

Lists with



Navigate up to the **07-Lists** folder.
Open on **07-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?

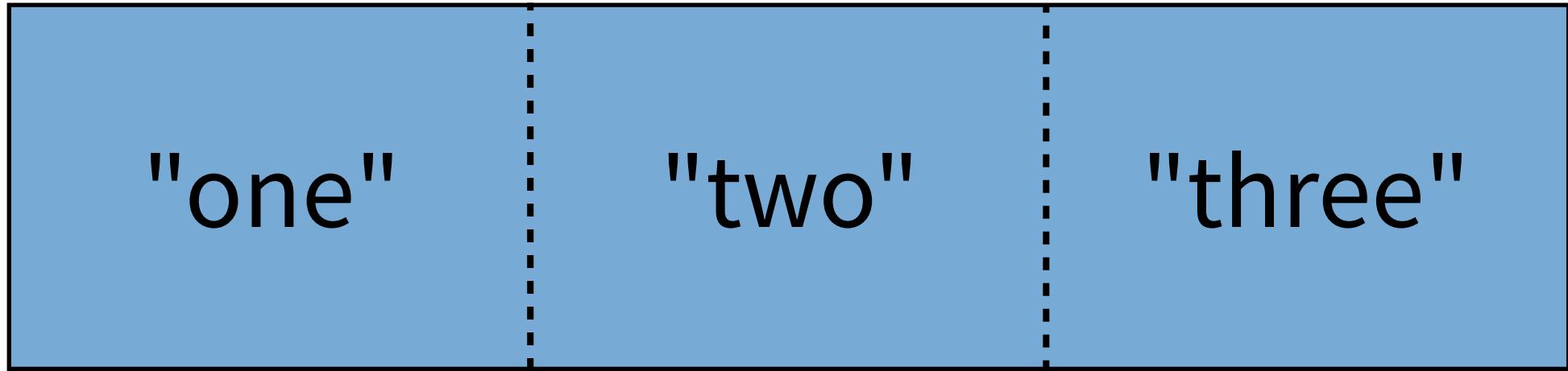
Atomic Vector



type

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

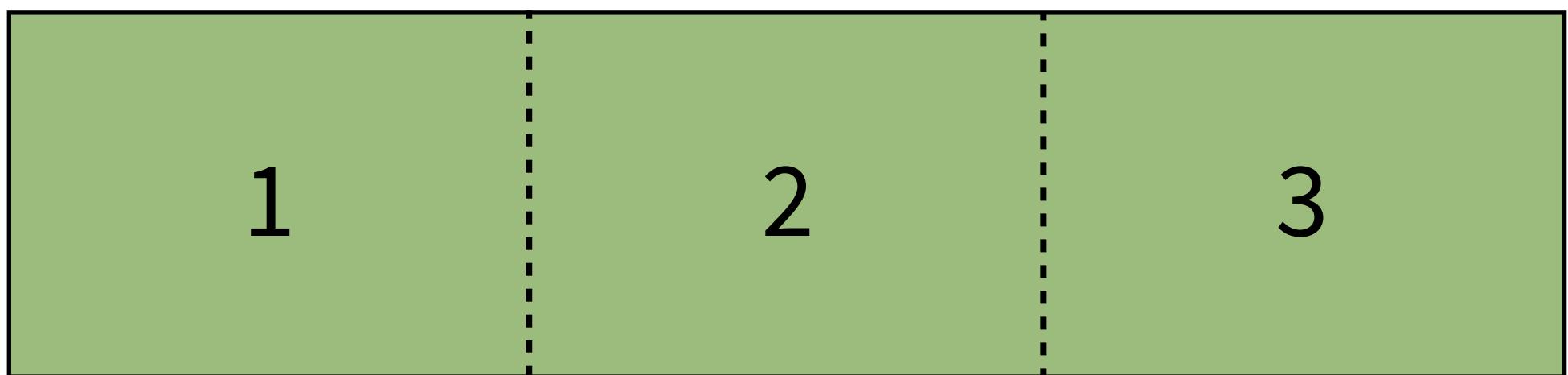
Atomic Vector



character

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

Atomic Vector



double

Atomic Vector

TRUE	FALSE	FALSE
------	-------	-------

logical

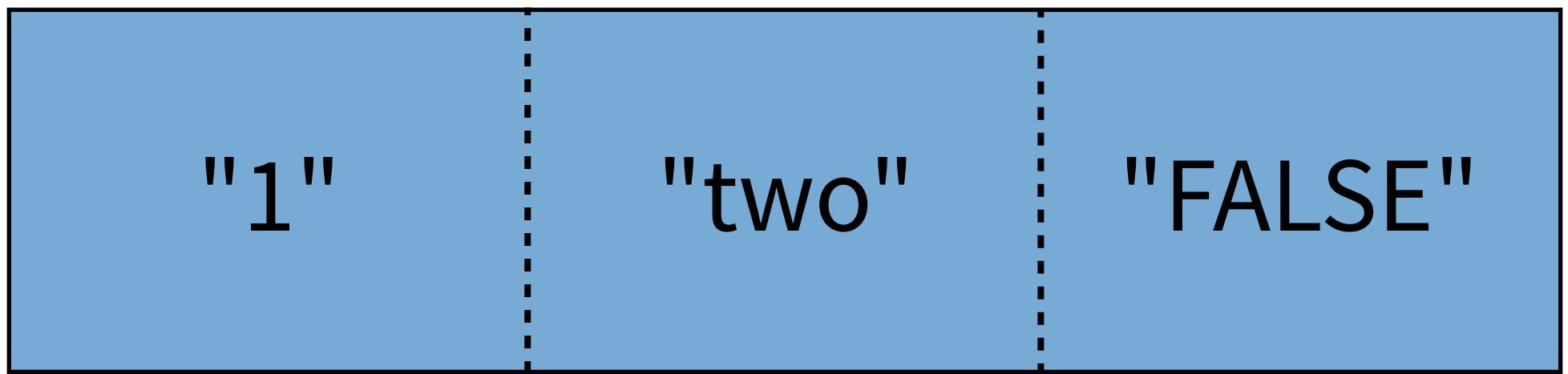
Atomic Vector

1	"two"	FALSE
---	-------	-------

?

