# Regular Expression Part 2

## Stats 102A

Miles Chen and Michael Tsiang

Department of Statistics

Week 4 Friday

UCLA

Section 1

# Regular Expression Part 2

# Character Classes

Closely related to character sets and character ranges are **character classes**, which are used to match a certain class of characters.

The most common character classes in most regex engines are:

| Pattern | Matches | Same as |
|---|---|---|
| `\\d` | Any digit | `[0-9]` |
| `\\D` | Any non-digit | `[^0-9]` |
| `\\w` | Any word character | `[a-zA-Z0-9_]` |
| `\\W` | Any non-word character | `[^a-zA-Z0-9_]` |
| `\\s` | Any whitespace character | `[\f\n\r\t\v]` |
| `\\S` | Any non-whitespace character | `[^\f\n\r\t\v]` |

Character classes can be thought of as another type of metacharacter or as shortcuts for special character sets.

# Whitespace

There are several types of whitespace characters, shown in the following table:

| Character | Description |
|---|---|
| `\f` | Form feed (page break) |
| `\n` | Line feed (new line) |
| `\r` | Carriage return |
| `\t` | Tab |
| `\v` | Vertical tab |

For situations with non-printing whitespace characters, it can be difficult to determine which exact character it is, so the whitespace class `\\s` is a useful way to match with all of them.

# Character Classes

For example:

```
pnx <-
  c("pan", "pen", "pin", "p0n", "p.n", "paun", "pwn3d")
```

```
str_detect(pnx, "p\\d") # p followed by digit
```

# Character Classes

For example:

```
pnx <-
  c("pan", "pen", "pin", "p0n", "p.n", "paun", "pwn3d")
```

```
str_detect(pnx, "p\\d") # p followed by digit
```

```
## [1] FALSE FALSE FALSE  TRUE FALSE FALSE FALSE
```

# Character Classes

For example:

```
pnx <-
  c("pan", "pen", "pin", "p0n", "p.n", "paun", "pwn3d")
```

```
str_detect(pnx, "p\\d") # p followed by digit
```

```
## [1] FALSE FALSE FALSE  TRUE FALSE FALSE FALSE
```

```
str_detect(pnx, "p\\D") # p followed by non-digit
```

# Character Classes

For example:

```
pnx <-
  c("pan", "pen", "pin", "p0n", "p.n", "paun", "pwn3d")
```

```
str_detect(pnx, "p\\d") # p followed by digit
```

```
## [1] FALSE FALSE FALSE  TRUE FALSE FALSE FALSE
```

```
str_detect(pnx, "p\\D") # p followed by non-digit
```

```
## [1]  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE
```

# Character Classes

For example:

```
pnx <-
  c("pan", "pen", "pin", "p0n", "p.n", "paun", "pwn3d")
```

```
str_detect(pnx, "p\\d") # p followed by digit
```

```
## [1] FALSE FALSE FALSE  TRUE FALSE FALSE FALSE
```

```
str_detect(pnx, "p\\D") # p followed by non-digit
```

```
## [1]  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE
```

```
str_detect(pnx, "p\\W") # p followed by non-word character
```

# Character Classes

For example:

```
pnx <-
  c("pan", "pen", "pin", "p0n", "p.n", "paun", "pwn3d")
```

```
str_detect(pnx, "p\\d") # p followed by digit
```

```
## [1] FALSE FALSE FALSE  TRUE FALSE FALSE FALSE
```

```
str_detect(pnx, "p\\D") # p followed by non-digit
```

```
## [1]  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE
```

```
str_detect(pnx, "p\\W") # p followed by non-word character
```

```
## [1] FALSE FALSE FALSE FALSE  TRUE FALSE FALSE
```

# POSIX Character Classes

There is another type of character classes known as **POSIX character classes** that is supported by the regex engine in R.

The main POSIX classes are:

| Class | Description | Same as |
|---|---|---|
| `[:alnum:]` | Any letter or digit | `[a-zA-Z0-9]` |
| `[:alpha:]` | Any letter | `[a-zA-Z]` |
| `[:digit:]` | Any digit | `[0-9]` |
| `[:lower:]` | Any lower case letter | `[a-z]` |
| `[:upper:]` | Any upper case letter | `[A-Z]` |
| `[:space:]` | Any whitespace, inluding space | `[\f\n\r\t\v ]` |
| `[:punct:]` | Any punctuation symbol | |
| `[:print:]` | Any printable character | |
| `[:graph:]` | Any printable character excluding space | |
| `[:xdigit:]` | Any hexadecimal digit | `[a-fA-F0-9]` |
| `[:cntrl:]` | ASCII control characters | |

# POSIX Character Classes

To use POSIX classes in R, the class needs to be wrapped inside a regex character class, i.e., the class needs to be inside a second set of square brackets.

For example:

```
pnx <-
  c("pan", "pen", "pin", "p0n", "p.n", "paun", "pwn3d")
str_detect(pnx, "[[:alpha:]]") # has any letter
```

```
## [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

```
str_detect(pnx, "[[:digit:]]") # has any digit
```

```
## [1] FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE
```

# 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

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

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

# Anchor Examples

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

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

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

# Anchor Examples

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

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

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

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

# Anchor Examples

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

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

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

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

```
## [1] "- 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, "the", "-") # 'the' anywhere
```

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

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

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

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

# Anchor Examples

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

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

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

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

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

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

```
## [1] "- 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, "the", "-") # 'the' anywhere
```

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

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

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

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

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

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

# Anchor Examples

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

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

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

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

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

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

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

```
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
```

# 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 - -"
```

# 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
```

# 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

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

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

# Anchor Examples

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

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

