

# String Manipulation With glue and stringr



# Outline

Intro to glue and stringr

String “injection” with glue

- Basics of glue: glue() and friends

- glue in the wild! (At least how I use it...)

String manipulation with stringr

- Detecting, Lengthening, Joining/Splitting, Subsetting, Mutating, Sorting/Ordering

- Regex's

- Data cleaning example

# glue and stringr

glue and stringr are tidyverse packages for string injection and manipulation

String manipulation is essential for data science

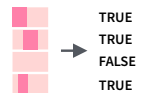
We will pay special attention to *regular expressions*: strings that signify special patterns of characters in other strings

# Work with strings with stringr : : CHEAT SHEET

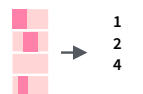


The **stringr** package provides a set of internally consistent tools for working with character strings, i.e. sequences of characters surrounded by quotation marks.

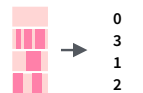
## Detect Matches



**str\_detect**(string, **pattern**) Detect the presence of a pattern match in a string.  
`str_detect(fruit, "a")`



**str\_which**(string, **pattern**) Find the indexes of strings that contain a pattern match.  
`str_which(fruit, "a")`



**str\_count**(string, **pattern**) Count the number of matches in a string.  
`str_count(fruit, "a")`



**str\_locate**(string, **pattern**) Locate the positions of pattern matches in a string. Also **str\_locate\_all**.  
`str_locate(fruit, "a")`

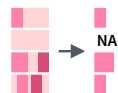
## Subset Strings



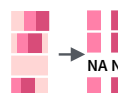
**str\_sub**(string, start = 1L, end = -1L) Extract substrings from a character vector.  
`str_sub(fruit, 1, 3); str_sub(fruit, -2)`



**str\_subset**(string, **pattern**) Return only the strings that contain a pattern match.  
`str_subset(fruit, "b")`

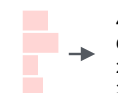


**str\_extract**(string, **pattern**) Return the first pattern match found in each string, as a vector. Also **str\_extract\_all** to return every pattern match.  
`str_extract(fruit, "[aeiou]")`



**str\_match**(string, **pattern**) Return the first pattern match found in each string, as a matrix with a column for each ( ) group in pattern. Also **str\_match\_all**.  
`str_match(sentences, "(a|the) ([^ ]+)")`

## Manage Lengths



**str\_length**(string) The width of strings (i.e. number of code points, which generally equals the number of characters). `str_length(fruit)`



**str\_pad**(string, width, side = c("left", "right", "both"), pad = " ") Pad strings to constant width. `str_pad(fruit, 17)`



**str\_trunc**(string, width, side = c("right", "left", "center"), ellipsis = "...") Truncate the width of strings, replacing content with ellipsis. `str_trunc(fruit, 3)`

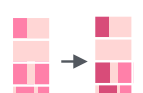


**str\_trim**(string, side = c("both", "left", "right")) Trim whitespace from the start and/or end of a string. `str_trim(fruit)`

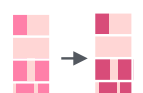
## Mutate Strings



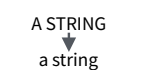
**str\_sub()** <- value. Replace substrings by identifying the substrings with `str_sub()` and assigning into the results.  
`str_sub(fruit, 1, 3) <- "str"`



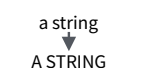
**str\_replace**(string, **pattern**, replacement) Replace the first matched pattern in each string. `str_replace(fruit, "a", "-")`



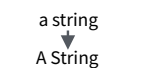
**str\_replace\_all**(string, **pattern**, replacement) Replace all matched patterns in each string. `str_replace_all(fruit, "a", "-")`



**str\_to\_lower**(string, locale = "en")<sup>1</sup> Convert strings to lower case.  
`str_to_lower(sentences)`

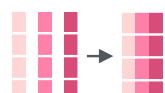


**str\_to\_upper**(string, locale = "en")<sup>1</sup> Convert strings to upper case.  
`str_to_upper(sentences)`



**str\_to\_title**(string, locale = "en")<sup>1</sup> Convert strings to title case. `str_to_title(sentences)`

## Join and Split



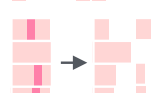
**str\_c**(..., sep = "", collapse = NULL) Join multiple strings into a single string.  
`str_c(letters, LETTERS)`



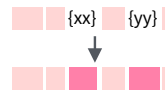
**str\_c**(..., sep = "", collapse = NULL) Collapse a vector of strings into a single string.  
`str_c(letters, collapse = "")`



