String Basics in the tidyverse
Winter Institute in Data Science

Ryan T. Moore

2026-01-05

Strings

Basic String Tools

Regular Expressions

Data Types

- Numeric
- Integer
- Complex
- Logical
- **Character**
- Factor

Strings

String Data

- Candidate names, donor names, employers
- School names, addresses
- Precinct labels

Basic String Tools

```r
paste("x", "y")
```

```
## [1] "x y"
```

```r
paste("x", "y")
```

```
## [1] "x y"
```

```r
paste0("x", "y")
```

```
## [1] "xy"
```

```r
paste(c("x", "y"), " + 10")
```

```
## [1] "x  + 10" "y  + 10"
```

```r
paste(c("x", "y"), " + 10")
```

```
## [1] "x  + 10" "y  + 10"
```

```r
paste0(c("x", "y"), " + 10")
```

```
## [1] "x + 10" "y + 10"
```

```r
library(stringr)
```

Concatenate Strings

```r
library(stringr)
str_c("x", "y")
```

```
## [1] "xy"
```

```r
str_c(c("x", "y"), collapse = ", ")
```

```
## [1] "x, y"
```

Escaping

Escaping

Escaping

\

Escaping

\

If character means something special, must *escape* it to refer to it literally.

Escaping

In R,

to refer to. . .	You must type . . .
`"`	`\"`
`'`	`\'`
`\`	`\\`
`<newline>`	`\n`
`<return>`	`\r`
`<tab>`	`\t`

String length

```r
ch <- c("Dem", "Rep", "Indep")
str_length(ch)
```

```
## [1] 3 3 5
```

String length

```r
ch <- c("Hello", "Hi!", "Good day")
str_length(ch)
```

String length

```r
ch <- c("Hello", "Hi!", "Good day")
str_length(ch)
```

```
## [1] 5 3 8
```

Substrings

```r
ch <- c("Dem", "Rep", "Indepen")
str_sub(ch, 2, 5)
```

```
## [1] "em"    "ep"    "ndep"
```

Substrings

```
ch <- c("Hello", "Hi!", "Good day")
str_sub(ch, 3)
```

Substrings

```r
ch <- c("Hello", "Hi!", "Good day")
str_sub(ch, 3)
```

```
## [1] "llo"    "!"      "od day"
```

String case

```r
ch <- c("Dem", "Rep", "Indepen")
str_to_upper(ch)
```

```
## [1] "DEM"     "REP"     "INDEPEN"
```

String case

```r
ch <- c("Hello", "Hi!", "Good day")
str_to_lower(ch)
```

String case

```r
ch <- c("Hello", "Hi!", "Good day")
str_to_lower(ch)
```

```
## [1] "hello"    "hi!"      "good day"
```

Trimming whitespace

```r
ch <- c(" Dem", " Rep ", "Indepen  dent")
str_trim(ch)
```

Trimming whitespace

```
ch <- c(" Dem", " Rep ", "Indepen  dent")
str_trim(ch)
```

```
## [1] "Dem"           "Rep"           "Indepen  dent"
```

Trimming whitespace

```
ch <- c(" Dem", " Rep ", "Indepen  dent")
str_trim(ch)
```

```
## [1] "Dem"           "Rep"           "Indepen  dent"
```

```
str_squish(ch)
```

```
## [1] "Dem"          "Rep"          "Indepen dent"
```

Trimming whitespace

```
ch <- "Hello, Hi, and   Good day!  "
str_trim(ch)
```

Trimming whitespace

```
ch <- "Hello, Hi, and   Good day!  "
str_trim(ch)
```

```
## [1] "Hello, Hi, and   Good day!"
```

Trimming whitespace

```r
ch <- "Hello, Hi, and   Good day!  "
str_squish(ch)
```

Trimming whitespace

```r
ch <- "Hello, Hi, and   Good day!  "
str_squish(ch)
```

```
## [1] "Hello, Hi, and Good day!"
```

Sorting strings

```r
ch <- c("Dem", " Rep", "Independent")
str_sort(ch, locale = "en")
```

Sorting strings

```
ch <- c("Dem", " Rep", "Independent")
str_sort(ch, locale = "en")
```

```
## [1] " Rep"        "Dem"         "Independen
```

Sorting strings

```r
ch <- c("Hello", "Hi!", "Good day")
str_sort(ch)
```

Sorting strings

```r
ch <- c("Hello", "Hi!", "Good day")
str_sort(ch)
```

```
## [1] "Good day" "Hello"    "Hi!"
```

Regular Expressions

Regular Expressions

A sequence of characters defining a string pattern.

