Video 26: regex capture groups

Stats 102A

Miles Chen, PhD

Grouping and Capturing

Parentheses `(  )` define a **group** that groups together parts of a regular expression.

Besides grouping part of a regular expression together, parentheses also create a numbered **capturing group**: Any matches to the part of the pattern defined by the parentheses can be referenced by group number, either for modification or replacement.

By including `?:` after the opening parenthesis, the group becomes a **non-capturing group**.

For example, in the pattern `(abc)(def)(?:ghi)`, the pattern `(abc)` creates capturing group 1, `(def)` creates capturing group 2, and `(ghi)` is a group that is not captured.

Groups are used in conjunction with `str_match()` and `str_match_all()`.

Grouping and Capturing

Some examples of the common syntax for groups:

Pattern	Meaning
`a(bc)d`	Match the text `abcd`, capture the text in the group `bc`
`(?:abc)`	Non-capturing group
`(abc)def(ghi)`	Match `abcdefghi`, group `abc` and `ghi`
`(Mrs|Ms|Mr)`	`Mrs` or `Ms` or `Mr` (preference in the order given)
`\1`, `\2`, etc.	The first, second, etc. matched group (for `str_replace()`)

Note: Notice that the vertical line `|` is used for "or", just like in logical expressions. The vertical line `|` is called the **alternation** operator.

Grouping and Capturing Examples

The output of `str_match()` is a character matrix whose first column is the complete match, followed by one column for each capturing group.

```
1 pattern <- "(bc)(def)(?:ghi)" # 'ghi' must be present but do not capture 'g
2 str_match("abcdefghijkl", pattern)
```

```
     [,1]       [,2] [,3]
[1,] "bcdefghi" "bc" "def"
```

```
1 str_match("abcdefghI", pattern)
```

```
     [,1] [,2] [,3]
[1,] NA   NA   NA
```

Grouping and Capturing Examples

```r
text <- "Mr. Smith, Mrs. Lee, Ms. Garcia"
pattern <- "(Mrs|Ms|Mr)" # match one of 'Mrs' or 'Ms' or 'Mr'
```

```r
str_match_all(text, pattern)
```

```
[[1]]
     [,1]  [,2]
[1,] "Mr"  "Mr"
[2,] "Mrs" "Mrs"
[3,] "Ms"  "Ms"
```

```r
# because Mr is listed before Mrs, it will match Mr and give preference to
wrong_order <- "(Mr|Mrs|Ms)"
str_match_all(text, wrong_order)
```

```
[[1]]
     [,1] [,2]
[1,] "Mr" "Mr"
[2,] "Mr" "Mr"
[3,] "Ms" "Ms"
```

Grouping and Capturing Examples

```r
text <- "Mr. Smith, Mrs. Lee, Ms. Garcia"
short_pattern <- "(Mr?s?)"
str_match_all(text, short_pattern)
```

```
[[1]]
     [,1]  [,2]
[1,] "Mr"  "Mr"
[2,] "Mrs" "Mrs"
[3,] "Ms"  "Ms"
```

Grouping and Capturing Examples

```r
text <- "Mr. Smith, Mrs. Lee, Ms. Garcia, Andy Hope"
capture <- "(Mrs|Ms|Mr)\\. (\\w+)"
```

```r
str_match_all(text, capture)
```

```
[[1]]
     [,1]          [,2]  [,3]
[1,] "Mr. Smith"   "Mr"  "Smith"
[2,] "Mrs. Lee"    "Mrs" "Lee"
[3,] "Ms. Garcia"  "Ms"  "Garcia"
```

Grouping and Capturing Examples

```r
text <- "Mr. Smith, Mrs. Lee, Ms. Garcia, Andy Hope"
non_capture <- "(?:Mrs|Ms|Mr)\\. (\\w+)"
```

```r
str_match_all(text, non_capture)
```

```
[[1]]
     [,1]          [,2]
[1,] "Mr. Smith"   "Smith"
[2,] "Mrs. Lee"    "Lee"
[3,] "Ms. Garcia"  "Garcia"
```

