RESEARCH INSTITUTE
NATURE AND FOREST

CODING CLUB

23 OCTOBER 2018

Herman Teirlinck,
01.72 - Kaat Tilley

```
> library(cowsay)
> say("Welcome to the coding club!", "turkey")
 -------------
Welcome to the coding club!
 --------------
              \
               \
                \
               .--.
              /} p \             /}
             `~)-) /           /` }
              ( / /          /`}.' }
               / / .-'""-.  / ' }-'}
              / (.'       \/ '.'}_.}
             |            `}   .}._}
             |     .-=-';   } ' }_.}
             \    `.-=-;'  } '.}.-}
              '.   -=-'    ;,}._.}
                `-,_  __.'` '-._}
              jgs   `|||
                   .=='=,
```

LOOPS

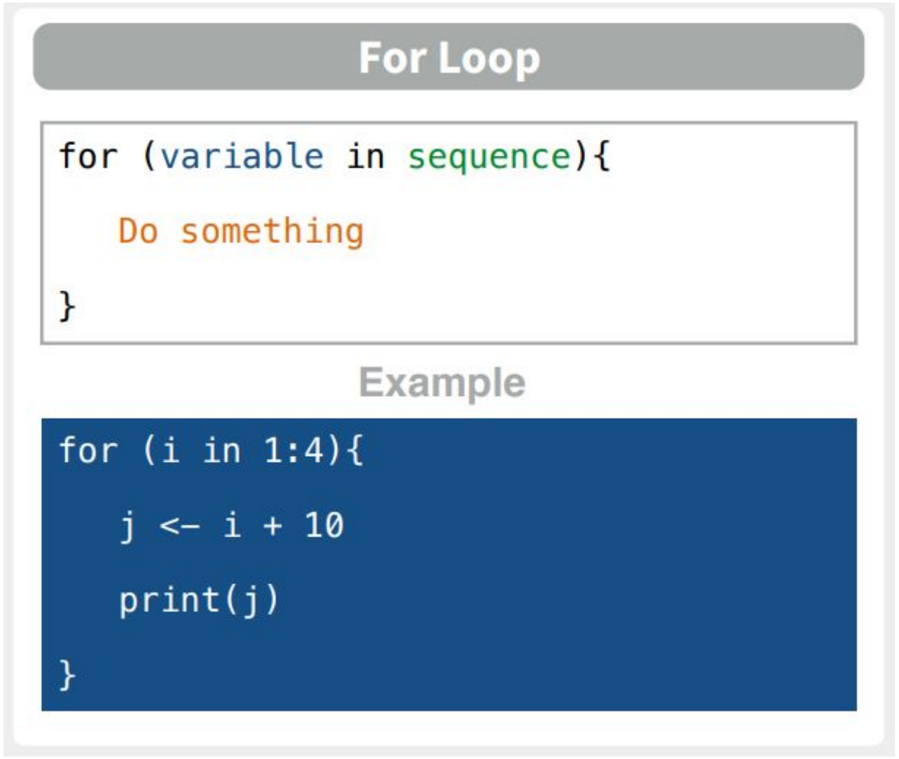
Install the package suite:

```
install.packages("tidyverse")
install.packages("rgbif")
```

Load the package suite:

```
library(tidyverse)
library(rgbif)
```

Old skool...


For Loop

```
for (variable in sequence){
   Do something
}
```

Example

```
for (i in 1:4){
   j <- i + 10
   print(j)
}
```

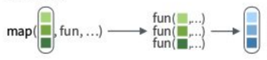
New kids:

purrr

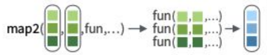
Apply functions with purrr : : **CHEAT SHEET**

Apply Functions

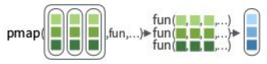
Map functions apply a function iteratively to each element of a list or vector.



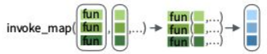
map(.x, .f, …) Apply a function to each element of a list or vector. *map(x, is.logical)*



map2(.x, .y, .f, …) Apply a function to pairs of elements from two lists, vectors. *map2(x, y, sum)*



pmap(.l, .f, …) Apply a function to groups of elements from list of lists, vectors. *pmap(list(x, y, z), sum, na.rm = TRUE)*



invoke_map(.f, .x = list(NULL), …, .env=NULL) Run each function in a list. Also **invoke**. *l <- list(var, sd); invoke_map(l, x = 1:9)*

lmap(.x, .f, ...) Apply function to each list-element of a list or vector.
imap(.x, .f, ...) Apply .f to each element of a list or vector and its index.

