CHARLOTTE WICKHAM

SOLVING ITERATION PROBLEMS WITH PURRR

GETTING SETUP

1. Download slides @ bit.ly/purrr-cascadia

2. Check you have packages:

```
library(tidyverse)
library(repurrrsive)   # devtools::install_github("jennybc/repurrrsive")
```

GETTING HELP

1. Your neighbors!

2. TAs

3. Slack: #purrr

SOLVE ITERATION PROBLEMS

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 slides @ bit.ly/purrr-cascadia

YOUR TURN

```
library(repurrrsive)

# includes objects: sw_films, sw_people, sw_vehicles,
# sw_starships, sw_planets & sw_species
```

1. How many elements are in `sw_people`?
2. Who is the first person listed in `sw_people`? What information is given for this person?
3. What is the difference between `sw_people[1]` and `sw_people[[1]]`?

BEWARE!
ANSWERS ON FOLLOWING SLIDE

Download slides @ bit.ly/purrr-cascadia

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

```
sw_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/"
```

Photo credit: https://www.flickr.com/photos/alreadytaken/ CC BY 2.0

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 STARSHIPS HAS EACH CHARACTER BEEN IN?

for each person in `sw_people`, **count the number of starships**

```
map(sw_people, ____ )
```

STRATEGY

1. Do it for one element
2. Turn it into a recipe
3. Use `map()` to do it for all elements

```
luke <- sw_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 <- sw_people[[1]]
```

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

DO IT FOR ONE

Solve the problem for one element

```
luke <- sw_people[[1]]
```

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

DO IT FOR ONE

Solve the problem for one element

```
leia <- sw_people[[5]]
```

```
length(leia$starships)
```

DO IT FOR ONE

Solve the problem for one element

```
____ <- sw_people[[?]]

length(____$starships)
```

TURN IT INTO A RECIPE

Make it a formula

Use .x as a pronoun

```
~ length(.x$starships)
```

A formula

purrr's "pronoun" for
one element of our vector

DO IT FOR ALL!

Your recipe is the second argument to map

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

A formula

purrr's "pronoun" for
one element of our vector

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

Copy and paste ME.

Create `planet_lookup` (ignore details for now):

```
planet_lookup <- map_chr(sw_planets, "name")  %>%
  set_names(map_chr(sw_planets, "url"))
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(sw_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?

ROAD `map()`

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

Other types of output

Other ways of specifying .f

Other iteration functions

ROAD `map()`

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

Other types of output

Other ways of specifying .f

Other iteration functions

ROAD `map()`

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

Other types of output

Other ways of specifying .f

Other iteration functions

map() details

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

SIMPLER OUTPUT:

`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
sw_people <- sw_people %>% set_names(map_chr(sw_people, "name"))
```

REPLACE map() WITH THE APPROPRIATELY TYPED FUNCTION

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

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

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

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

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

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

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

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

```
# How heavy is each character?

map_dbl(sw_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(sw_people, ~ .x[["mass"]])

## [[1]]

## [1] "77"

##

## [[2]]

## [1] "75"

...
```

```
# A little risky

map_dbl(sw_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(sw_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(sw_people, ~ length(.x[["starships"]]))
map_chr(sw_people, ~ .x[["hair_color"]])
map_chr(sw_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(sw_people, ~ .x[["hair_color"]])
# becomes
map_chr(sw_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(sw_people, "starships")
map_int(char_starships, length)

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

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

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

FROM EARLIER...

Create `planet_lookup` ~~(ignore details for now)~~:

```
planet_lookup <- map_chr(sw_planets, "name") %>%
  set_names(map_chr(sw_planets, "url"))
planet_lookup
```