**str\_dup**(string, times) Repeat strings times times. `str_dup(fruit, times = 2)`



**str\_split\_fixed**(string, **pattern**, n) Split a vector of strings into a matrix of substrings (splitting at occurrences of a pattern match). Also **str\_split** to return a list of substrings.  
`str_split_fixed(fruit, " ", n=2)`



**glue::glue**(..., .sep = "", .envir = parent.frame(), .open = "{", .close = "}") Create a string from strings and {expressions} to evaluate. `glue::glue("Pi is {pi}")`



**glue::glue\_data**(.x, ..., .sep = "", .envir = parent.frame(), .open = "{", .close = "}") Use a data frame, list, or environment to create a string from strings and {expressions} to evaluate. `glue::glue_data(mtcars, "{rownames(mtcars)} has {hp} hp")`

## Order Strings



**str\_order**(x, decreasing = FALSE, na\_last = TRUE, locale = "en", numeric = FALSE, ...) <sup>1</sup> Return the vector of indexes that sorts a character vector. `x[str_order(x)]`



**str\_sort**(x, decreasing = FALSE, na\_last = TRUE, locale = "en", numeric = FALSE, ...) <sup>1</sup> Sort a character vector. `str_sort(x)`

## Helpers

apple  
banana  
pear

**str\_conv**(string, encoding) Override the encoding of a string. `str_conv(fruit, "ISO-8859-1")`

apple  
banana  
pear

**str\_view**(string, **pattern**, match = NA) View HTML rendering of first regex match in each string. `str_view(fruit, "[aeiou]")`

**str\_view\_all**(string, **pattern**, match = NA) View HTML rendering of all regex matches. `str_view_all(fruit, "[aeiou]")`

**str\_wrap**(string, width = 80, indent = 0, exdent = 0) Wrap strings into nicely formatted paragraphs. `str_wrap(sentences, 20)`

<sup>1</sup> See [bit.ly/ISO639-1](https://bit.ly/ISO639-1) for a complete list of locales.

# glue

glue's main function is `glue()` which creates strings and can inject R objects into strings

`glue()` has two arguments we are interested in:

1. String(s) with ...
2. How to separate the elements with `sep`

```
glue("Hello, my name is Grant Innerst")
```

```
Hello, my name is Grant Innerst
```

You “inject” R code into the string with `{ }`

```
first_name <- "Grant"; last_name <- "Innerst"
```

```
glue("Hello, my name is {first_name} {last_name}")
```

```
Hello, my name is Grant Innerst
```

# glue

```
name <- "Grant"  
age <- 29  
anniversary <- as.Date("2018-07-14")
```

```
glue('My name is {name},',  
     ' my age next year is {age + 1},',  
     ' my anniversary is {format(anniversary, "%A, %B %d, %Y")}.'.')
```

My name is Grant, my age next year is 30, my anniversary is Saturday,  
July 14, 2018.

# glue

glue has other functions that perform more specific tasks

`glue_data()` works well with pipes and exposes LHS names

`glue_collapse()` collapses a character vector into one string

`glue_col()` colors console output of strings

etc....

# glue in the wild!

Scraping weather data for some NFL data analysis.



# stringr

stringr is much more extensive package for string manipulation

stringr functions will almost always start with str\_

stringr functions are wrappers of a very fast C++ library called stringi. stringi is powerful but complicated

stringr functions can be grouped into six categories

- Detecting
- Lengthening
- Joining/Splitting
- Subsetting
- Mutating
- Sorting/Ordering

# Detecting

The fruit vector:

```
glimpse(fruit)
```

```
chr [1:80] "apple" "apricot" "avocado" "banana" "bell pepper"  
"bilberry" ...
```

```
str_detect(fruit, "apple")
```

```
[1] TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
[10] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
[19] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
[28] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
[37] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
[46] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
[55] FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE  
[64] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
[73] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

# Detecting

```
fruit %>% str_which("berry")
```

```
[1] 6 7 10 11 19 21 29 32 33 38 50 70 73 76
```

```
fruit %>% str_count("a")
```

```
[1] 1 1 2 3 0 0 1 2 1 0 0 1 2 2 1 0 0 0 0 0 1 0 1 1 1 1 1 1  
[29] 0 1 0 0 0 1 1 2 0 0 1 1 0 0 1 0 0 1 0 2 1 0 1 0 0 1 1 3  
[57] 1 1 1 0 1 1 0 2 0 1 0 1 2 1 1 0 2 2 1 1 2 1 0 1
```

# Subsetting