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

Atomic Vector



character

Atomic Vector



type

List



type

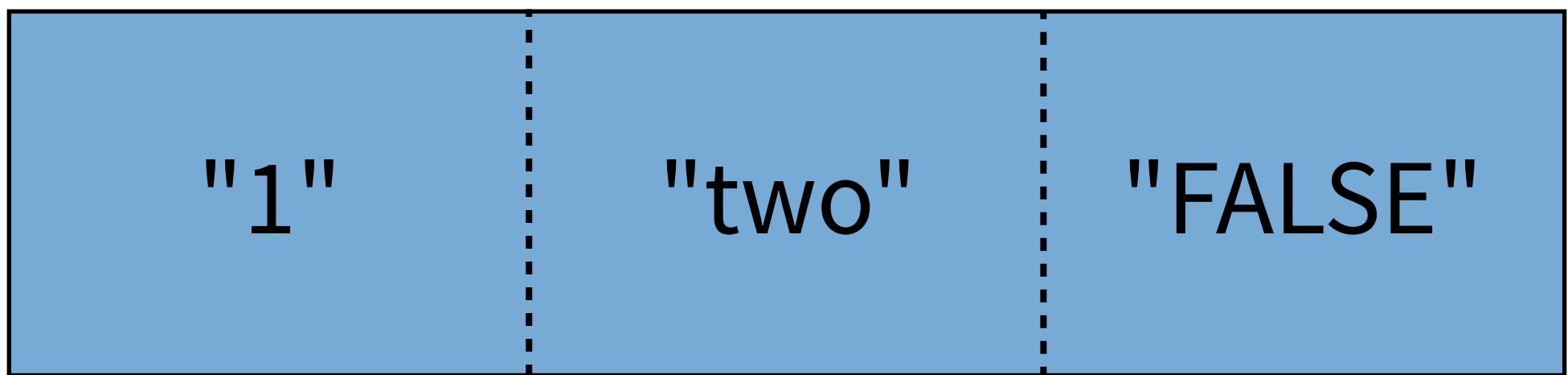


type



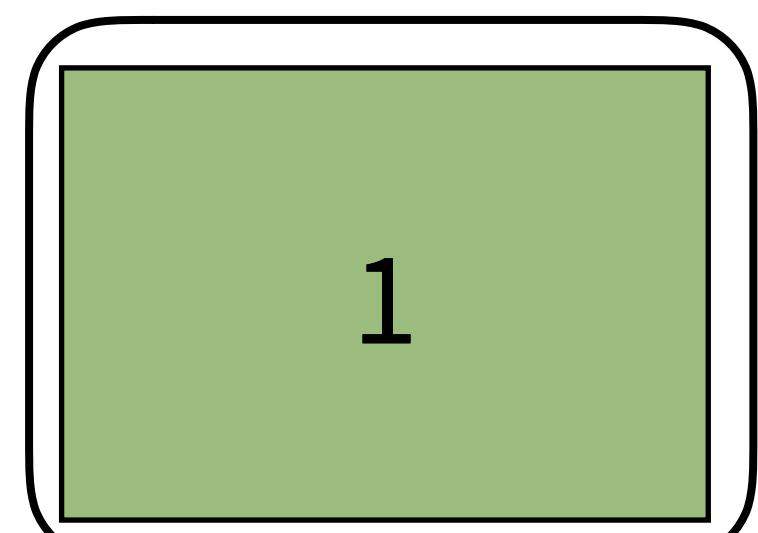
type

Atomic Vector

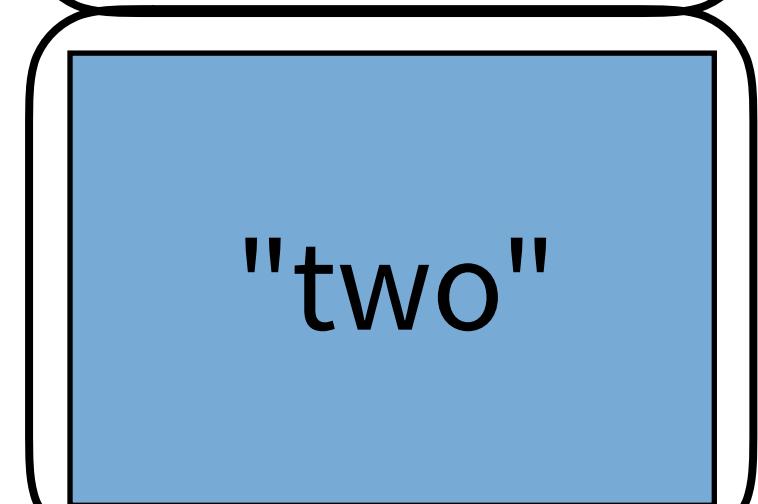


character

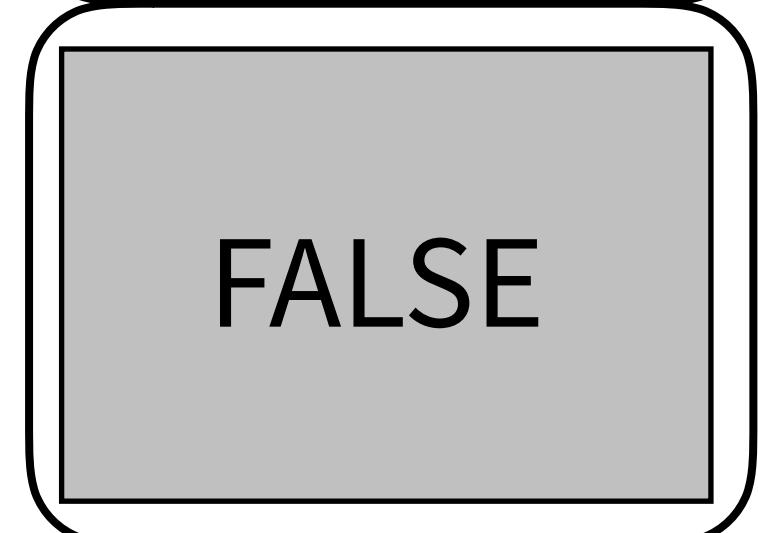