```
x %>% set_names(y)
```

equivalent to

```
names(x) <- y
x
```

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 `sw_films`) has the most characters?

Which `sw_species` has the most possible eye colors?

Which `sw_planets` do we know the least about (i.e. have the most `"unknown"` entries)?

BREAK?

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

map(sw_films, "characters") %>%
  map_int(length) %>%
  set_names(map_chr(sw_films, "title")) %>%
  sort()
```

```
# Which species has the most possible eye colors?
sw_species[[1]]$eye_colors


map_chr(sw_species, "eye_colors") %>%
  strsplit(", ") %>%
  map_int(length)
# this is lazy, what about n/a and unknown?
```

More iteration functions

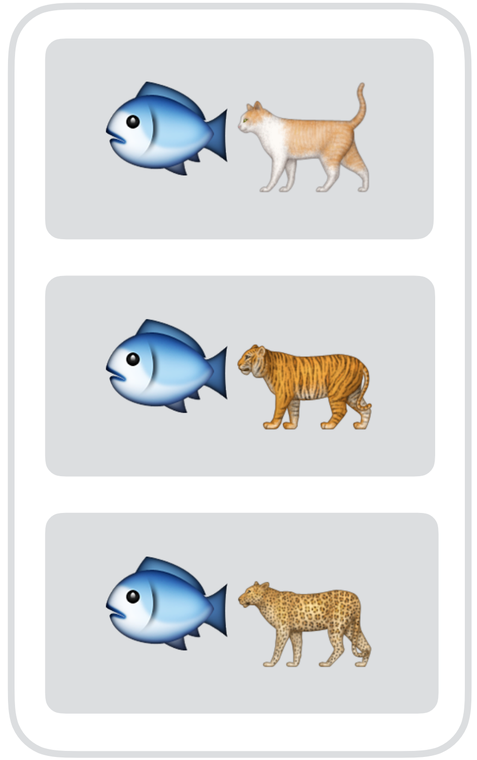
to each element of **.x** apply **.f**

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

to each `cat` apply `give_fish`

```
map(
```

```
, give_fish)
```



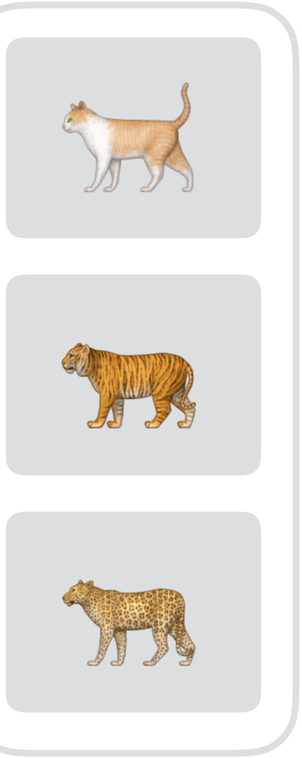
to each element of **.x** apply **.f**

```
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**

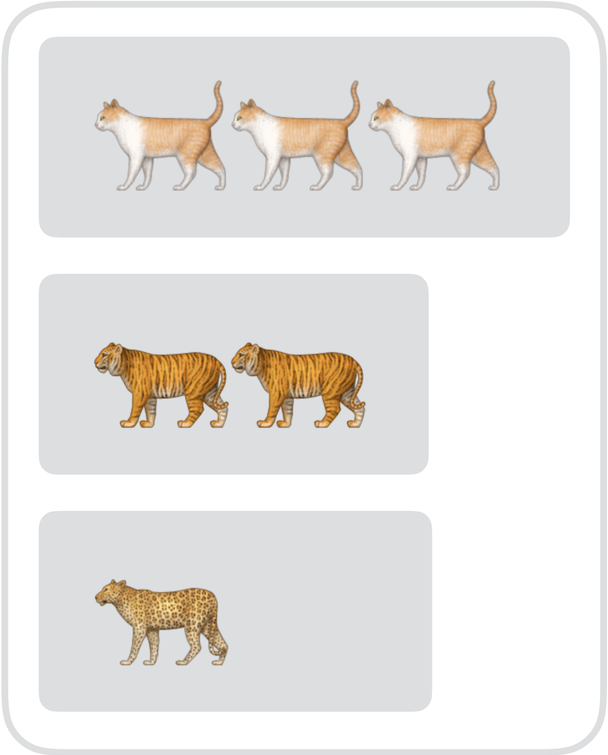
```
map2(  .x ,  .y , .f)
```

to each `cat` and corresponding `times` apply `rep`

