CHARLOTTE WICKHAM

HAPPY R USERS PURRR: USING FUNCTIONAL PROGRAMMING TO SOLVE ITERATION PROBLEMS

GETTING SETUP

1. Download .zip (Download->Direct Download) of slides and code

   @ bit.ly/purrr-rstudioconf

2. Extract and open purrr_workshop.Rproj

3. Check you have packages:

   ```
   library(purrr)
   library(tidyverse)
   ```

FOR EACH ___ DO ___

You are already solving them:

copy & paste, for loops, `(l/s)apply()`

I'll show you an alternative `purrr::map()` & friends

Download .zip of slides and code @ bit.ly/purrr-rstudioconf

FUNCTIONAL PROGRAMMING

a programming paradigm

has some central concepts

you don't need to know them to use `purrr`, but I'll point them out

FP

Download .zip of slides and code @ bit.ly/purrr-rstudioconf

Star Wars API - http://swapi.co/
Data extracted using https://github.com/Ironholds/rwars/

```
# loads objects: films, people, vehicles, starships,
# planets & species

load("data/swapi.rda")
```

1. How many elements are in `people`?

2. Who is the first person listed in `people`? What information is given for this person?

3. What is the difference between `people[1]` and `people[[1]]`?

BEWARE!
ANSWERS ON FOLLOWING SLIDE

Download .zip of slides and code @ bit.ly/purrr-rstudioconf

```
length(people)
## [1] 87
```

```
people[[1]]
## $name
## [1] "Luke Skywalker"
##
## $height
## [1] "172"
##
## $mass
## [1] "77"
##
## $hair_color
## [1] "blond"
##
## $skin_color
## [1] "fair"
##
## $eye_color
## [1] "blue"
##
## $birth_year
## [1] "19BBY"
##
## $gender
## [1] "male"
##
## $homeworld
## [1] "http://swapi.co/api/planets/1/"
##
```

```
## $films
## [1] "http://swapi.co/api/films/6/"
## [2] "http://swapi.co/api/films/3/"
## [3] "http://swapi.co/api/films/2/"
## [4] "http://swapi.co/api/films/1/"
## [5] "http://swapi.co/api/films/7/"
##
## $species
## [1] "http://swapi.co/api/species/1/"
##
## $vehicles
## [1] "http://swapi.co/api/vehicles/14/"
## [2] "http://swapi.co/api/vehicles/30/"
##
## $starships
## [1] "http://swapi.co/api/starships/
12/"
## [2] "http://swapi.co/api/starships/
22/"
##
## $created
## [1] "2014-12-09T13:50:51.644000Z"
##
## $edited
## [1] "2014-12-20T21:17:56.891000Z"
##
## $url
## [1] "http://swapi.co/api/people/1/"
```

map()

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

for each element of .x do .f

.x

▸ **a vector**

▸ a list

▸ a data frame (for each column)

.f

We'll get to that...

HOW MANY FILMS HAS EACH CHARACTER BEEN IN?

for each person in `people`, **count the number of films**

$$\texttt{map(people, \_\_\_\_)}$$

STRATEGY

1. Do it for one element

2. Turn it into a recipe

3. Use `map()` to do it for all elements

```
luke <- people[[1]]
```

HOW MANY STARSHIPS HAS LUKE BEEN IN?

Write a line of code to find out.

Bored? Find the names of those starships…

DO IT FOR ONE

Solve the problem for one element

```
luke <- people[[1]]
```

```
length(luke$starships)
```

DO IT FOR ONE

Solve the problem for one element

```
luke <- people[[1]]
```

length(luke$starships)

DO IT FOR ONE

Solve the problem for one element

```
leia <- people[[5]]
```

length(**leia**$starships)

DO IT FOR ONE

Solve the problem for one element

```
____ <- people[[?]]
```

```
length(____$starships)
```

TURN IT INTO A RECIPE

Make it a formula

Use .x as a placeholder

$$\sim \text{length}(\underline{\quad.x}\$\text{starships})$$

A formula

purrr's placeholder for
one element of our vector

DO IT FOR ALL!

