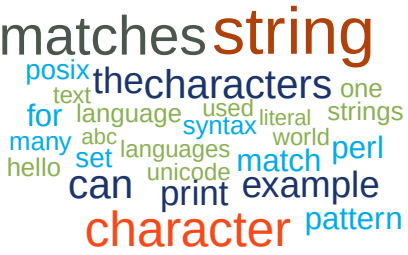
# Introduction to Web Scraping with R

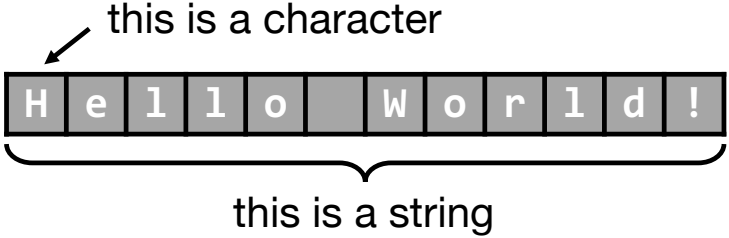
Regular Expressions in R




Simon Munzert | IPSDS

# Regular expressions in R

# Regular expressions in R



# Regular expressions in R

An example string:

R code

```r
example.obj <- "1. A small sentence. - 2. Another tiny sentence."
```

end

We are going to use the `str_extract()` function and the `str_extract_all()` function from the `stringr` package to apply regular expressions in strings. The generic syntax is:

```
str_extract(string, pattern)
str_extract_all(string, pattern)
```

`str_extract()` returns the first match, `str_extract_all()` returns all matches.

# Regular expressions in R

```
example.obj <- "1.  A small sentence.  - 2.  Another tiny sentence."
```

## Strings match themselves

R code

```
2  str_extract(example.obj, "small")
   [1] "small"
3  str_extract(example.obj, "banana")
   [1] NA
```

end

# Regular expressions in R

`example.obj <- "1. A small sentence. - 2. Another tiny sentence."`

## Multiple matches are returned as a list

R code

```
4 (out <- str_extract_all(c("text", "manipulation", "basics"), "a"))
```

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

[[2]]
[1] "a" "a"

[[3]]
[1] "a"
```

end

# Regular expressions in R

```
example.obj <- "1.  A small sentence.  - 2.  Another tiny sentence."
```

## Character matching is case sensitive

R code

```
5  str_extract(example.obj, "small")
   [1] "small"
6  str_extract(example.obj, "SMALL")
   [1] NA
7  str_extract(example.obj, ignore.case("SMALL"))
   [1] "small"
```

end

# Regular expressions in R

```
example.obj <- "1.  A small sentence.  - 2.  Another tiny sentence."
```

## Character matching is case sensitive

R code

```
9  str_extract(example.obj, "small")
   [1] "small"
10 str_extract(example.obj, "SMALL")
   [1] NA
11 str_extract(example.obj, ignore.case("SMALL"))
   [1] "small"
```

end

## We can match arbitrary combinations of characters

R code

```
12 str_extract(example.obj, "mall sent")
   [1] "mall sent"
```

end

# Regular expressions in R

`example.obj <- "1. A small sentence. - 2. Another tiny sentence."`

## Matching the beginning of a string

R code

```
13 str_extract(example.obj, "^1")
   [1] "1"
14 str_extract(example.obj, "^2")
   [1] NA
```

end

# Regular expressions in R

`example.obj <- "1. A small sentence. - 2. Another tiny sentence."`

## Matching the beginning of a string

R code

```
17 str_extract(example.obj, "^1")
   [1] "1"
18 str_extract(example.obj, "^2")
   [1] NA
```

end

## Matching the ending of a string

R code

```
19 str_extract(example.obj, "sentence$")
   [1] NA
20 str_extract(example.obj, "sentence.$")
   [1] "sentence."
```

end

# Regular expressions in R

`example.obj <- "1.  A small sentence.  - 2.  Another tiny sentence."`

## Express an "or" with the pipe operator

R code

```
21  unlist(str_extract_all(example.obj, "tiny|sentence"))
    [1] "sentence" "tiny"     "sentence"
```

end

# Regular expressions in R

`example.obj <- "1. A small sentence. - 2. Another tiny sentence."`

## Express an "or" with the pipe operator

R code

```
23 unlist(str_extract_all(example.obj, "tiny|sentence"))
   [1] "sentence" "tiny"     "sentence"
```

end

## The dot: the ultimate wildcard

R code

