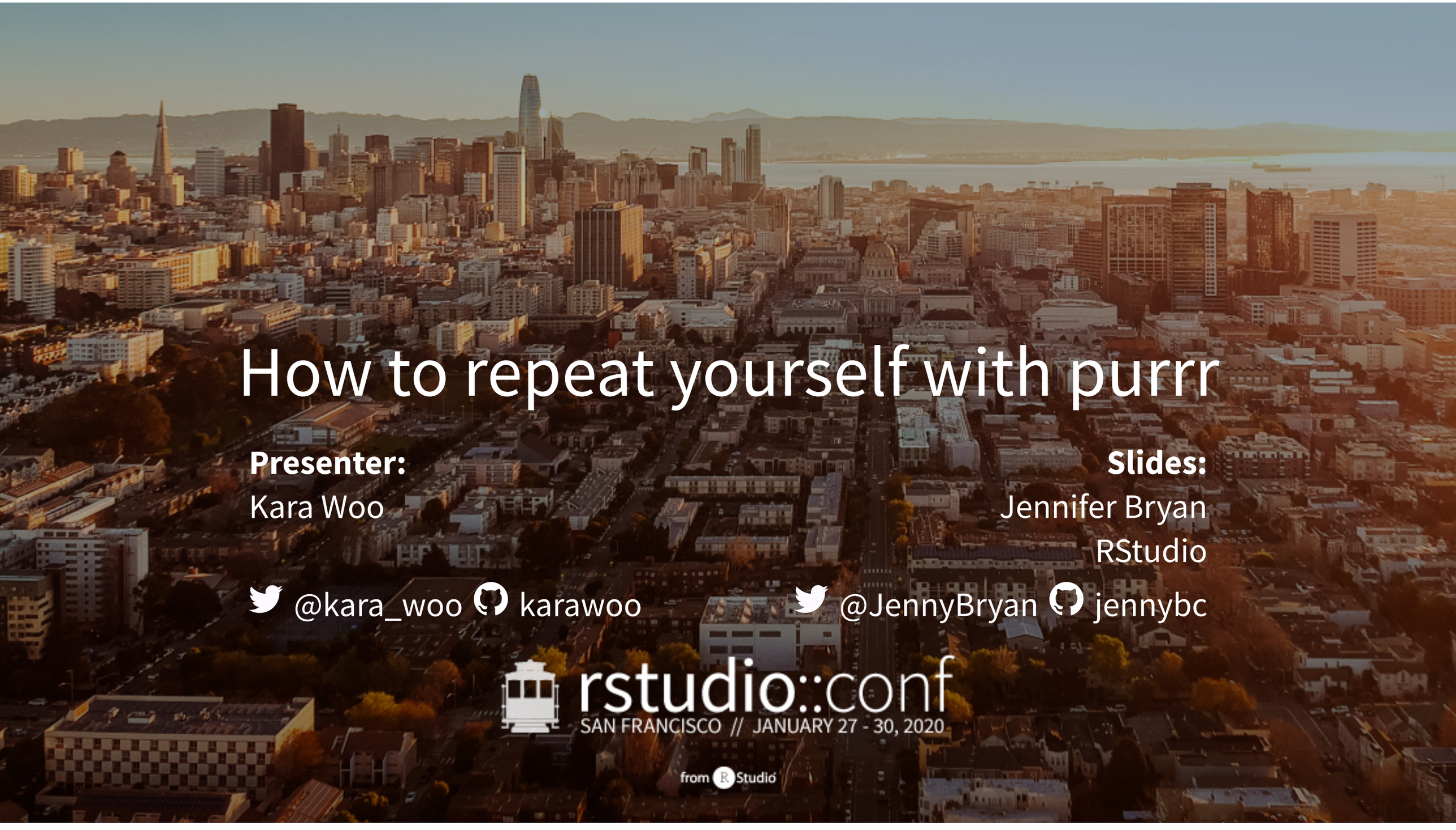

How to repeat yourself with purrr
Presenter:
Kara Woo
@kara_woo karawoo
Slides:
Jennifer Bryan
RStudio
@JennyBryan jennybc
rstudio::conf
SAN FRANCISCO // JANUARY 27 - 30, 2020
from RStudio

This work is licensed under a
Creative Commons
Attribution-ShareAlike 4.0
International License.

To view a copy of this license, visit
http://creativecommons.org/licenses/by-sa/4.0/

bit.ly/rladies-wtf-2020

R installed? Pretty recent?

- Current version: 3.6.2

RStudio installed?

- Current Preview: 1.2.5036

Have these packages?

- tidyverse (includes purrr)
- repurrrsive

Get some help NOW if you need/want to do some setup during the intro!

bit.ly/kara-purrr-2020-rladies

Resources

My purrr materials:
https://jennybc.github.io/purrr-tutorial/

Charlotte Wickham's purrr materials:
https://github.com/cwickham/purrr-tutorial

My "row-oriented workflows" materials:
rstd.io/row-work

"Functionals" chapter of 2nd of Advanced R by Wickham
https://adv-r.hadley.nz/functionals.html

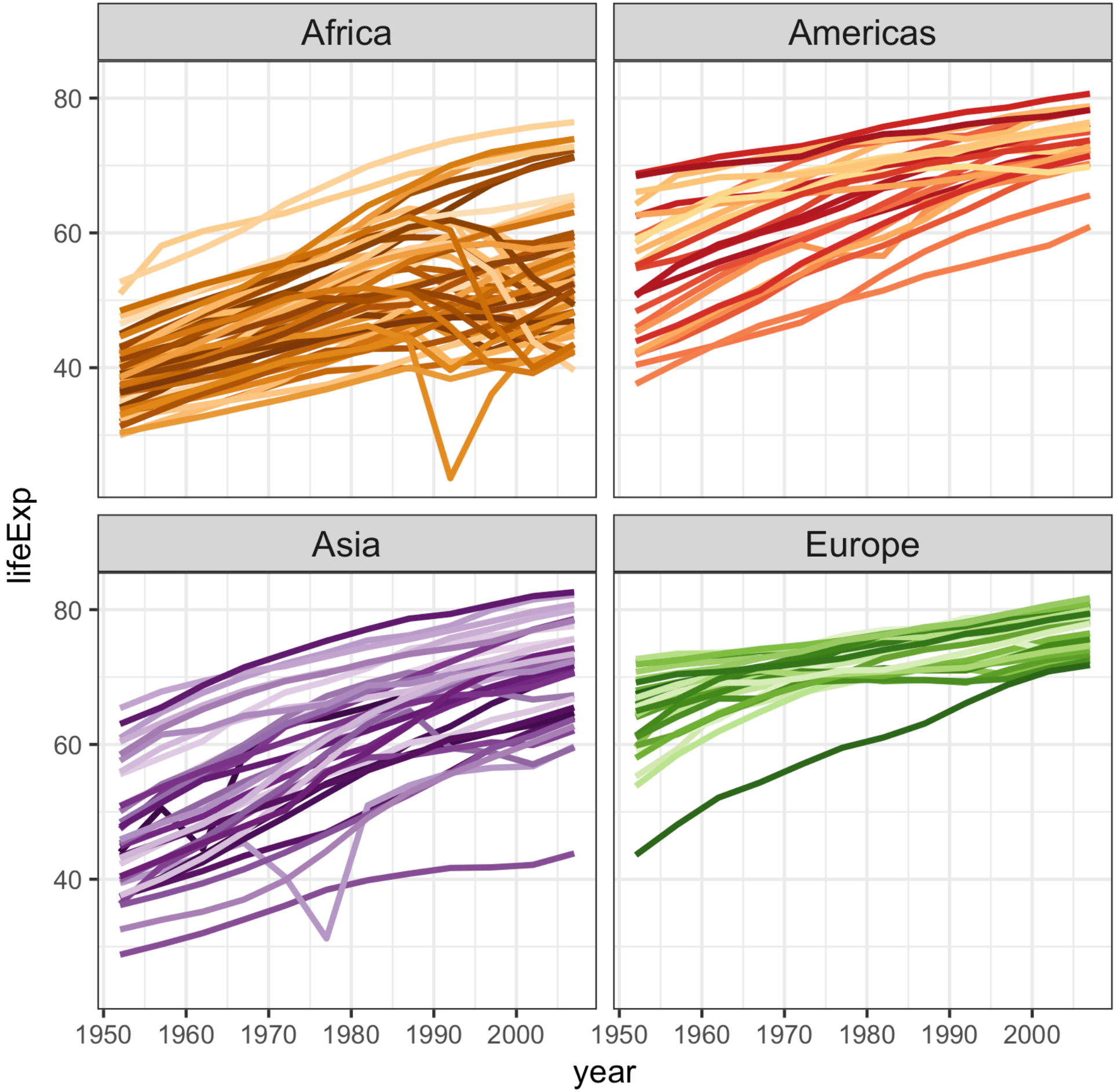
1. What is the harm with copy/paste and repetitive code?