Your recipe is the second argument to map

map(people ,
    ~ length(.x$starships))

A formula

purrr's placeholder for
one element of our vector

```
map(people, ~ length(.x$starships))
```

Copy and paste ME.

Load then look at `planet_lookup`:

```
load("data/planet_lookup.rda")

planet_lookup
```

FIND THE NAME OF EACH CHARACTERS HOME WORLD.

Bored? Find the body mass index (BMI) of all characters.

bmi = (mass in kg) / ((height in m)^2)

```
luke$homeworld
## [1] "http://swapi.co/api/planets/1/"


planet_lookup[luke$homeworld]
## http://swapi.co/api/planets/1/
##                              "Tatooine"


map(people, ~ planet_lookup[.x$homeworld])
## [[1]]
## http://swapi.co/api/planets/1/
##                              "Tatooine"


## [[2]]
## http://swapi.co/api/planets/1/
##                              "Tatooine"


## [[3]]
## http://swapi.co/api/planets/8/
##                                 "Naboo"
...
```

ARE YOU PURRRING YET?

ROADmap()

map_lgl(.x, .f, ...)

Other types of output

Other ways of specifying .f

Other iteration functions

ROADmap()

```
map( .x, length, ...)
```

Other types of output

Other ways of specifying .f

Other iteration functions

ROADmap()

map2(.x, .y, .f, ...)

Other types of output

Other ways of specifying .f

Other iteration functions

map() details

`map()` **always** returns a **list**

`map_lgl()` **logical** vector

`map_int()` **integer** vector

`map_dbl()` **double** vector

`map_chr()` **character** vector

`walk()` - when you want nothing at all,
use a function for its side effects

Result: No surprises!

vector same length as `.x` or an ERROR

```
# names can be useful
people <- people %>% set_names(map_chr(people, "name"))
```

REPLACE map() WITH THE APPROPRIATELY TYPED FUNCTION

```
# How many starships has each character been in?
map(people, ~ length(.x[["starships"]]))


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


# Is the character male?
map(people, ~ .x[["gender"]] == "male")


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

```
# How many starships has each character been in?
map_int(people, ~ length(.x[["starships"]]))

##    Luke Skywalker    C-3PO    R2-D2    Darth Vader
##                 2        0        0              1 ...

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

##    Luke Skywalker    C-3PO    R2-D2    Darth Vader
##           "blond"    "n/a"    "n/a"         "none" ...

# Is the character male?
map_lgl(people, ~ .x[["gender"]] == "male")

##    Luke Skywalker    C-3PO    R2-D2    Darth Vader
##              TRUE    FALSE    FALSE           TRUE ...
```

```r
# How heavy is each character?

map_dbl(people, ~ .x[["mass"]])

## Error: Can't coerce element 1 from a character to a double

# Doesn't work...because we get a string back

map(people, ~ .x[["mass"]])

## [[1]]

## [1] "77"

## 

## [[2]]

## [1] "75"

...
```

```r
# A little risky

map_dbl(people, ~ as.numeric(.x[["mass"]]))

## [1]  77.0   75.0   32.0   136.0  49.0  120.0   75.0   32.0   84.0
## ...
## There were 29 warnings (use warnings() to see them)


# Probably want something like:

map_chr(people, ~ .x[["mass"]]) %>%

  readr::parse_number(na = "unknown")

## [1]   77.0   75.0   32.0  136.0   49.0  120.0   75.0   32.0   84.0
## ...
```

.f CAN BE A FORMULA

```
map(.x, .f =  ~ DO SOMETHING WITH .x)
```

```
map_int(people, ~ length(.x[["starships"]]))
map_chr(people, ~ .x[["hair_color"]])
map_chr(people, ~ .x[["mass"]])
```

.f CAN BE A STRING OR INTEGER

For each element, extract the named/numbered element

map(.x, .f = "some_name")

equivalent to

map(.x, ~ .x[[some_name]])

.f CAN BE A STRING OR INTEGER

For each element, extract the named/numbered element

map(.x, .f =   some_number)

equivalent to

map(.x, ~ .x[[some_number]])

