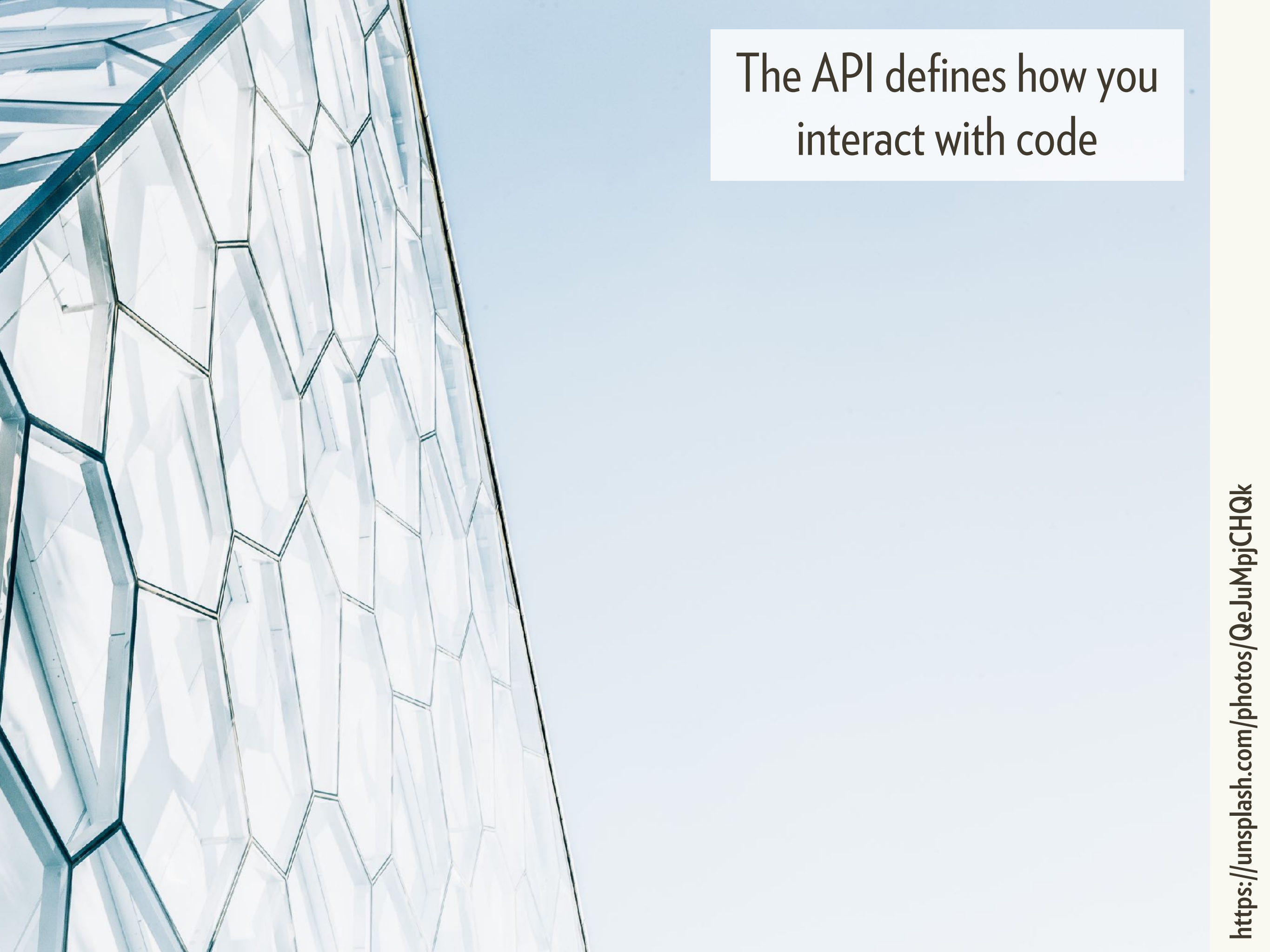


API design

January 2019

Hadley Wickham
[@hadleywickham](#)
Chief Scientist, RStudio



The API defines how you
interact with code



The *interface*, not
the internals

Case study

What makes base R functions hard to learn?