List



double

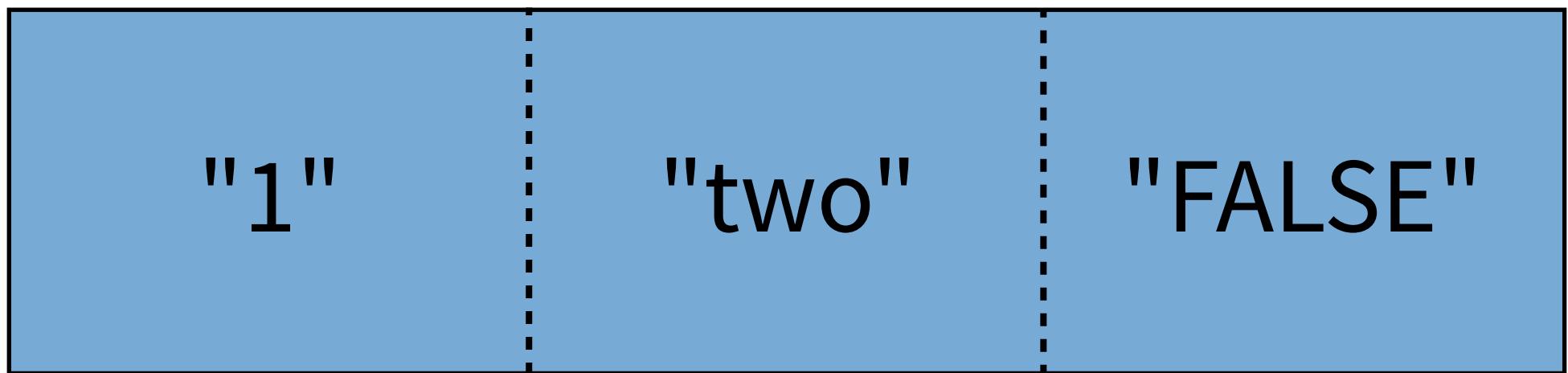


character



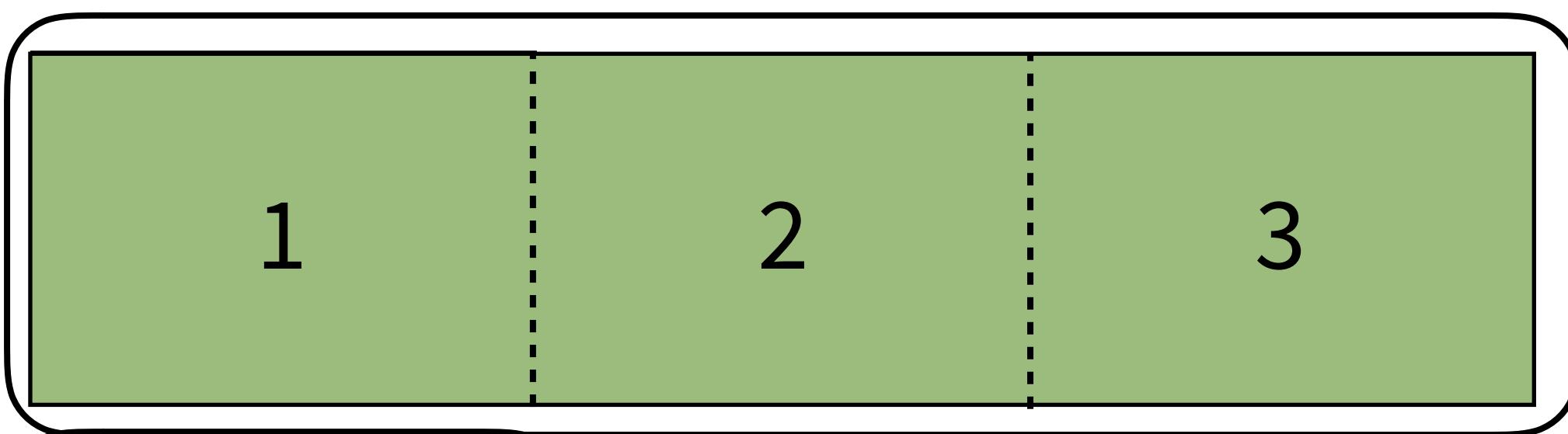
logical

Atomic Vector

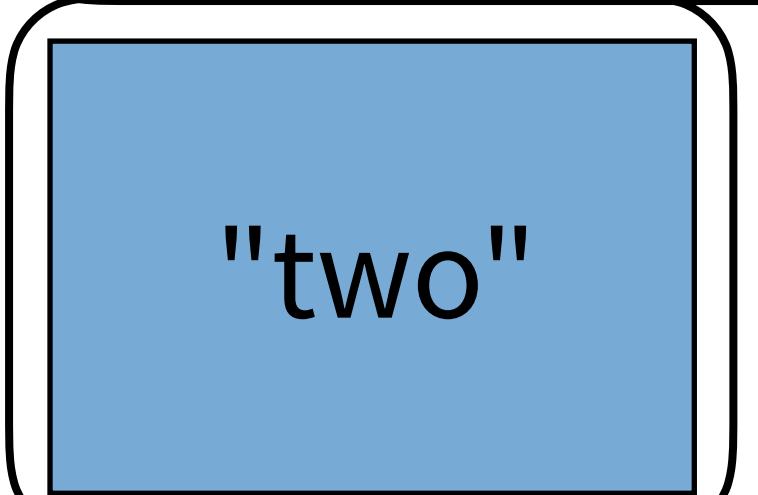


character

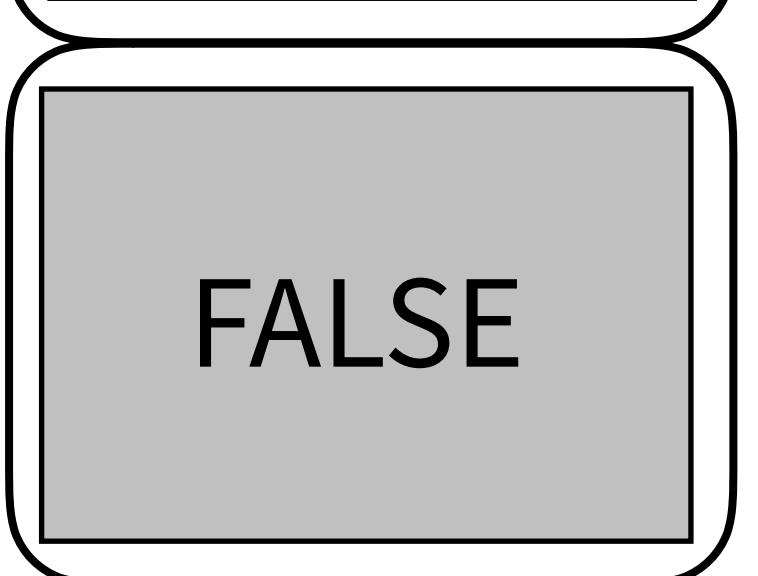
List



double

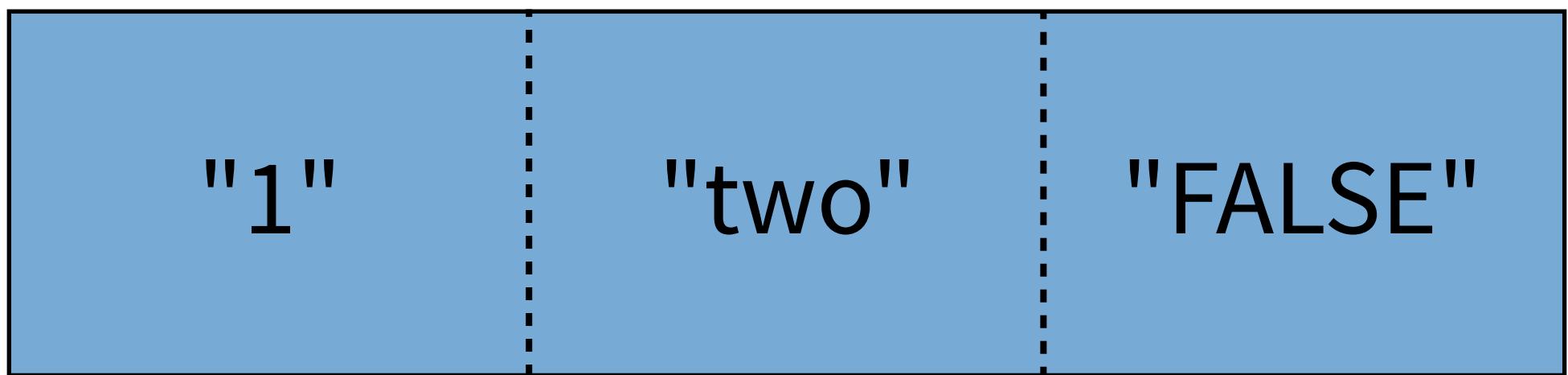


character