Regular Expressions

A sequence of characters defining a string pattern.

We define a regex, then find matches to the pattern in our strings.

Regular Expressions: Is this interesting?

Regular Expressions: Is this interesting?

Yes.

Regular Expressions: Is this interesting?

Yes.

In March 2021, Russia's state Internet censor wanted to block Twitter URL shortener domain `t.co`.

Regular Expressions: Is this interesting?

Yes.

In March 2021, Russia's state Internet censor wanted to block Twitter URL shortener domain `t.co`.

Clumsy regular expression blocked

- `microsoft.com`

Regular Expressions: Is this interesting?

Yes.

In March 2021, Russia's state Internet censor wanted to block Twitter URL shortener domain `t.co`.

Clumsy regular expression blocked

- `microsoft.com`
- `reddit.com`

Regular Expressions: Is this interesting?

Yes.

In March 2021, Russia's state Internet censor wanted to block Twitter URL shortener domain `t.co`.

Clumsy regular expression blocked

- `microsoft.com`
- `reddit.com`
- `rt.com` (. . . Russian state media!)

Regular Expressions: Is this interesting?

Yes.

In March 2021, Russia's state Internet censor wanted to block Twitter URL shortener domain `t.co`.

Clumsy regular expression blocked

- `microsoft.com`
- `reddit.com`
- `rt.com` (…Russian state media!)

Regular Expressions: Is this interesting?

Yes.

In March 2021, Russia's state Internet censor wanted to block Twitter URL shortener domain `t.co`.

Clumsy regular expression blocked

- `microsoft.com`
- `reddit.com`
- `rt.com` (…Russian state media!)

https://whyisthisinteresting.substack.com/p/the-regular-expression-edition

Use `str_view()`/`str_view_all()` to locate matches visually.

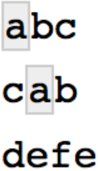
Use `str_view`()/`str_view_all()` to locate matches visually. (Test your regex before deploying it).

Use `str_view()`/`str_view_all()` to locate matches visually. (Test your regex before deploying it).

```r
ch <- c("abc", "cab", "defe")
str_view(ch, "a")
```

Use `str_view()`/`str_view_all()` to locate matches visually. (Test your regex before deploying it).

```r
ch <- c("abc", "cab", "defe")
str_view(ch, "a")
```



```
abc
cab
defe
```

Regex chars are either *literal* or *meta*.

Regex chars are either *literal* or *meta*.

So far, all literal matches.

Regex chars are either *literal* or *meta*.

So far, all literal matches.

Metacharacters match something else, unless you escape them.

Regex Metacharacters

Metachar	Meaning
`.`	any char
`\`	escape
`|`	or
`^`	starts with (`^a`)
`$`	ends with (`z$`)
`[ ]`	any of
`[^ ]`	none of
`[ - ]`	from … to …
`( )`	group
`?`	0 or 1 match
`*`	0 or more matches
`+`	1 or more matches
`{n}`	`n` matches
`{n,}`	$\geq$ `n` matches
`{,m}`	$\leq$ `m` matches

Regex Shorthands

Regex	Meaning
`\b`	word *boundary*
`\d`	any digit
`\D`	any non-digit
`\s`	any whitespace char
`\S`	any non-whitespace char
`\w`	any alpha-numeric char
`\W`	any non-alpha-numeric char

Example

Find all words in `words` that have second letter `x`:

Example

Find all words in `words` that have second letter `x`:

```
words[str_detect(words, "\\b.x")]
```

Example

Find all words in `words` that have second letter `x`:

```
words[str_detect(words, "\\b.x")]
```

```
##  [1] "exact"       "example"     "except"      "excuse"
##  [6] "exist"       "expect"      "expense"     "experie
## [11] "express"     "extra"
```

`str_count()` returns number of matches:

```r
str_count(c("aab2", "a1b2"), "a")
```

`str_count()` returns number of matches:

```r
str_count(c("aab2", "a1b2"), "a")
```

```
## [1] 2 1
```

`str_subset()` returns only the strings that have a match.

