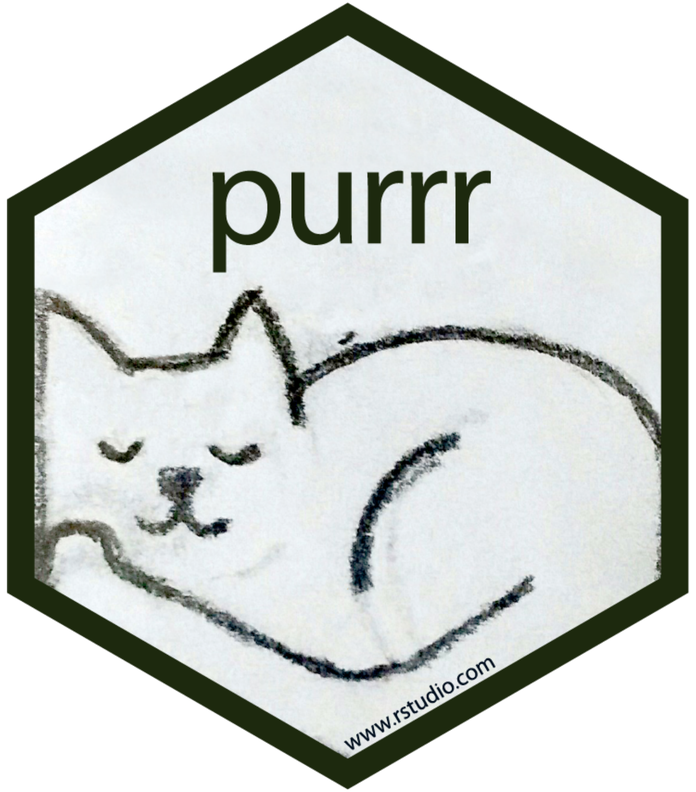
Lists with



Navigate up to the **05-Lists** folder.

Open on **05-Lists-Exercises**.

Lists

Your Turn 1

Run the code below, which displays a list. What do you see?

```
sw_people
```

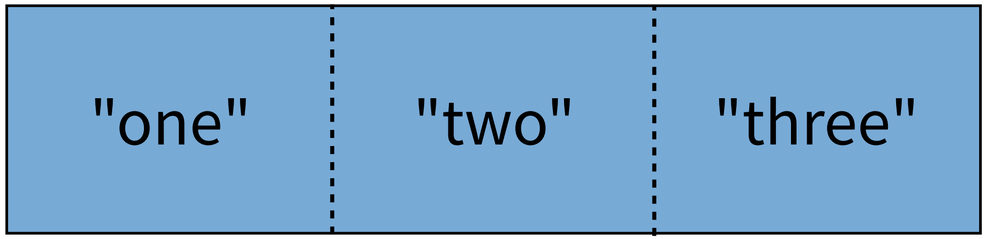
Quiz

What is the difference between an atomic vector and a list?