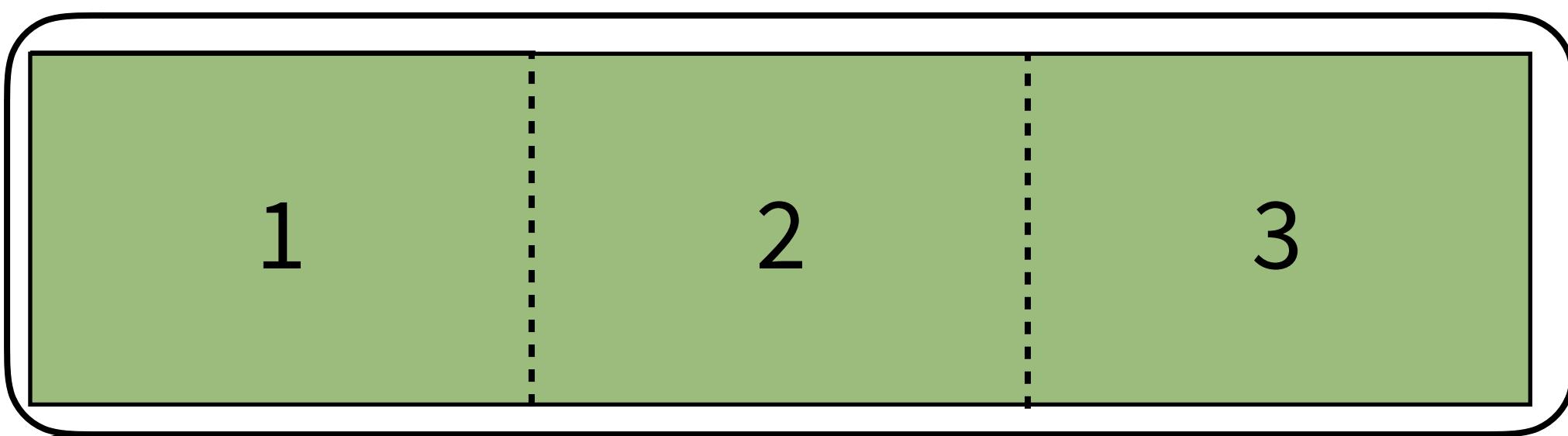
logical

Atomic Vector

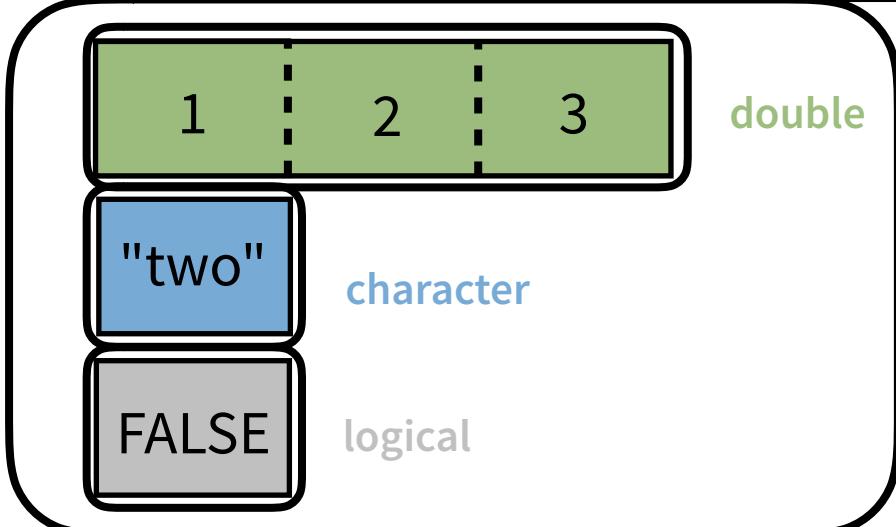


character

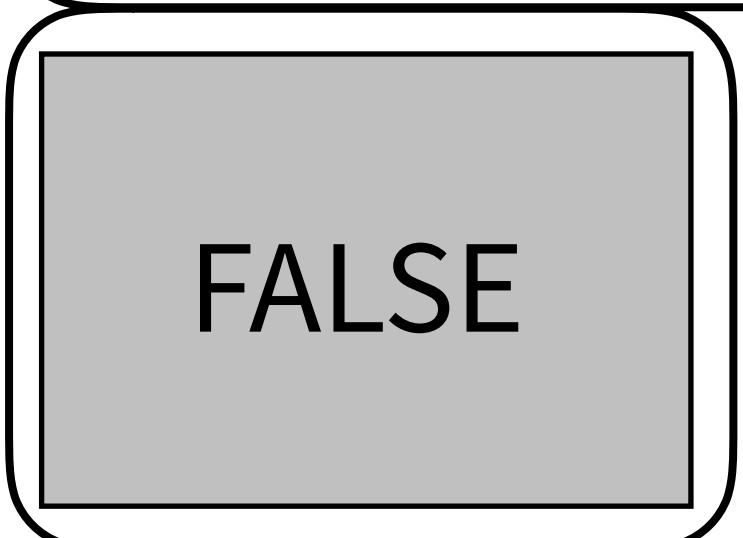
List



double



list



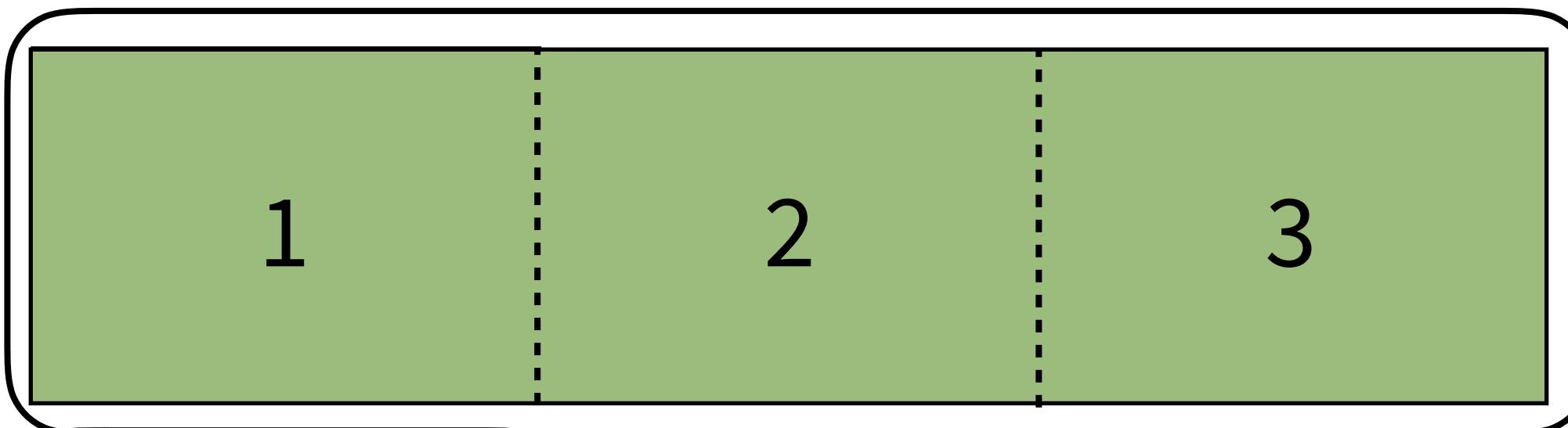
logical

list()

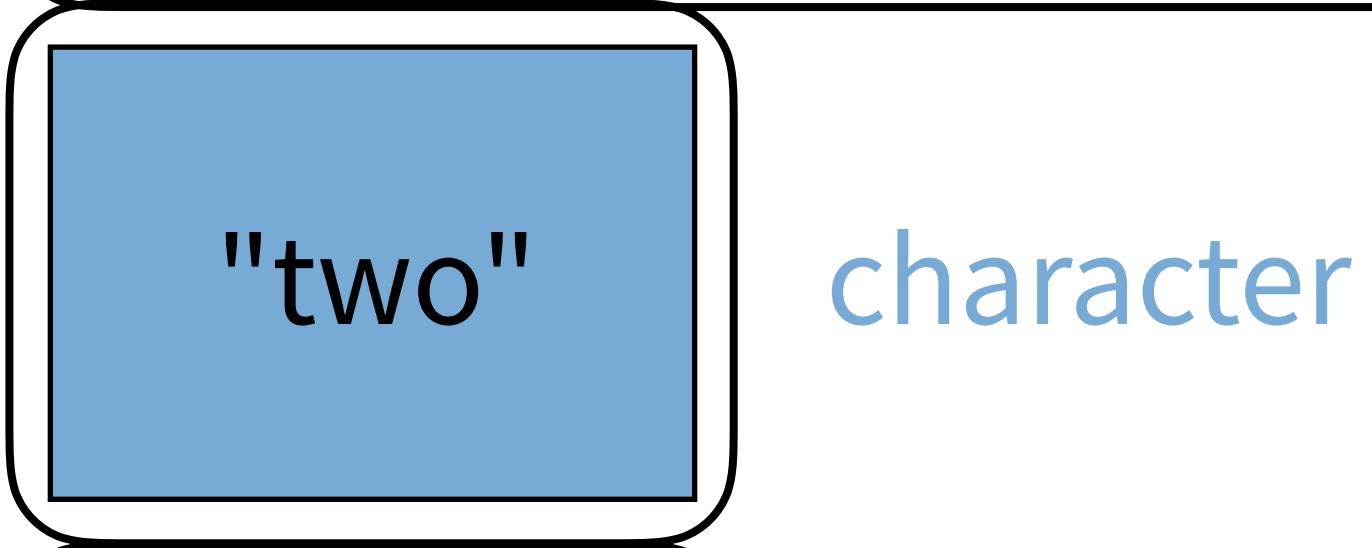
Combines objects into a list.

