Video 25: regex anchors and quantifiers

Stats 102A

Miles Chen, PhD

Anchors

An **anchor** is a pattern that does not match a character but rather a position before, after, or between characters. Anchors are used to "anchor" a match at a certain position.

Pattern	Meaning
`^` or `\A`	Start of string
`$` or `\Z`	End of string
`\b`	Word boundary (i.e., the edge of a word)
`\B`	Not a word boundary

Anchor Examples

```r
text <- "the quick brown fox jumps over the lazy dog dog"
```

```r
str_replace_all(text, "the", "-") # 'the' anywhere
```

```
[1] "- quick brown fox jumps over - lazy dog dog"
```

```r
str_replace_all(text, "^the", "-") # 'the' only at the start
```

```
[1] "- quick brown fox jumps over the lazy dog dog"
```

```r
str_replace_all(text, "\\Athe", "-") # same thing
```

```
[1] "- quick brown fox jumps over the lazy dog dog"
```

```r
str_replace_all(text, "the$", "-") # 'the' only at the end
```

```
[1] "the quick brown fox jumps over the lazy dog dog"
```

Anchor Examples

```
text <- "the quick brown fox jumps over the lazy dog dog"
```

```
str_replace_all(text, "dog", "-") # 'dog' anywhere
```

```
[1] "the quick brown fox jumps over the lazy - -"
```

```
str_replace_all(text, "dog$", "-") # 'dog' only at the end
```

```
[1] "the quick brown fox jumps over the lazy dog -"
```

Anchor Examples

```r
text <- "words jump jumping umpire pump umpteenth lumps"
```

```r
str_replace_all(text, "(\\b.|.\\b)", "-") # word boundaries
```

```
[1] "-ord---um---umpin---mpir---um---mpteent---ump-"
```

```r
str_replace_all(text, "\\B.\\B", "-") # non-word-boundaries
```

```
[1] "w---s j--p j-----g u----e p--p u-------h l---s"
```

Anchor Examples

```r
text <- "words jump jumping umpire pump umpteenth lumps"
```

```r
str_replace_all(text, "\\bump", "-") # 'ump' at the beginning of a word
```

```
[1] "words jump jumping -ire pump -teenth lumps"
```

```r
str_replace_all(text, "\\Bump", "-") # 'ump' not at the beginning of a word
```

```
[1] "words j- j-ing umpire p- umpteenth l-s"
```

```r
str_replace_all(text, "ump\\b", "-") # 'ump' at the end of a word
```

```
[1] "words j- jumping umpire p- umpteenth lumps"
```

```r
str_replace_all(text, "ump\\B", "-") # 'ump' not at the end of a word
```

```
[1] "words jump j-ing -ire pump -teenth l-s"
```

The Caret Metacharacter Revisited

Question: What is the difference between `^[0-9]`, `[^0-9]`, and `[0-9^]`?

The caret `^` outside of the character set is an anchor, so `^[0-9]` matches strings that begin with a digit.

The caret `^` at the start of the character set is a negation, so `[^0-9]` matches a character that is not a digit.

The caret `^` inside a character set but not at the start is the literal caret character, so `[0-9^]` matches a character that is a digit or the caret.

Quantifiers

Quantifiers can be attached to literal characters, character classes, or groups to match repeats.

Pattern	Meaning
`*`	Match 0 or more (is greedy)
`+`	Match 1 or more (is greedy)
`?`	Match 0 or 1
`{3}`	Match Exactly 3
`{3,}`	Match 3 or more
`{3,5}`	Match 3, 4 or 5

Quantifier Examples

```
text <- "words or numbers 9,876 and combos123 like password_1234"
```

```
str_replace_all(text, "\\s", "-") # any whitespace
```

```
[1] "words-or-numbers-9,876-and-combos123-like-password_1234"
```

```
str_replace_all(text, "\\S", "-") # anything but whitespace
```

```
[1] "----- -- ------- ----- --- --------- ---- -------------"
```

```
str_replace_all(text, "\\S+", "-") # one or more non-whitespace
```

```
[1] "- - - - - - - - -"
```

```
str_replace_all(text, "\\w+", "-") # one or more word characters
```

```
[1] "- - - -,- - - - -"
```

Quantifier Examples

```r
text <- "words or numbers 9,876 and combos123 like password_1234"
```

```r
str_replace_all(text, "\\d", "-") # any digit
```

```
[1] "words or numbers -,--- and combos--- like password_----"
```

```r
str_replace_all(text, "\\D", "-") # any non-digit
```

```
[1] "-----------------9-876-----------123---------------1234"
```

```r
str_replace_all(text, "\\d+", "-") # one or more digits
```

```
[1] "words or numbers -,- and combos- like password_-"
```

```r
str_replace_all(text, "\\D+", "-") # one or more nondigits
```

```
[1] "-9-876-123-1234"
```

Quantifier Examples

```
text <-
"year 1996 area code 310 combo123 password_1234 singledigit5"
```

```
str_replace_all(text, "\\d{3}", "-") # 3 adjacent digits. reads left to rig
```

```
[1] "year -6 area code - combo- password_-4 singledigit5"
```

```
str_replace_all(text, "\\d{2,4}", "-") # 2 to 4 adjacent digits
```

```
[1] "year - area code - combo- password_- singledigit5"
```

Quantifier Examples for ?

```
text <- c("Momma","Mama","Mamma","Mommy","Mom","Mother")
```

```
str_match(text, "M[ao]m{1,2}[ay]") |> as.vector() # [ay] req'd as last char
```

```
[1] "Momma" "Mama"  "Mamma" "Mommy" NA      NA
```

```
str_match(text, "M[ao]m{1,2}[ay]?") |> as.vector() #? [ay] opt as last char
```

```
[1] "Momma" "Mama"  "Mamma" "Mommy" "Mom"   NA
```

Greedy vs Ungreedy Matching

Quantifiers are by default **greedy** in the sense that they will return the longest match.

Adding **?** to a quantifier will make it ungreedy (or **lazy**), so it will return the shortest match.

```
text <- "Peter Piper picked a peck of pickled peppers"
```

```
str_extract(text, "P.*r") # 'P' to 'r' anything in between greedy
```

```
[1] "Peter Piper picked a peck of pickled pepper"
```

```
str_extract(text, "P.*?r") # 'P' to 'r' anything in between ungreedy
```

```
[1] "Peter"
```

Greedy vs Ungreedy Matching, `str_extrac_all()`

```
1 text <- "Peter Piper picked a peck of pickled peppers"
```

```
1 str_extract_all(text, "P.*?r") # ungreedy
```

```
[[1]]
[1] "Peter" "Piper"
```

```
1 str_extract_all(text, "[Pp].*?r") # ungreedy
```

```
[[1]]
[1] "Peter"                                "Piper"
[3] "picked a peck of pickled pepper"
```