Atomic Vector  type

```
c("one", "two", "three")
```

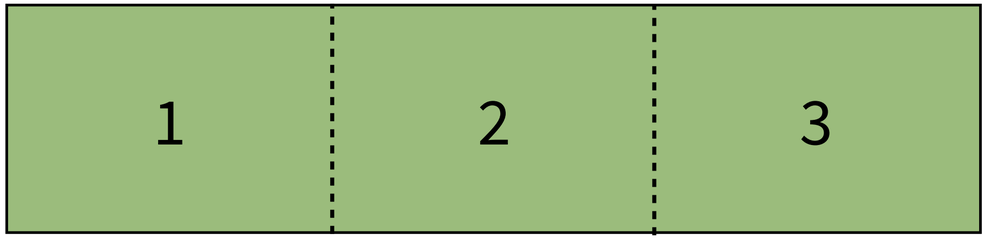
Atomic Vector



character

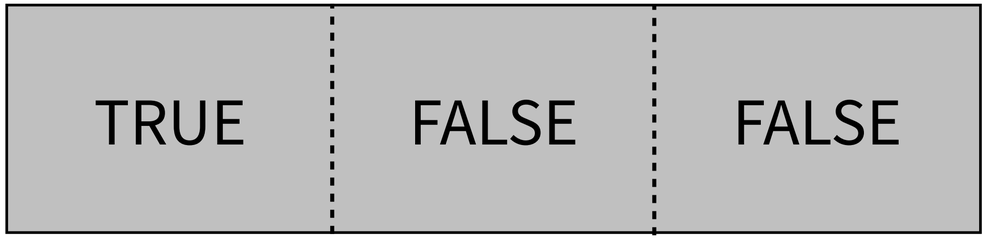
```
c("one", "two", "three")
```

Atomic Vector



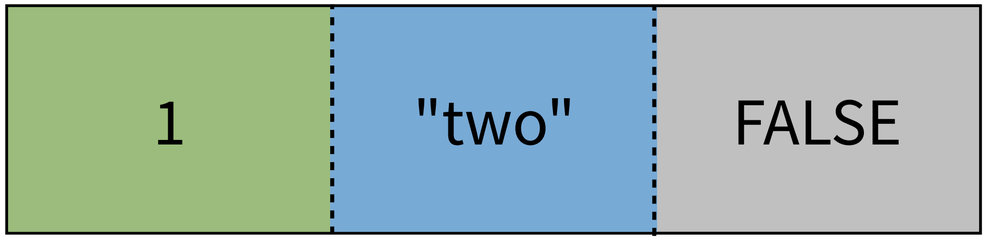
double

Atomic Vector



logical

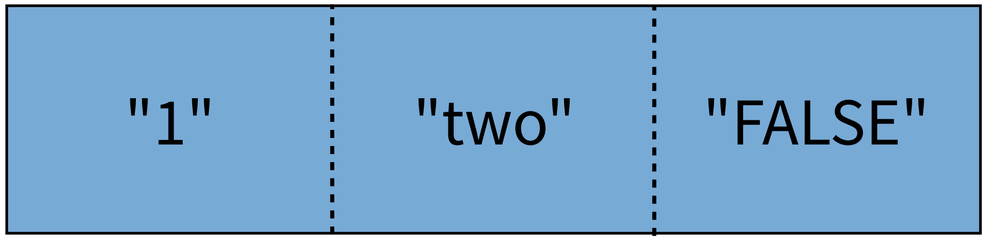
Atomic Vector



?

```
c(1, "two", FALSE)
```

Atomic Vector

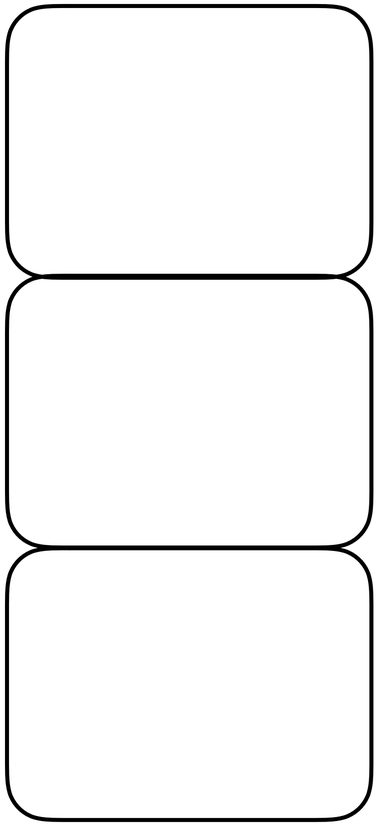


character

Atomic Vector

type

List

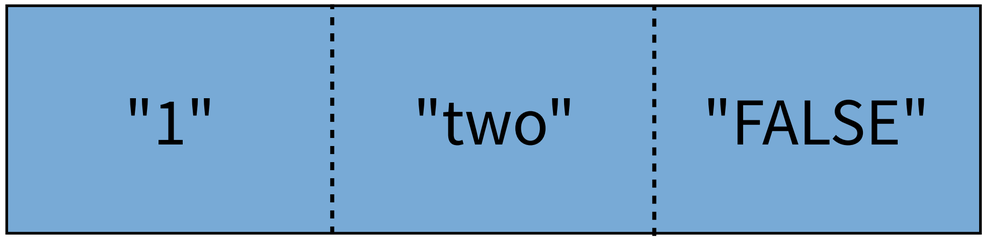


type

type

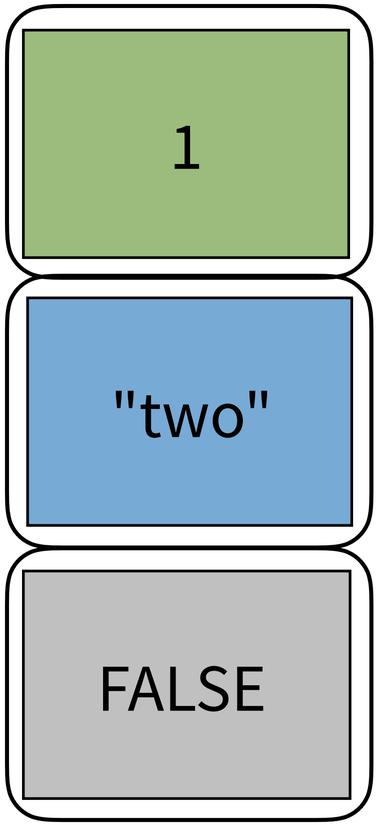
type

Atomic Vector



character

List

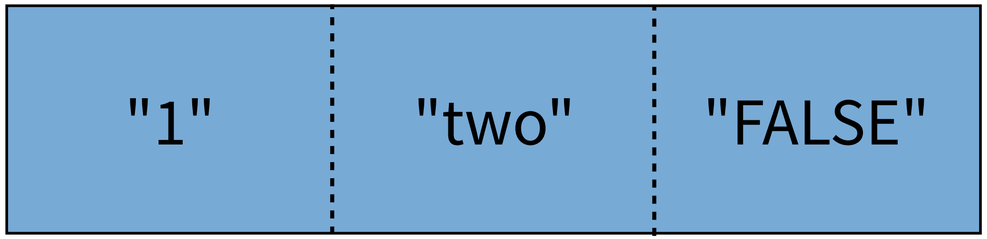


double

character

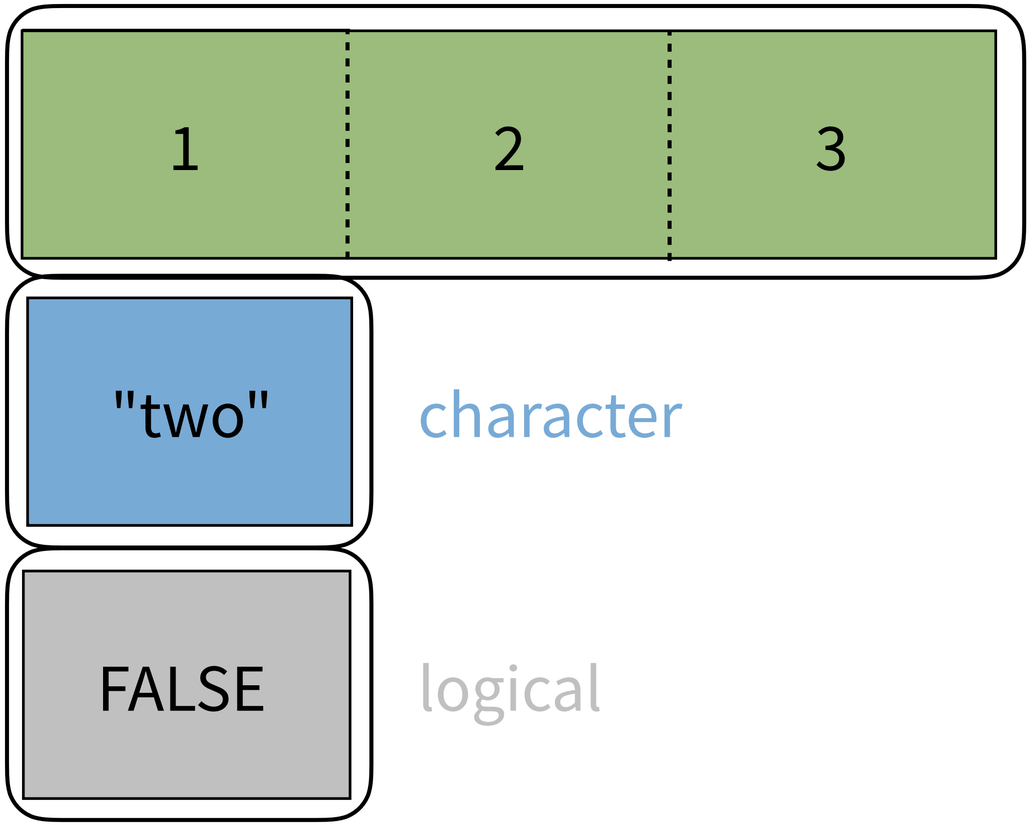
logical

Atomic Vector



character

List

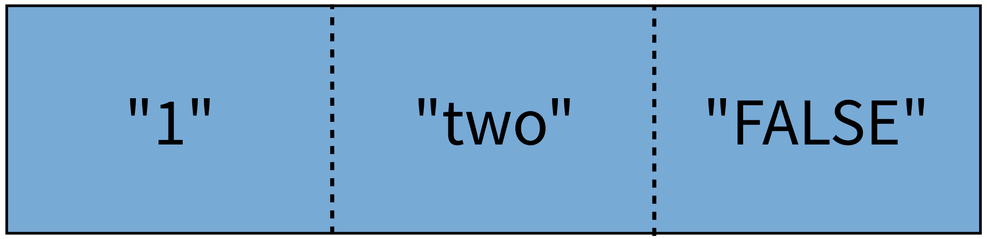


double

character

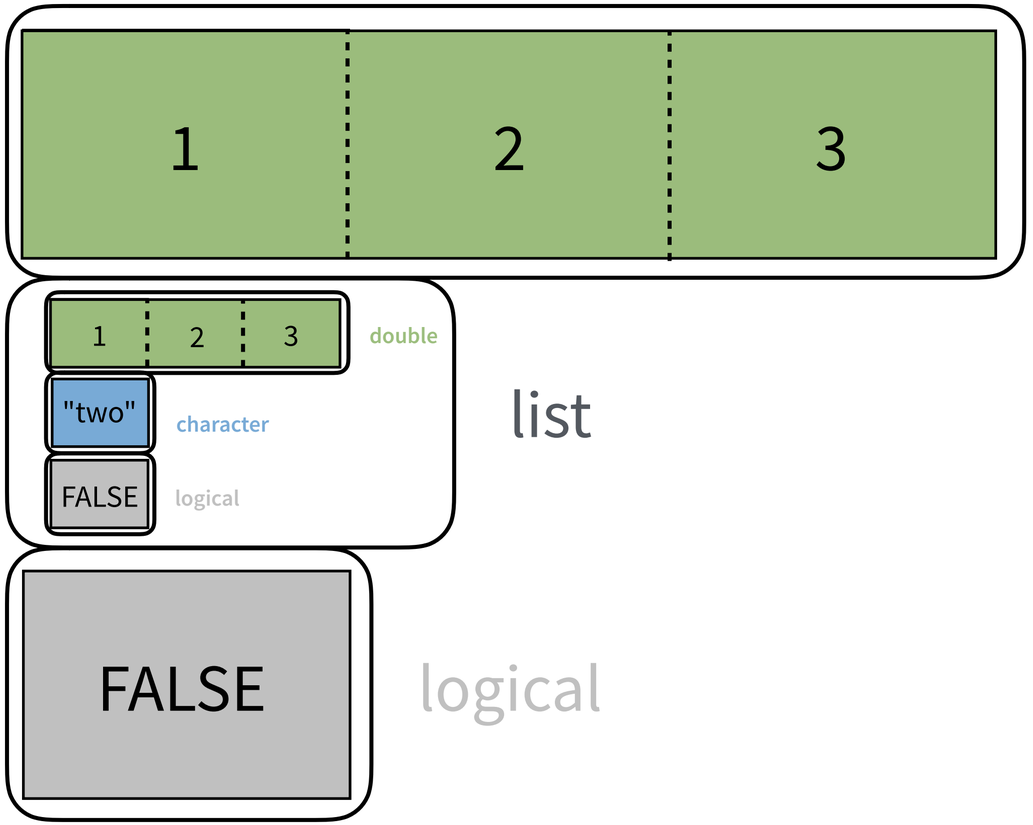
logical

Atomic Vector



character

List



list()

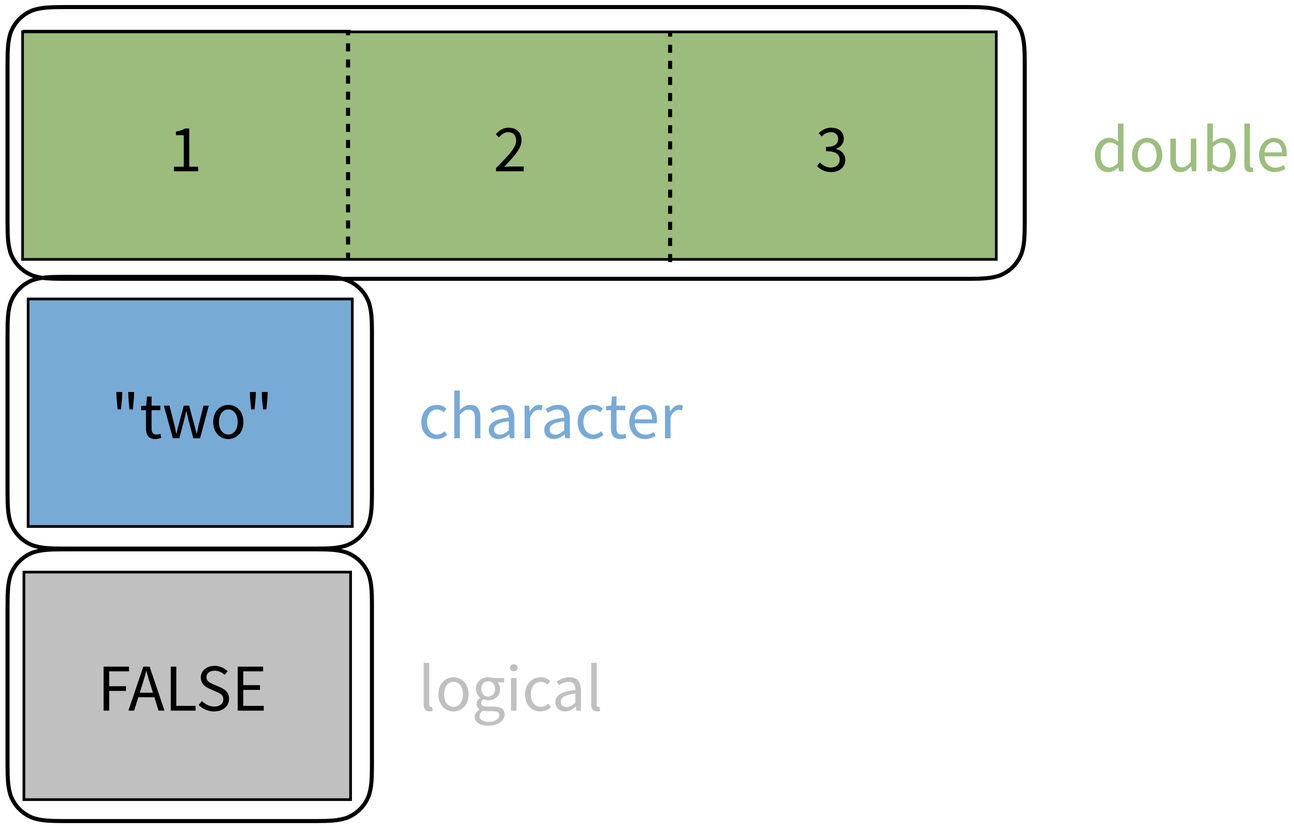
Combines objects into a list.

```
list(x = c(1,2,3), y = "two", z = "FALSE")
```

```
$x
[1] 1 2 3

$y
[1] "two"

$z
[1] "FALSE"
```



Where you find lists in R

1. JSON/XML data
2. Model objects
3. Plots
4. Function Output

EVERYWHERE

Main difficulties with lists

1. **Viewing** contents
2. **Extracting** contents

Quiz

```
vec <- c(-2, -1, 0)
lst <- list(-2, -1, 0)
```

What will each of these return?

```
abs(vec)
# 2 1 0
```

```
abs(lst)
# Error in abs(lst) :
# non-numeric argument to mathematical function
```