strsplit(x, split, ...)

grep(pattern, x, value = FALSE, ...)

grep1(pattern, x, ...)

sub(pattern, replacement, x, ...)

gsub(pattern, replacement, x, ...)

regexpr(pattern, text, ...)

gregexpr(pattern, text, ...)

regexec(pattern, text, ...)

substr(x, start, stop)

nchar(x, type, ...)

A few issues

Names: Function names have no common theme, and no common prefix. Names are concise at expense of expressiveness.

Arguments: Argument names & order are not consistent, and data isn't the first argument (2nd or 3rd). Sometimes text, sometimes x.

Type stability: `grep()` is not type stable: can return string or integer. Can't feed output of `grepexpr()` into `substr()`



Each individual problem is small



<https://stringr.tidyverse.org>

“Each [function] is perfect the way it is ... and it can use a little improvement.”

—*Shunryu Suzuki*

Carefully
contemplate names

“A rose by any other name
would smell as sweet.”
— *Shakespeare*



“A **function** by any other name
would **not** smell as sweet.”

— *Hadley*



Principle:

Use verbs for action
functions

stringr uses evocative verbs

`str_split()`

`str_detect()`

`str_locate()`

`str_subset()`

`str_extract()`

`str_replace()`

But good verbs don't always exist

`str_to_lower()`

`str_to_upper()`

ggplot2 uses nouns

`geom_line()`

`scale_x_continuous()`

`coord_fixed()`

Past mistakes

Avoid verbs with dual meanings

`filter()`

`weather()`

`cleave()`

Avoid words with UK/US variants

`summarise()` / `summarize()`

`scale_colour_grey()` / `scale_color_grey()`

Principle:

Use prefixes to group
related functions together

Most stringr functions start with str_

`str_split()`

`str_detect()`

`str_locate()`

`str_replace()`

...

Principle:

Use suffixes for variations
on a theme

Use suffixes for variations on a theme

```
str_extract()
```

```
str_extract_all()
```

```
str_replace()
```

```
str_replace_all()
```

```
str_split()
```

```
str_split_fixed()
```

```
# Why not arguments?
```

```
str_extract(all = TRUE)
```

```
str_split(fixed = TRUE)
```

Your turn

What stringr functions violate these principles?

What other tidyverse functions violate these principles?

```
# Don't start with str_  
invert_match()  
word()  
fixed()  
regexp()
```

```
# Aren't verbs  
str_which()  
str_c()  
str_length()  
str_sub()
```

Plan for the pipe

Why is the pipe useful?

```
library(dplyr)
library(nycflights13)

by_dest <- group_by(flights, dest)
dest_delay <- summarise(by_dest,
  delay = mean(dep_delay, na.rm = TRUE),
  n = n()
)
big_dest <- filter(dest_delay, n > 100)
arrange(big_dest, desc(delay))
```

But naming is hard work

```
foo <- group_by(flights, dest)
foo <- summarise(foo,
  delay = mean(dep_delay, na.rm = TRUE),
  n = n()
)
foo <- filter(foo, n > 100)
arrange(foo, desc(delay))
```