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

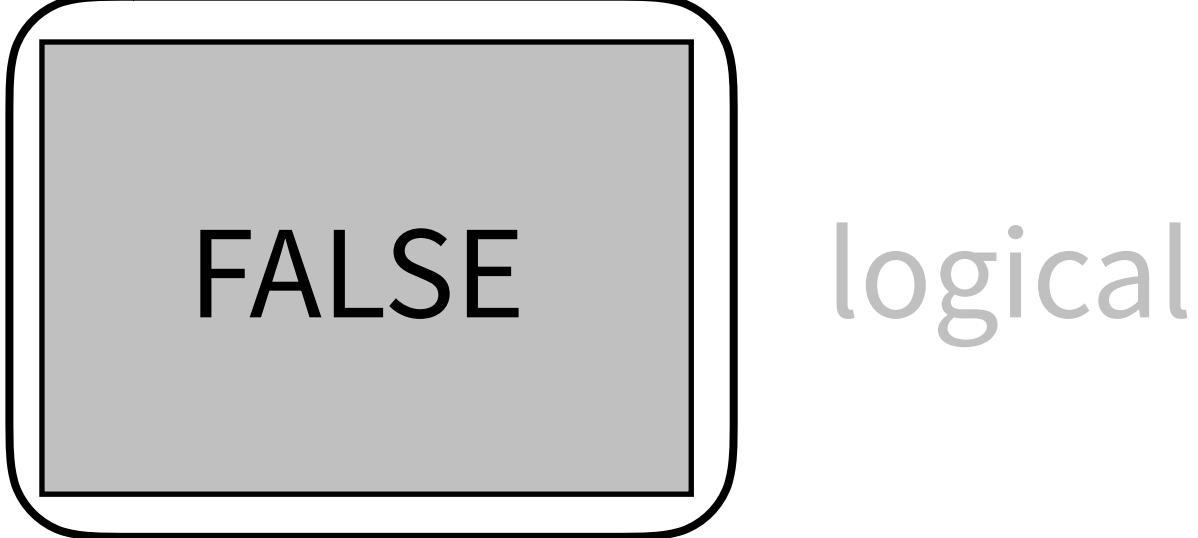
```
$x  
[1] 1 2 3
```



```
$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

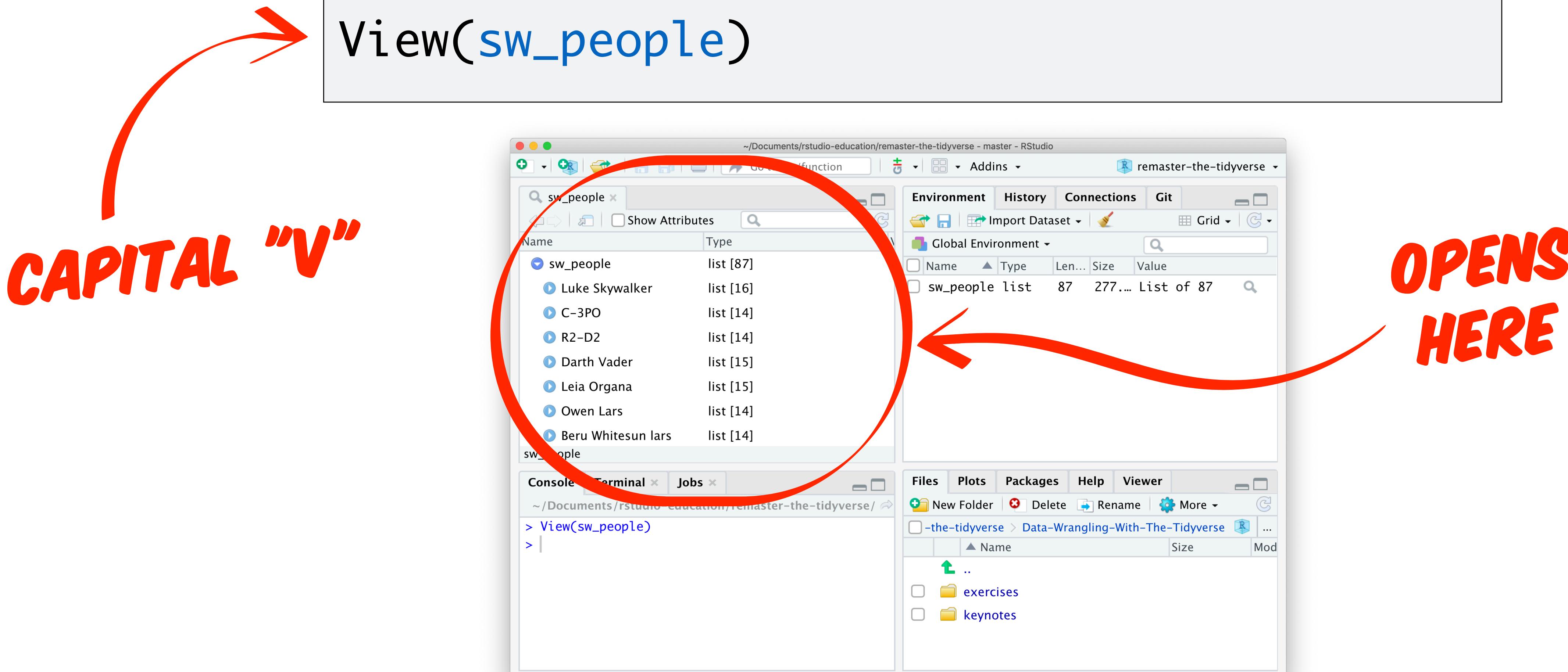
1. **Viewing** contents
2. **Extracting** contents
3. **Using** with functions

Viewing List Contents

R

View()

Opens RStudio IDE's interactive list viewer.



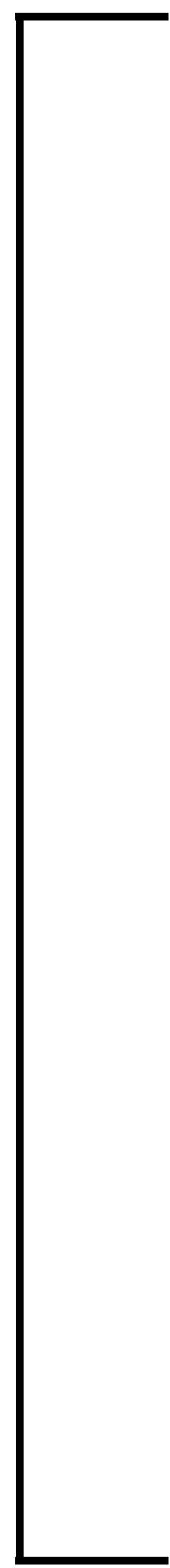
Your Turn 2

Who was taller, Anakin Skywalker or Darth Vader?

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



188 cm



Anakin

24

202 cm



Darth ✓



Extracting List Contents

R

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
```

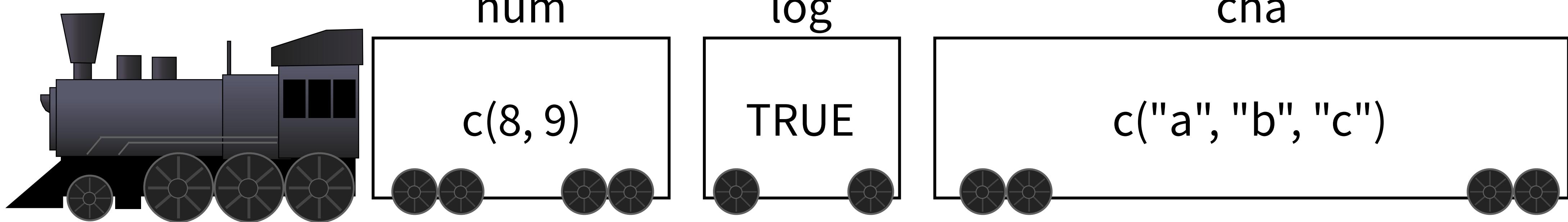
A list

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

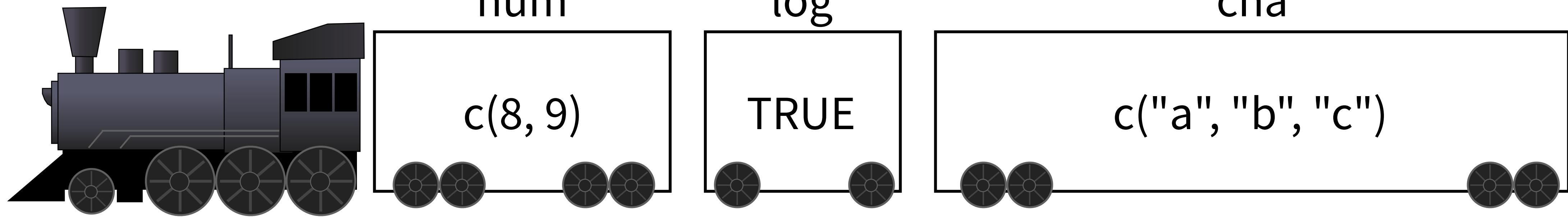
```
a_list[["num"]]
```

```
[1] 8 9
```