Main difficulties with lists

1. **Viewing** contents
2. **Extracting** contents
3. **Using** with functions

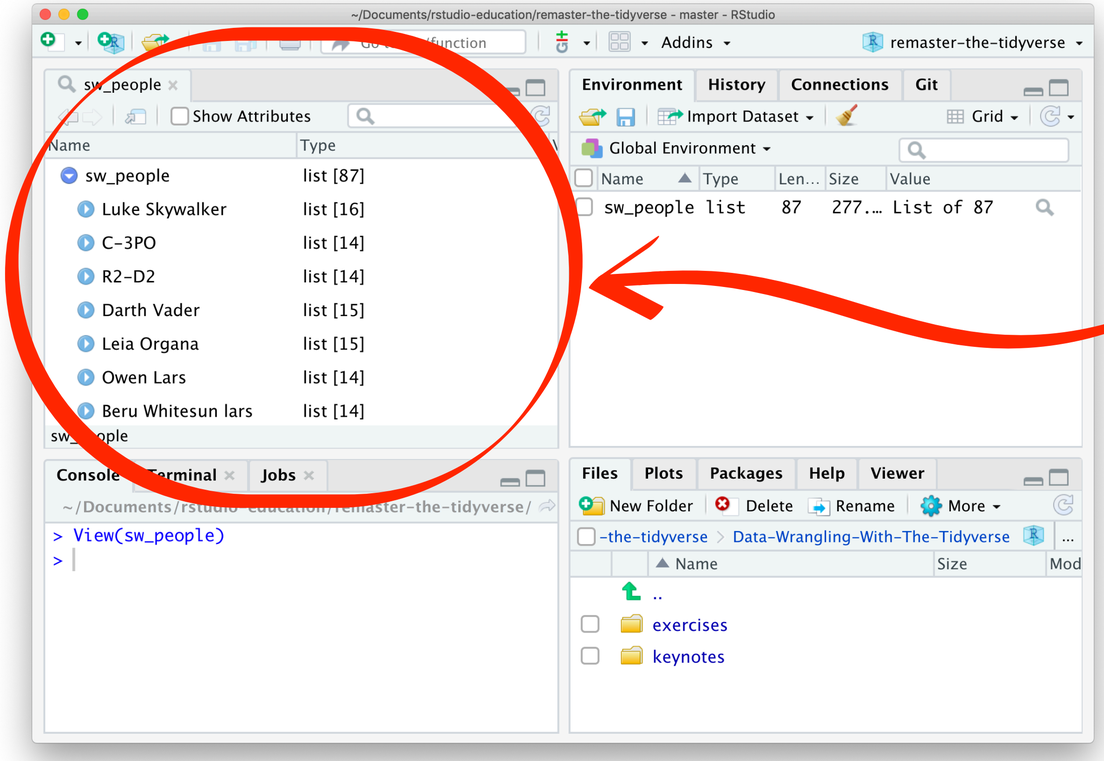
Viewing List Contents

View()

Opens RStudio IDE's interactive list viewer.

```
View(sw_people)
```

CAPITAL "V"



OPENS HERE

Your Turn 2

Who was taller Anakin Skywalker or Darth Vader?

Use the RStudio Viewer to find the answer (in cm).

188 cm

Anakin

Darth ✓

202 cm

Extracting List Contents

Your Turn 3

Here is a list:

```
a_list <- list(num = c(8, 9),
               log = TRUE,
               cha = c("a", "b", "c"))
```

Here are two subsetting commands. Do they return the same values? Run the code chunks to confirm

```
a_list["num"]
```

```
a_list[["num"]]
```

```
a_list["num"]
```

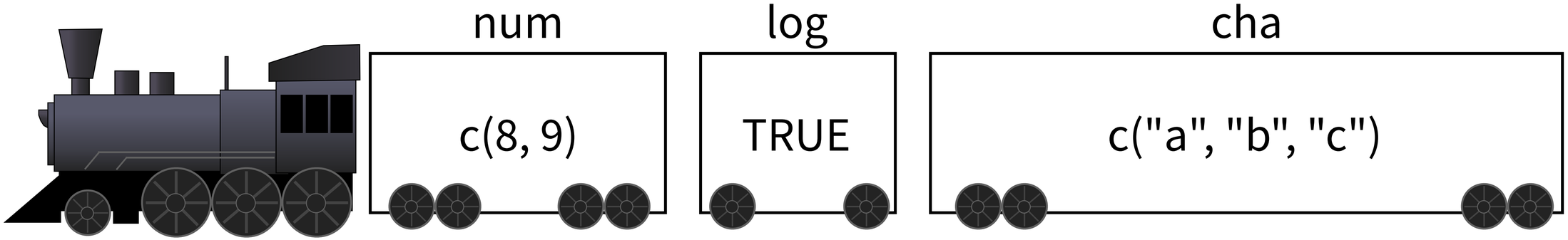
```
$num
[1] 8 9
```

A list
(with one element named num that contains an atomic vector)

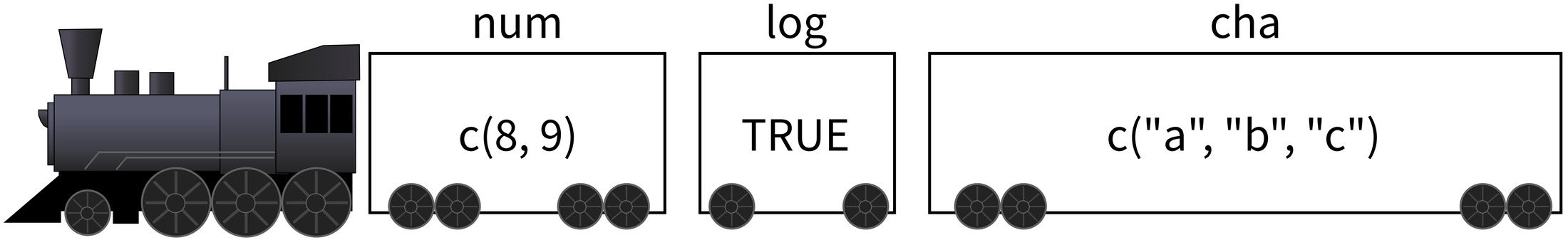
```
a_list[["num"]]
```

```
[1] 8 9
```

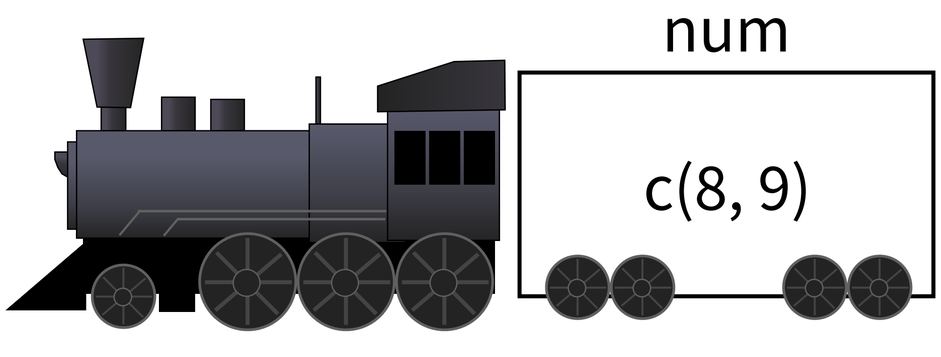
An atomic vector

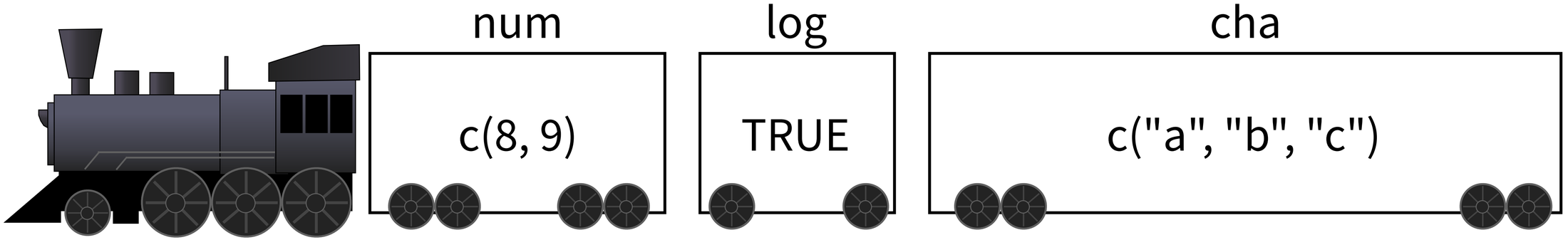



```
a_list <- list(num = c(8,9), log = TRUE, cha = c("a", "b", "c"))
```

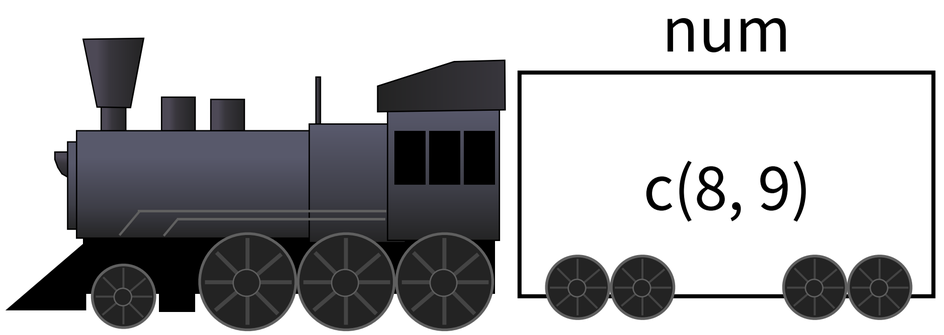



```
a_list["num"]
```



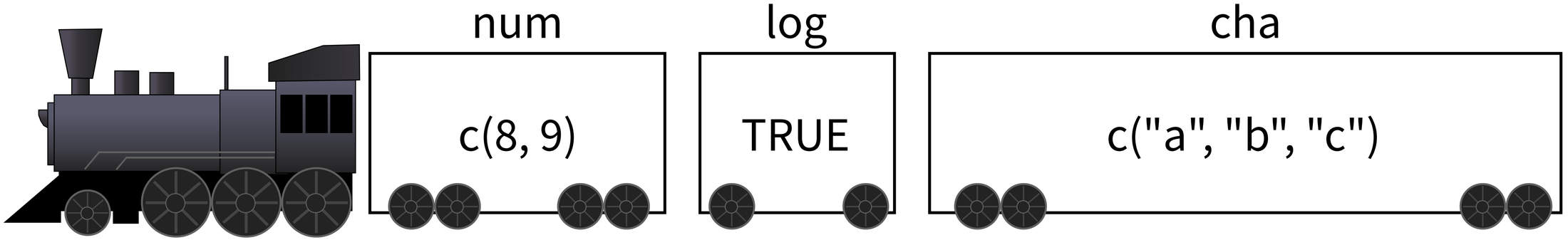


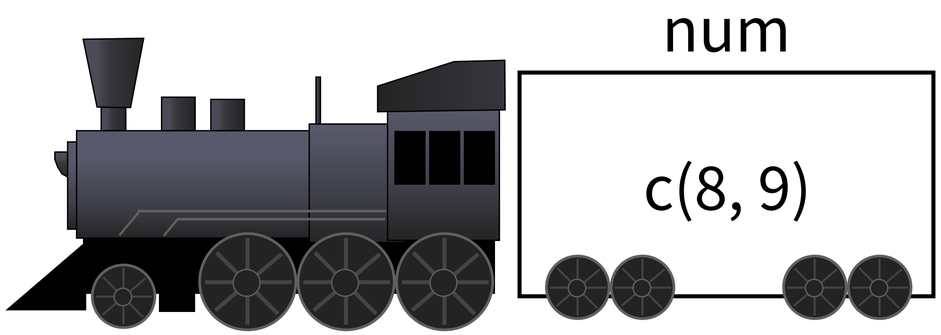

```
a_list["num"]
```




```
a_list[["num"]]
```

c(8, 9)






```
a_list["num"]
```

```
a_list[["num"]]
```

c(8, 9)

```
a_list$num
```

c(8, 9)

x

x

x[1]

x

x[1]

x[[1]]