OUTPUT

map(), map2(), pmap(), imap and **invoke_map** each return a list. Use a suffixed version to return the results as a specific type of flat vector, e.g. **map2_chr**, **pmap_lgl**, etc.

Use **walk**, **walk2**, and **pwalk** to trigger side effects. Each return its input invisibly.

function	returns
map	list
map_chr	character vector
map_dbl	double (numeric) vector
map_dfc	data frame (column bind)
map_dfr	data frame (row bind)
map_int	integer vector
map_lgl	logical vector
walk	triggers side effects, returns the input invisibly

SHORTCUTS - within a purrr function:

"name" becomes **function(x) x[["name"]]**, e.g. *map(l, "a")* extracts *a* from each element of *l*

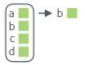
~ .x becomes **function(x) x**, e.g. *map(l, ~ 2 +.x)* becomes *map(l, function(x) 2 + x)*

~ .x .y becomes **function(.x, .y) .x .y**, e.g. *map2(l, p, ~ .x +.y)* becomes *map2(l, p, function(l, p) l + p)*

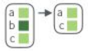
~ ..1 ..2 etc becomes **function(..1, ..2**, etc) **..1 ..2** etc, e.g. *pmap(list(a, b, c), ~ ..3 + ..1 - ..2)* becomes *pmap(list(a, b, c), function(a, b, c) c + a - b)*

Work with Lists

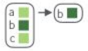
FILTER LISTS



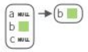
pluck(.x, ..., .default=NULL) Select an element by name or index, *pluck(x,"b")* ,or its attribute with **attr_getter**. *pluck(x,"b",attr_getter("n"))*



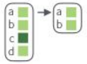
keep(.x, .p, …) Select elements that pass a logical test. *keep(x, is.na)*



discard(.x, .p, …) Select elements that do not pass a logical test. *discard(x, is.na)*

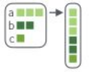


compact(.x, .p = identity) Drop empty elements. *compact(x)*

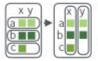


head_while(.x, .p, …) Return head elements until one does not pass. Also **tail_while**. *head_while(x, is.character)*

RESHAPE LISTS

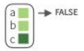


flatten(.x) Remove a level of indexes from a list. Also **flatten_chr**, **flatten_dbl**, **flatten_dfc**, **flatten_dfr**, **flatten_int**, **flatten_lgl**. *flatten(x)*

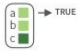


transpose(.l, .names = NULL) Transposes the index order in a multi-level list. *transpose(x)*

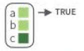
SUMMARISE LISTS



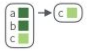
every(.x, .p, …) Do all elements pass a test? *every(x, is.character)*



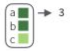
some(.x, .p, …) Do some elements pass a test? *some(x, is.character)*



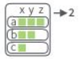
has_element(.x, .y) Does a list contain an element? *has_element(x, "foo")*



detect(.x, .f, ..., .right=FALSE, .p) Find first element to pass. *detect(x, is.character)*

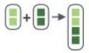


detect_index(.x, .f, ..., .right = FALSE, .p) Find index of first element to pass. *detect_index(x, is.character)*

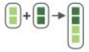


vec_depth(x) Return depth (number of levels of indexes). *vec_depth(x)*

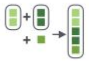
JOIN (TO) LISTS



append(x, values, after = length(x)) Add to end of list. *append(x, list(d = 1))*

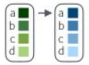


prepend(x, values, before = 1) Add to start of list. *prepend(x, list(d = 1))*

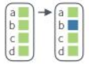


splice(…) Combine objects into a list, storing S3 objects as sub-lists. *splice(x, y, "foo")*

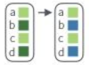
TRANSFORM LISTS



modify(.x, .f, ...) Apply function to each element. Also **map**, **map_chr**, **map_dbl**, **map_dfc**, **map_dfr**, **map_int**, **map_lgl**. *modify(x, ~.+ 2)*