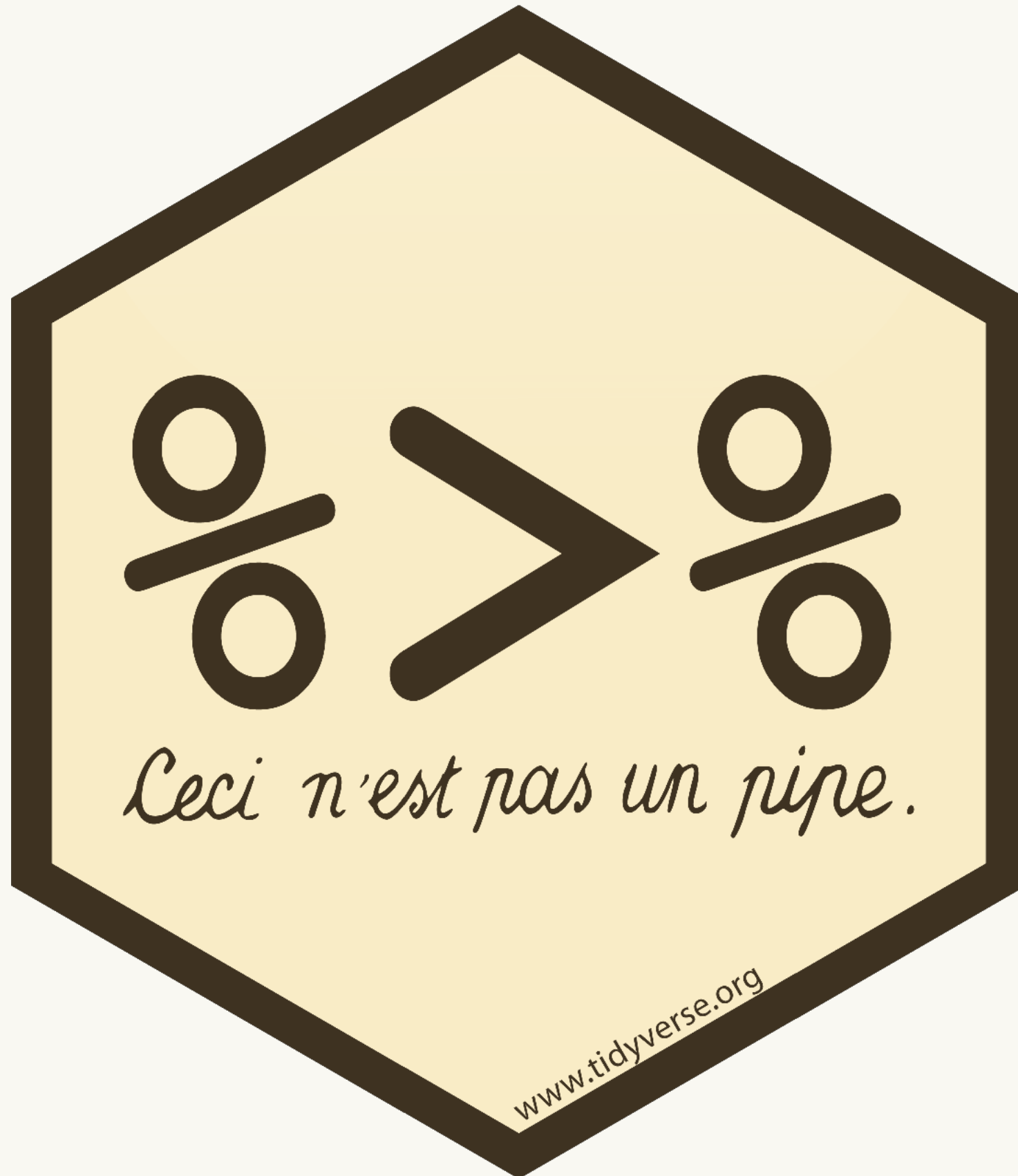
But naming is hard work

```
foo1 <- group_by(flights, dest)
foo2 <- summarise(foo1,
  delay = mean(dep_delay, na.rm = TRUE),
  n = n()
)
foo3 <- filter(foo2, n > 100)
arrange(foo2, desc(delay))
```

Alternatively, you *could* nest function calls

```
arrange(  
  filter(  
    summarise(  
      group_by(flights, dest),  
      delay = mean(dep_delay, na.rm = TRUE),  
      n = n()  
    ),  
    n > 100  
  ),  
  desc(delay)  
)
```


magrittr provides a third option



No intermediaries; read from left-to-right

```
flights %>%  
  group_by(dest) %>%  
  summarise(  
    delay = mean(dep_delay, na.rm = TRUE),  
    n = n()  
  ) %>%  
  filter(n > 100) %>%  
  arrange(desc(delay))
```

