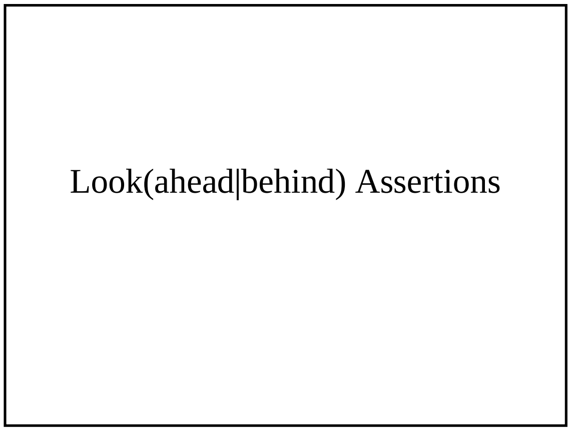
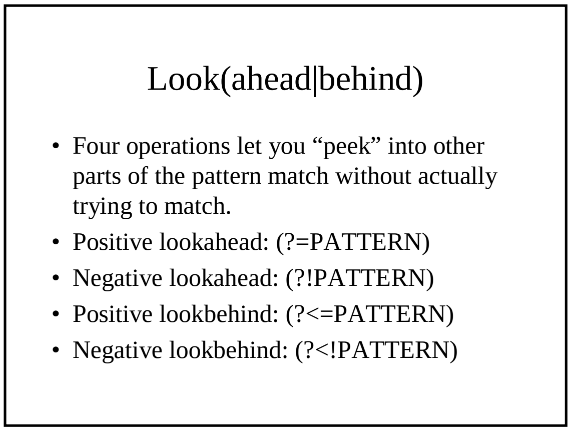
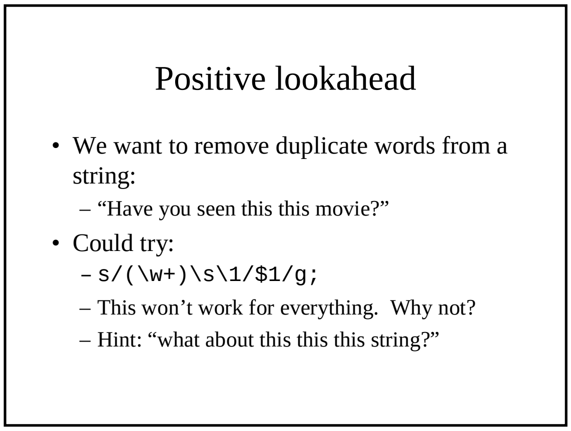


Look(ahead|behind) Assertions



Look(ahead|behind)

- Four operations let you “peek” into other parts of the pattern match without actually trying to match.
- Positive lookahead: (?=PATTERN)
- Negative lookahead: (?!PATTERN)
- Positive lookbehind: (?<=PATTERN)
- Negative lookbehind: (?<!PATTERN)



Positive lookahead

- We want to remove duplicate words from a string:
  - “Have you seen this this movie?”
- Could try:
  - `s/(\w+)\s\1/$1/g;`
  - This won’t work for everything. Why not?
  - Hint: “what about this this this string?”

Lookaheads to the rescue

- The problem is that the regular expression is “eating up” too much of the string.
- We instead just want to check if a duplicate word exists, but not actually match it.
- Instead of checking for a pair of duplicate words and replacing with first instance, delete any word if it’s going to be followed by a duplicate
- `s/(\w+) \s (?= \1 )//gx;`
- “Search for any word (and save it) followed by a space, then *check to see* if it’s followed by the same word, and replace the word and space with nothing”

Negative Lookahead

- (?!PATTERN)
- Same concept. This time, *check to see* if PATTERN does NOT come next in the string.
- `s/(\w+) \s (?= \1 )//gx;`
  - this cancels “the team that won won’t play.”
- We want to insure that the duplicate word isn’t followed by an apostrophe.
- `s/(\w+) \s (?= \1 (?! ’\w))//gx;`
- “Search for any word (and save it), followed by a space, then check to see if it’s followed by the same word, NOT followed by an apostrophe and a word character”

Lookbehind

- Positive: (?<=PATTERN)
- Negative: (?<!PATTERN)
- Same concept as look-ahead. This time, ensure that PATTERN did or did not occur *before* current position.
- ex: `s/(?<!c)ei/ie/g;`
  - Search string for all “ei” not preceded by a ‘c’ and replace with “ie”
  - “i before e except after c”
- NOTE: only ‘fixed-length’ assertions can be used for look-behind (ie, c* doesn’t work)