Advanced String Manipulation

Spring 2023

April 24 2023

Last time: Quantifiers and Special Characters

Preceding characters are matched ... .

- `*` = 0 or more
- `?` = 0 or 1
- `+` = 1 or more
- `{n}` = exactly n times

Matching character types

- `\\d` = digit
- `\\s` = white space
- `\\w` = word
- `\\t` = tab
- `\\n` = newline

More quantifiers

useful when you want to match a pattern a specific number of times

- `{n, }` = n or more times
- `{, m}` = at most m times
- `{n, m}` = between n & m times

Alternatives

useful for matching patterns more flexibly

- `[abc]` = one of **a**, **b**, or **c**
- `[e-z]` = a letter from **e** to **z**
- `[^abc]` = anything other than **a**, **b**, or **c**

Duplicating Groups

Use escaped numbers (\\1, \\2, etc) to repeat a group based on position

Which numbers have the same 1st and 3rd digits?

```
phone_numbers <- c("515 111 2244",
                   "310 549 6892",
                   "474 234 7548")
str_view(phone_numbers, "(\\d)\\d\\1")
[1] │ <515> <111> 2244
[3] │ <474> 234 7548
```

▶ Explanation

str_view_all()

```
name_phone <- c("Moly Robins: 250-999-8878",
                "Ali Duluth: 416-908-2044",
                "Eli Mitchell: 204.192.9829",
                "May Flowers: 250.209.7047")
```

```
str_view_all(name_phone,
             pattern = "([2-9][0-9]{2})[.-]([0-9]{3})[.-]([0-9]{4})")
[1] | Moly Robins: <250-999-8878>
[2] | Ali Duluth: <416-908-2044>
[3] | Eli Mitchell: <204.192.9829>
[4] | May Flowers: <250.209.7047>
```

▶ Explanation

str_replace_all()

```
str_replace_all(name_phone,
pattern = "([2-9][0-9]{2})[.-]([0-9]{3})[.-]([0-9]{4})",
replacement = "XXX-XXX-XXXX"
)
[1] "Moly Robins: XXX-XXX-XXXX"  "Ali Duluth: XXX-XXX-XXXX"
[3] "Eli Mitchell: XXX-XXX-XXXX" "May Flowers: XXX-XXX-XXXX"
```

```
str_replace_all(name_phone,
                pattern = "([2-9][0-9]{2})[.-]([0-9]{3})[.-]([0-9]{4})",
                replacement = "\\1-\\2-XXXX")
[1] "Moly Robins: 250-999-XXXX"  "Ali Duluth: 416-908-XXXX"
[3] "Eli Mitchell: 204-192-XXXX" "May Flowers: 250-209-XXXX"
```

str_extract_all()

pull all set of values matching the specified pattern

```
name_phone <- c("Moly Robins: 250-999-8878",
                "Ali Duluth: 416-908-2044",
                "Eli Mitchell: 204-192-9829",
                "May Flowers: 250-209-7047")
```

```
str_extract_all(name_phone, "[:alpha:]{2,}", simplify = TRUE)
     [,1]   [,2]
[1,] "Moly" "Robins"
[2,] "Ali"  "Duluth"
[3,] "Eli"  "Mitchell"
[4,] "May"  "Flowers"
```

Repetition

```
aboutMe <- c("my SSN is 536-76-9423 and my age is 55")
```

Repetition using ?

```
str_view_all(aboutMe, "\\s\\d?") # space followed by 0 or 1 digit
[1] │ my< >SSN< >is< 5>36-76-9423< >and< >my< >age< >is< 5>5
```

Repetition using +

```
str_view_all(aboutMe, "\\s\\d+")  # space followed by 1 or more digits
[1] │ my SSN is< 536>-76-9423 and my age is< 55>
```

Repetition using *

```
str_view_all(aboutMe, "\\s\\d*")  # space followed by 0 or more digits
[1] │ my< >SSN< >is< 536>-76-9423< >and< >my< >age< >is< 55>
```

Case conversion

```
str_to_lower("BEAUTY is in the EYE of the BEHOLDER")
[1] "beauty is in the eye of the beholder"
```

```
str_to_upper("one small step for man, one giant leap for mankind")
[1] "ONE SMALL STEP FOR MAN, ONE GIANT LEAP FOR MANKIND"
```

```
str_to_title("Aspire to inspire before we expire")
[1] "Aspire To Inspire Before We Expire"
```

```
str_to_sentence("everything you can imagine is real")
[1] "Everything you can imagine is real"
```

Alternates: OR

```
aboutMe <- c("My phone number is 236-748-4508.")
```

```
str_view(aboutMe,"8|6-")
[1] │ My phone number is 23<6->74<8>-450<8>.
```

```
str_view_all(aboutMe,"(8|6)-")
[1] │ My phone number is 23<6->74<8->4508.
```

More Duplicating Groups

```
foo <- c("addidas", "racecar")
```

```
# anything then repeat anything
str_view(foo, "(.)\\1")
[1] │ a<dd>idas
```

```
# strings like `xyzzyx`
str_view(foo, "(.)(.)(.).\\3\\2\\1"
[2] │ <racecar>
```

```
str_view(foo, "(.)(.)\\1")
[1] │ ad<did>as
[2] │ ra<cec>ar
```

Finding patterns

```
# find the last word in a sentence
str_view_all("it's a goat.",
             "[a-z]+\\.")
[1] │ it's a <goat.>
```

```
# find word with  `'s`
str_view_all("it's a goat.",
             "[a-z]+\\'\\w")
[1] │ <it's> a goat.
```

```
# find a single letter word separated by spaces
str_view_all("it's a goat.",
             "(\\s)(\\w)\\s")
[1] │ it's< a >goat.
```

What are these?