```
map_chr(people, ~ .x[["hair_color"]])
# becomes
map_chr(people, "hair_color")
```

.f CAN BE A FUNCTION

map(.x, .f = some_function, ...)

equivalent to

map(.x, ~ some_function(.x, ...))

gets passed on to .f

```
char_starships <- map(people, "starships")
map_int(char_starships, length)


# In one go
map(people, "starships") %>% map_int(length)


# equivalent to
map_int(people, ~ length(.x[["starships"]]))
```

don't be afraid to do things in
little steps and pipe them
together

WHAT ABOUT `sapply()` & `lapply()`?

What type of object does `sapply()` return?   It depends.

Motivation for `purrr`:

- consistent return type,

- useful shortcuts,

- consistent syntax for more complicated iteration

STAR WARS CHALLENGES

Which film (see `films`) has the most characters?

Create the `planet_lookup` vector from earlier.

Which `species` has the most possible eye colors?

```r
# Which film (see films) has the most characters?

map(films, "characters") %>%

  map_int(length) %>%

  set_names(map_chr(films, "title")) %>%

  sort()


# Create the planet_lookup vector from earlier.

planet_lookup <- map_chr(planets, "name") %>%

  set_names(map(planets, "url"))
```

```r
# Which species has the most possible eye colors?

species[[1]]$eye_colors


map_chr(species, "eye_colors") %>%

  strsplit(", ") %>%

  map_int(length)
# this is lazy, what about n/a and unknown?
```

FUNCTIONS CAN BE ARGUMENTS

FP

Functions are **first class** citizens in R, they can occur anywhere a number could:
as arguments,
as return values,
assigning them to variables,
storing in data structures.

Higher order function: a function that takes a function as input or returns a function

`map()` is a higher order function.

purrr and list columns

PURRR AND LIST COLUMNS

Data should be in a data frame as soon as it makes sense!

Data frame: **cases** in rows, **variables** in columns

YOUR TURN:

What are the **cases** and **variables** in the `people` data?

```
# A tibble: 87 × 4

              name        films height                    species
             <chr>       <list>  <dbl>                      <chr>
1  Luke Skywalker    <chr [5]>     172 http://swapi.co/api/species/1/
2           C-3PO    <chr [6]>     167 http://swapi.co/api/species/2/
3           R2-D2    <chr [7]>      96 http://swapi.co/api/species/2/
4     Darth Vader    <chr [4]>     202 http://swapi.co/api/species/1/
5     Leia Organa    <chr [5]>     150 http://swapi.co/api/species/1/
# ... with 82 more rows
```

PURRR CAN HELP TURN LISTS INTO TIBBLES

```
people_tbl <- tibble(

  name    =                          ,

  films   =                     ,

  height  =

                                  ,

  species =
)
```

Full code in `code/star_wars-tbl.R`

PURRR CAN HELP TURN LISTS INTO TIBBLES

```
people_tbl <- tibble(

  name     = people %>% map_chr("name"),

  films    = people %>% map("films"),          will result in list column

  height   = people %>% map_chr("height") %>%
                readr::parse_number(na = "unknown"),   needs some parsing

  species = people %>% map_chr("species", .null = NA_character_)
)                                                isn't in every element
```

Full code in `code/star_wars-tbl.R`

COMBINE PURRR WITH DPLYR TO WORK WITH LIST COLUMNS

```
people_tbl$films


people_tbl %>%

  mutate(

    film_numbers = map(films, ~ film_number_lookup[.x]),

    n_films = map_int(films, length)
)
```

Code to create tibble in `04-purrr-list-columns.R`

Create a new character column that collapses the film numbers into a single string,

e.g. for Luke: " 6, 3, 2, 1, 7"

