Python Regular Expressions — Q&A; (Set 16)

Q1. What is the benefit of regular expressions?

Regular expressions (regex) provide a concise, powerful way to search, match, and manipulate text patterns. They replace many lines of parsing logic with compact patterns, enable grouping/alternation/repetition, and work consistently across languages.

Q2. Difference between (ab)c+ and a(bc)+? Which one is abc+?

- (ab)c+: matches 'ab' followed by one or more 'c'. Examples: 'abc', 'abcc', 'abccc'.
- a(bc)+: matches 'a' followed by one or more 'bc'. Examples: 'abc', 'abcbc', 'abcbcbc'.
- abc+: equivalent to (ab)c+ since + applies only to the c.

Q3. How much do you need to use import re?

Always required, as regex functions live in the re module:
import re
re.search(r'\d+', 'abc123')

Q4. Which characters have special significance in square brackets, and when?

- '-' denotes ranges: [a-z].
- '^' negates if first: [^0-9].
- ']' ends the set; escape it for literal.

Other metacharacters lose special meaning inside brackets.

Q5. How does compiling a regex object help?

re.compile(pattern) pre-compiles regex, faster for repeated matches. Produces an object with match/search/findall methods. Improves readability and avoids recompilation.

Q6. Examples of using the match object (re.match/re.search)

m = re.search(r'(\d+)', 'Order 1234')
m.group(0) → '1234'
m.group(1) → '1234'
m.start(), m.end() → (6,10)
m.span() → (6,10)

Q7. Difference: vertical bar (|) vs square brackets []

- '|' = alternation between subpatterns: (cat|dog).
- '[]' = character class: [cd]og matches 'cog' or 'dog' but not 'cat'.

Q8. Why use raw strings (r'...') in patterns and replacements?

- In patterns: avoids double escaping (r'\d+' is clearer than '\\d+').
- In replacements: prevents Python misinterpreting backslashes. Example:

re.sub(r'(\w+) (\w+)', r'\2, \1', 'John Doe') → 'Doe, John'