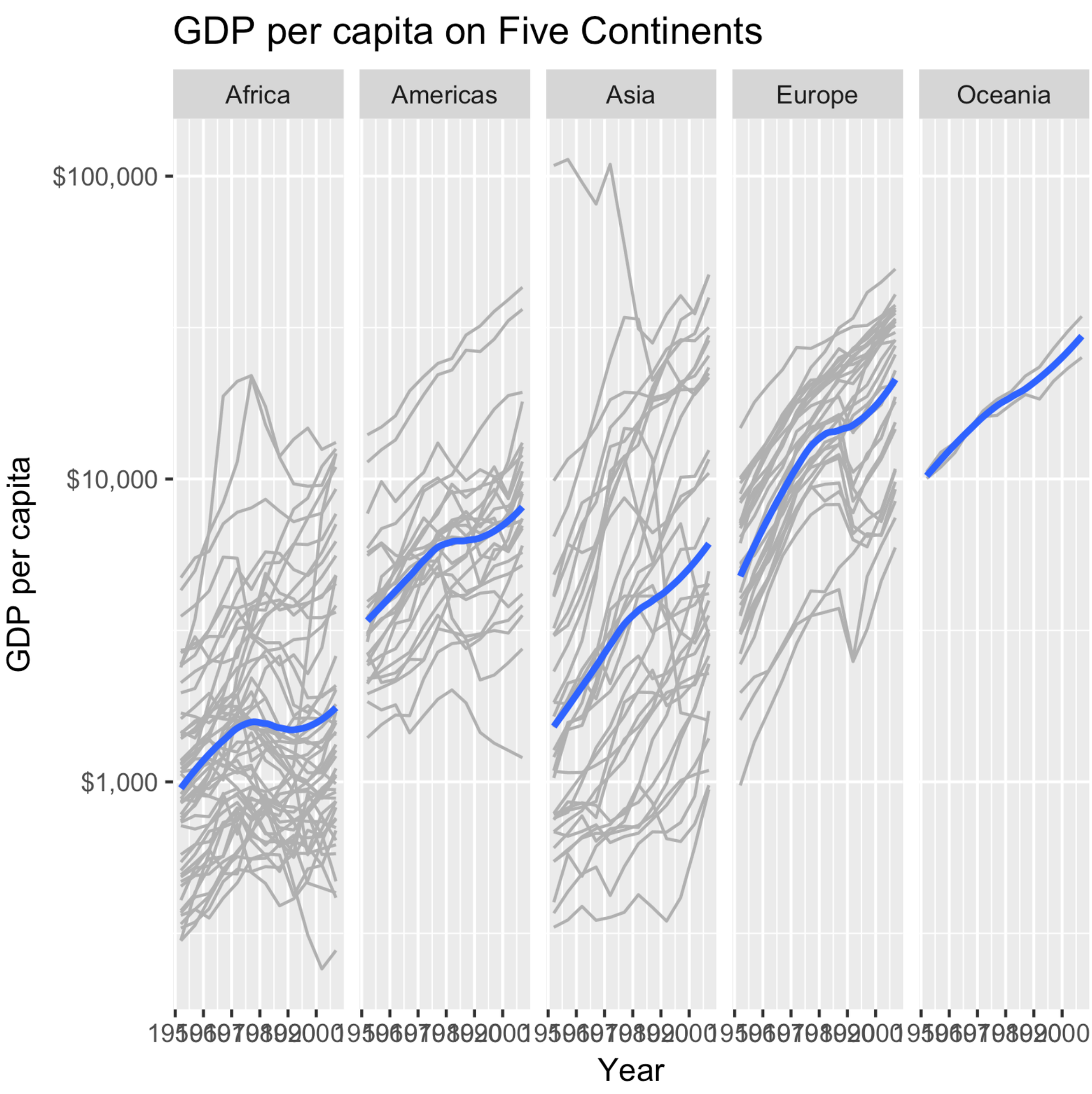
2. What should I do instead?
   - write functions
   - use formal tools to iterate the R way

3. Hands-on practice with the purrr package for iteration

```
library(gapminder)
library(tidyverse)

gapminder
#> # A tibble: 1,704 x 6
#>    country     continent  year lifeExp      pop gdpPercap
#>    <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
#>  1 Afghanistan Asia       1952    28.8  8425333      779.
#>  2 Afghanistan Asia       1957    30.3  9240934      821.
#>  3 Afghanistan Asia       1962    32.0 10267083      853.
#>  4 Afghanistan Asia       1967    34.0 11537966      836.
#>  5 Afghanistan Asia       1972    36.1 13079460      740.
#>  6 Afghanistan Asia       1977    38.4 14880372      786.
#>  7 Afghanistan Asia       1982    39.9 12881816      978.
#>  8 Afghanistan Asia       1987    40.8 13867957      852.
#>  9 Afghanistan Asia       1992    41.7 16317921      649.
#> 10 Afghanistan Asia       1997    41.8 22227415      635.
#> # ... with 1,694 more rows
```

```
gapminder %>%
  count(continent)
#> # A tibble: 5 x 2
#>   continent     n
#>   <fct>     <int>
#> 1 Africa      624
#> 2 Americas    300
#> 3 Asia        396
#> 4 Europe      360
#> 5 Oceania      24
```


Africa
Americas
Asia
Europe
80
60
40
lifeExp
1950
1960
1970
1980
1990
2000
year


GDP per capita on Five Continents
Africa
Americas
Asia
Europe
Oceania
$100,000
$10,000
$1,000
GDP per capita
1952 1967 1982 1997 2007
Year

```r
africa <- gapminder[gapminder$continent == "Africa", ]
africa_mm <- max(africa$lifeExp) - min(africa$lifeExp)

americas <- gapminder[gapminder$continent == "Americas", ]
americas_mm <- max(americas$lifeExp) - min(americas$lifeExp)

asia <- gapminder[gapminder$continent == "Asia", ]
asia_mm <- max(asia$lifeExp) - min(africa$lifeExp)

europe <- gapminder[gapminder$continent == "Europe", ]
europe_mm <- max(europe$lifeExp) - min(europe$lifeExp)

oceania <- gapminder[gapminder$continent == "Oceania", ]
oceania_mm <- max(europe$lifeExp) - min(oceania$lifeExp)

cbind(
  continent = c("Africa", "Asias", "Europe", "Oceania"),
  max_minus_min = c(africa_mm, americas_mm, asia_mm,
                    europe_mm, oceania_mm)
)
```

What am I trying to do?

Have I even done it?*

* Can you find my mistakes?