```
24 str_extract(example.obj, "sm.ll")
   [1] "small"
```

end

# Regular expressions in R

`example.obj <- "1. A small sentence. - 2. Another tiny sentence."`

## Matching of meta characters

- some symbols have a special meaning in the regex syntax: `.`, `|`, `(`, `)`, `[`, `]`, `{`, `}`, `^`, `$`, `*`, `+`, `?` and `-`.
- if we want to match them literally, we have to use an escape sequence: `\symbol`
- as `\` is a meta character itself, we have to escape it with `\`, so we always write `\\symbol` (weird, isn't it?!)
- alternatively, use `fixed("symbols")` to let the parser interpret a chain of symbols literally

R code

```
25 unlist(str_extract_all(example.obj, "\\."))
   [1] "." "." "." "."
26 unlist(str_extract_all(example.obj, fixed(".")))
   [1] "." "." "." "."
```

end

# Character classes

# Regular expressions in R

```
example.obj <- "1.  A small sentence.  - 2.  Another tiny sentence."
```

## Square brackets define character classes

Character classes help define special wild cards. The idea is that any of the characters within the brackets can be matched.

R code

```
27  str_extract(example.obj, "sm[abc]ll")
    [1] "small"
```

end

# Regular expressions in R

```
example.obj <- "1.  A small sentence.  - 2.  Another tiny sentence."
```

## Square brackets define character classes

Character classes help define special wild cards. The idea is that any of the characters within the brackets can be matched.

R code

```
29  str_extract(example.obj, "sm[abc]ll")
    [1] "small"
```

end

## The hyphen defines a range of characters

R code

```
30  str_extract(example.obj, "sm[a-p]ll")
    [1] "small"
```

end

# Regular expressions in R

```
example.obj <- "1.  A small sentence.  - 2.  Another tiny sentence."
```

## Predefined character classes

| | |
|---|---|
| `[:digit:]` | Digits: 0 1 2 3 4 5 6 7 8 9 |
| `[:lower:]` | Lower-case characters: a–z |
| `[:upper:]` | Upper-case characters: A–Z |
| `[:alpha:]` | Alphabetic characters: a–z and A–Z |
| `[:alnum:]` | Digits and alphabetic characters |
| `[:punct:]` | Punctuation characters: '.', ',', ';', etc. |
| `[:graph:]` | Graphical characters: `[:alnum:]` and `[:punct:]` |
| `[:blank:]` | Blank characters: Space and tab |
| `[:space:]` | Space characters: Space, tab, newline, and other space characters |
| `[:print:]` | Printable characters: `[:alnum:]`, `[:punct:]` and `[:space:]` |

```
example.obj <- "1.  A small sentence.  - 2.  Another tiny sentence."
```

## Predefined character classes in action

R code

```
31  unlist(str_extract_all(example.obj, "[:punct:]"))
    [1] "." "." "-" "." "."
32  unlist(str_extract_all(example.obj, "[:alpha:]"))
     [1] "A" "s" "m" "a" "l" "l" "s" "e" "n" "t" "e" "n" "c" "e" "A" "n" "o"
    [18] "t" "h" "e" "r" "t" "i" "n" "y" "s" "e" "n" "t" "e" "n" "c" "e"
```

end

# Regular expressions in R

`example.obj <- "1.  A small sentence.  - 2.  Another tiny sentence."`

## Predefined character classes are useful because they are efficient

- combine different kinds of characters
- facilitate reading of an expression
- include special characters, e.g., ß, ö, ...
- can be extended

R code

```
33 unlist(str_extract_all(example.obj, "[[:punct:]ABC]"))
   [1] "." "A" "." "-" "." "A" "."
34 unlist(str_extract_all(example.obj, "[^[:alnum:]]"))
    [1] "." " " " " " " " " "." " " " "-" " " " "." " " " " " " " "."
```

end

# Regular expressions in R

```
example.obj <- "1.  A small sentence.  - 2.  Another tiny sentence."
```

## Alternative character classes

| | |
|---|---|
| `\w` | Word characters: `[[:alnum:]_]` |
| `\W` | No word characters: `[^[:alnum:]_]` |
| `\s` | Space characters: `[[:blank:]]` |
| `\S` | No space characters: `[^[:blank:]]` |
| `\d` | Digits: `[[:digit:]]` |
| `\D` | No digits: `[^[:digit:]]` |
| `\b` | Word edge |
| `\B` | No word edge |
| `\<` | Word beginning |
| `\>` | Word end |