```
people_tbl <- people_tbl %>%

  mutate(

    films_squashed = map_chr(film_numbers, paste,

                             collapse = ", "))


people_tbl %>% select(name, n_films, films_squashed)
```
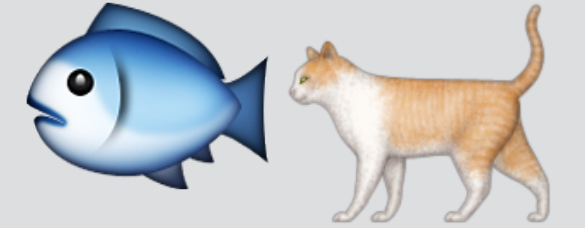
More iteration functions

to each element of **.x** apply **.f**

map(.x , .f   )

to each `cat` apply `give_fish`
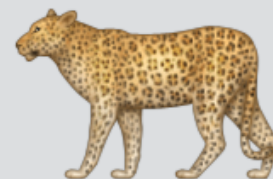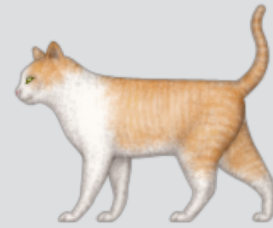
map(🐈 , give_fish)
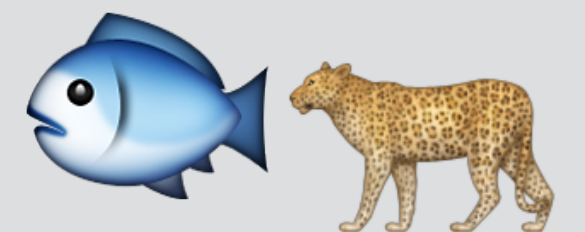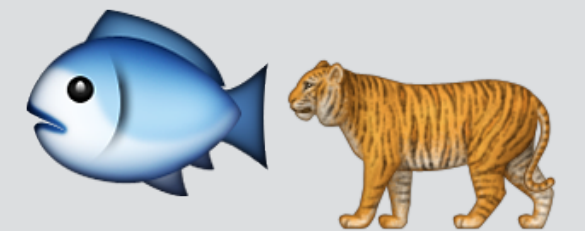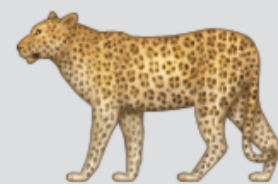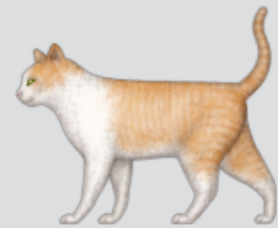
to each element of **.x** apply **.f**

`walk( .x  ,.f    )`

Expect nothing in return

You actually get .x invisibly back,
good for piping

to each `cat` apply `love`

`walk(` `, love )`

Expect nothing in return

You actually get .x invisibly back,
good for piping

For functions called for their side effects:

▸ printing to screen

▸ plotting to graphics device

▸ file manipulation (saving, writing, moving etc.)

▸ system calls

to each element of **.x** and corresponding element of **.y** apply **.f**

map2(.x,   .y, .f)

to each `cat` and corresponding `times` apply `rep`
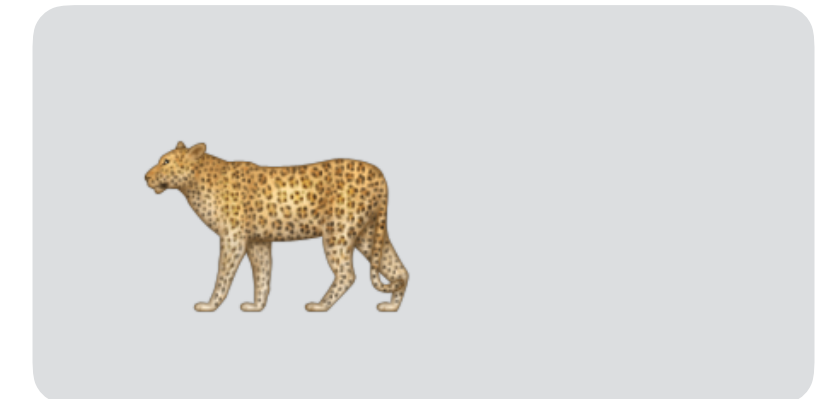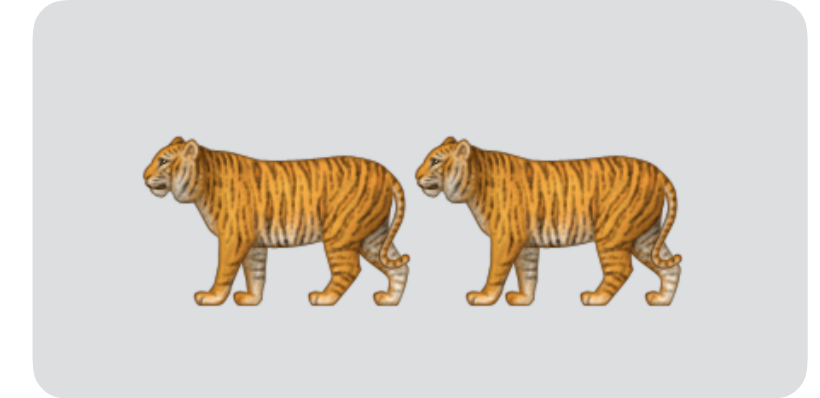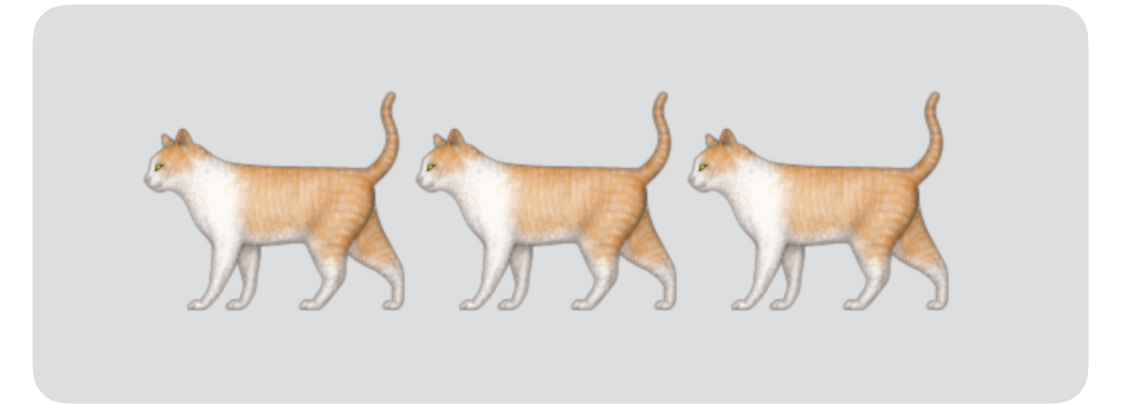
map2(🐈 , 3 , rep)

map2(🐅 , 2 , rep)

map2(🐆 , 1 , rep)

Always get a list back, or use:

walk2(), map2_lgl(), map2_int(), map2_dbl(), map2_chr()

DISCUSS WITH YOUR NEIGHBOR

1. For each function, which two arguments might be useful to iterate over?

   `download.file()`

   `rnorm()`

   `lm()`

   `predict.lm()`

   `write.csv()`

2. Which functions should we use `walk2()` or a typed version of `map2()`?

`download.file()` **for each** `url` **download to** `destfile`   `walk2(), map2_int()`

`rnorm()` **for each** `n` **generate a Normal sample with mean** `mean` **(or** `sd`**)**
(See `purrr::rerun()` **for repeating a function many times)**

`lm()` **for each** `data` **fit a model** (`formula`)

`predict.lm()` **for each model** (`object`)**, generate predictions at data**
(`newdata`)

`readr::write_csv()` **for each data frame** (`x`) **save to** `path`   `walk2()`
Similar for `ggplot::ggsave()` **for each** `plot` **save to** `filename`

DID YOU KNOW HOW DANGEROUS CHAIRS CAN BE?

NATIONAL ELECTRONIC INJURY SURVEILLANCE SYSTEM (NEISS)

From https://github.com/hadley/neiss