How would *you* compute this?

for each continent

max life exp - min life exp

put result in a data frame

Here's how I would do it.

```
gapminder %>%
  group_by(continent) %>%
  summarize(max_minus_min = max(lifeExp) - min(lifeExp))
#> # A tibble: 5 x 2
#>   continent max_minus_min
#>   <fct>             <dbl>
#> 1 Africa             52.8
#> 2 Americas           43.1
#> 3 Asia               53.8
#> 4 Europe             38.2
#> 5 Oceania            12.1
```

Conclusion: there are many ways to write a `for` loop in R!

sidebar on %>%

```r
filter(gapminder, country == "Canada")
gapminder %>%
  filter (country == "Canada")
```

```r
mean(x)
x %>% mean()
```

```
whatever(arg1, arg2, arg3, …)
```

```
arg1 %>%
  whatever(arg2, arg3, …)
```

```
foo_foo <- little_bunny()


bop_on(
  scoop_up(
    hop_through(foo_foo, forest),
    field_mouse
  ),
  head
)


# vs
foo_foo %>%
  hop_through(forest) %>%
  scoop_up(field_mouse) %>%
  bop_on(head)
```

from various Hadley Wickham talks

the magrittr package
provides %>%
which is used heavily and re-exported in the tidyverse

https://github.com/smbache/magrittr
https://cran.r-project.org/web/packages/magrittr/index.html

https://github.com/hadley/tidyverse
https://cran.r-project.org/web/packages/tidyverse/index.html

New example: making strings

```
child <- c("Reed", "Wesley", "Eli", "Toby")
age   <- c(    14,        12,    12,      1)

s <- rep_len("", length(child))
for (i in seq_along(s)) {
  s[i] <- paste(child[i], "is", age[i], "years old")
}
s
#> [1] "Reed is 14 years old"   "Wesley is 12 years old"
#> [3] "Eli is 12 years old"    "Toby is 1 years old"
```

Here's how I would do it.

```
child <- c("Reed", "Wesley", "Eli", "Toby")
age   <- c(    14,        12,    12,      1)

paste(child, "is", age, "years old")
#> [1] "Reed is 14 years old"   "Wesley is 12 years old"
#> [3] "Eli is 12 years old"    "Toby is 1 years old"
glue::glue("{child} is {age} years old")
#> Reed is 14 years old
#> Wesley is 12 years old
#> Eli is 12 years old
#> Toby is 1 years old
```

Conclusion: maybe someone already wrote that `for` loop for you!

But what if you really do need to iterate?

https://purrr.tidyverse.org

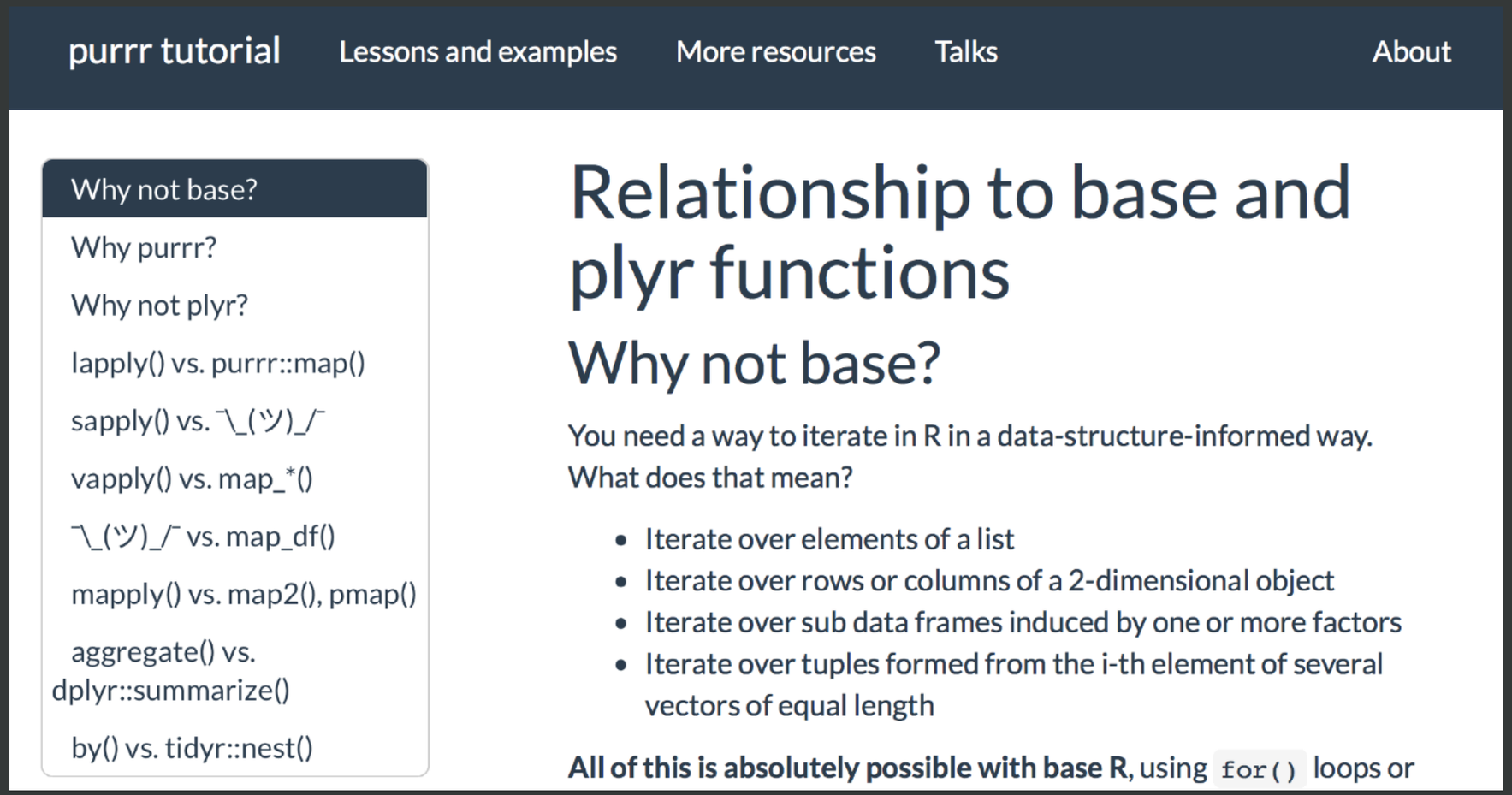
Part of the tidyverse

A "core" package in the tidyverse meta-package

```
install.packages("tidyverse") # <-- install purrr + much more
install.packages("purrr")     # <-- installs only purrr

library(tidyverse) # <-- loads purrr + much more
library(purrr)     # <-- loads only purrr
```

purrr is an alternative to "apply" functions

`purrr::map()` ≈ `base::lapply()`



```
library(purrr)
library(repurrrsive)
help(package = "repurrrsive")
```

Get comfortable with lists!

atomic vectors are familar:
logical, integer, double, character, etc

a list = a generalized vector
a list can hold almost anything

"working with lists"

How many elements are in `got_chars`?

Who is the 9th person listed in `got_chars`?
What information is given for this person?

What is the difference between `got_chars[9]` and `got_chars[[9]]`?