`x` `x[1]` `x[[1]]` `x[[1]][[1]]`

Name	Type	Value
Leia Organa	list [15]	List of length 15
Owen Lars	list [14]	List of length 14
Beru Whitesun lars	list [14]	List of length 14
R5-D4	list [14]	List of length 14
Biggs Darklighter	list [15]	List of length 15
Obi-Wan Kenobi	list [16]	List of length 16
Anakin Skywalker	list [16]	List of length 16
name	character [1]	'Anakin Skywalker'
height	character [1]	'188'
mass	character [1]	'84'
hair_color	character [1]	'blond'
skin_color	character [1]	'fair'

sw_people

Sends the code to extract the item to the console

```
sw_people[["Anakin Skywalker"]][["height"]]
## 188
```

pluck()

Extracts an element from a list, equivalent of [[ ]]

```
sw_people %>% pluck(11, height)
```

A list

Number or name of element to extract from list

Number or name of element to extract from that element (and so on)

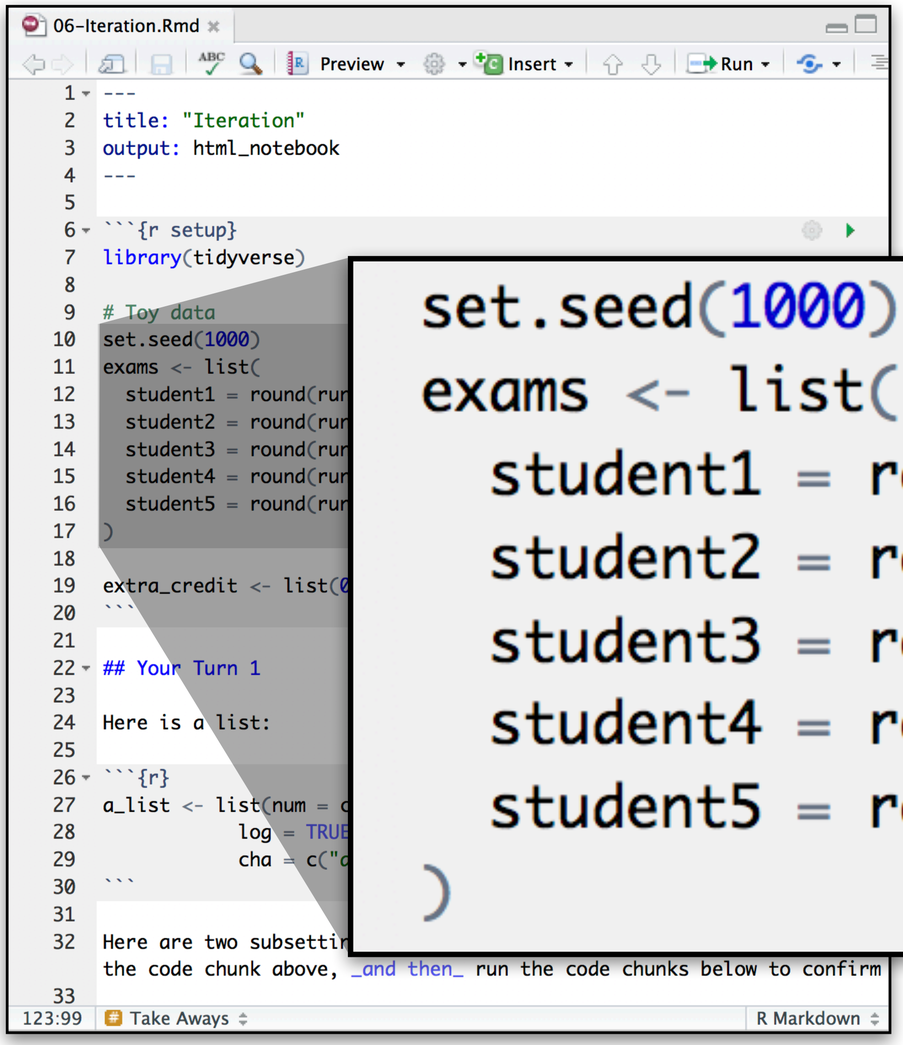
Main difficulties with lists

1. **Viewing** contents
2. **Extracting** contents
3. **Using** with functions

Mapping

Toy data

Suppose we have the exam scores of five students…



Ensures that you and I generate the same "random" values

```
set.seed(1000)
exams <- list(
  student1 = round(runif(10, 50, 100)),
  student2 = round(runif(10, 50, 100)),
  student3 = round(runif(10, 50, 100)),
  student4 = round(runif(10, 50, 100)),
  student5 = round(runif(10, 50, 100))
)
```

Suppose we have the exam scores of five students…

```
exams
```

```
$student1
 [1] 66 88 56 85 76 53 87 79 61 63

$student2
 [1] 67 88 66 93 88 54 75 82 54 79

$student3
 [1] 58 90 64 54 77 84 73 91 55 56

$student4
 [1] 78 52 78 98 75 85 51 89 79 66

$student5
 [1] 100  77  55  82  90  86  85  78  63  75
```

How can we compute the mean grade for each student?

We want the **mean grade** for each student…

```
exams
```

```
$student1
 [1] 66 88 56 85 76 53 87 79 61 63

$student2
 [1] 67 88 66 93 88 54 75 82 54 79

$student3
 [1] 58 90 64 54 77 84 73 91 55 56

$student4
 [1] 78 52 78 98 75 85 51 89 79 66

$student5
 [1] 100  77  55  82  90  86  85  78  63  75
```

```
$student1
[1] 71.4

$student2
[1] 74.6

$student3
[1] 70.2

$student4
[1] 75.1

$student5
[1] 79.1
```

How could we compute the mean grade?

```
mean(exams)
```

```
argument is not numeric or logical: returning NA[1] NA
```

How could we compute the average grade?

```
mean(exams$student1)
mean(exams$student2)
mean(exams$student3)
mean(exams$student4)
mean(exams$student5)
```

```
[1] 71.4
[1] 74.6
[1] 70.2
[1] 75.1
[1] 79.1
```

How could we compute the average grade?

```
list(student1 = mean(exams$student1),
     student2 = mean(exams$student2),
     student3 = mean(exams$student3),
     student4 = mean(exams$student4),
     student5 = mean(exams$student5))
```

```
$student1
[1] 71.4

$student2
[1] 74.6

$student3
[1] 70.2

$student4
[1] 75.1

$student5
[1] 79.1
```

Is there a better way?

purrr

purrr

Functions for working with lists.

```
# install.packages("tidyverse")
library(tidyverse)
```

Your Turn 4

Run the code in the chunk. What does it do?

```
exams %>% map(mean)
```

```
exams %>% map(mean)
```

```
$student1
[1] 71.4

$student2
[1] 74.6

$student3
[1] 70.2

$student4
[1] 75.1

$student5
[1] 79.1
```

map()

Applies a function to every element of a list.
Returns the results as a list.

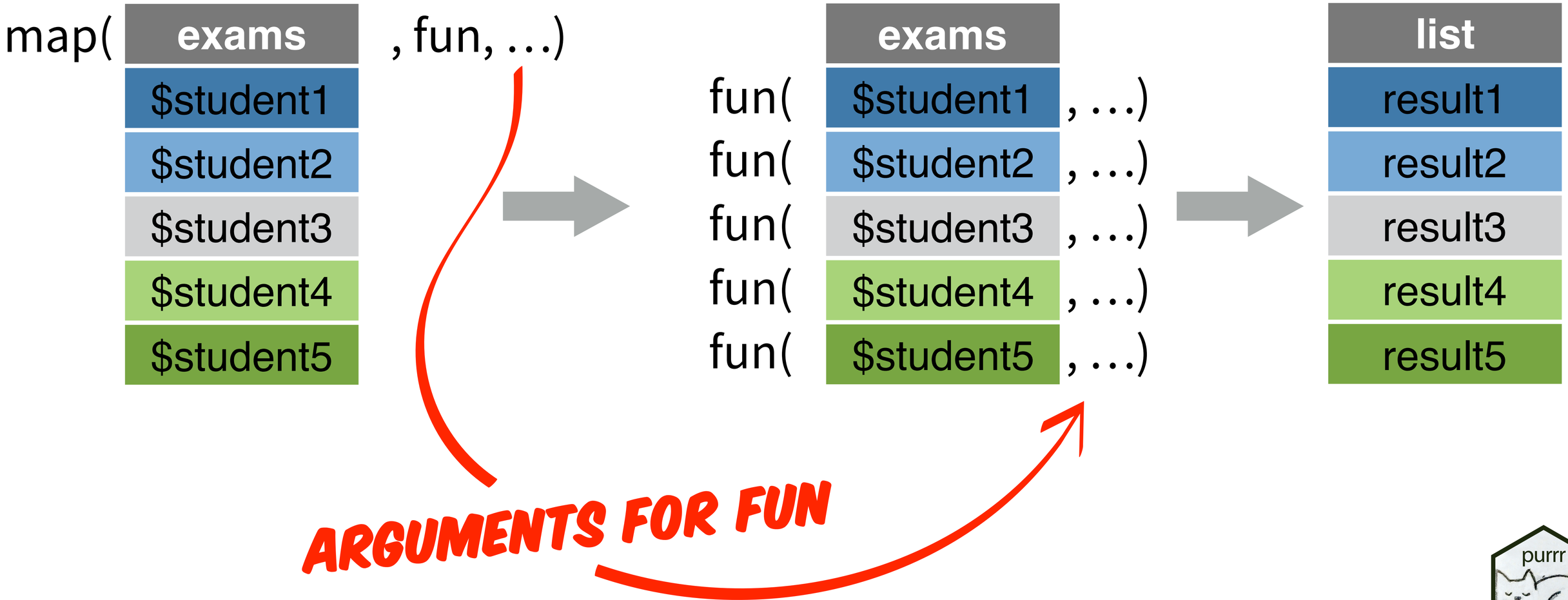
```
map(.x, .f, …)
```

A list

A function to apply to each element of the list (element become first argument)