```
example.obj <- "1.  A small sentence.  - 2.  Another tiny sentence."
```

## Alternative character classes in action

R code

```
35  unlist(str_extract_all(example.obj, "\\w+"))
    [1] "1"        "A"        "small"    "sentence" "2"        "Another"
    [7] "tiny"     "sentence"

36  unlist(str_extract_all(example.obj, "e\\b"))
    [1] "e" "e"
```

end

# Regular expressions in R

`example.obj <- "1.  A small sentence.  - 2.  Another tiny sentence."`

## Use of quantifiers

| | |
|---|---|
| `?` | The preceding item is optional and will be matched at most once |
| `*` | The preceding item will be matched zero or more time |
| `+` | The preceding item will be matched one or more times |
| `{n}` | The preceding item is matched exactly n times |
| `{n,}` | The preceding item is matched n or more times |
| `{n,m}` | The preceding item is matched between n and m times |

R code

```
37 str_extract(example.obj, "s[[:alpha:]][[:alpha:]][[:alpha:]]l")
   [1] "small"
38 str_extract(example.obj, "s[[:alpha:]]{3}l")
   [1] "small"
39 str_extract(example.obj, "A.+sentence")
   [1] "A small sentence. - 2. Another tiny sentence"
```

end

# Regular expressions in R

`example.obj <- "1.  A small sentence.  - 2.  Another tiny sentence."`

## Greedy quantification

- the use of '`.+`' results in 'greedy' matching, i.e. the parser tries to match as many characters as possible
- not always desired, but '`.+?`' helps avoid greedy quantification

R code

```
40 str_extract(example.obj, "A.+sentence")
   [1] "A small sentence. - 2. Another tiny sentence"
41 str_extract(example.obj, "A.+?sentence")
   [1] "A small sentence"
```

end

# Regular expressions in R

`example.obj <- "1. A small sentence. - 2. Another tiny sentence."`

## Meta symbols in character classes

- within a *character class*, most meta symbols lose their special meaning
- exceptions: `^` and `-`
- `-` at the beginning or end matches the hyphen

R code

```
42 unlist(str_extract_all(example.obj, "[1-2]"))
   [1] "1" "2"
43 unlist(str_extract_all(example.obj, "[12-]"))
   [1] "1" "-" "2"
```

end

# Backreferencing

# Regular expressions in R

`example.obj <- "1. A small sentence. - 2. Another tiny sentence."`

## *Backreferencing*

- regular expression 'with memory'
- repeated match of previously matched pattern
- we refer to the first match (defined with round brackets) using `\1`, to the second match with `\2` etc. (up to 9)

R code

```
44 str_extract(example.obj, "([[:alpha:]]).+?\\1")
   [1] "A small sentence. - 2. A"
```

end

**Logic:** Match the first letter, then anything until you find the first letter again (not greedy)

# Regular expressions in R

```
example.obj <- "1.  A small sentence.  - 2.  Another tiny sentence."
```

## *Backreferencing:* a bit more complicated

**Goal:** match a word that does not include 'a' until the word appears the second time

# Regular expressions in R

```
example.obj <- "1.  A small sentence.  - 2.  Another tiny sentence."
```

## *Backreferencing:* a bit more complicated

**Goal:** match a word that does not include 'a' until the word appears the second time

**Solution:**

R code

```
46  str_extract(example.obj, "(\\b[b-z]+\\b).+?\\1")
    [1] "sentence. - 2. Another tiny sentence"
```

end

# Regular expressions in R

`example.obj <- "1. A small sentence. - 2. Another tiny sentence."`

## *Backreferencing:* a bit more complicated

**Goal:** match a word that does not include 'a' until the word appears the second time

**Solution:**

R code

```
47 str_extract(example.obj, "(\\b[b-z]+\\b).+?\\1")
   [1] "sentence. - 2. Another tiny sentence"
```

end

**Mechanic:**

1. match all letters without 'a': `[b-z]+`
2. match complete words (have beginning and end): `\\b`
3. refer to the word: `()`
4. match anything in between: `.+?\\1`

# Summary

# Summary

- regular expressions are magic
- regular expressions are non-trivial
- regular expressions are unreadable
- the good news: for scraping purposes, we can (and should!) rely on other, simpler and more robust tools
- still, regex can prove incredibly powerful under certain circumstances