`str_subset()` returns only the strings that have a match.

```r
str_subset(c("aab2", "a1b2"), "a1")
```

```
## [1] "a1b2"
```

```r
str_subset(c("aab2", "a1b2"), ".[0-9].")
str_subset(c("aab2", "a1b2"), ".[0-9].")
```

```
## [1] "a1b2"
```

`str_extract()` returns only the matching parts.

`str_extract()` returns only the matching parts.

```r
str_extract(c("aab2", "a1b2"), "a1")
```

```
## [1] NA   "a1"
```

```r
str_extract(c("aab2", "a1b2"), "a")
str_extract(c("aab2", "a1b2"), "a")
```

```
## [1] "a" "a"
```

```
str_extract_all(c("aab2", "a1b2"), "a")
```

```r
str_extract_all(c("aab2", "a1b2"), "a")
```

```
## [[1]]
## [1] "a" "a"
##
## [[2]]
## [1] "a"
```

`str_match()` is like `str_extract()`, but returns components of the match separately.

`str_match()` is like `str_extract()`, but returns components of the match separately.

```
str(sentences)
```

```
##  chr [1:720] "The birch canoe slid on the smooth pla
```

`str_match()` is like `str_extract()`, but returns components of the match separately.

```
str(sentences)
```

```
##  chr [1:720] "The birch canoe slid on the smooth pla
```

```
a_phr <- str_subset(sentences, "\\b[Aa] ([^ ]+)")
```

str_match() is like str_extract(), but returns components of the match separately.

```
str(sentences)
```

```
##  chr [1:720] "The birch canoe slid on the smooth pla
```

```
a_phr <- str_subset(sentences, "\\b[Aa] ([^ ]+)")
```

```
str_match(a_phr, "\\b([Aa]) ([^ ]+)")
```

```
##         [,1]          [,2] [,3]
##    [1,] "a well."     "a"  "well."
##    [2,] "a chicken"   "a"  "chicken"
##    [3,] "A large"     "A"  "large"
##    [4,] "A rod"       "A"  "rod"
##    [5,] "A pot"       "A"  "pot"
##    [6,] "a hole"      "a"  "hole"
##    [7,] "a button"    "a"  "button"
##    [8,] "A king"      "A"  "king"
```

`str_replace()` replaces first match in string.

`str_replace()` replaces first match in string.

```
str_replace(words[1:10], "s", "*")
```

```
##  [1] "a"          "able"       "about"      "ab*olute" "ac
##  [7] "achieve"  "acro*s"    "act"        "active"
```

```r
str_replace_all(words[1:10], "s", "*")
str_replace_all(words[1:10], "s", "*")
```

```
##  [1] "a"          "able"       "about"      "ab*olute"   "accept
##  [7] "achieve"  "acro**"   "act"        "active"
```

Split strings

`str_split()` returns a list or matrix of components.

Split strings

str_split() returns a list or matrix of components.

```
str_split(words[1:10], "c")
```

```
## [[1]]
## [1] "a"
##
## [[2]]
## [1] "able"
##
## [[3]]
## [1] "about"
##
## [[4]]
## [1] "absolute"
##
## [[5]]
## [1] "a"   ""    "ept"
```

Split strings

```r
str_split(words[1:10], "c", simplify = TRUE)
```

```
##       [,1]       [,2]    [,3]  
##  [1,] "a"        ""      ""    
##  [2,] "able"     ""      ""    
##  [3,] "about"    ""      ""    
##  [4,] "absolute" ""      ""    
##  [5,] "a"        ""      "ept" 
##  [6,] "a"        ""      "ount"
##  [7,] "a"        "hieve" ""    
##  [8,] "a"        "ross"  ""    
##  [9,] "a"        "t"     ""    
## [10,] "a"        "tive"  ""    
```

Exercises §14.4.3.1

2. From the Harvard sentences data, extract:

a) The first word from each sentence.
b) All words ending in `ing`.
c) All plurals.