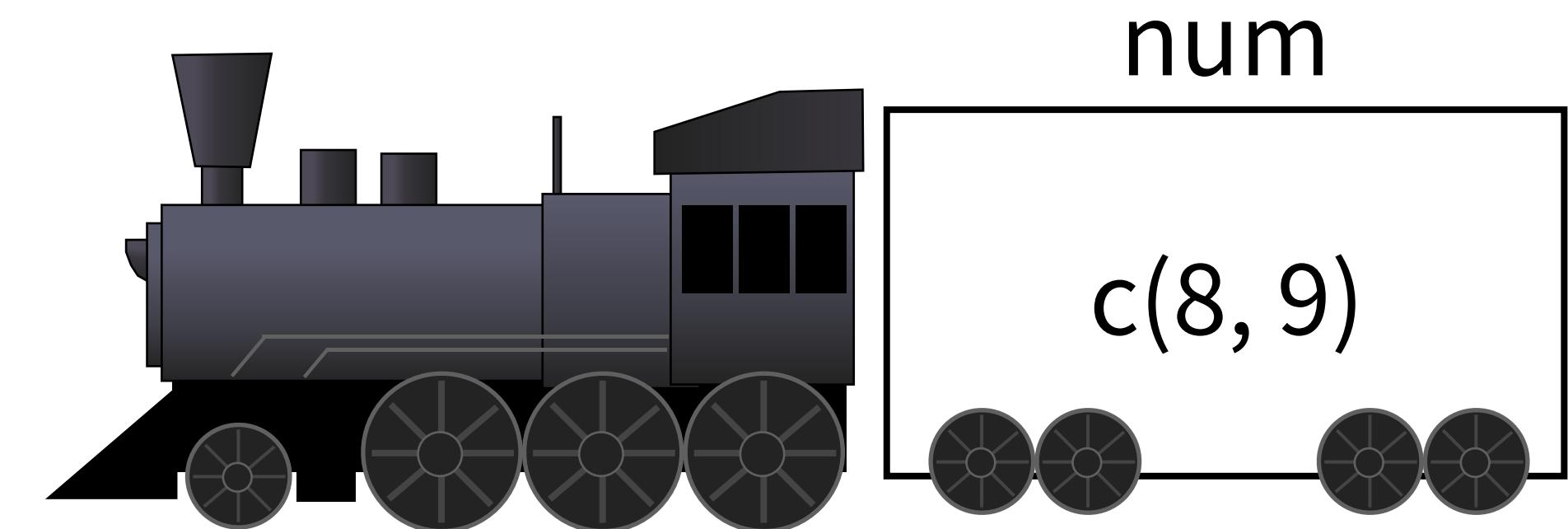
An atomic vector

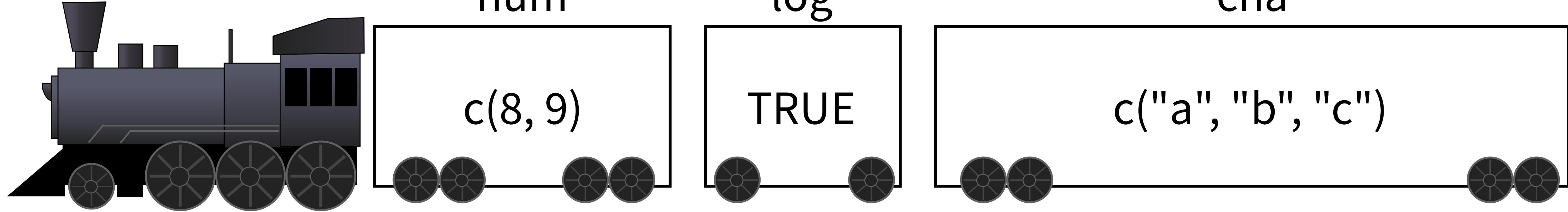


```
a_list <- list(num = c(8,9), log = TRUE, cha = c("a", "b", "c"))
```



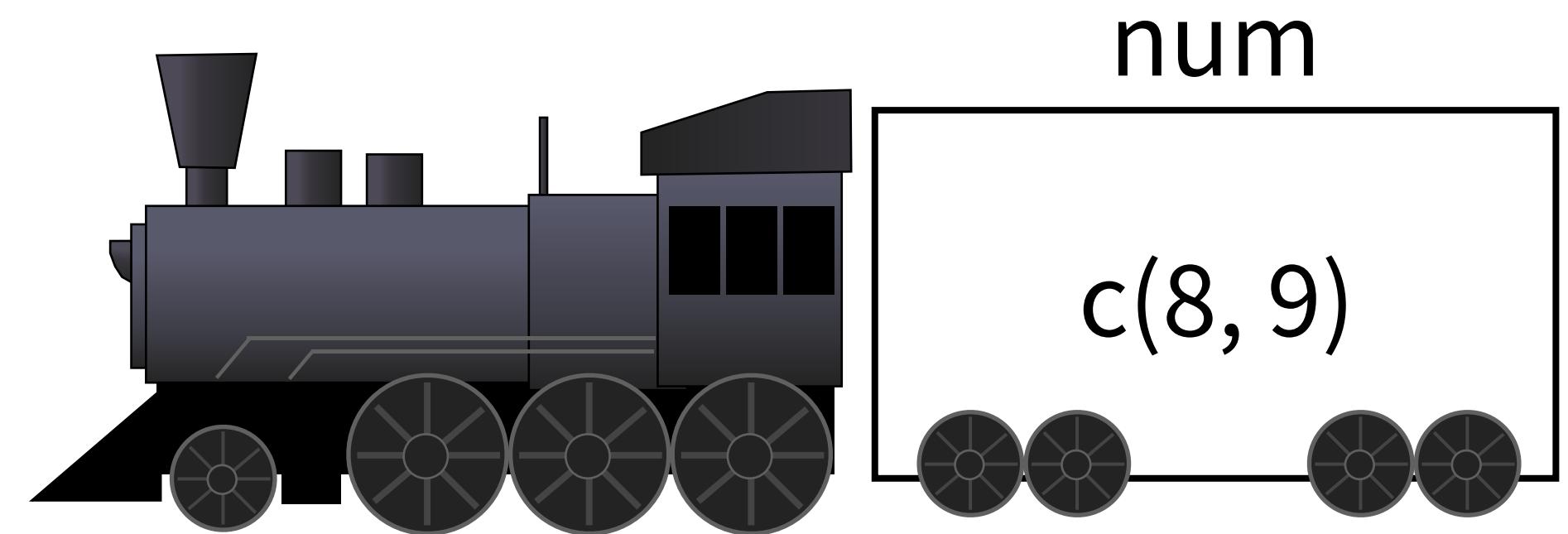
a_list["num"]



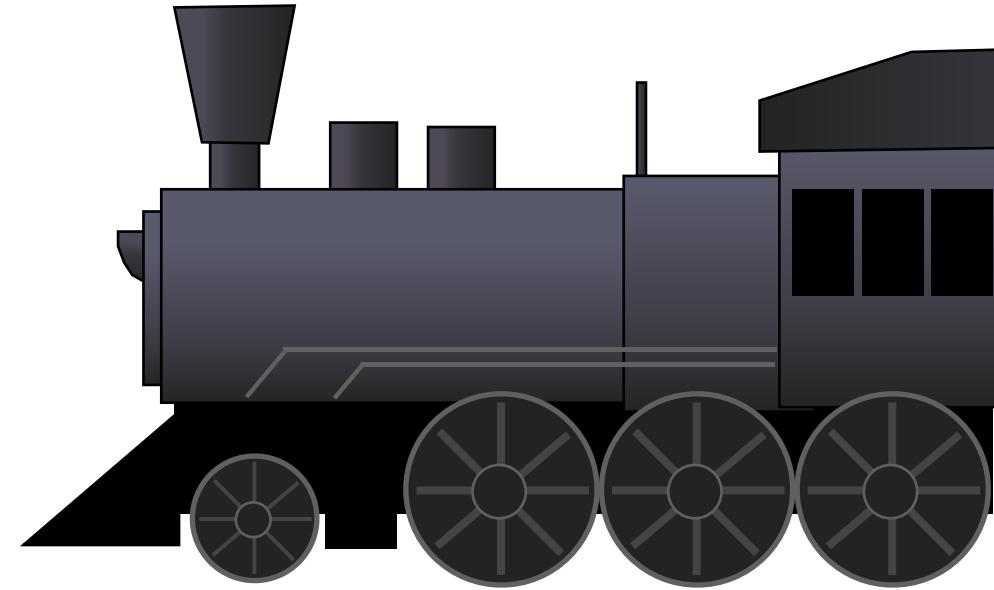


```
a_list["num"]
```

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



c(8, 9)



num

c(8, 9)