```
## [1] "-words- -jump- -jumping- -umpire- -pump- -umpteenth- -lumps-"
```

# Anchor Examples

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

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

```
## [1] "-words- -jump- -jumping- -umpire- -pump- -umpteenth- -lumps-"
```

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

# Anchor Examples

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

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

```
## [1] "-words- -jump- -jumping- -umpire- -pump- -umpteenth- -lumps-"
```

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

```
## [1] "w-o-r-d-s j-u-m-p j-u-m-p-i-n-g u-m-p-i-r-e p-u-m-p u-m-p-t-e-e-n-t-h l-u-m-p
```

# Anchor Examples

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

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

# Anchor Examples

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

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

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

# Anchor Examples

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

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

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

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

# Anchor Examples

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

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

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

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

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

# Anchor Examples

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

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

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

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

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

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

# Anchor Examples

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

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

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

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

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

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

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

# Anchor Examples

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

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

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

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

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

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

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

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

# Anchor Examples

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

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

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

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

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

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

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

```
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 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
```

# 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"
```

# 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
```

# 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] "----- -- ------- ----- --- --------- ---- -------------"
```

# 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
```

# 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] "- - - - - - - -"
```

# 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
```

# 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

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

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

# Quantifier Examples

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

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

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

# Quantifier Examples

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

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

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

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

# Quantifier Examples

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

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

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

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

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

# Quantifier Examples

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

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

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

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

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

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

# Quantifier Examples

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

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

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

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

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

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

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

# Quantifier Examples

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

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

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

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

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

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

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

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

# Quantifier Examples

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

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

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

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

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

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

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

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

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

# Quantifier Examples

```
text <-
"the year 1996 area code 310 combo123 password_1234 singledigit 5"
```

```
str_replace_all(text, "\\d?", "-") # 0 or 1 digits
```

# Quantifier Examples

```
text <-
"the year 1996 area code 310 combo123 password_1234 singledigit 5"
```

```
str_replace_all(text, "\\d?", "-") # 0 or 1 digits
```

```
## [1] "-t-h-e- -y-e-a-r- ----- -a-r-e-a- -c-o-d-e- ---- -c-o-m-b-o---- -p-a-s-s-w-o-r-d-_----
```

# Quantifier Examples

```
text <-
"the year 1996 area code 310 combo123 password_1234 singledigit 5"
```

```
str_replace_all(text, "\\d?", "-") # 0 or 1 digits
```

```
## [1] "-t-h-e- -y-e-a-r- ----- -a-r-e-a- -c-o-d-e- ---- -c-o-m-b-o---- -p-a-s-s-w-o-r-d-_----
```

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

# Quantifier Examples

```
text <-
"the year 1996 area code 310 combo123 password_1234 singledigit 5"
```

```
str_replace_all(text, "\\d?", "-") # 0 or 1 digits
```

```
## [1] "-t-h-e- -y-e-a-r- ----- -a-r-e-a- -c-o-d-e- ---- -c-o-m-b-o---- -p-a-s-s-w-o-r-d-_----
```

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

```
## [1] "the year -6 area code - combo- password_-4 singledigit 5"
```

# Quantifier Examples

```
text <-
"the year 1996 area code 310 combo123 password_1234 singledigit 5"
```

```
str_replace_all(text, "\\d?", "-") # 0 or 1 digits
```

```
## [1] "-t-h-e- -y-e-a-r- ----- -a-r-e-a- -c-o-d-e- ---- -c-o-m-b-o---- -p-a-s-s-w-o-r-d-_----
```

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

```
## [1] "the year -6 area code - combo- password_-4 singledigit 5"
```

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

# Quantifier Examples

```
text <-
"the year 1996 area code 310 combo123 password_1234 singledigit 5"
```

```
str_replace_all(text, "\\d?", "-") # 0 or 1 digits
```

```
## [1] "-t-h-e- -y-e-a-r- ----- -a-r-e-a- -c-o-d-e- ---- -c-o-m-b-o---- -p-a-s-s-w-o-r-d-_----
```

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

```
## [1] "the year -6 area code - combo- password_-4 singledigit 5"
```

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

```
## [1] "the year - area code - combo- password_- singledigit 5"
```

# 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
```

# 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"
```

# 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
```

# 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()

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

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

# Greedy vs Ungreedy Matching, str_extrac_all()

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

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

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

# Greedy vs Ungreedy Matching, str_extrac_all()

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

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

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

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

# Greedy vs Ungreedy Matching, str_extrac_all()

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

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

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

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

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

# 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.

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

# 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.

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

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

# 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.

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

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

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

# 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.

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

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

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

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

# Grouping and Capturing Examples

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

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

# Grouping and Capturing Examples

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

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

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

# Grouping and Capturing Examples

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

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

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

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

# Grouping and Capturing Examples

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

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

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

```
# because Mr is listed before Mrs, it will match Mr and give preference to it
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

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

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

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

```
# because Mr is listed before Mrs, it will match Mr and give preference to it
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

```
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

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

# Grouping and Capturing Examples

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

```
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

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

# Grouping and Capturing Examples

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

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

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

# Grouping and Capturing Examples: Backreferences

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

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

# Grouping and Capturing Examples: Backreferences

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

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

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

# Grouping and Capturing Examples: Backreferences

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

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

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

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

# Grouping and Capturing Examples: Backreferences

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

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

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

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

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

# Grouping and Capturing Examples: 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)
```

# Grouping and Capturing Examples: 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.

```
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

```
# 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

```
# 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

```
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"
## [2,] "(323)224-2621" "323" "224-2621"
##
## [[7]]
##      [,1] [,2] [,3]
```

# 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 Example - 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.

```
phone_matrix[,3]
```

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

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

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

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