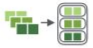
modify_at(.x, .at, .f, ...) Apply function to elements by name or index. Also **map_at**. *modify_at(x, "b", ~.+ 2)*



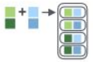
modify_if(.x, .p, .f, ...) Apply function to elements that pass a test. Also **map_if**. *modify_if(x, is.numeric,~.+2)*

modify_depth(.x,.depth,.f,...) Apply function to each element at a given level of a list. *modify_depth(x, 1, ~.+ 2)*

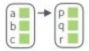
WORK WITH LISTS



array_tree(array, margin = NULL) Turn array into list. Also **array_branch**. *array_tree(x, margin = 3)*

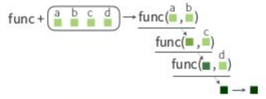


cross2(.x, .y, .filter = NULL) All combinations of .x and .y. Also **cross**, **cross3**, **cross_df**. *cross2(1:3, 4:6)*

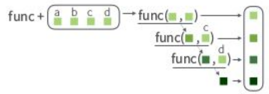


set_names(x, nm = x) Set the names of a vector/list directly or with a function. *set_names(x, c("p", "q", "r"))* *set_names(x, tolower)*

Reduce Lists



reduce(.x, .f, ..., .init) Apply function recursively to each element of a list or vector. Also **reduce_right**, **reduce2**, **reduce2_right**. *reduce(x, sum)*



accumulate(.x, .f, ..., .init) Reduce, but also return intermediate results. Also **accumulate_right**. *accumulate(x, sum)*

Modify function behavior

compose() Compose multiple functions.

lift() Change the type of input a function takes. Also **lift_dl**, **lift_dv**, **lift_ld**, **lift_lv**, **lift_vd**, **lift_vl**.

rerun() Rerun expression n times.

negate() Negate a predicate function (a pipe friendly !)

partial() Create a version of a function that has some args preset to values.

safely() Modify func to return list of results and errors.

quietly() Modify function to return list of results, output, messages, warnings.

possibly() Modify function to return default value whenever an error occurs (instead of error).

Share your snippets and solutions during the coding session:

Go to https://hackmd.io/jwSucdiFQDCcIFSbHgLCCg and post your code in between backticks:

For example:

~~~~
```

library(tidyverse)

my_data <- ...

```
~~~~

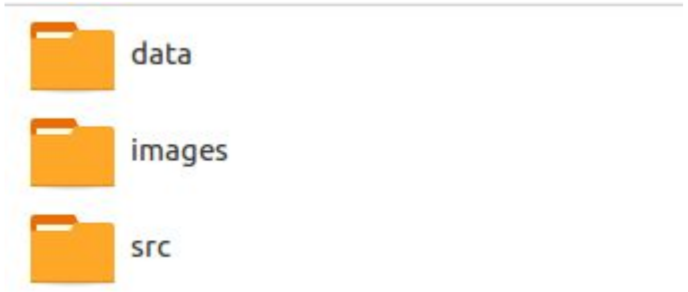
The concept

We defined a number of challenges. If you were able to achieve a challenge, add a to r laptop screen.

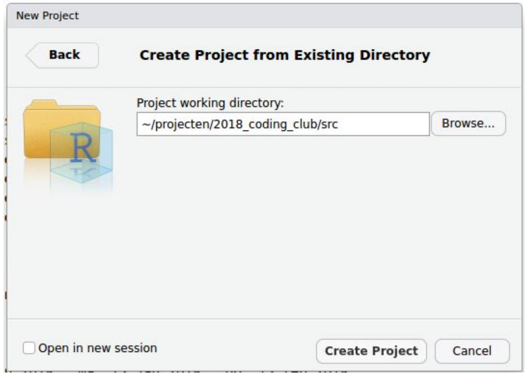
The objective is that **everyone** achieves !