```
map2(
```

```
,
```


```
,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. For which functions above 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`

YOUR TURN

```
jan_sales <- read_csv("jan.csv")

jan_sales <- mutate(jan_sales, month = "jan")


feb_sales <- read_csv("feb.csv")

feb_sales <- mutate(feb_sales, month = "feb")


mar_sales <- read_csv("mar.csv")

mar_sales <- mutate(mar_sales, month = "mar")


sales <- bind_rows(jan_sales, feb_sales, mar_sales)
```

WHAT DOES THIS CODE DO?

REDUCE DUPLICATION (AND MISTAKES) WITH PURRR

```
months <- c("jan", "feb", "mar")

files <- paste0(months, ".csv")

sales_list <- map(files, read_csv)
```

Now...For each **element** (do) **add a month column**

USE THE SAME STRATEGY!

DO IT FOR ONE

Solve the problem for one element

```
mutate(sales_list[[1]],
       month = months[[1]])
```

DO IT FOR ONE

Solve the problem for one element

```
mutate(sales_list[[1]],
       month = months[[1]])
```

DO IT FOR ONE

Solve the problem for one element

```
mutate(sales_list[[2]],
       month = months[[2]])
```

DO IT FOR ONE

Solve the problem for one element

```
mutate(sales_list[[2]],
       month = months[[2]])
```

Iterating over two objects!

TURN IT INTO A RECIPE

Make it a formula

Use .x and .y

```
mutate(sales_list[[2]],
       month = months[[2]])
```

TURN IT INTO A RECIPE

Make it a formula

Use .x and .y

```
~ mutate(sales_list[[2]],
         month = months[[2]])
```

A formula

TURN IT INTO A RECIPE

Make it a formula

Use .x and .y

```
~ mutate(    .x          ,
         month =    .y      )
```

A formula

DO IT FOR ALL!

Your recipe is the .f argument to map2

```
map2(.x = sales_files,
     .y = months,
  ~ mutate(    .x          ,
           month =    .y        ))
```

A formula

```
months <- c("jan", "feb", "mar")

files <- paste0(months, ".csv")

sales_list <- map(files, read_csv)

sales_list_months <- map2(.x = sales_list,
                          .y = months,
                          .f = ~ mutate(.x, month = .y)

bind_rows(sales_list_months)
```

```
library(repurrrsive)

gap_split_small <- gap_split[1:10]

countries <- names(gap_split_small)
```

FOR EACH COUNTRY CREATE A GGPLOT OF LIFE EXPECTANCY THROUGH TIME WITH A TITLE

Need a hint? For one country, see next slide

Bored? For each plot, save it to a .pdf, with an appropriate file name

```r
# For one country

ggplot(gap_split[[1]], aes(year, lifeExp)) +

  geom_line() +

  labs(title = countries[[1]])
```

```
# For all countries

plots <- map2(gap_split_small, countries,

  ~ ggplot(.x, aes(year, lifeExp)) +

      geom_line() +

      labs(title = .y))


plots[[1]]


# Display all plots

walk(plots, print) # this might take awhile
```

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 `sw_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

```
library(tidyverse)

people_tbl <- tibble(
  name    =                                  ,
  films   =                                ,
  height  =

                                                  ,
  species =
)
```

PURRR CAN HELP TURN LISTS INTO TIBBLES

```
library(tidyverse)

people_tbl <- tibble(
  name    = sw_people %>% map_chr("name"),
  films   = sw_people %>% map("films"),
  height  = sw_people %>% map_chr("height") %>%
              readr::parse_number(na = "unknown"),
  species = sw_people %>% map_chr("species", .null = NA_character_)
)
```

will result in list column

needs some parsing

isn't in every element

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)
)
```

```
library(tidyverse)
library(repurrrsive)