Or ... do same for `sw_people` or the n-th person

List exploration

`str(x, list.len = ?, max.level = ?)`

`x[i]`

`x[[i]]`

`str(x[[i]], ...)`

`View(x)`, in RStudio

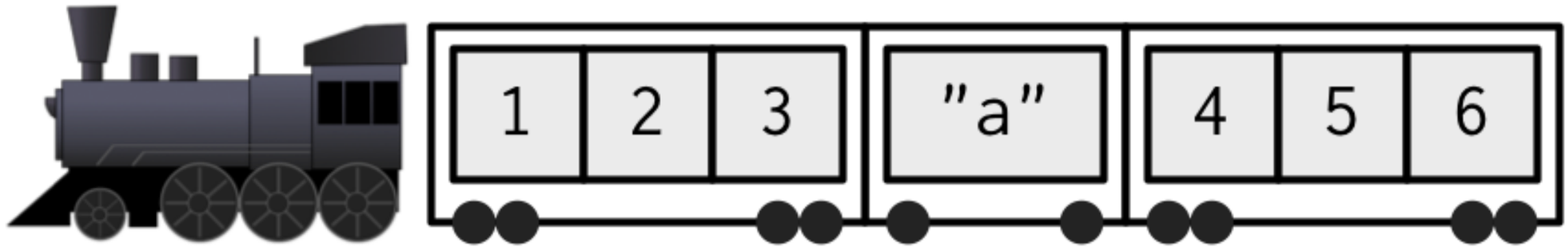
If list x is a train carrying objects:

`x[[5]]` is the object in car 5

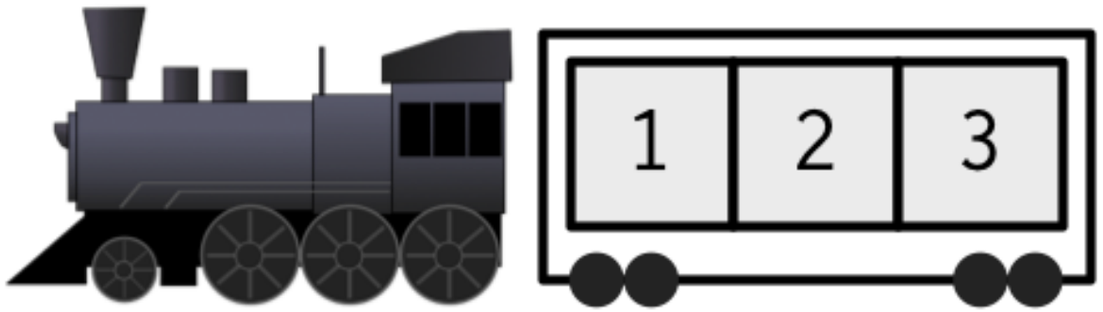
`x[4:6]` is a train of cars 4-6.

-- Tweet by @RLangTip

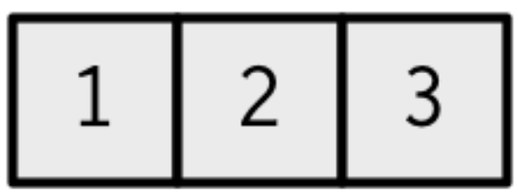
x



x[1]

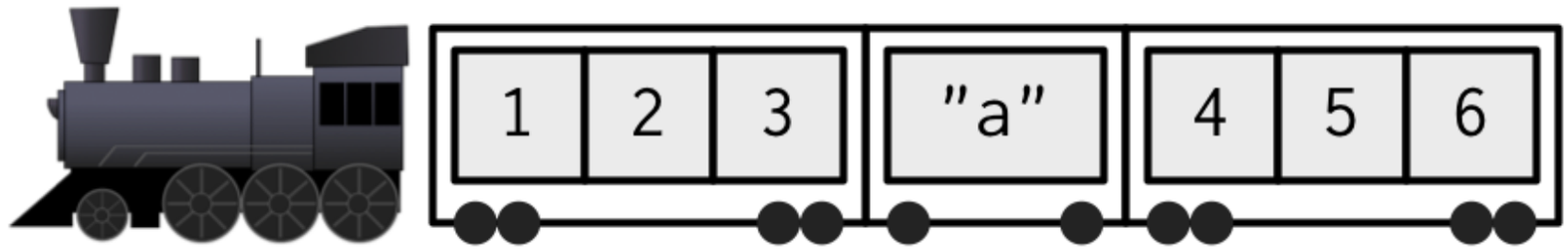


x[[1]]

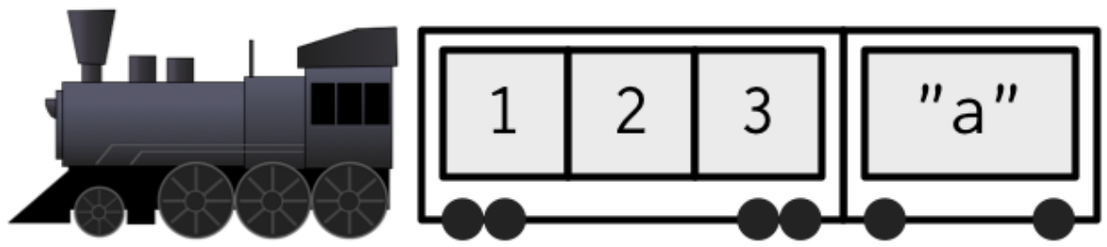


from Subsetting chapter of 2nd ed Advanced R

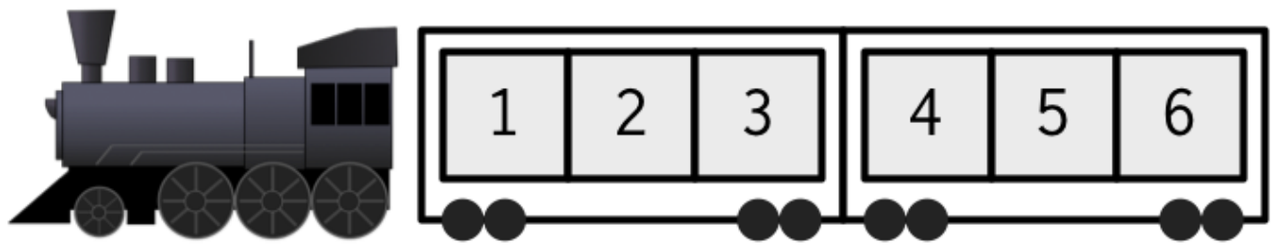
x



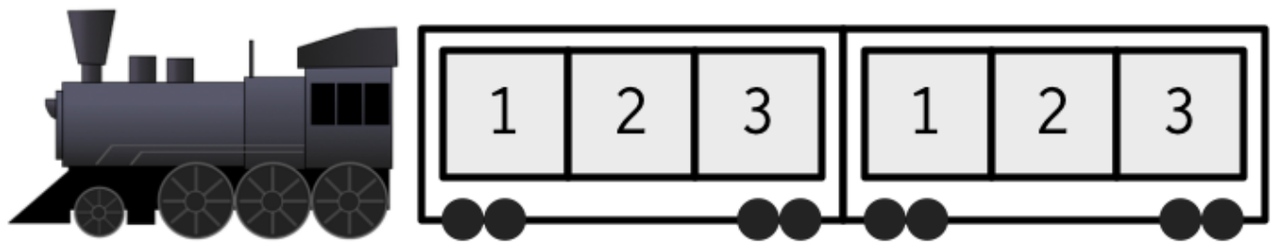
x[1:2]



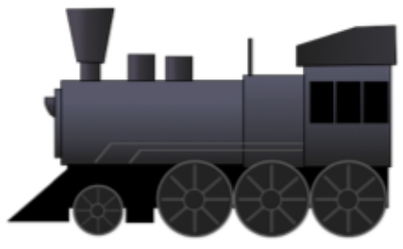
x[-2]



x[c(1,1)]



x[0]



from Subsetting chapter of 2nd ed Advanced R

`x`

`x[i]`

`x[[i]]`

from

http://r4ds.had.co.nz/vectors.html#lists-of-condiments

```
purrr::
map(.x, .f, ...)
```