Backreferences

```
1 text = 'George Washington, John Adams, Thomas Jefferson'
2 pattern <- "(\\w+) (\\w+),?" # first group becomes \\1, second becomes \\2
```

```
1 str_match_all(text, pattern)
```

```
[[1]]
     [,1]                 [,2]     [,3]
[1,] "George Washington," "George" "Washington"
[2,] "John Adams,"        "John"   "Adams"
[3,] "Thomas Jefferson"   "Thomas" "Jefferson"
```

```
1 str_replace_all(text, pattern, "\\2, \\1;")
```

```
[1] "Washington, George; Adams, John; Jefferson, Thomas;"
```

Backreferences

```
text = 'the quick brown fox jumps over the the lazy dog'
pattern <- "\\b(\\w+)\\s+\\1\\b"
```

The pattern says:

- Word boundary
- followed by a capture group of one or more word characters
- followed by one or more spaces
- followed by the group of text that was captured earlier
- followed by a word boundary

```
str_match_all(text, pattern)
```

```
[[1]]
     [,1]       [,2]
[1,] "the the" "the"
```

The pattern will match words that are repeated.

Telephone Numbers Example

For a more complicated example, we can define a regular expression to extract phone numbers.

```
1 phone_pattern <- "\\(?([2-9]\\d{2})\\)?[- .]?([2-9]\\d{2}[- .]?\\d{4})"
```

The pattern searches for:

- an optional opening parenthesis
- a capture group consisting of:
  - a digit between 2 and 9, followed by any 2 digits
- an optional closing parenthesis
- an optional character: one of dash, space, or dot
- a capture group consisting of:
  - a digit between 2 and 9, followed by any 2 digits
  - an optional character: one of dash, space, or dot
  - any four digits

Telephone Numbers Example

```
text <- c("apple", "1-800-786-1000", "(310) 209-1626", "310.208.0448",
  "3108258430", "Work: 323 224 2611; Home: (323)224-2621", "123-456-7890")
phone_pattern <- "\\(?([2-9]\\d{2})\\)?[- .]?([2-9]\\d{2}[- .]?\\d{4})"
str_extract(text, phone_pattern)
```

```
[1] NA               "800-786-1000"   "(310) 209-1626" "310.208.0448"
[5] "3108258430"     "323 224 2611"   NA
```

Telephone Numbers Example

```r
# text <- c("apple", "1-800-786-1000", "(310) 209-1626", "310.208.0448",
#   "3108258430", "Work: 323 224 2611; Home: (323)224-2621", "123-456-7890")
# phone_pattern <- "\\(?([2-9]\\d{2})\\)?[- .]?([2-9]\\d{2}[- .]?\\d{4})"
str_extract_all(text, phone_pattern)
```

```
[[1]]
character(0)

[[2]]
[1] "800-786-1000"

[[3]]
[1] "(310) 209-1626"

[[4]]
[1] "310.208.0448"

[[5]]
[1] "3108258430"

[[6]]
[1] "323 224 2611"  "(323)224-2621"

[[7]]
character(0)
```

Telephone Numbers Example

```r
# text <- c("apple", "1-800-786-1000", "(310) 209-1626", "310.208.0448",
#   "3108258430", "Work: 323 224 2611; Home: (323)224-2621", "123-456-7890"
# phone_pattern <- "\\(?([2-9]\\d{2})\\)?[- .]?([2-9]\\d{2}[- .]?\\d{4})"
str_match(text, phone_pattern)
```

```
     [,1]              [,2]  [,3]
[1,] NA                NA    NA
[2,] "800-786-1000"    "800" "786-1000"
[3,] "(310) 209-1626" "310" "209-1626"
[4,] "310.208.0448"    "310" "208.0448"
[5,] "3108258430"      "310" "8258430"
[6,] "323 224 2611"    "323" "224 2611"
[7,] NA                NA    NA
```

Telephone Numbers Example

```
1 str_match_all(text, phone_pattern)
```

```
[[1]]
     [,1] [,2] [,3]

[[2]]
     [,1]           [,2]  [,3]
[1,] "800-786-1000" "800" "786-1000"

[[3]]
     [,1]             [,2]  [,3]
[1,] "(310) 209-1626" "310" "209-1626"

[[4]]
     [,1]           [,2]  [,3]
[1,] "310.208.0448" "310" "208.0448"

[[5]]
     [,1]         [,2]  [,3]
[1,] "3108258430" "310" "8258430"

[[6]]
     [,1]            [,2]  [,3]
[1,] "323 224 2611"  "323" "224 2611"
```

