# Python Advanced Regular Expressions — Q&A (Set 17)

## Q1. Greedy vs Non-greedy (visual & minimal change)

Greedy: expands as far right as possible (.\*).  
Non-greedy: stops at the first match (.\*?).  
Bare minimum change: add ? after quantifier (\*, +, ?, {m,n}).

## Q2. When does greedy vs non-greedy matter?

Matters when multiple matches possible:  
 'a.\*b' on 'axxbxxb' → greedy: 'axxbxxb'.  
 'a.\*?b' → non-greedy: 'axxb'.  
If only one match possible, both same. If non-greedy impossible, greedy match is used.

## Q3. Does a nontagged group matter in a simple single match?

No. (abc) vs (?:abc) both match 'abc'. Only difference is whether group is stored.

## Q4. Scenario where non-tagged groups matter

When using alternation but avoiding group numbering:  
 (foo|bar)baz → group(1).  
 (?:foo|bar)baz → no group(1). Preserves numbering for other groups.

## Q5. Why does look-ahead not consuming characters matter?

Allows assertions without consuming characters. Example:  
 foo(?=bar) on 'foobar' → matches 'foo' without consuming 'bar'.

## Q6. Positive vs Negative Lookahead

Positive (?=...): match if followed by pattern (foo(?=bar)).  
Negative (?!...): match if not followed by pattern (foo(?!bar)).

## Q7. Benefit of named groups over numbered

Improves readability and avoids miscounting. Access with m.group('name') instead of m.group(1).

## Q8. Identify repeated items with named groups

Use backreferences:  
pat = r'(?P<word>\b\w+\b).\*\b(?P=word)\b'  
Matches repeated word. Example finds 'cow' in 'the cow ... the cow'.

## Q9. What can Scanner do that re.findall cannot?

re.Scanner supports multiple patterns with actions, sequential scanning, tokenization, and custom callbacks. re.findall just returns matches without actions.

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

No. Any variable name can hold re.Scanner object. 'scanner' is just a convention.