```
load("data/neiss_by_day.rda")
```

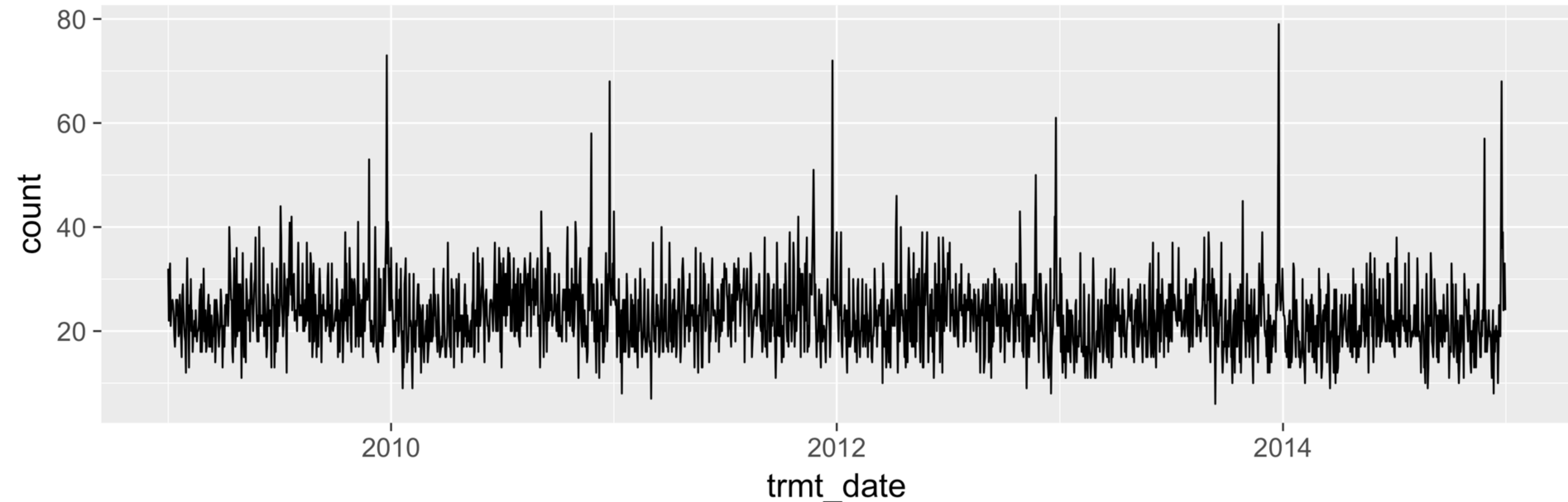`common_prods`: 11 product codes with at least 50,000 injuries 2009-2014

`common_names`: corresponding product description

`per_day`: a list with 11 elements, one for each product, injuries summarized to daily counts

ARE THERE PRODUCTS WITH PERIODIC PATTERNS IN INCIDENCE?

TAKING A LOOK