Telephone Numbers Example

Getting the previous results into something workable:

```
phone_list <- str_match_all(text, phone_pattern)
phone_matrix <- matrix(nrow = 0, ncol = 3)
for(i in seq_len(length(phone_list))){
  phone_matrix <- rbind(phone_matrix, phone_list[[i]])
}
phone_matrix
```

```
     [,1]             [,2]  [,3]
[1,] "800-786-1000"   "800" "786-1000"
[2,] "(310) 209-1626" "310" "209-1626"
[3,] "310.208.0448"   "310" "208.0448"
[4,] "3108258430"     "310" "8258430"
[5,] "323 224 2611"   "323" "224 2611"
[6,] "(323)224-2621"  "323" "224-2621"
```

Telephone - replacements with capture groups

We might not like the fact the phone numbers in column 3 have a non-standard appearance.

Fixing the problem is not as simple as replacing a dot or space with a dash because some phone numbers don't have either a dot or a space.

I define a new pattern: Three digits as capture group 1; an optional delimiter; then 4 digits which is capture group 2.

My replacement pattern is capture group 1, dash, capture group 2.

```
1 phone_matrix[,3]
```

```
[1] "786-1000" "209-1626" "208.0448" "8258430"  "224 2611" "224-2621"
```

```
1 phone_pattern <- "(\\d{3})[- .]?(\\d{4})"
2 replace_pattern <- "\\1-\\2"
3 str_replace(phone_matrix[,3], phone_pattern, replace_pattern)
```

```
[1] "786-1000" "209-1626" "208-0448" "825-8430" "224-2611" "224-2621"
```

Lookarounds

Occasionally we want to match characters that have a certain pattern before or after it. There are statements called **lookahead** and **lookbehind**, collectively called **lookarounds**, that look ahead or behind a pattern to check if a pattern does or does not exist.

Pattern	Name
`(?=...)`	Positive lookahead
`(?!...)`	Negative lookahead
`(?<=...)`	Positive lookbehind
`(?<!...)`	Negative lookbehind

The lookbehind patterns must have a bounded length (no `*` or `+`).

Positive Lookahead

Positive lookahead with `(?=...)` looks ahead of the current match to ensure that the `...` subpattern matches.

```
1 text <- "I put a grey hat on my grey greyhound."
2 pattern <- "grey(?=hound)"
3 str_locate_all(text, pattern)
```

```
[[1]]
     start end
[1,]    29  32
```

The word `grey` is matched *only* if it is followed by `hound`. Note that `hound` itself is not part of the match.

Negative Lookahead

Negative lookahead with `(?!...)` looks ahead of the current match to ensure that the `...` subpattern does *not* match.

```
1 text <- "I put a grey hat on my grey greyhound."
2 pattern <- "grey(?!hound)"
3 str_locate_all(text, pattern)
```

```
[[1]]
     start end
[1,]     9  12
[2,]    24  27
```

The word `grey` is matched *only* if it is *not* followed by `hound`.

Positive Lookbehind

Positive lookbehind with `(?<=...)` looks behind the current position to ensure that the `...` subpattern immediately precedes the current match.

```
text <-
  "I withdrew 100 $1 bills, 20 $5 bills, and 5 $20 bills."
pattern <- "(?<=\\$)[[:digit:]]+"
str_extract_all(text, pattern)
```

```
[[1]]
[1] "1"  "5"  "20"
```

The digits are matched *only* if they are immediately preceded by a dollar `$` sign.

Negative Lookbehind

Positive lookbehind with `(?<!...)` looks behind the current position to ensure that the `...` subpattern does *not* immediately precede the current match.

```
text <-
  "I withdrew 100 $1 bills, 20 $5 bills, and 5 $20 bills."
pattern <- "(?<!\\$)[[:digit:]]+"
str_extract_all(text, pattern)
```

```
[[1]]
[1] "100" "20"  "5"   "0"
```

The digits are matched *only* if they are *not* immediately preceded by a dollar `$` sign.