finditer(target\_string):

# extract words

#print(match.group(1))

ll.append(match.group(1))

for i in ll:

if ll.count(i) >1:

print(i)

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?

A lexer and a parser work in sequence: the lexer scans the input and produces the matching tokens, the parser scans the tokens and produces the parsing result.

Whereas in the re.findall it will directly perform the operations on the texts whereas this is not the case with the scanner interfaces.

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

No.