```
plots <- map(per_day, ~ ggplot(.x, aes(trmt_date, count)) + geom_line())

plots[[1]]    # try: walk(plots, print)
```


CAN WE SAVE THIS PLOT FOR ALL PRODUCTS?

DO IT FOR ONE

Solve the problem for one pair of elements

```r
one_plot <- plots[[1]]

one_code <- common_codes[[1]]


    ggsave(paste0(one_code, ".png"), one_plot)
```

TURN IT INTO A RECIPE

Make it a formula

Use .x and .y as placeholders

```
~ ggsave(paste0(   .x   , ".png"),   .y   )
```
                              code   plot

DO IT FOR ALL!

Your recipe is the .f argument to map2

```r
walk2( common_codes, plots,
    ~ ggsave(paste0(  .x  , ".png"),  .y  ))
                        code              plot
```

WHEN THE SHORTCUT, ISN'T A SHORTCUT

```
walk2(paste0(common_codes, ".png"),
      plots, ggsave)
```

WHEN THE SHORTCUT, ISN'T A SHORTCUT

```
walk2(paste0(common_codes, ".png"),
      plots, ggsave,
      width = 10, height = 3)
```

This fits a **naive** model with effects for month and day of the week to the first product:

```
lm(count ~ month + wday, data = per_day[[1]])
```

1. Fit the model to all products

2. Use `modelr::rsquare` to find the R-squared for each model (you'll probably want to look at `?modelr::rsquare`)

Bored? Repeat the plots but title them with the product names in `common_names`

```
models <- map(per_day, ~ lm(count ~ month + wday, data = .x))

map2_dbl(models, per_day, modelr::rsquare)


plots[[3]]

common_names[[3]]
```

SHOULD REALLY BE USING LIST COLUMNS...