- Someone has more than you? **Ask for help!**
- Someone has less than you? **Provide help!**

- Download coding club material and work locally, **not in sync** with the Google drive



- Create new Rstudio project in the **/src** folder


New Project

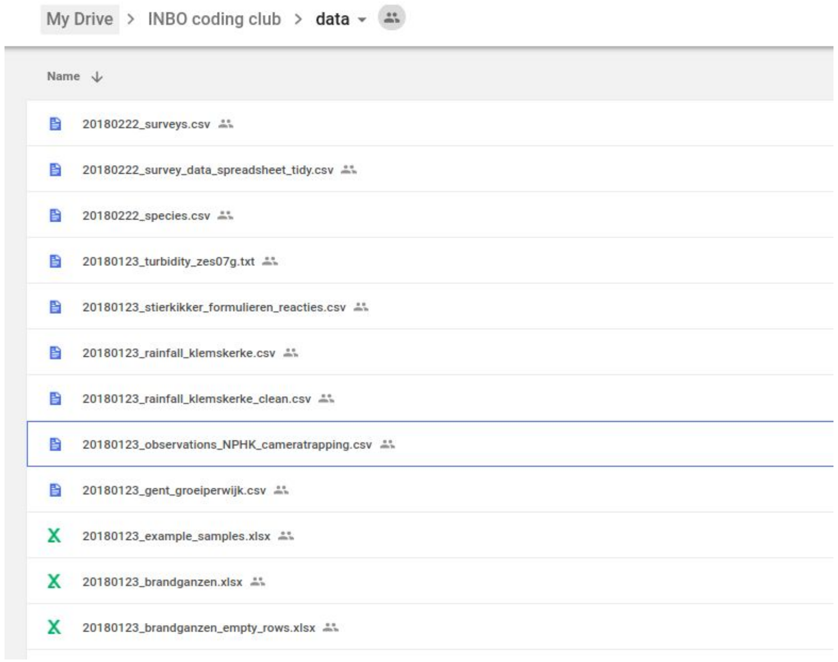
Create Project from Existing Directory

Project working directory:

~/projecten/2018_coding_club/src

- Download coding club material and work locally, not in sync with the Google drive
- Create new Rstudio project in the **src** folder…
- Use relative paths to data files!

```
> library(readr)
> read_csv2("../data/20180123_gent_groeiperwijk.csv")
```

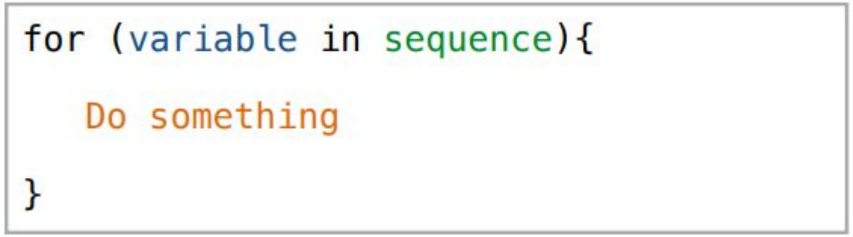


20180222_surveys.csv
20180222_survey_data_spreadsheet_tidy.csv
20180222_species.csv
20180123_turbidity_zes07g.txt
20180123_stierkikker_formulieren_reacties.csv
20180123_rainfall_klemskerke.csv
20180123_rainfall_klemskerke_clean.csv
20180123_observations_NPHK_cameratrapping.csv
20180123_gent_groeiperwijk.csv
20180123_example_samples.xlsx
20180123_brandganzen.xlsx
20180123_brandganzen_empty_rows.xlsx

For this coding club:

`20180522_gent_groeiperwijk_tidy.csv`

`20180222_species.csv`

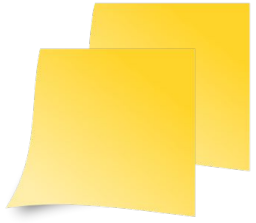


```
for (variable in sequence){

   Do something

}
```

This code makes and saves a plot of the demographic evolution of Ghent’s districts for year 2000.

```
library(readr)

groei_gent_df <- read_csv("../data/20180522_gent_groeiperwijk_tidy.csv" )

year_plot <- groei_gent_df %>% filter(year == 2000) %>%

  ggplot(aes(x = wijk, y = growth)) + geom_bar(stat = "identity") +

  coord_flip()

ggsave(file.path("..", "images", "district_evol_2000.png"), year_plot)
```

How to do the same for all years from 2000 to 2003 using a `for` loop?