```
fruit %>% str_sub(2, 4)
```

```
[1] "ppl" "pri" "voc" "ana" "ell" "ilb" "lac" "lac" "loo" "lue" "oys" "rea" "ana" "ant" "her"  
[16] "her" "hil" "lem" "lou" "oco" "ran" "ucu" "urr" "ams" "ate" "rag" "uri" "ggp" "lde" "eij"  
[31] "ig" "oji" "oos" "rap" "rap" "uav" "one" "uck" "ack" "amb" "uju" "iwi" "umq" "emo" "ime"  
[46] "oqu" "ych" "and" "ang" "ulb" "ect" "ut" "liv" "ran" "ame" "apa" "ass" "eac" "ear" "ers"  
[61] "hys" "ine" "lum" "ome" "ome" "urp" "uin" "ais" "amb" "asp" "edc" "ock" "ala" "ats" "tar"  
[76] "tra" "ama" "ang" "gli" "ate"
```

```
fruit %>% str_sub(-2)
```

```
[1] "le" "ot" "do" "na" "er" "ry" "ry" "nt" "ge" "ry" "ry" "it" "on" "pe" "ya"  
[16] "ry" "er" "ne" "ry" "ut" "ry" "er" "nt" "on" "te" "it" "an" "nt" "ry" "oa"  
[31] "ig" "ry" "ry" "pe" "it" "va" "ew" "ry" "it" "ul" "be" "it" "at" "on" "me"  
[46] "at" "ee" "ne" "go" "ry" "ne" "ut" "ve" "ge" "lo" "ya" "it" "ch" "ar" "on"  
[61] "is" "le" "um" "te" "lo" "en" "ce" "in" "an" "ry" "nt" "on" "ry" "ma" "it"  
[76] "ry" "lo" "ne" "it" "on"
```

# Subsetting

```
fruit %>% str_subset("apple")
```

```
[1] "apple"      "pineapple"
```

```
fruit %>% str_extract("app")
```

[1]	"app"	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
[12]	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
[23]	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
[34]	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
[45]	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
[56]	NA	NA	NA	NA	NA	NA	"app"	NA	NA	NA	NA
[67]	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
[78]	NA	NA	NA								

# Lengthening

```
fruit[1:5]
```

```
[1] "apple" "apricot" "avocado" "banana" "bell pepper"
```

```
fruit[1:5] %>% str_length()
```

```
[1] 5 7 7 6 11
```

```
fruit[1:5] %>% str_pad(11L)
```

```
[1] "      apple" "      apricot" "      avocado" "      banana"
[5] "bell pepper"
```

You can pad from the left or right and with whatever symbol you want

# Lengthening

```
fruit[1:5] %>% str_trunc(6L)
```

```
[1] "apple" "apr..." "avo..." "banana" "bel..."
```

```
fruit[1:5] %>% str_pad(11L) %>% str_trim()
```

```
[1] "apple" "apricot" "avocado" "banana" "bell pepper"
```

# Mutating

```
fruit[1:5] %>% str_replace("p", "_")
```

```
[1] "a_ple"      "a_ricot"    "avocado"    "banana"     "bell _epper"
```

```
fruit[1:5] %>% str_replace_all("p", "_")
```

```
[1] "a__le"      "a_ricot"    "avocado"    "banana"     "bell _e__er"
```

```
fruit[1:5] %>% str_to_upper()
```

```
[1] "APPLE"      "APRICOT"    "AVOCADO"    "BANANA"     "BELL PEPPER"
```

```
fruit[1:5] %>% str_to_title()
```

```
[1] "Apple"      "Apricot"    "Avocado"    "Banana"     "Bell Pepper"
```



# Joining and Splitting

```
fruit[1:5] %>% str_c(collapse = " ")
```

```
[1] "apple apricot avocado banana bell pepper"
```

```
fruit[1:5] %>% str_c(rep("x_", 5), .)
```

```
[1] "x_apple" "x_apricot" "x_avocado" "x_banana" "x_bell pepper"
```

```
fruit[1:5] %>% str_c(collapse = " ") %>% str_split(" ")
```

```
[[1]]
```

```
[1] "apple" "apricot" "avocado" "banana" "bell"
```

```
[6] "pepper"
```

# Sorting and Ordering

```
fruit[1:5] %>% str_sort(decreasing = TRUE)
```

```
[1] "bell pepper" "banana"    "avocado"   "apricot"   "apple"
```

```
fruit[1:5] %>% str_order()
```

```
[1] 1 2 3 4 5
```