```r
accidents <- tibble(
  name = common_names,
  code = common_codes,
  data = per_day)

accidents %>%
  mutate(
    model = map(data, ~ lm(count ~ month + wday, data = .x)),
    rsquare = map2_dbl(model, data, modelr::rsquare)) %>%
  arrange(rsquare) %>%
  select(name, rsquare)
```

CHALLENGES:

`challenges/01-mtcars.R` - Fit and summarise many regression models

`challenges/02-word_count.R` - Count the number of words of all files in a directory

`challenges/03-starwars.R` - Print who used which vehicles in the films

`challenges/04-weather.R` - Download, tidy, plot and save daily temperatures

`challenges/05-swapi.R` - Download all Star Wars data using `rwars` package

Next up: a few remaining iteration functions, a comment about other functions in purrr, wrap up.

to each element of each vector in .l, apply **.f**

pmap(.l , .f , ...)

.f(.l[[1]][[1]], .l[[2]][[1]], .l[[3]][[1]], ...)

.f(.l[[1]][[2]], .l[[2]][[2]], .l[[3]][[1]], ...)

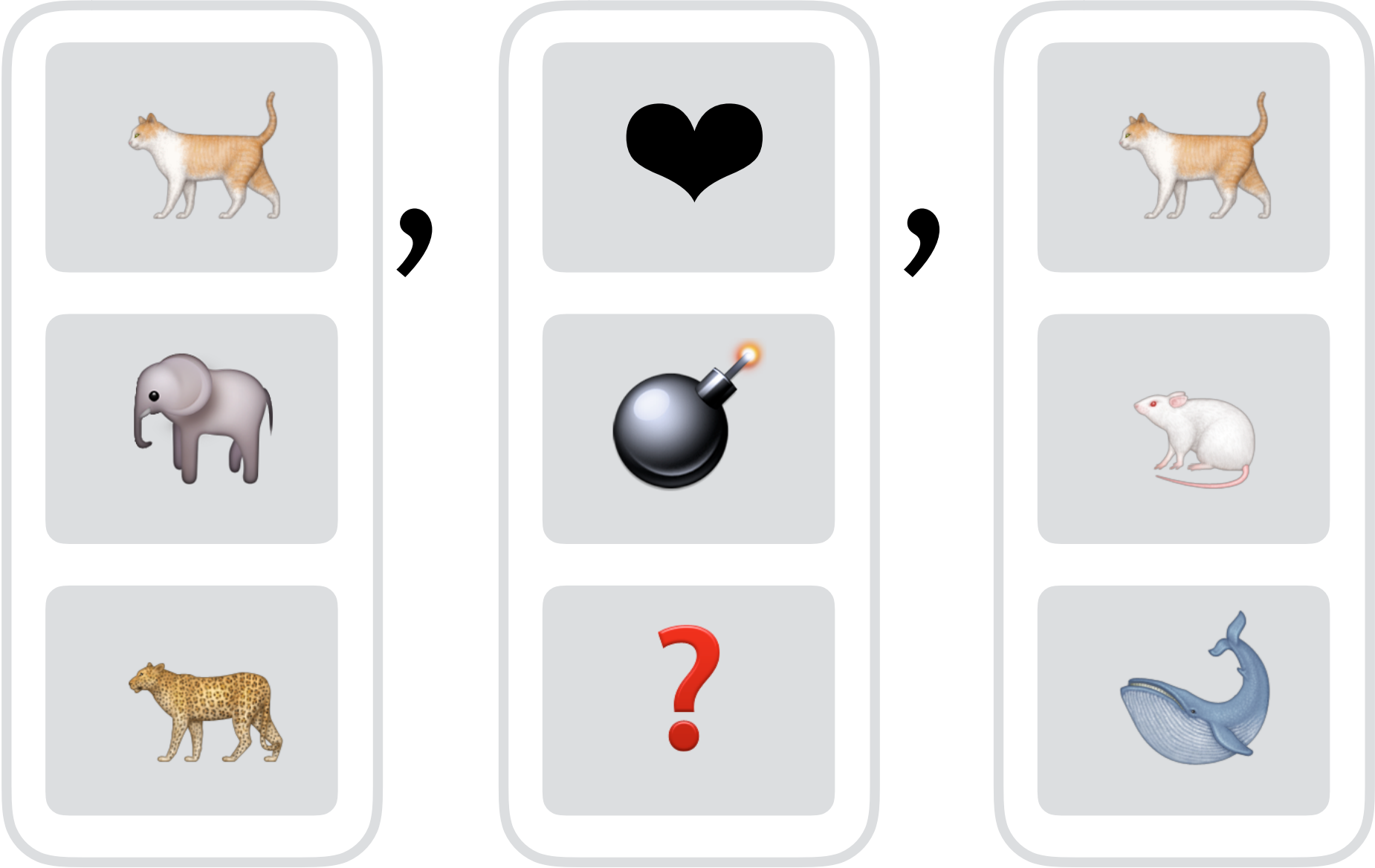.f(.l[[1]][[3]], .l[[2]][[3]], .l[[3]][[1]], ...)

and so on

or by name if supplied

to each element , apply c

pmap(data.frame(🐈, ❤, 🐈), c)

🐘 💣 🐁

🐆 ❓ 🐋

🐈❤🐈

🐘💣🐁

🐆❓🐋

no more formula shortcut

for each function in **.f**, apply it to **.x**

invoke_map(.f, .x, ...)