Other arguments to pass to the function

map()


map(
exams
$student1
$student2
$student3
$student4
$student5
, fun, …)
exams
fun($student1 , …)
fun($student2 , …)
fun($student3 , …)
fun($student4 , …)
fun($student5 , …)
list
result1
result2
result3
result4
result5
ARGUMENTS FOR FUN

```r
x <- c("hw1","hw2","hw3","hw4","hw5","hw6","hw7","hw8","hw9","hw10")
set_names(exams$student1, nm = x)
```

```
hw1  hw2  hw3  hw4  hw5  hw6  hw7  hw8  hw9 hw10
 66   88   56   85   76   53   87   79   61   63
```

```r
x <- c("hw1","hw2","hw3","hw4","hw5","hw6","hw7","hw8","hw9","hw10")
set_names(exams$student1, nm = x)
```

```r
exams %>% map(set_names, nm = x)
```

```
$student1
 hw1  hw2  hw3  hw4  hw5  hw6  hw7  hw8  hw9 hw10
  66   88   56   85   76   53   87   79   61   63

$student2
 hw1  hw2  hw3  hw4  hw5  hw6  hw7  hw8  hw9 hw10
  67   88   66   93   88   54   75   82   54   79

$student3
 hw1  hw2  hw3  hw4  hw5  hw6  hw7  hw8  hw9 hw10
  58   90   64   54   77   84   73   91   55   56
```

```r
x <- c("hw1","hw2","hw3","hw4","hw5","hw6","hw7","hw8","hw9","hw10")
set_names(exams$student1, nm = x)
```

```r
exams %>%
  map(set_names, nm = x) %>%
  map(bind_rows)
```

```
$student1
# A tibble: 1 x 10
    hw1   hw2   hw3   hw4   hw5   hw6   hw7   hw8   hw9  hw10
1    66    88    56    85    76    53    87    79    61    63

$student2
# A tibble: 1 x 10
    hw1   hw2   hw3   hw4   hw5   hw6   hw7   hw8   hw9  hw10
1    67    88    66    93    88    54    75    82    54    79
```

map family

function	returns results as
map()	list
map_chr()	character vector
map_dbl()	double vector (numeric)
map_int()	integer vector
map_lgl()	logical vector
map_dfc()	data frame (column-wise)
map_dfr()	dataframe (row-wise)

```
exams %>%
  map(set_names, nm = x) %>%
  map(bind_rows)
```

```
$student1
# A tibble: 1 x 10
    hw1   hw2   hw3   hw4   hw5   hw6   hw7   hw8   hw9  hw10
1    66    88    56    85    76    53    87    79    61    63

$student2
# A tibble: 1 x 10
    hw1   hw2   hw3   hw4   hw5   hw6   hw7   hw8   hw9  hw10
1    67    88    66    93    88    54    75    82    54    79


$student3
# A tibble: 1 x 10
```

```
exams %>%
  map(set_names, nm = x) %>%
  map_dfr(bind_rows)
```

```
# A tibble: 5 x 10
    hw1   hw2   hw3   hw4   hw5   hw6   hw7   hw8   hw9  hw10
  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1    66    88    56    85    76    53    87    79    61    63
2    67    88    66    93    88    54    75    82    54    79
3    58    90    64    54    77    84    73    91    55    56
4    78    52    78    98    75    85    51    89    79    66
5   100    77    55    82    90    86    85    78    63    75
```

```
exams %>%
  map(set_names, nm = x) %>%
  map_dfc(bind_rows)
```

```
# A tibble: 1 x 50
    hw1   hw2   hw3   hw4   hw5   hw6   hw7   hw8   hw9  hw10  hw11
  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1    66    88    56    85    76    53    87    79    61    63    67
# … with 39 more variables: hw21 <dbl>, hw31 <dbl>, hw41 <dbl>,
#   hw51 <dbl>, hw61 <dbl>, hw71 <dbl>, hw81 <dbl>, hw91 <dbl>,
#   hw101 <dbl>, hw12 <dbl>, hw22 <dbl>, hw32 <dbl>, hw42 <dbl>,
#   hw52 <dbl>, hw62 <dbl>, hw72 <dbl>, hw82 <dbl>, hw92 <dbl>,
#   hw102 <dbl>, hw13 <dbl>, hw23 <dbl>, hw33 <dbl>, hw43 <dbl>,
#   hw53 <dbl>, hw63 <dbl>, hw73 <dbl>, hw83 <dbl>, hw93 <dbl>,
```

```
exams %>% map(mean)
```

```
$student1
[1] 71.4

$student2
[1] 74.6

$student3
[1] 70.2

$student4
[1] 75.1

$student5
[1] 79.1
```


```
exams %>% map_dbl(mean)
```

```
student1 student2 student3 student4 student5
    71.4     74.6     70.2     75.1     79.1
```

Quiz

What if what we want to do is not a function? e.g.

length(exams$student1) == **10**

map shorthand

~ + .

To map a generic R expression…

```
exams %>% map(~length(.) == 10)
```

Begin the expression with ~

Use a . to indicate where the inputs should go

Your Turn 5

Complete the code to apply the test below to every element of the list. Return the results as a vector.

length(<input>) **== 10**

```
exams %>% map_lgl(~length(.) == 10)
```

```
student1 student2 student3 student4 student5
    TRUE     TRUE     TRUE     TRUE     TRUE
```

Your Turn 6

Recompute the grades by dropping the lowest score and then taking the mean of the rest, e.g.

(sum(<input>) **- min**(<input>)) **/ 9**

```
exams %>% map_dbl(~(sum(.) - min(.)) / 9)
```

```
student1 student2 student3 student4 student5
73.44444 76.88889 72.00000 77.77778 81.77778
```

Quiz

What does this return?

```
add_1 <- function(x) x + 1

add_1(1)
```

Quiz

What does this return?

```
add_1 <- function(x) x + 1

add_1(1)
```

```
# 2
```

Quiz

What does this return?

```
add_2 <- function(x, y) x + y

add_2(2, 3)
```

Quiz

What does this return?

```
add_2 <- function(x, y) x + y

add_2(2, 3)
```

```
# 5
```

If functions can take two arguments, how can you pass two lists as the arguments?

map2()

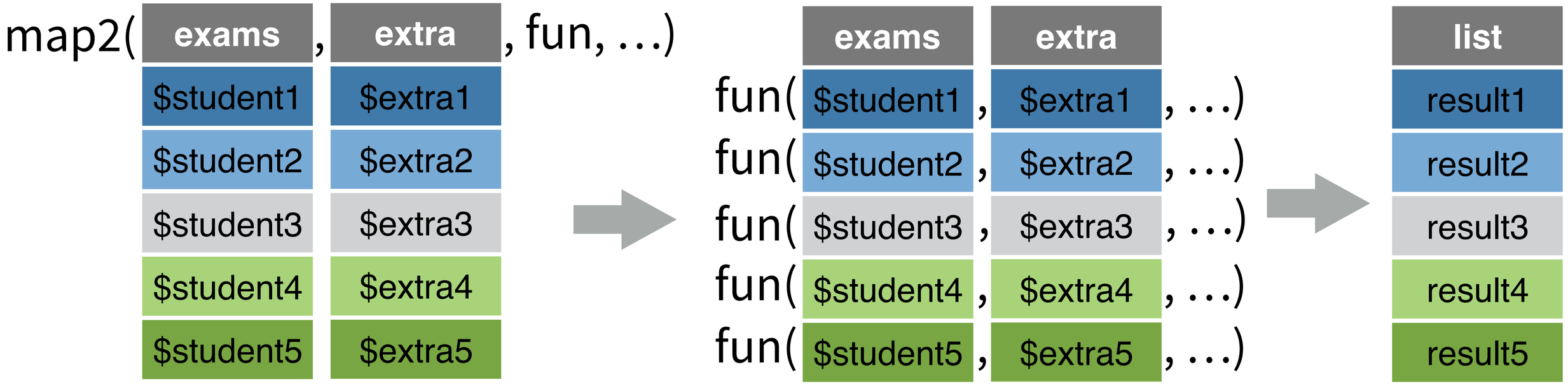
Applies a function to every element of two lists.
Returns the results as a list.

```
map2(.x, .y, .f, …)
```

A list of elements to pass to the first argument of .f

A list of elements to pass to the second argument of .f

map2()


map2(
exams
,
extra
, fun, ...)
$student1
$extra1
$student2
$extra2
$student3
$extra3
$student4
$extra4
$student5
$extra5
exams
extra
fun($student1 , $extra1 , ...)
fun($student2 , $extra2 , ...)
fun($student3 , $extra3 , ...)
fun($student4 , $extra4 , ...)
fun($student5 , $extra5 , ...)
list
result1
result2
result3
result4
result5


purrr

map family

single list	two lists	returns results as
map()	map2()	list
map_chr()	map2_chr()	character vector
map_dbl()	map2_dbl()	double vector (numeric)
map_int()	map2_int()	integer vector
map_lgl()	map2_lgl()	logical vector
map_dfc()	map2_dfc()	data frame (columnwise)
map_dfr()	map2_dfr()	data frame (rowwise)