# A useful lookup table ----------------------------------------------
film_number_lookup <- map_chr(sw_films, "url") %>%
  map(~ stringr::str_split_fixed(.x, "/", 7)[, 6])  %>%
  as.numeric() %>%
  set_names(map_chr(sw_films, "url"))

people_tbl <- tibble(
  name    = sw_people %>% map_chr("name"),
  films   = sw_people %>% map("films"),
  height  = sw_people %>% map_chr("height") %>%
    readr::parse_number(na = "unknown"),
  species = sw_people %>% map_chr("species", .null = NA_character_)
)

# Turning parts of our list to a tibble ---------------------------------
people_tbl$films

# Use map with mutate to manipulate list columns
people_tbl <- people_tbl %>%
  mutate(
    film_numbers = map(films,
      ~ film_number_lookup[.x]),
    n_films = map_int(films, length)
  )

people_tbl %>% select(name, film_numbers, n_films)
```

Copy and paste ME.

YOUR TURN

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

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

```
?paste
```

```
people_tbl <- people_tbl %>%
  mutate(films_squashed = map_chr(film_numbers, paste,
                                  collapse = ", "))

people_tbl %>% select(name, n_films, films_squashed)
```

CHALLENGES @ https://github.com/cwickham/purrr-tutorial

`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.

OTHER FEATURES OF PURRR

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

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

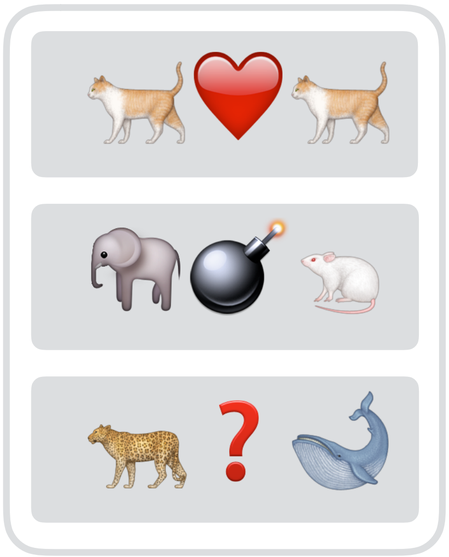
```
.f(.l[[1]][[1]], .l[[2]][[1]], .l[[3]][[1]], ...)

.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 `in animal, reaction, and animal2`, apply `c`

```
pmap(data.frame( 🐈 , ❤️ , 🐈 ), c )
     🐘   💣   🐀
     🐆   ❓   🐋
```



no formula shortcut

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

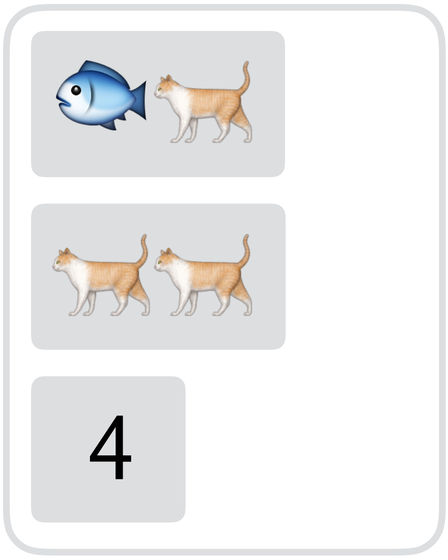
`invoke_map(.f, .x, ...)`

```
.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
```



LISTS AND FUNCTIONS

Key objects in `purrr`

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

Functions: `safely()`, `possibly()`, `partial()`

Lists: `transpose()`, `accumulate()`, `reduce()`, `every()`, `order_by()`

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

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 @ bit.ly/purrr-cascadia

All materials (code files too): https://github.com/cwickham/purrr-tutorial

@cvwickham

http://cwick.co.nz

cwickham@gmail.com