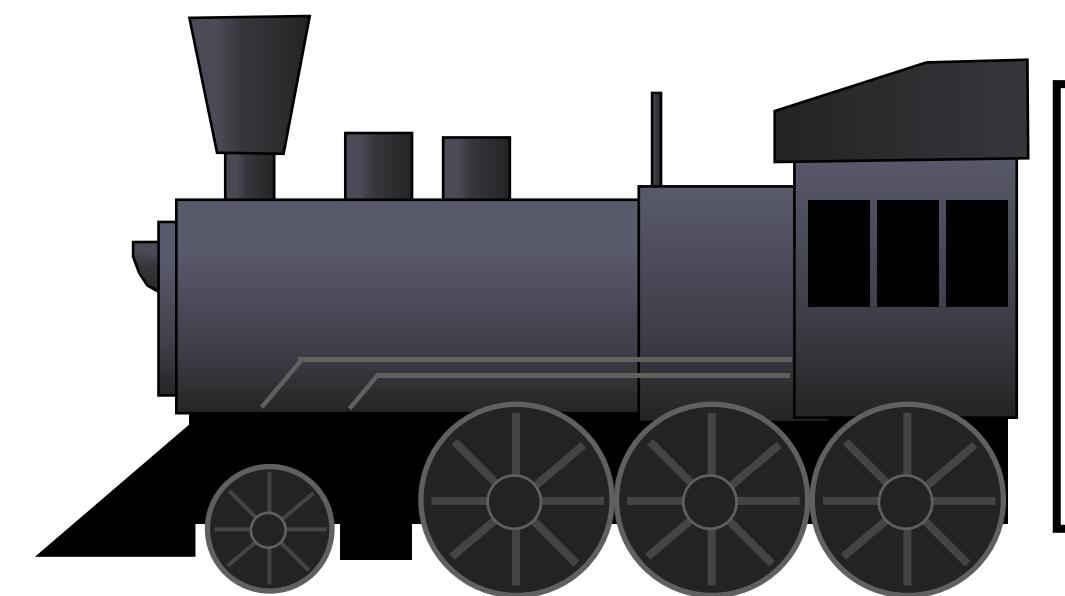
log

TRUE

cha

c("a", "b", "c")

a_list["num"]



num

c(8, 9)

a_list[["num"]]

c(8, 9)

a_list\$num

c(8, 9)



X

32

Q R



X

33



x[1]

R



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("Anakin Skywalker", "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)



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)



pluck()

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

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

A list

Number or name
of element to
extract from list



Main difficulties with lists

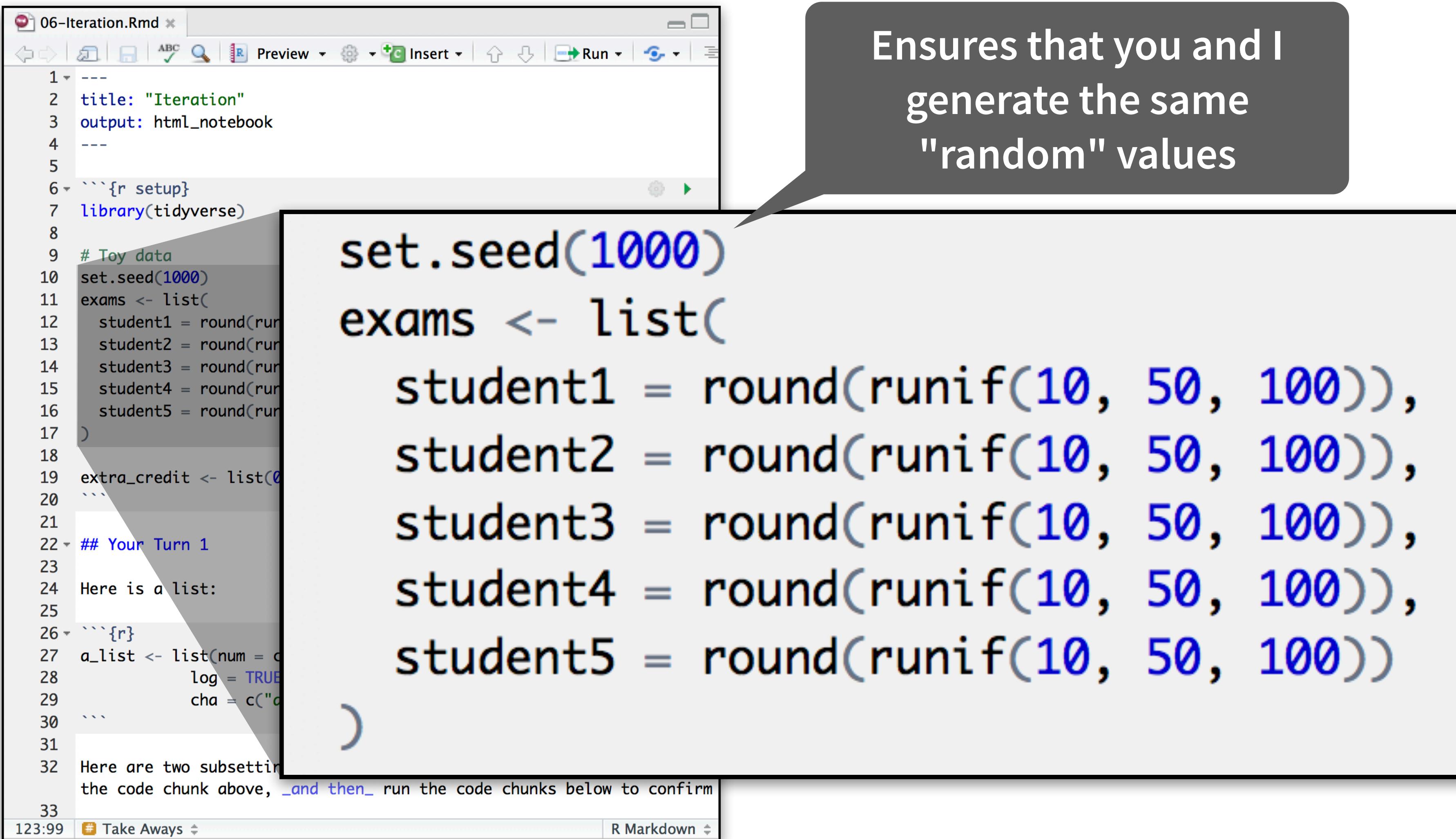
1. **Viewing** contents
2. **Extracting** contents
3. **Using** with functions

Mapping



Toy data

Suppose we have the exam scores of five students...



The screenshot shows an R Markdown file titled "06-Iteration.Rmd". The code is as follows:

```
1 ---  
2 title: "Iteration"  
3 output: html_notebook  
4 ---  
5  
6 ```{r setup}  
7 library(tidyverse)  
8  
9 # Toy data  
10 set.seed(1000)  
11 exams <- list(  
12   student1 = round(runif(10, 50, 100)),  
13   student2 = round(runif(10, 50, 100)),  
14   student3 = round(runif(10, 50, 100)),  
15   student4 = round(runif(10, 50, 100)),  
16   student5 = round(runif(10, 50, 100))  
17 )  
18  
19 extra_credit <- list()  
20 ...  
21  
22 ## Your Turn 1  
23  
24 Here is a list:  
25  
26 ```{r}  
27 a_list <- list(num = c(1, 2, 3),  
28                 log = TRUE,  
29                 cha = c("cat", "dog"))  
30 ...  
31  
32 Here are two subsetting  
33 the code chunk above, and then run the code chunks below to confirm
```

A callout bubble points to the line `set.seed(1000)` with the text: "Ensures that you and I generate the same "random" values".