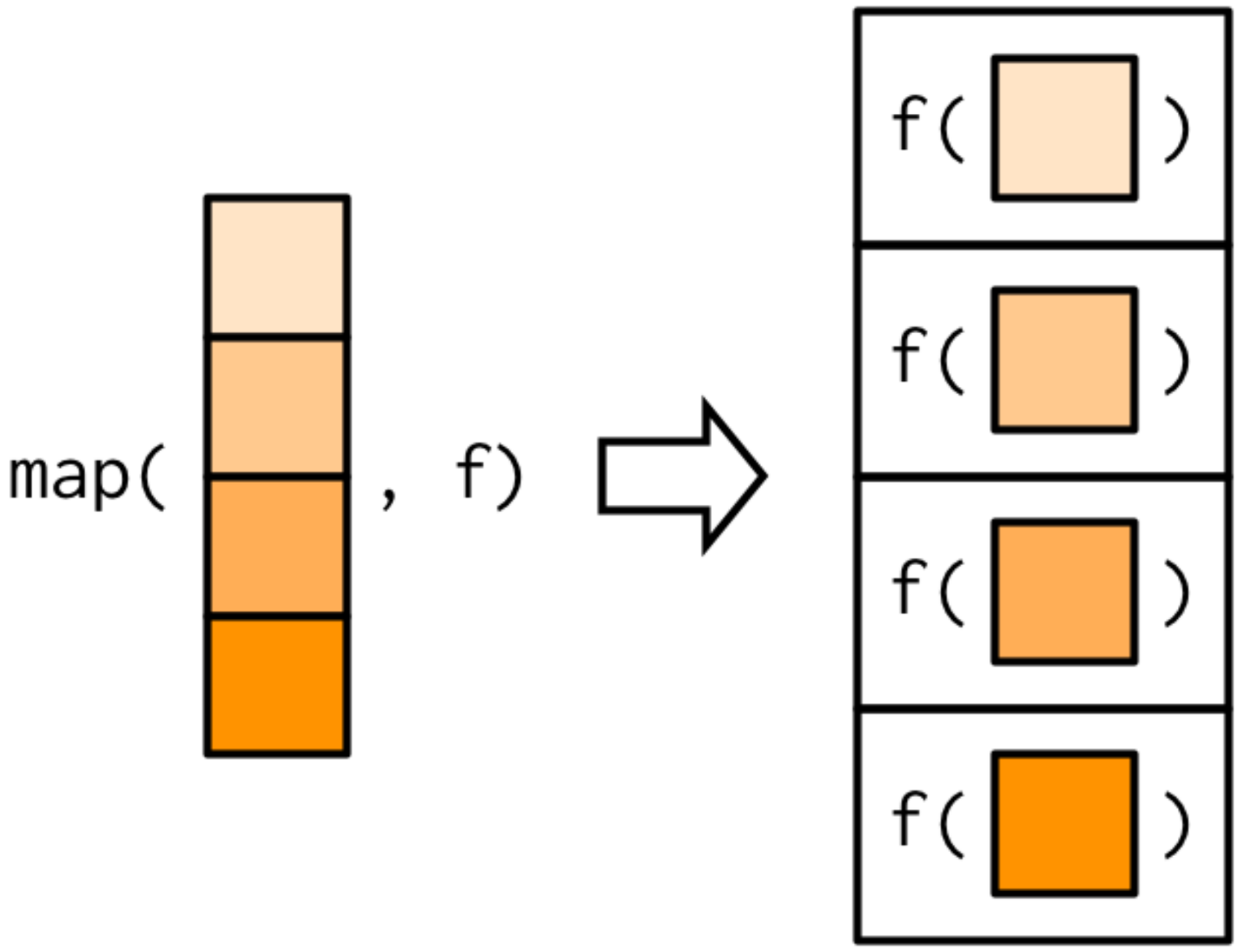
```
purrr::
map(.x, .f, ...)
```

for every element of **.x**

do **.f**

.x = minis

```
map(minis, antennate)
```



from Functionals chapter of 2nd ed Advanced R

purrr::
map(.x, .f)

```
.x <- SOME VECTOR OR LIST
out <- vector(mode = "list", length = length(.x))
for (i in seq_along(out)) {
  out[[i]] <- .f(.x[[i]])
}
out
```

purrr::
map(.x, .f)

```
.x <- SOME VECTOR OR LIST
out <- vector(mode = "list", length = length(.x))
for (i in seq_along(out)) {
  out[[i]] <- .f(.x[[i]])
}
out
```

purrr::map() is a nice way to write a for loop.

How many aliases does each GoT character have?

```
map(got_chars, .f = 🤷)
```

or

```
map(sw_people, .f = 🤷)
```

Workflow:

1. Do it for one element.
2. Find the general recipe.
3. Drop into `map()` to do for all.

Step 1: Do it for one element

```
daenerys <- got_chars[[9]]
## View(daenerys)

daenerys[["aliases"]]
#>  [1] "Dany"                      "Daenerys Stormborn"
#>  [3] "The Unburnt"               "Mother of Dragons"
#>  [5] "Mother"                    "Mhysa"
#>  [7] "The Silver Queen"          "Silver Lady"
#>  [9] "Dragonmother"              "The Dragon Queen"
#> [11] "The Mad King's daughter"

length(daenerys[["aliases"]])
#> [1] 11
```

Step 1: Do it for one element

```r
asha <- got_chars[[13]]
## View(asha)

asha[["aliases"]]
#> [1] "Esgred"                    "The Kraken's Daughter"
```

```r
length(asha[["aliases"]])
#> [1] 2
```

Step 2: Find the general recipe

```
.x <- got_chars[[?]]
length(.x[["aliases"]])
```

Step 2: Find the general recipe

```
.x <- got_chars[[?]]
length(.x[["aliases"]])
```

.x is a pronoun, like "it"
means "the current element"

Step 3: Drop into `map()` to do for all

```
map(got_chars, ~ length(.x[["aliases"]]))
#> [[1]]
#> [1] 4
#>
#> [[2]]
#> [1] 11
#>
#> [[3]]
#> [1] 1
#>
#> [[4]]
#> [1] 1
#> ...
```

Step 3: Drop into `map()` to do for all

```
map(got_chars, ~ length(.x[["aliases"]]))
#> [[1]]
#> [1] 4
#>
#> [[2]]
#> [1] 11
#>
#> [[3]]
#> [1] 1
#>
#> [[4]]
#> [1] 1
#> ...
```

formula method of specifying `.f`

`.x` means "the current element"

concise syntax for anonymous functions

a.k.a. lambda functions

Challenge (pick one or more!)

How many x does each (GoT or SW) character have? (x = titles, allegiances, vehicles, starships)

```
map(got_chars, ~ length(.x[["aliases"]]))
```

Oh, would you prefer an integer vector?

```
map_int(got_chars, ~ length(.x[["aliases"]]))
#>  [1]  4 11  1  1  1  1  1  1 11  5 16
#> [12]  1  2  5  3  3  3  5  0  3  4  1
#> [25]  8  2  1  5  1  4  7  3
```

```
map()
map_lgl()
map_int()
map_dbl()
map_chr()
```

type-specific
variants of map()

Challenge:

Replace map() with type-specific map()

```
# What's each character's name?
map(got_chars, ~.x[["name"]])
map(sw_people, ~.x[["name"]])

# What color is each SW character's hair?
map(sw_people, ~ .x[["hair_color"]])

# Is the GoT character alive?
map(got_chars, ~ .x[["alive"]])

# Is the SW character female?
map(sw_people, ~ .x[["gender"]] == "female")

# How heavy is each SW character?
map(sw_people, ~ .x[["mass"]])
```

Review

Lists can be awkward

Lists are necessary

Get to know your list

```
purrr::
map(.x, .f, ...)
```

for every element of **.x**

do **.f**

purrr::
map(.x, .f)

```
map(got_chars, ~ length(.x[["aliases"]]))
```

quick anonymous functions
via formula

```r
map_lgl(sw_people, ~ .x[["gender"]] == "female")

map_int(got_chars, ~ length(.x[["aliases"]]))

map_chr(got_chars, ~ .x[["name"]])
```

Onwards!

