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 are 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 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?

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

expression?

Q8. Can you identify repeated items within a target string using named groups, as in &quot;The cow

jumped over the moon&quot;?

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?

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

### **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**: Tries to match as much text as possible.
  + **Example**: a.\*b matches a...b (longest match).
* **Non-Greedy Syntax**: Tries to match as little text as possible.
  + **Example**: a.\*?b matches a.b (shortest match).
* **Transformation**: Change \* or + to \*? or +?.
  + **Greedy**: .\*
  + **Non-Greedy**: .\*?

### **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?**

* **Difference**: Greedy matches the longest possible string, while non-greedy matches the shortest string. This affects which part of the text is matched when multiple valid matches exist.
  + **Example**: For a.\*b on a1b2b, greedy matches a1b2b, non-greedy matches a1b.
* **Non-Greedy Match Availability**: If only a greedy match is available, you might capture more text than intended. Adjusting to non-greedy ensures minimal capture.
  + **Workaround**: Use non-greedy quantifiers like \*? or +?.

### **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?**

* **Practical Difference**: For a single match, the use of a nontagged (unnamed) group vs. a tagged (named) group does not make a practical difference if you are not referencing the group later.
  + **Example**: re.search(r'(abc)', 'abc') vs. re.search(r'(?P<name>abc)', 'abc'). Both will find abc in the string.

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

* **Scenario**: If you need to reference the group’s match later, such as in replacement operations or further processing, naming the group becomes essential.
  + **Example**: In re.sub(r'(?P<name>abc)', r'\g<name>', 'abc abc'), using (?P<name>abc) allows you to refer to name in the replacement string, preserving the matched text.

### **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 program.**

* **Situation**: When you need to assert the presence of a pattern without including it in the match result or affecting the subsequent pattern matching.
  + **Example**: To match "abc" only if it is followed by "123" but not include "123" in the result, use abc(?=123). This matches "abc" but "123" is not part of the match.

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

* **Positive Look-Ahead**: Asserts that a pattern must follow the current position.
  + **Syntax**: X(?=Y) — Matches X only if it is followed by Y.
  + **Example**: \d(?=\D) matches a digit only if it is followed by a non-digit character.
* **Negative Look-Ahead**: Asserts that a pattern must not follow the current position.
  + **Syntax**: X(?!Y) — Matches X only if it is not followed by Y.
  + **Example**: \d(?!\d) matches a digit only if it is not followed by another digit.

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

* **Clarity**: Named groups improve code readability and maintainability by providing descriptive names for capturing groups.
  + **Example**: (?P<year>\d{4})-(?P<month>\d{2})-(?P<day>\d{2}) allows access to groups via group('year'), group('month'), and group('day'), making the code more understandable.

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

* **Identification**: Named groups are useful for capturing and referencing specific patterns but are not inherently used for finding repeated items. For repetitions, you would typically use quantifiers or backreferences.
  + **Example**: To find repeated words, use a pattern like (\b\w+\b).\*\1 where \1 refers to the first captured group.

### **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**: Provides an iterative and stateful way to parse and process strings, allowing for scanning and tokenizing text efficiently, and handling more complex parsing scenarios.
  + **Feature**: Supports skipping and advancing through text, which is more flexible than the re.findall method, which only finds all matches without maintaining state.

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

* **Naming**: No, a scanner object does not have to be named scanner. The name is arbitrary and chosen by the programmer.
  + **Example**: You could name it parser, tokenizer, or any other valid identifier.