~ + .x + .y

To map a generic R expression…

```
exams %>% map2(extra_credit, ~.x + .y)
```

Use a .x to indicate the first input

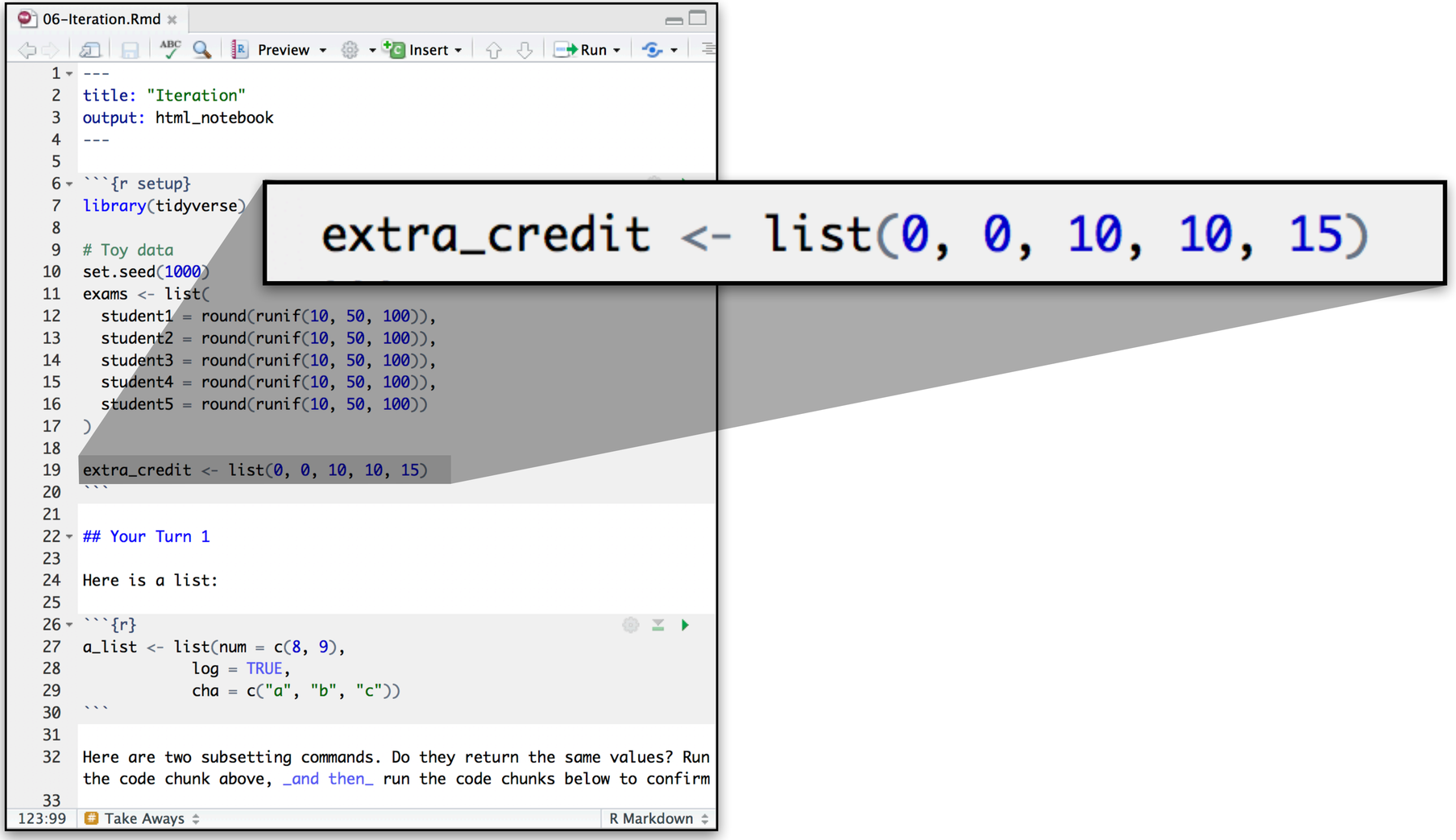
Second list

Begin the expression with ~

Use a .y to indicate the second

Toy data

Suppose we have extra credit for the five students…



Your Turn 7

Compute a final grade for each student, where the final grade is the average test score plus any extra credit assigned to the student. Return the results as a double (i.e. numeric) vector.

mean(<exams>) + <extra_credit>

The grades with extra credit…

```
exams %>%
  map2_dbl(extra_credit, ~mean(.x) + .y)
```

```
student1  student2  student3  student4  student5
    71.4      74.6      80.2      85.1      94.1
```

Other mapping functions

THESE WORK FOR ALL VECTORS

map and walk functions

single list	two lists	n lists	returns results as
map()	map2()	pmap()	list
map_chr()	map2_chr()	pmap_chr()	character vector
map_dbl()	map2_dbl()	pmap_dbl()	double vector
map_int()	map2_int()	pmap_int()	integer vector
map_lgl()	map2_lgl()	pmap_lgl()	logical vector
map_df()	map2_df()	pmap_df()	data frame
walk()	walk2()	pwalk()	side effect

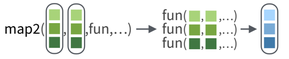
purrr

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)*

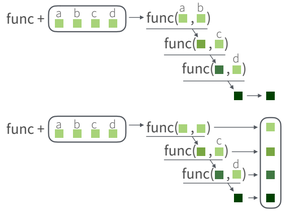
WORK WITH LISTS

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).

List Columns

Quiz

How is a data frame/tibble similar to a list?

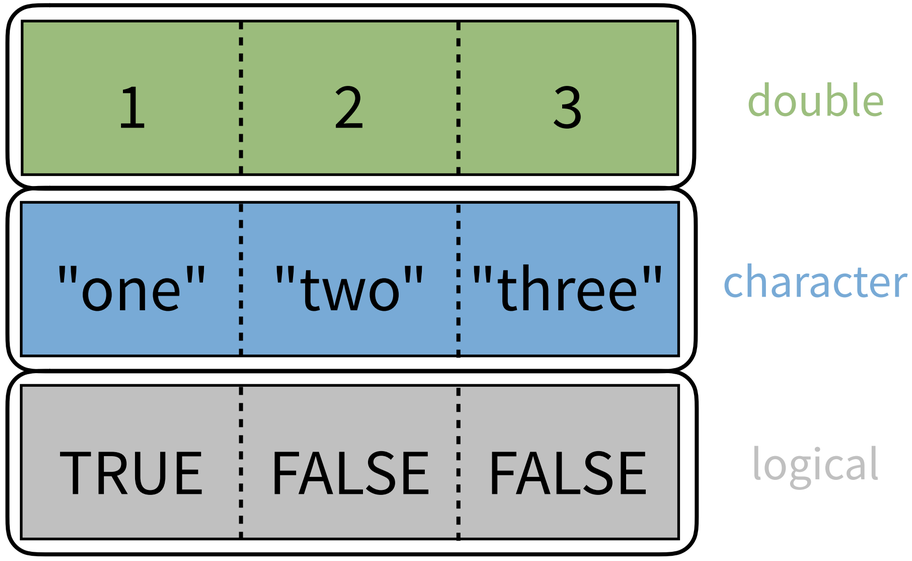
A data frame/tibble is a list!

data frame

num	cha	log
1	"one"	TRUE
2	"two"	FALSE
3	"three"	FALSE

=

List



\+ class = "data.frame"

A data frame/tibble is a list!

data frame

num	cha	log
1	"one"	TRUE
2	"two"	FALSE
3	"three"	FALSE

`df["num"]`

num
1
2
3

`df[["num"]]`

`df$num`

`c(1, 2, 3)`

A data frame/tibble is a list!

data frame

num	cha	log
1	"one"	TRUE
2	"two"	FALSE
3	"three"	FALSE

`df %>% select(num)`

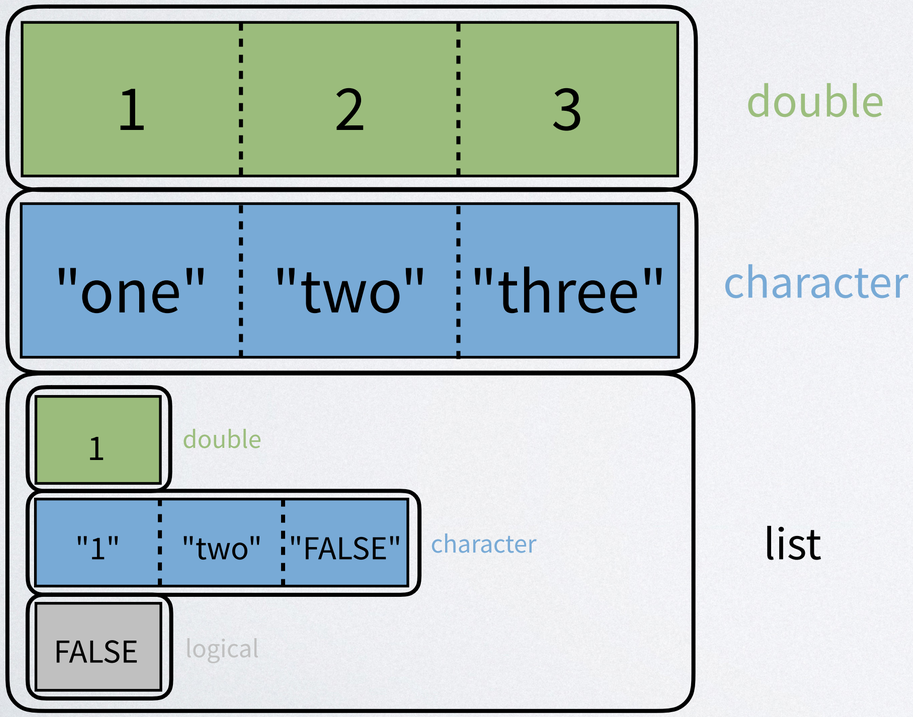
num
1
2
3

Quiz

If one of the elements of a list can be another list,

can one of the columns of a data frame be another list?

List



?
=

data frame

num	cha	listcol
1	"one"	1
2	"two"	c("1", "two", "FALSE")
3	"three"	FALSE

Yes.

```
df <- tibble(
  num = c(1, 2, 3),
  cha = c("one", "two", "three"),
  listcol = list(1, c("1", "two", "FALSE"), FALSE)
)
```

num <dbl>	cha <chr>	listcol <list>
1	one	<dbl [1]>
2	two	<chr [3]>
3	three	<lgl [1]>

3 rows

Yes.

```
df %>%
  mutate(listcol2 = list(c(1,2), c(3,4), "Hi"))
```

num <dbl>	cha <chr>	listcol <list>	listcol2 <list>
1	one	<dbl [1]>	<dbl [2]>
2	two	<chr [3]>	<dbl [2]>
3	three	<lgl [1]>	<chr [1]>

3 rows

```
[1] 1 2
```

Putting it all together

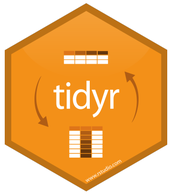
who

Tuberculosis (TB) cases broken down by year, country, age, gender, and diagnosis method from the *2014 World Health Organization Global Tuberculosis Report*

```
View(who)
```

```
tb <-
  who %>%
  gather("codes", "n", 5:60) %>%
  separate(codes, c("new", "type", "sexage"), sep = "_") %>%
  select(-new, -iso2, -iso3) %>%
  separate(sexage, c("sex", "age"), sep = 1) %>%
  drop_na(n)
```

country	year	type	sex	age	n
Afghanistan	1997	sp	m	014	0
Afghanistan	1998	sp	m	014	30
Afghanistan	1999	sp	m	014	8
Afghanistan	2000	sp	m	014	52
Afghanistan	2001	sp	m	014	129
Afghanistan	2002	sp	m	014	90



Can we predict the number of cases for each country in 2020?