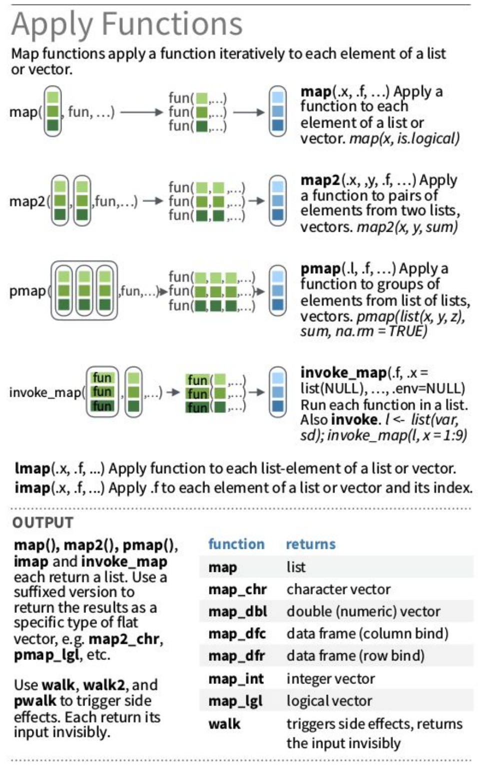


Apply function name_backbone() to a vector/list of taxa (`animals` in the example below).

`name_backbone()` : lookup names in the GBIF backbone taxonomy.

```
library(rgbif)

name_backbone("Branta", rank = "GENUS")
name_backbone("Sus", rank = "GENUS")
animals <- c("Branta", "Sus")
...
```

Can you get the result as a `data.frame` using `purrr` package as well?



Apply Functions

Map functions apply a function iteratively to each element of a list or vector.

map(.x, .f, …) Apply a function to each element of a list or vector. *map(x, is.logical)*

map2(.x, .y, .f, …) Apply a function to pairs of elements from two lists, vectors. *map2(x, y, sum)*

pmap(.l, .f, …) Apply a function to groups of elements from list of lists, vectors. *pmap(list(x, y, z), sum, na.rm = TRUE)*

invoke_map(.f, .x = list(NULL), …, .env=NULL) Run each function in a list. Also **invoke**. *l <- list(var, sd); invoke_map(l, x = 1:9)*

lmap(.x, .f, ...) Apply function to each list-element of a list or vector.
imap(.x, .f, ...) Apply .f to each element of a list or vector and its index.

OUTPUT

map(), map2(), pmap(), imap and **invoke_map** each return a list. Use a suffixed version to return the results as a specific type of flat vector, e.g. **map2_chr, pmap_lgl**, etc.

Use **walk, walk2**, and **pwalk** to trigger side effects. Each return its input invisibly.

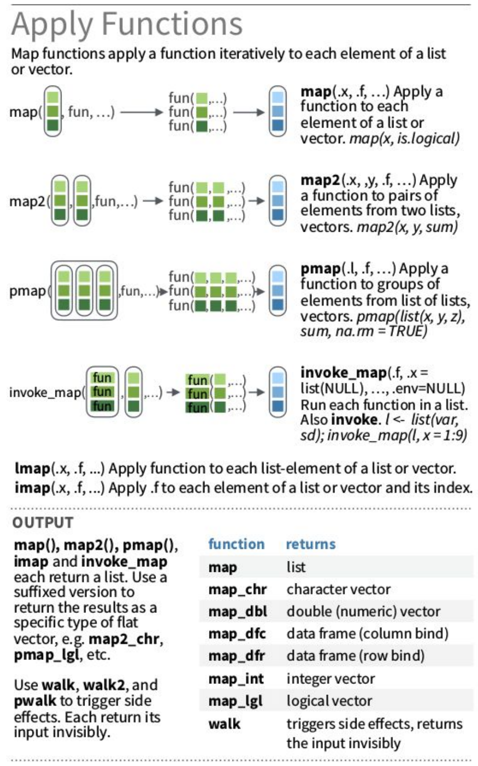
function	returns
map	list
map_chr	character vector
map_dbl	double (numeric) vector
map_dfc	data frame (column bind)
map_dfr	data frame (row bind)
map_int	integer vector
map_lgl	logical vector
walk	triggers side effects, returns the input invisibly