Notice:

We extract by name a lot

```
# What's each character's name?
map(got_chars, ~.x[["name"]])

# What color is each SW character's hair?
map(sw_people, ~ .x[["hair_color"]])
```

```
# Is the GoT character alive?
map(got_chars, ~ .x[["alive"]])

# How heavy is each SW character?
map(sw_people, ~ .x[["mass"]])
```

```
map_chr(got_chars, ~ .x[["name"]])
```

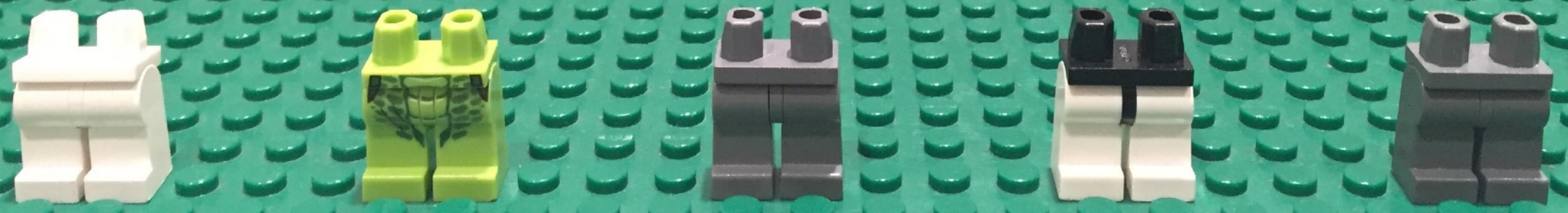
```
map_chr(got_chars, "name")
```

Shortcut!

.f accepts a name or position

.x = minis

```
map(minis, "pants")
```



Challenge:

Explore a GoT or SW list and find a new element to look at

Extract it across the whole list with name and position shortcuts for `.f`

Use `map_TYPE()` to get an atomic vector as output

```
map_??(got_??, ??)
map_??( sw_??, ??)
```

Common problem

I'm using `map_TYPE()` but some individual elements aren't of length `1`.

They are absent or have length `> 1`.

Solutions

Missing elements?

Specify a `.default` value.

Elements of length > 1?

You can't make an atomic vector.*

Get happy with a list or list-column.

Or pick one element, e.g., the first.

* You can, if you are willing to `flatten()` or `squash()`.

```
map(sw_vehicles, "pilots", .default = NA)
#> [[1]]
#> [1] NA
#>
#> ...
#>
#> [[19]]
#> [1] "http://swapi.co/api/people/10/" "http://swapi.co/api/people/32/"
#>
#> [[20]]
#> [1] "http://swapi.co/api/people/44/"
#>
#> ...
#>
#> [[37]]
#> [1] "http://swapi.co/api/people/67/"
#>
#> [[38]]
#> [1] NA
#>
#> [[39]]
#> [1] NA
```

```
map_chr(sw_vehicles, list("pilots", 1), .default = NA)
#>  [1] NA                                    NA
#>  [3] NA                                    NA
#>  [5] "http://swapi.co/api/people/1/"       NA
#>  [7] NA                                    "http://swapi.co/api/people/13/"
#>  [9] NA                                    NA
#> [11] NA                                    NA
#> [13] "http://swapi.co/api/people/1/"       NA
#> [15] NA                                    NA
#> [17] NA                                    NA
#> [19] "http://swapi.co/api/people/10/" "http://swapi.co/api/people/44/"
#> [21] "http://swapi.co/api/people/11/" "http://swapi.co/api/people/70/"
#> [23] "http://swapi.co/api/people/11/" NA
#> [25] NA                                    "http://swapi.co/api/people/79/"
#> [27] NA                                    NA
#> [29] NA                                    NA
#> [31] NA                                    NA
#> [33] NA                                    NA
#> [35] NA                                    NA
#> [37] "http://swapi.co/api/people/67/" NA
#> [39] NA
```

```
map(got_chars, c(14, 1))

map(sw_vehicles, list("pilots", 1))
```

Shortcut!

.f accepts a ~~name or position~~ vector of names or positions or a list of names and positions

Names make life nicer!

```
map_chr(got_chars, "name")
#>  [1] "Theon Greyjoy"      "Tyrion Lannister"   "Victarion Greyjoy"
#>  ...

got_chars_named <- set_names(got_chars, map_chr(got_chars, "name"))

got_chars_named %>%
  map_lgl("alive")
#>      Theon Greyjoy   Tyrion Lannister  Victarion Greyjoy
#>               TRUE               TRUE               TRUE
#>      ...
```

Names propagate in purrr pipelines.

Set them early and enjoy!

`tibble::enframe()` does this:

named list → df w/ names & list-column

```
allegiances <- map(got_chars_named, "allegiances")
tibble::enframe(allegiances, value = "allegiances")
#> # A tibble: 30 x 2
#>    name                allegiances
#>    <chr>               <list>
#>  1 Theon Greyjoy       <chr [1]>
#>  2 Tyrion Lannister    <chr [1]>
#>  3 Victarion Greyjoy   <chr [1]>
#>  4 Will                <NULL>
#>  5 Areo Hotah          <chr [1]>
#>  6 Chett               <NULL>
#>  7 Cressen             <NULL>
#>  8 Arianne Martell     <chr [1]>
#>  9 Daenerys Targaryen  <chr [1]>
#> 10 Davos Seaworth      <chr [2]>
#> # ... with 20 more rows
```

Review #2

Set list names for a happier life.

```
got_chars_named <- set_names(got_chars, map_chr(got_chars, "name"))
```

There are many ways to specify .f.

```
map(got_chars, ~ length(.x[["aliases"]]))
map_chr(got_chars, "name")
map(sw_vehicles, list("pilots", 1))
```

`.default` is useful for missing things.

```
map(sw_vehicles, "pilots", .default = NA)
map_chr(sw_vehicles, list("pilots", 1), .default = NA)
```

Challenge:

Create a named copy of a GoT or SW list with `set_names()`.

Find an element with tricky presence/absence or length.

Extract it many ways:

- by name
- by position
- by `list("name", pos)` or `c(pos, pos)`
- use `.default` for missing data
- use `map_TYPE()` to coerce output to atomic vector

Challenge (pick one or more):

Which SW film has the most characters?

Which SW species has the most possible eye colors?

Which GoT character has the most allegiances? Aliases? Titles?