Lookaround	Name	What it Does
`(?=foo)`	Lookahead	Asserts that what immediately follows the current position in the string is *foo*
`(?<=foo)`	Lookbehind	Asserts that what immediately precedes the current position in the string is *foo*
`(?!foo)`	Negative Lookahead	Asserts that what immediately follows the current position in the string is not *foo*
`(?<!foo)`	Negative Lookbehind	Asserts that what immediately precedes the current position in the string is not *foo*

Source: click here

Look ahead example

Positive look ahead operator x(?=[y]) will find x when it comes before y

Negative version is x(?![y]) (x when it comes before something that isn't y)

```
# t before a period
str_view_all("it's a goat.", "t(?=[\\.])")
[1] │ it's a goa<t>.
```

Look ahead example

Positive look ahead operator x(?=[y]) will find x when it comes before y

Negative version is x(?![y]) (x when it comes before something that isn't y)

```
# 1+ letters before a period
str_view_all("it's a goat.","[a-z]+(?=[\\.])")
[1] │ it's a <goat>.
```

Look ahead example

Positive look ahead operator x(?=[y]) will find x when it comes before y

Negative version is x(?![y]) (x when it comes before something that isn't y)

```
# t NOT followed by a period
str_view_all("it's a goat.", "t(?![\\.])")
[1] │ i<t>'s a goat.
```

Look behind example

Positive look behind operator (?<=[x])y will find y when it follows x

Negative version is (?<![x])y (y when it does not follow x)

```
# one or more t, if preceded by a letter
str_view_all("that is a top cat.","(?<=[a-z])t+")
[1] │ tha<t> is a top ca<t>.
```

Look behind example

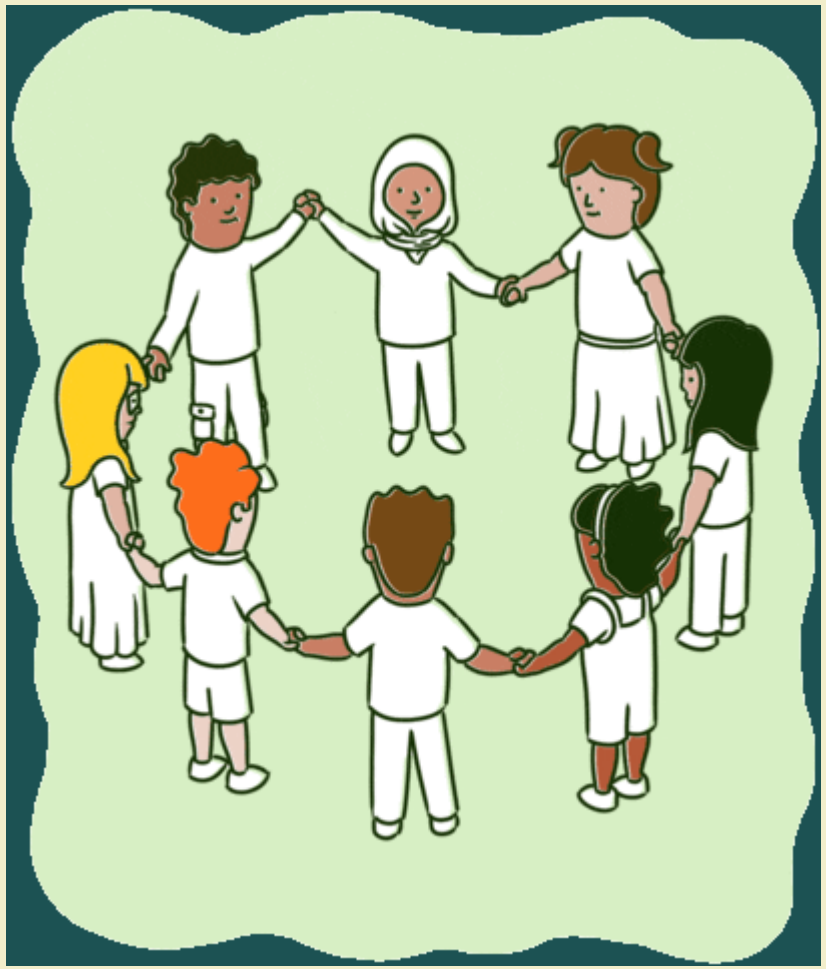
Positive look behind operator *(?<=[x])y* will find *y* when it follows *x*

Negative version is *(?<![x])y* (*y* when it does not follow *x*)

```
# t and one or more letter not preceded by a letter
str_view_all("that is a top cat.","(?<![a-z])t[a-z]+")
[1] │ <that> is a <top> cat.
```

GROUP ACTIVITY 1

15:00



- *Let's go over to maize server/ local Rstudio and our class moodle*
- *Get the class activity 13.Rmd file*
- *Skim through the problems*