Note: Many functions are capable of dealing with a **vector as input**, cfr. https://jennybc.github.io/purrr-tutorial/bk00_vectors-and-lists.html#vectorized_operations

Apply function `name_backbone()` to all scientific names of the species data (`genus` + `species`) and add the results (multiple columns) to original data frame `df_species`

```
df_species <- read_csv("../data/20180222_species.csv")
df_species %>%
    mutate(scientific_name = str_c(genus, species,
                                           sep = " ")) %>%
    ...
```

```
# A tibble: 55 x 27
  usageKey scientificName canonicalName rank  status confidence matchType kingdom phylum order family genus species
     <int> <chr>          <chr>         <chr> <chr>       <int> <chr>     <chr>   <chr>  <chr> <chr>  <chr> <chr>
 1 2491757 Amphispiza bi… Amphispiza b… SPEC… ACCEP…         98 EXACT     Animal… Chord… Pass… Ember… Amph… Amphis…
 2 2437568 Ammospermophi… Ammospermoph… SPEC… ACCEP…         96 FUZZY     Animal… Chord… Rode… Sciur… Ammo… Ammosp…
 3 2491123 Ammodramus sa… Ammodramus s… SPEC… ACCEP…         98 EXACT     Animal… Chord… Pass… Ember… Ammo… Ammodr…
...
 8 2444480 Crotalus scut… Crotalus scu… SPEC… ACCEP…         92 FUZZY     Animal… Chord… Squa… Viper… Crot… Crotal…
 9 8071886 Cnemidophorus… Cnemidophoru… SPEC… SYNON…         97 EXACT     Animal… Chord… Squa… Teiid… Aspi… Aspido…
10 5227544 Cnemidophorus… Cnemidophoru… SPEC… SYNON…         98 EXACT     Animal… Chord… Squa… Teiid… Aspi… Aspido…
# ... with 45 more rows, and 14 more variables: kingdomKey <int>, phylumKey <int>, classKey <int>, orderKey <int>,
#   familyKey <int>, genusKey <int>, speciesKey <int>, synonym <lgl>, class <chr>, acceptedUsageKey <int>,
#   species_id <chr>, genus <chr>, species1 <chr>, taxa <chr>
```


map(🟩 , fun, …)
fun(🟩 ,…)
map2(🟩 , 🟩 ,fun,…)
pmap(🟩 🟩 🟩 ,fun,…)
invoke_map(fun fun fun , 🟩 ,…)
fun(🟩 ,…)

Apply Functions

Map functions apply a function iteratively to each element of a list or vector.

map(.x, .f, …) Apply a function to each element of a list or vector. *map(x, is.logical)*

map2(.x, ,y, .f, …) Apply a function to pairs of elements from two lists, vectors. *map2(x, y, sum)*

pmap(.l, .f, …) Apply a function to groups of elements from list of lists, vectors. *pmap(list(x, y, z), sum, na.rm = TRUE)*

invoke_map(.f, .x = list(NULL), …, .env=NULL) Run each function in a list. Also **invoke**. *l <- list(var, sd); invoke_map(l, x = 1:9)*

lmap(.x, .f, ...) Apply function to each list-element of a list or vector.
imap(.x, .f, ...) Apply .f to each element of a list or vector and its index.

OUTPUT

map(), map2(), pmap(), imap and **invoke_map** each return a list. Use a suffixed version to return the results as a specific type of flat vector, e.g. **map2_chr, pmap_lgl**, etc.

Use **walk**, **walk2**, and **pwalk** to trigger side effects. Each return its input invisibly.

function	returns
map	list
map_chr	character vector
map_dbl	double (numeric) vector
map_dfc	data frame (column bind)
map_dfr	data frame (row bind)
map_int	integer vector
map_lgl	logical vector
walk	triggers side effects, returns the input invisibly

Go to https://hackmd.io/jwSucdiFQDCcIFSbHgLCCg...

RESEARCH INSTITUTE
NATURE AND FOREST

CODING
CLUB

Zaal: Herman Teirlinck - 01.71 - Frans Breziers

Datum: 2018-11-29, van 10:00 tot 12:00

(registration announced via DG_useR@inbo.be)