First, try for one country

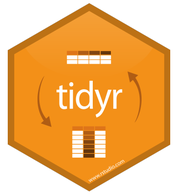
nest()

Nest rows into a list column by group.

```
nest(data,.key = "data")
```

A grouped data frame

name for the new list column



nest()

Nest rows into a list column by group.

```
iris
```

Species	S.L	S.W	P.L	P.W
setosa	5.1	3.5	1.4	0.2
setosa	4.9	3.0	1.4	0.2
setosa	4.7	3.2	1.3	0.2
setosa	4.6	3.1	1.5	0.2
setosa	5.0	3.6	1.4	0.2
versi	7.0	3.2	4.7	1.4
versi	6.4	3.2	4.5	1.5
versi	6.9	3.1	4.9	1.5
versi	5.5	2.3	4.0	1.3
versi	6.5	2.8	4.6	1.5
virgini	6.3	3.3	6.0	2.5
virgini	5.8	2.7	5.1	1.9
virgini	7.1	3.0	5.9	2.1
virgini	6.3	2.9	5.6	1.8
virgini	6.5	3.0	5.8	2.2

nest()

Nest rows into a list column by group.

```
iris %>% group_by(Species)
```

Species	S.L	S.W	P.L	P.W
setosa	5.1	3.5	1.4	0.2
setosa	4.9	3.0	1.4	0.2
setosa	4.7	3.2	1.3	0.2
setosa	4.6	3.1	1.5	0.2
setosa	5.0	3.6	1.4	0.2
versi	7.0	3.2	4.7	1.4
versi	6.4	3.2	4.5	1.5
versi	6.9	3.1	4.9	1.5
versi	5.5	2.3	4.0	1.3
versi	6.5	2.8	4.6	1.5
virgini	6.3	3.3	6.0	2.5
virgini	5.8	2.7	5.1	1.9
virgini	7.1	3.0	5.9	2.1
virgini	6.3	2.9	5.6	1.8
virgini	6.5	3.0	5.8	2.2

Species	S.L	S.W	P.L	P.W
setosa	5.1	3.5	1.4	0.2
setosa	4.9	3.0	1.4	0.2
setosa	4.7	3.2	1.3	0.2
setosa	4.6	3.1	1.5	0.2
setosa	5.0	3.6	1.4	0.2
versi	7.0	3.2	4.7	1.4
versi	6.4	3.2	4.5	1.5
versi	6.9	3.1	4.9	1.5
versi	5.5	2.3	4.0	1.3
versi	6.5	2.8	4.6	1.5
virgini	6.3	3.3	6.0	2.5
virgini	5.8	2.7	5.1	1.9
virgini	7.1	3.0	5.9	2.1
virgini	6.3	2.9	5.6	1.8
virgini	6.5	3.0	5.8	2.2

nest()

Nest rows into a list column by group.

```
iris %>% group_by(Species) %>% nest()
```

Species	S.L	S.W	P.L	P.W
setosa	5.1	3.5	1.4	0.2
setosa	4.9	3.0	1.4	0.2
setosa	4.7	3.2	1.3	0.2
setosa	4.6	3.1	1.5	0.2
setosa	5.0	3.6	1.4	0.2
versi	7.0	3.2	4.7	1.4
versi	6.4	3.2	4.5	1.5
versi	6.9	3.1	4.9	1.5
versi	5.5	2.3	4.0	1.3
versi	6.5	2.8	4.6	1.5
virgini	6.3	3.3	6.0	2.5
virgini	5.8	2.7	5.1	1.9
virgini	7.1	3.0	5.9	2.1
virgini	6.3	2.9	5.6	1.8
virgini	6.5	3.0	5.8	2.2

→

Species	S.L	S.W	P.L	P.W
setosa	5.1	3.5	1.4	0.2
setosa	4.9	3.0	1.4	0.2
setosa	4.7	3.2	1.3	0.2
setosa	4.6	3.1	1.5	0.2
setosa	5.0	3.6	1.4	0.2
versi	7.0	3.2	4.7	1.4
versi	6.4	3.2	4.5	1.5
versi	6.9	3.1	4.9	1.5
versi	5.5	2.3	4.0	1.3
versi	6.5	2.8	4.6	1.5
virgini	6.3	3.3	6.0	2.5
virgini	5.8	2.7	5.1	1.9
virgini	7.1	3.0	5.9	2.1
virgini	6.3	2.9	5.6	1.8
virgini	6.5	3.0	5.8	2.2

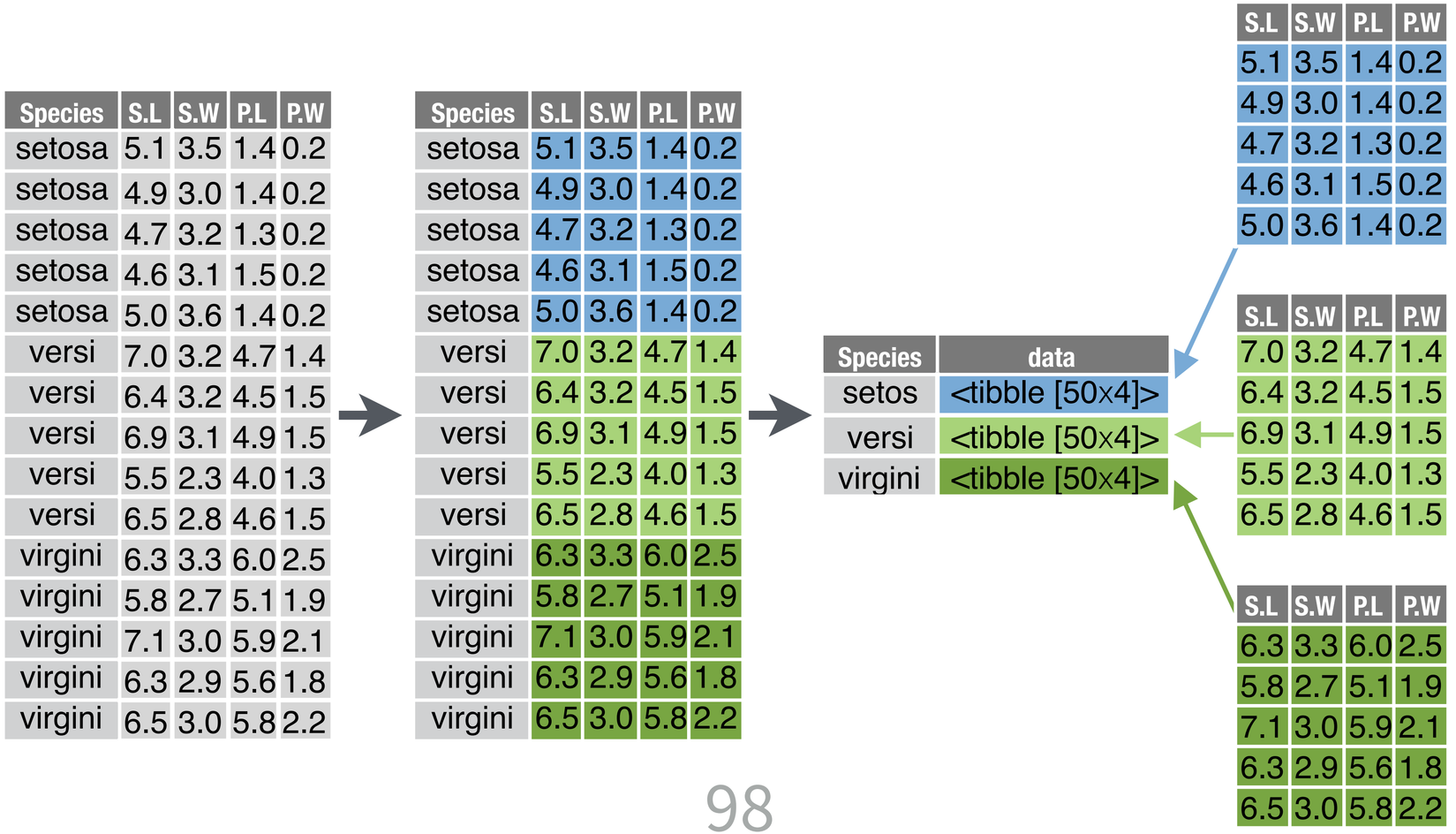
→

Species	data
setos	<tibble [50x4]>
versi	<tibble [50x4]>
virgini	<tibble [50x4]>

nest()

Nest rows into a list column by group.

```
iris %>% group_by(Species) %>% nest()
```



Species	S.L	S.W	P.L	P.W
setosa	5.1	3.5	1.4	0.2
setosa	4.9	3.0	1.4	0.2
setosa	4.7	3.2	1.3	0.2
setosa	4.6	3.1	1.5	0.2
setosa	5.0	3.6	1.4	0.2
versi	7.0	3.2	4.7	1.4
versi	6.4	3.2	4.5	1.5
versi	6.9	3.1	4.9	1.5
versi	5.5	2.3	4.0	1.3
versi	6.5	2.8	4.6	1.5
virgini	6.3	3.3	6.0	2.5
virgini	5.8	2.7	5.1	1.9
virgini	7.1	3.0	5.9	2.1
virgini	6.3	2.9	5.6	1.8
virgini	6.5	3.0	5.8	2.2

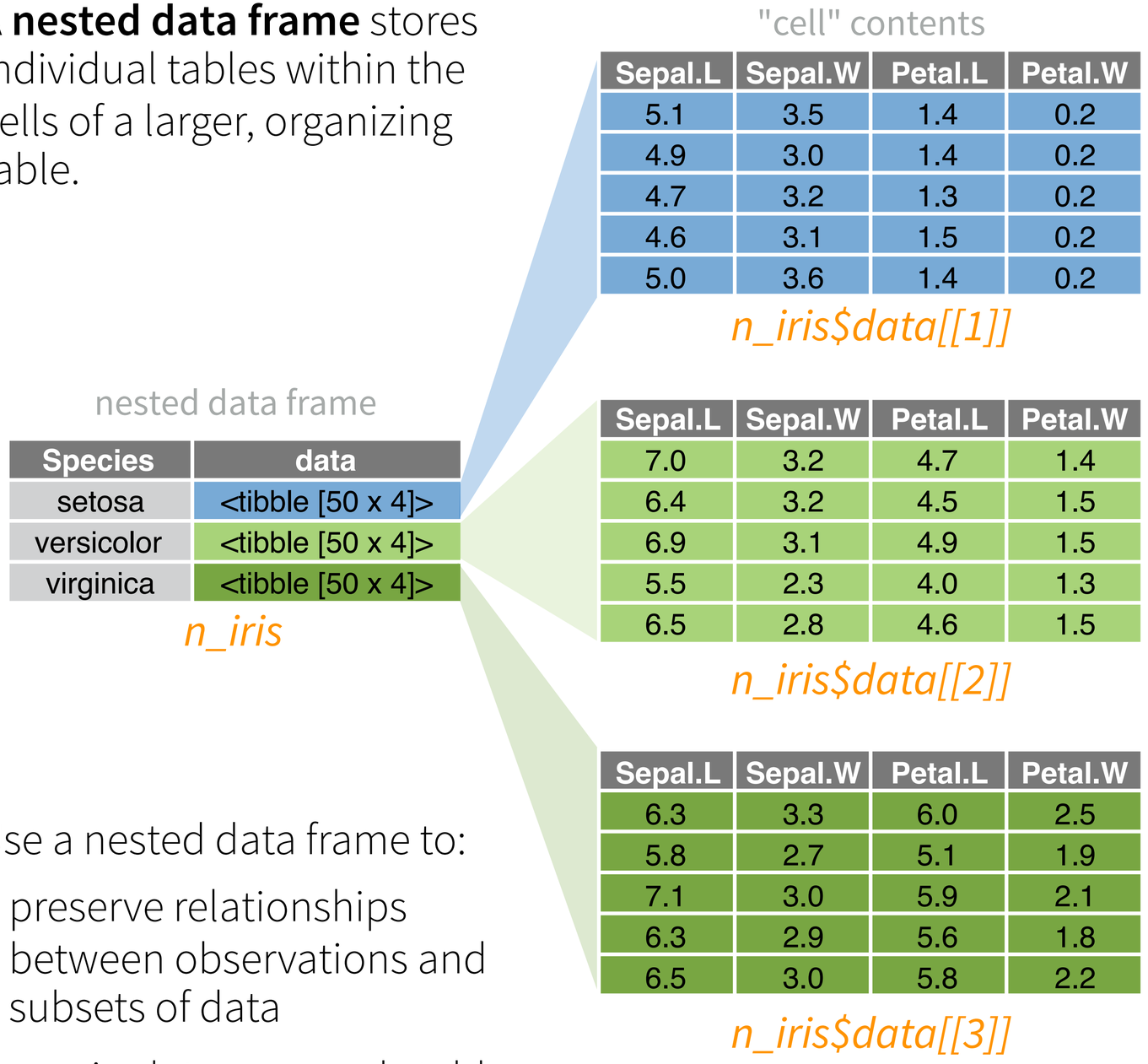
Species	data
setos	<tibble [50x4]>
versi	<tibble [50x4]>
virgini	<tibble [50x4]>

S.L	S.W	P.L	P.W
5.1	3.5	1.4	0.2
4.9	3.0	1.4	0.2
4.7	3.2	1.3	0.2
4.6	3.1	1.5	0.2
5.0	3.6	1.4	0.2

S.L	S.W	P.L	P.W
7.0	3.2	4.7	1.4
6.4	3.2	4.5	1.5
6.9	3.1	4.9	1.5
5.5	2.3	4.0	1.3
6.5	2.8	4.6	1.5

S.L	S.W	P.L	P.W
6.3	3.3	6.0	2.5
5.8	2.7	5.1	1.9
7.1	3.0	5.9	2.1
6.3	2.9	5.6	1.8
6.5	3.0	5.8	2.2

A **nested data frame** stores individual tables within the cells of a larger, organizing table.