	Read left-to-right	Can omit intermediate names	Non-linear
$y \leftarrow f(x)$ $g(y)$	✓		✓
$g(f(x))$		✓	✓
$x \%>\%$ $f() \%>\%$ $g()$	✓	✓	

Principle:

Data arguments should
come first

Most arguments fall in one of two classes

Data	Details
Required	Optional
Core data	Additional options
Often vectorised	Scalar
Often called x or data	Names are important

These affect how you call functions

Omit names of the data arguments

```
ggplot(mtcars, aes(x = disp, y = cyl))
```

Not

```
ggplot(data = mtcars, mapping = aes(...))
```

Provide names of details arguments

```
mean(1:10, na.rm = TRUE)
```

Not

```
mean(1:10, , TRUE)
```

Never use partial names

Your turn

Which are the data arguments in `grepl()`?

Which are the details?

Which are the data arguments in `strsplit()`?

Which are the details?

Which are the data arguments in `substr()`?

Which are the details?

Which are the data arguments in `merge()`?

Which are the details?

In pipe, use . to override position

```
x %>%
```

```
  str_replace("a", "A") %>%
```

```
  str_replace("b", "B")
```

```
x %>%
```

```
  gsub("a", "A", .) %>%
```

```
  gsub("b", "B", .)
```


Principle:

Match outputs and inputs

Your turn

```
x <- c("bbaab", "bbb", "bbaaba")  
loc <- regexpr("a+", x)
```

```
# What does regexpr() return? What data  
# structure does it use?
```

```
# How do you use substr() with the result  
# of regexpr() to extract the match?
```

Output of regexp() not compatible with substr()

```
x <- c("bbaab", "bbb", "bbaaba")
```

```
regexpr("a+", x)
```

```
loc <- regexpr("a", x)
```

```
substr(x, loc, loc + attr(loc, "match.length"))
```

```
# And only works because this returns ""
```

```
substr(x, -1, -2)
```

```
# regmatches() has a different problem
```

```
regmatches(x, loc)
```

Equivalent stringr code is much simpler

```
x <- c("bbaab", "bbb", "bbaaba")  
str_sub(x, str_locate(x, "a+"))
```

```
# All matches
```

```
loc <- str_locate_all(x, "a+")  
map2(x, loc, str_sub)
```

Principle:
Type stability



Why doesn't `str_replace()` use an argument?

Instead of suffixes

`str_replace()`

`str_replace_all()`

could use an argument

`str_replace(n = 1)`

`str_replace(n = Inf)`

which generalises better

`str_replace(n = 2)`

`str_replace(n = -1)`

but we want `str_replace()` to be

consistent with `str_extract()`

str_extract() vs. str_extract_all()?

```
strings <- c("x1 y", "x2 y x3", "z")
```

```
str_extract(strings, "x.")
```

```
#> [1] "x1" "x2" NA
```

```
str_extract_all(strings, "x.")
```

```
#> [[1]]
```

```
#> [1] "x1"
```

```
#>
```

```
#> [[2]]
```

```
#> [1] "x2" "x3"
```

```
#>
```

```
#> [[3]]
```

```
#> character(0)
```

And we want functions to have consistent outputs

```
str_extract(x, pattern, n = 1)
```

```
# character vector?
```

```
str_extract(x, pattern, n = 2)
```

```
# character matrix?
```

```
str_extract(x, pattern, n = Inf)
```

```
# list?
```

```
x <- str_extract(x, pattern, n = n)
```

```
# What is x?
```


This work is licensed under the
Creative Commons Attribution-Noncommercial 3.0
United States License.

To view a copy of this license, visit
<http://creativecommons.org/licenses/by-nc/3.0/us/>