```
.f[[1]](.x, ...)

.f[[2]](.x, ...)

.f[[3]](.x, ...)
```

and so on
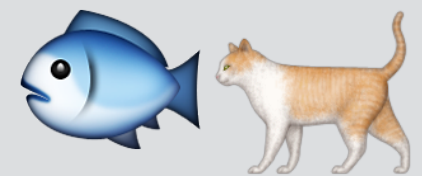
for each function in **.f**, apply it to **.x**

invoke_map(give_fish , 🐈)

double

count_legs

🐟🐈

🐈🐈

4

OTHER FEATURES OF PURRR

`06-other-features.R`

LISTS AND FUNCTIONS

Key objects in `purrr`

`purrr` provides a pile of functions to make working with them easier

WITH YOUR NEIGHBOUR

Look at the help for `safely()` and `transpose()`

What kind of objects do they expect as input?

What kind of objects are returned as output?

SAFELY() TO HANDLE ERRORS

```
urls <- list(
  example = "http://example.org",
  asdf = "http://asdfasdasdkfjlda"
)


map(urls, read_lines)


safe_readLines <- safely(readLines)
safe_readlines

# Use the safe_readLines() function with map(): html
html <- map(urls, safe_readLines)
```

TRANSPOSE() TO HANDLE RESULTS

```r
# Easier to handle transposed
str(html)
str(transpose(html))

# Extract the results: res
res <- transpose(html)[["result"]]

# Extract the errors: errs
errs <- transpose(html)[["error"]]
```

WRAP UP

purrr provides:

‣ functions that write for loops for you

‣ with consistent syntax, and

‣ convenient shortcuts for specifying functions to iterate

Choosing the right function depends on:

• type of iteration

• type of output

Check out "Bonus" cheatsheet
in your conference packet

LEARNING MORE

R for Data Science:

‣ http://r4ds.had.co.nz/iteration.html

‣ http://r4ds.had.co.nz/many-models.html

DataCamp Writing functions in R
https://www.datacamp.com/courses/writing-functions-in-r

Jenny Bryan's purrr tutorial
https://github.com/jennybc/purrr-tutorial

THANK YOU

Slides and code @ bit.ly/purrr-rstudioconf

🐦 @cvwickham

http://cwick.co.nz

cwickham@gmail.com