Use a nested data frame to:

- preserve relationships between observations and subsets of data
- manipulate many sub-tables at once with the **purrr** functions **map()**, **map2()**, or **pmap()**.

```
tb_predictions <- tb %>%
  group_by(country, year) %>%
  summarise(n = sum(n)) %>%
  nest()
```

country <chr>	data <list>
Afghanistan	<tibble>
Albania	<tibble>
Algeria	<tibble>
American Samoa	<tibble>
Andorra	<tibble>
Angola	<tibble>
Anguilla	<tibble>
Antigua and Barbuda	<tibble>
Argentina	<tibble>
Armenia	<tibble>

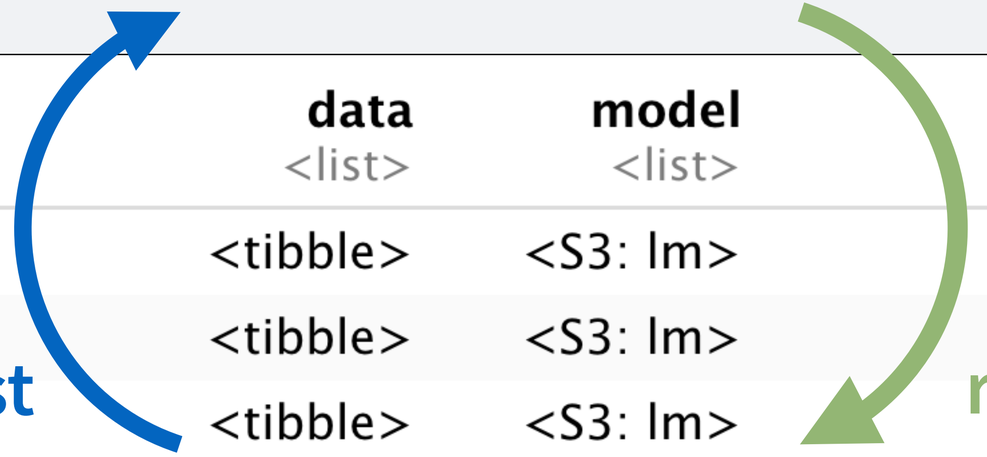
```
tb_predictions$data[[1]]
```

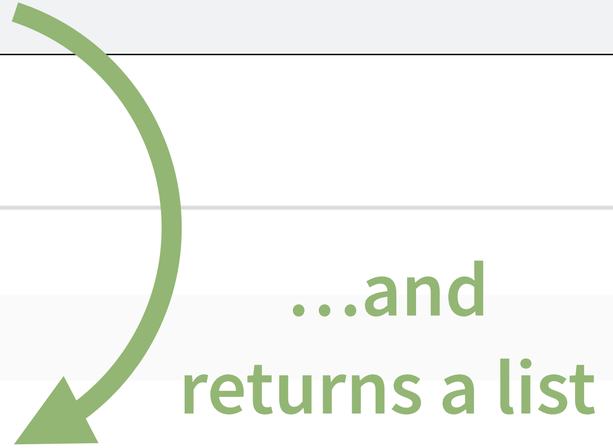
country <chr>	data <list>
Afghanistan	<tibble>
Albania	<tibble>
Algeria	<tibble>
American Samoa	<tibble>
Andorra	<tibble>
Angola	<tibble>
Anguilla	<tibble>
Antigua and Barbuda	<tibble>
Argentina	<tibble>
Armenia	<tibble>

year <int>	n <int>
1997	128
1998	1778
1999	745
2000	2666
2001	4639
2002	6509
2003	6528
2004	8245
2005	9949
2006	12469

1–10 of 17 ro… Previous 1 2 Next

```
tb %>% group_by(country, year) %>%
  summarise(n = sum(n)) %>% nest() %>%
  mutate(model = map(data, ~lm(lifeExp ~ year, data = .)))
```





country <chr>	data <list>	model <list>
Afghanistan	<tibble>	<S3: lm>
Albania	<tibble>	<S3: lm>
Algeria	<tibble>	<S3: lm>
American Samoa	<tibble>	<S3: lm>
Andorra	<tibble>	<S3: lm>
Angola	<tibble>	<S3: lm>
Anguilla	<tibble>	<S3: lm>
Antigua and Barbuda	<tibble>	<S3: lm>
Argentina	<tibble>	<S3: lm>

```
tb_prediction$model[[1]]
```

country <chr>	data <list>	model <list>
Afghanistan	<tibble>	<S3: lm>
Albania	<tibble>	<S3: lm>
Algeria	<tibble>	<S3: lm>
American Samoa	<tibble>	<S3: lm>
Andorra	<tibble>	<S3: lm>
Angola	<tibble>	<S3: lm>
Anguilla	<tibble>	<S3: lm>
Antigua and Barbuda	<tibble>	<S3: lm>
Argentina	<tibble>	<S3: lm>

```
Call:
lm(formula = n ~ year, data = .)

Coefficients:
(Intercept)         year
 -1631615.4         817.9
```

```
tb %>% group_by(country, year) %>%
  summarise(n = sum(n)) %>% nest() %>%
  mutate(model = map(data, ~lm(lifeExp ~ year, data = .)) %>%
  mutate(pred = map_dbl(model, predict, tibble(year = 2020)))
```

map_dbl() takes a list

…and returns a number

country <chr>	data <list>	model <list>	pred <dbl>
Afghanistan	<tibble>	<S3: lm>	20516.838235
Albania	<tibble>	<S3: lm>	592.979876
Algeria	<tibble>	<S3: lm>	7670.311150
American Samoa	<tibble>	<S3: lm>	3.031548
Andorra	<tibble>	<S3: lm>	5.002408
Angola	<tibble>	<S3: lm>	34342.129825
Anguilla	<tibble>	<S3: lm>	-0.240367
Antigua and Barbuda	<tibble>	<S3: lm>	8.014420
Argentina	<tibble>	<S3: lm>	10504.915377

unnest()

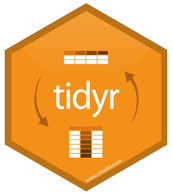
Unnests one or more list columns

```
tb_predictions %>% unnest(data,.drop = FALSE)
```

A nested data frame

list column(s) to unnest (should contain data frames)

Drop remaining list columns from the result?



Take Away

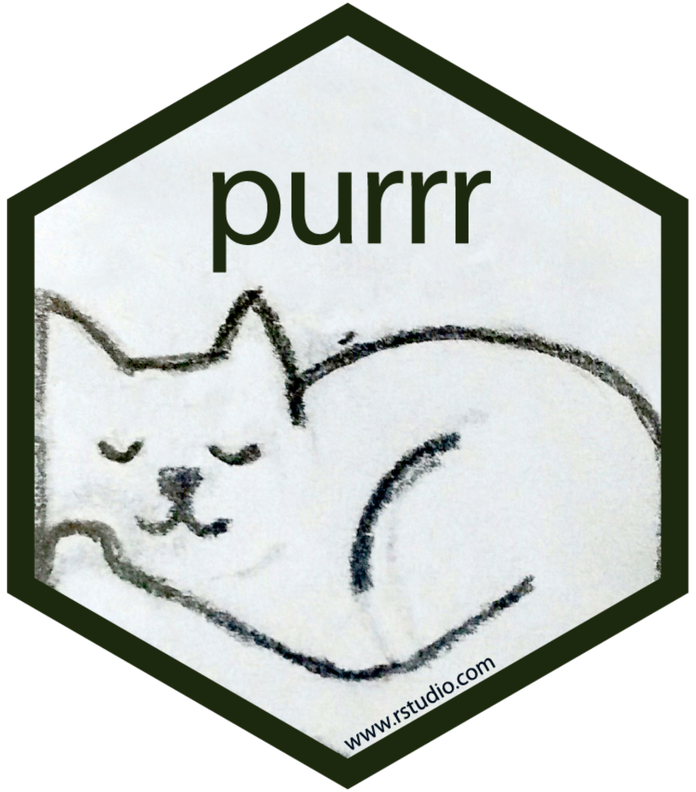
A table is …an organizational structure …that you can manipulate.

country	data	model	pred
Afghanistan	year | n 1997 | 128 1998 | 1778 1999 | 745 2000 | 2666 2001 | 4639 2002 | 6509 2003 | 6528 2004 | 8245 2005 | 9949 2006 | 12469	Call: lm(formula = n ~ year, data = .) Coefficients: (Intercept) year -1631615.4 817.9	20517
Albania	year | .resid 1995 | 139 1997 | 241 1998 | 212 1999 | 168 2000 | 171 2001 | 171 2002 | 225 2003 | 211 2004 | 201 2005 | 196	Call: lm(formula = n ~ year, data = .) Coefficients: (Intercept) year -37921.48 19.07	593

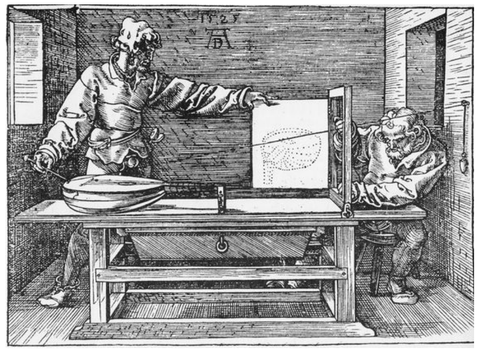
"Better experimental design = simpler statistics.
Better data model = simpler analysis."

- Jenny Bryan (2016)

Lists with



Thank you



Please take the class survey

rstd.io/class-survey