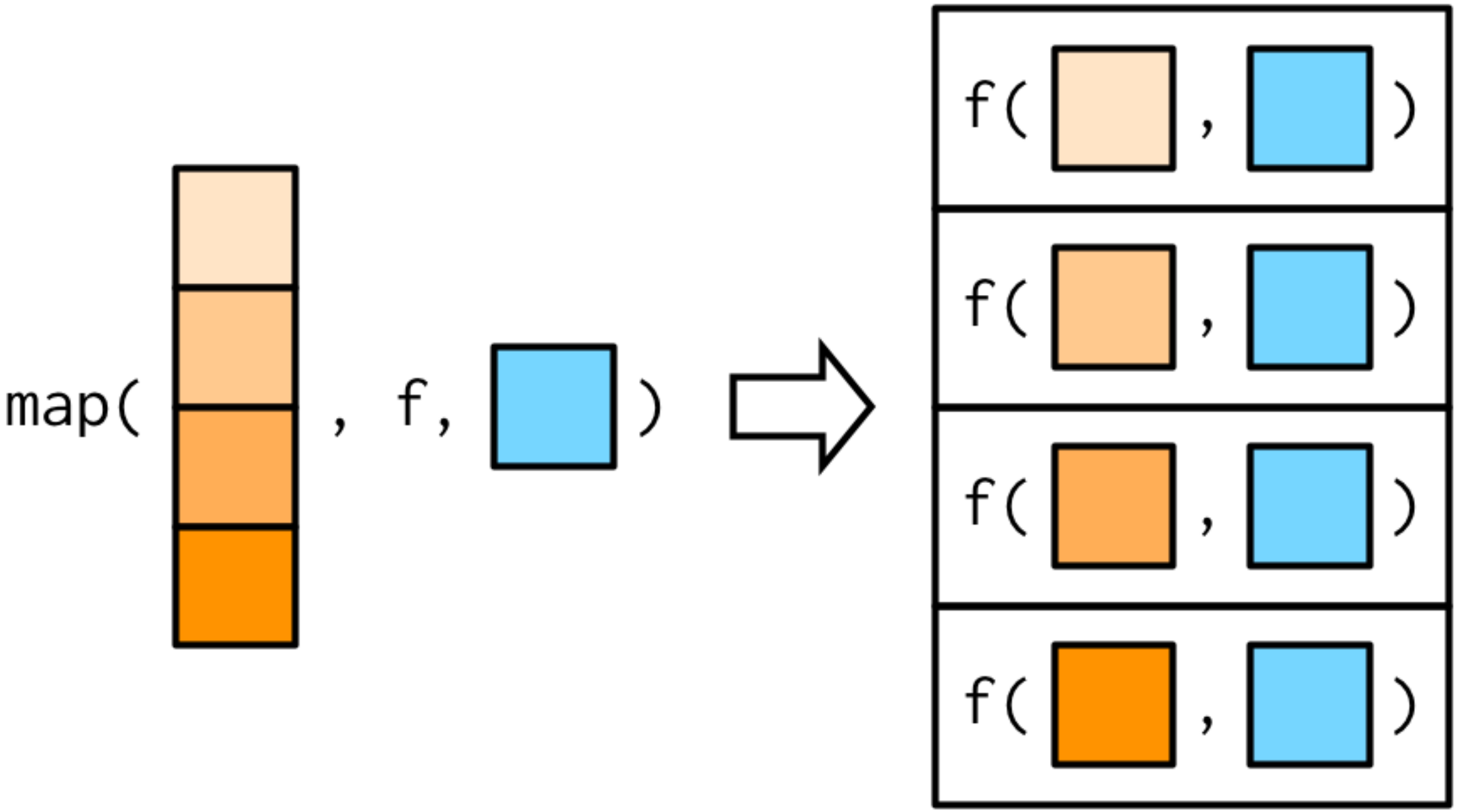
Which GoT character has been played by multiple actors?

Inspiration for your future purrr work

map(.x, .f, ...)

```
books <- map(got_chars_named, "books")

map_chr(books[1:2], paste, collapse = ", ")
#>                                                  Theon Greyjoy
#> "A Game of Thrones, A Storm of Swords, A Feast for Crows"
#>                                                 Tyrion Lannister
#>             "A Feast for Crows, The World of Ice and Fire"
map_chr(books[1:2], ~ paste(.x, collapse = ", "))

#>                                                  Theon Greyjoy
#> "A Game of Thrones, A Storm of Swords, A Feast for Crows"
#>                                                 Tyrion Lannister
#>             "A Feast for Crows, The World of Ice and Fire"
```

map(.x, .f, ...)



from Functionals chapter of 2nd ed Advanced R

map(.x, .f, ...)

```
books <- map(got_chars_named, "books")

map_chr(books[1:2], paste, collapse = ", ")
#>                                                  Theon Greyjoy
#> "A Game of Thrones, A Storm of Swords, A Feast for Crows"
#>                                                 Tyrion Lannister
#>             "A Feast for Crows, The World of Ice and Fire"
map_chr(books[1:2], ~ paste(.x, collapse = ", "))

#>                                                  Theon Greyjoy
#> "A Game of Thrones, A Storm of Swords, A Feast for Crows"
#>                                                 Tyrion Lannister
#>             "A Feast for Crows, The World of Ice and Fire"
```

So, yes,

there are **many** ways to specify .f.

```
map(got_chars, ~ length(.x[["aliases"]]))
map_chr(got_chars, "name")
map_chr(books[1:2], paste, collapse = ", ")
map(sw_vehicles, list("pilots", 1))
```

```r
library(tidyverse)
library(gapminder)

countries <- c("Argentina", "Brazil", "Canada")
gap_small <- gapminder %>%
  filter(country %in% countries, year > 1996)
gap_small
#> # A tibble: 9 x 6
#>   country   continent  year lifeExp      pop gdpPercap
#>   <fct>     <fct>     <int>   <dbl>    <int>     <dbl>
#> 1 Argentina Americas   1997    73.3  36203463    10967.
#> 2 Argentina Americas   2002    74.3  38331121     8798.
#> 3 Argentina Americas   2007    75.3  40301927    12779.
#> 4 Brazil    Americas   1997    69.4 168546719     7958.
#> 5 Brazil    Americas   2002    71.0 179914212     8131.
#> 6 Brazil    Americas   2007    72.4 190010647     9066.
#> 7 Canada    Americas   1997    78.6  30305843    28955.
#> 8 Canada    Americas   2002    79.8  31902268    33329.
#> 9 Canada    Americas   2007    80.7  33390141    36319.

write_one <- function(x) {
  filename <- paste0(x, ".csv")
  dataset <- filter(gap_small, country == x)
  write_csv(dataset, filename)
}

walk(countries, write_one)
list.files(pattern = "*.csv")
#> [1] "Argentina.csv" "Brazil.csv"    "Canada.csv"
```

walk() is map() but returns no output

map_dfr() rowbinds a list of data frames

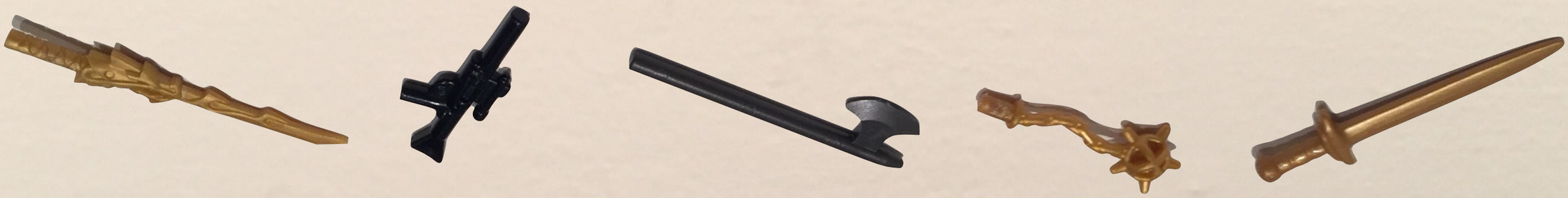
```
library(tidyverse)

csv_files <- list.files(pattern = "*.csv")
csv_files
#> [1] "Argentina.csv" "Brazil.csv"    "Canada.csv"

map_dfr(csv_files, ~ read_csv(.x))
#> # A tibble: 9 x 6
#>   country   continent  year lifeExp      pop gdpPercap
#>   <fct>     <fct>     <int>   <dbl>    <int>     <dbl>
#> 1 Argentina Americas   1997    73.3  36203463    10967.
#> 2 Argentina Americas   2002    74.3  38331121     8798.
#> 3 Argentina Americas   2007    75.3  40301927    12779.
#> 4 Brazil    Americas   1997    69.4 168546719     7958.
#> 5 Brazil    Americas   2002    71.0 179914212     8131.
#> 6 Brazil    Americas   2007    72.4 190010647     9066.
#> 7 Canada    Americas   1997    78.6  30305843    28955.
#> 8 Canada    Americas   2002    79.8  31902268    33329.
#> 9 Canada    Americas   2007    80.7  33390141    36319.
```

mapping over 2 or more things in parallel

.y = hair

.x = minis

```
map2(minis, hair, enhair)
```

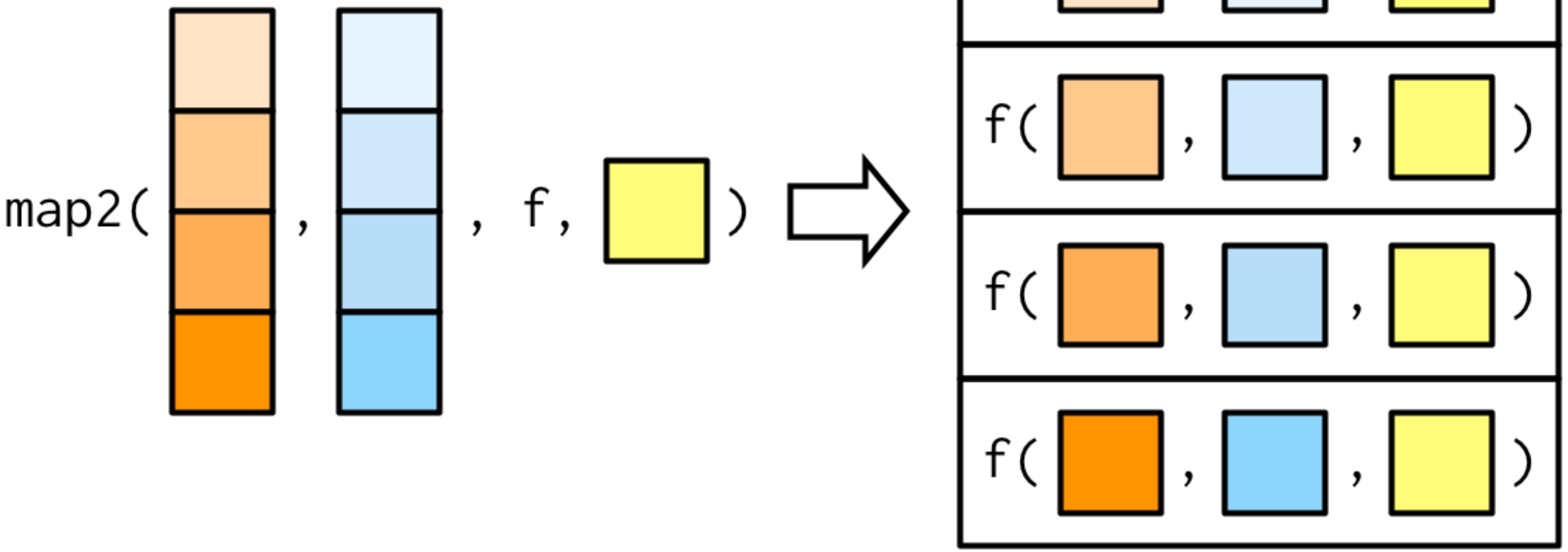


.y = weapons

.x = minis

```
map2(minis, weapons, arm)
```

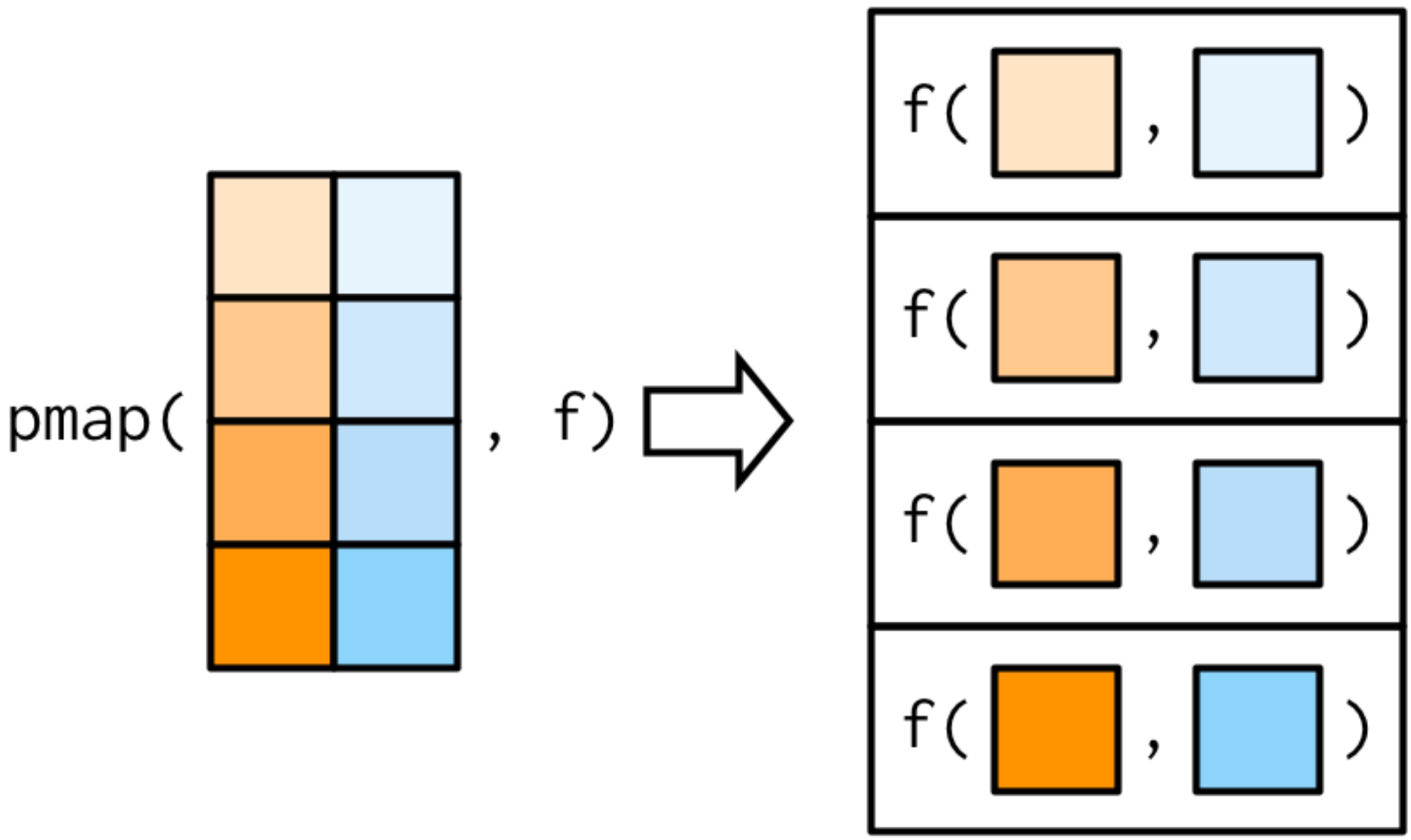
```
minis %>%
   map2(hair, enhair) %>%
   map2(weapons, arm)
```



from Functionals chapter of 2nd ed Advanced R

```
df <- tibble(pants, torso, head)
embody <- function(pants, torso, head)
  insert(insert(pants, torso), head)
```

pmap(df, embody)



from Functionals chapter of 2nd ed Advanced R

```
map_dfr(minis, `[`,
        c("pants", "torso", "head")
```

For much more on this: rstd.io/row-work

Row-oriented workflows in R with the tidyverse

Materials for RStudio webinar *recording available at this link!*:

Thinking inside the box: you can do that inside a data frame?!
Jenny Bryan
Wednesday, April 11 at 1:00pm ET / 10:00am PT
rstd.io/row-work <-- *shortlink to this repo*
Slides available on SpeakerDeck

You have the basis for exploring the world of purrr now!

	List	Atomic	Same type	Nothing
One argument	`map()`	`map_lgl()`, ...	`modify()`	`walk()`
Two arguments	`map2()`	`map2_lgl()`, ...	`modify2()`	`walk2()`
One argument + index	`imap()`	`imap_lgl()`, ...	`imodify()`	`iwalk()`
N arguments	`pmap()`	`pmap_lgl()`, ...	—	`pwalk()`

from Functionals chapter of 2nd ed Advanced R