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



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, ...)
```

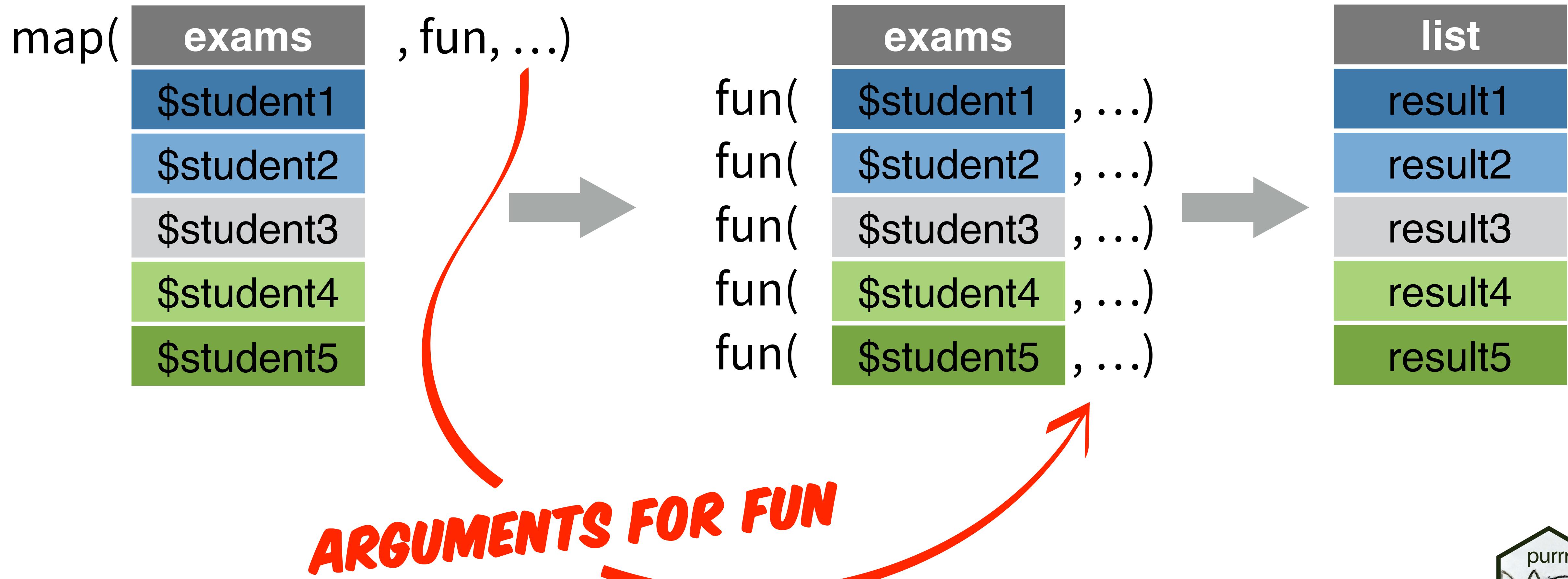
A list

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

Other
arguments to
pass to the
function



map()



```
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

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

```
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



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

\$student1

[1] 71.4

\$student2

[1] 74.6

\$student3

[1] 70.2

\$student4

[1] 75.1

\$student5

[1] 79.1



```
exams %>% ?
```

student1 student2 student3 student4 student5

71.4

74.6

70.2

75.1

79.1



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_db1(mean)
```

student1 student2 student3 student4 student5

71.4

74.6

70.2

75.1

79.1



```
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

\$student4



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

\$student1

A tibble: 1 × 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 × 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 × 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>									
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>
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>,
```



What if what we want to read multiple similar data files into one data frame?

```
list.files("./data",
  pattern = "*csv",
  full.names = TRUE)
```

"./data/0/03-01-2020.csv" "./data/03-02-2020.csv" "./data/03-03-2020.csv"

.....

Province/State,Country/Region,Last
Update,Confirmed,Deaths,Recovered,La
titude,Longitude
Hubei,Mainland China,
2020-03-01T10:13:19,66907,2761,31536

Province/State,Country/Region,Last
Update,Confirmed,Deaths,Recovered,Latitude,
Longitude
Hubei,Mainland China,
2020-03-03T11:43:02,67217,2835,36208,30.975

Province/State,Country/Region,Last
Update,Confirmed,Deaths,Recovered,Latitude,Longitude
Hubei,Mainland China,2020-03-02T15:03:23,67103,2803,33934,30.9756,112.2707
,South Korea,2020-03-02T20:23:16,4335,28,30,36.0000,128.0000
,Italy,2020-03-02T20:23:16,2036,52,149,43.0000,12.0000

Data source: <https://github.com/CSSEGISandData/COVID-19>



```
list.files("./data",
  pattern = "*csv",
  full.names = TRUE) %>%
map_dfr(read_csv)
```

	# A tibble: 1,429 x 8	`Province/State`	`Country/Region`	`Last Update`	Confirmed	Deaths	Recovered
		<chr>	<chr>	<dttm>	<int>	<int>	<int>
1	Hubei		Mainland China	2020-03-01 10:13:19	66907	2761	31536
2	NA		South Korea	2020-03-01 23:43:03	3736	17	30
3	NA		Italy	2020-03-01 23:23:02	1694	34	83
4	Guangdong		Mainland China	2020-03-01 14:13:18	1349	7	1016
5	Henan		Mainland China	2020-03-01 14:13:18	1272	22	1198
6	Zhejiang		Mainland China	2020-03-01 10:13:33	1205	1	1046
7	Hunan		Mainland China	2020-03-01 10:03:03	1018	4	866
8	Anhui		Mainland China	2020-03-01 10:03:03	990	6	873



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



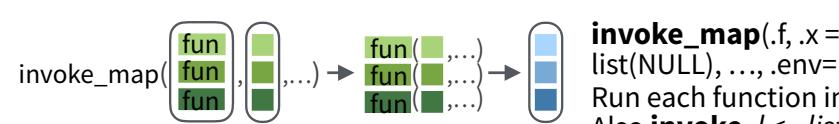
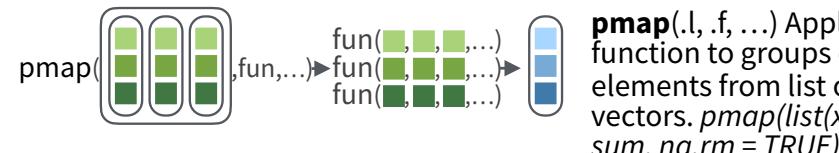
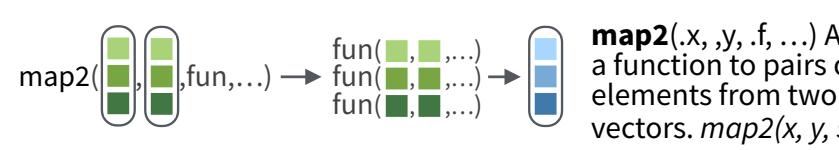
purrr

Apply functions with purrr :: CHEAT SHEET



Apply Functions

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



imap(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.

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)**

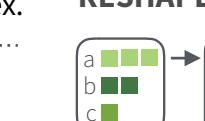
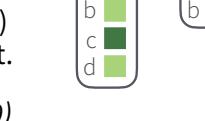
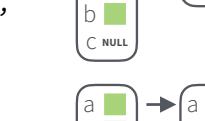
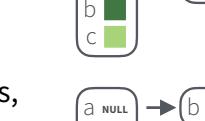
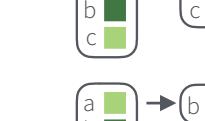
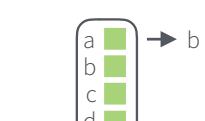
~ .x.y becomes **function(x, .y) .x.y**, e.g. **map2(l, p, ~.x + .y)** becomes **map2(l, p, function(l, p) l + p)**

~ ..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)**

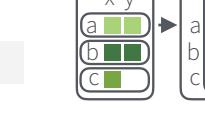
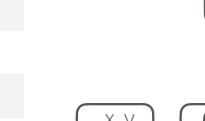
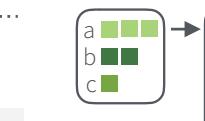


Work with Lists

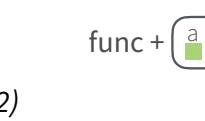
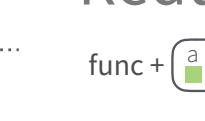
FILTER LISTS



RESHAPE LISTS



JOIN (TO) LISTS



SUMMARISE LISTS

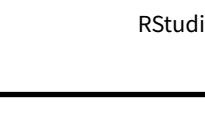
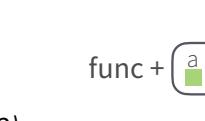
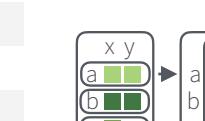
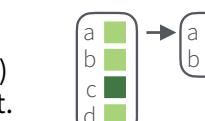
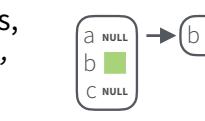
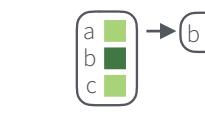
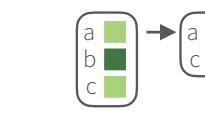


TRANSFORM LISTS

WORK WITH LISTS

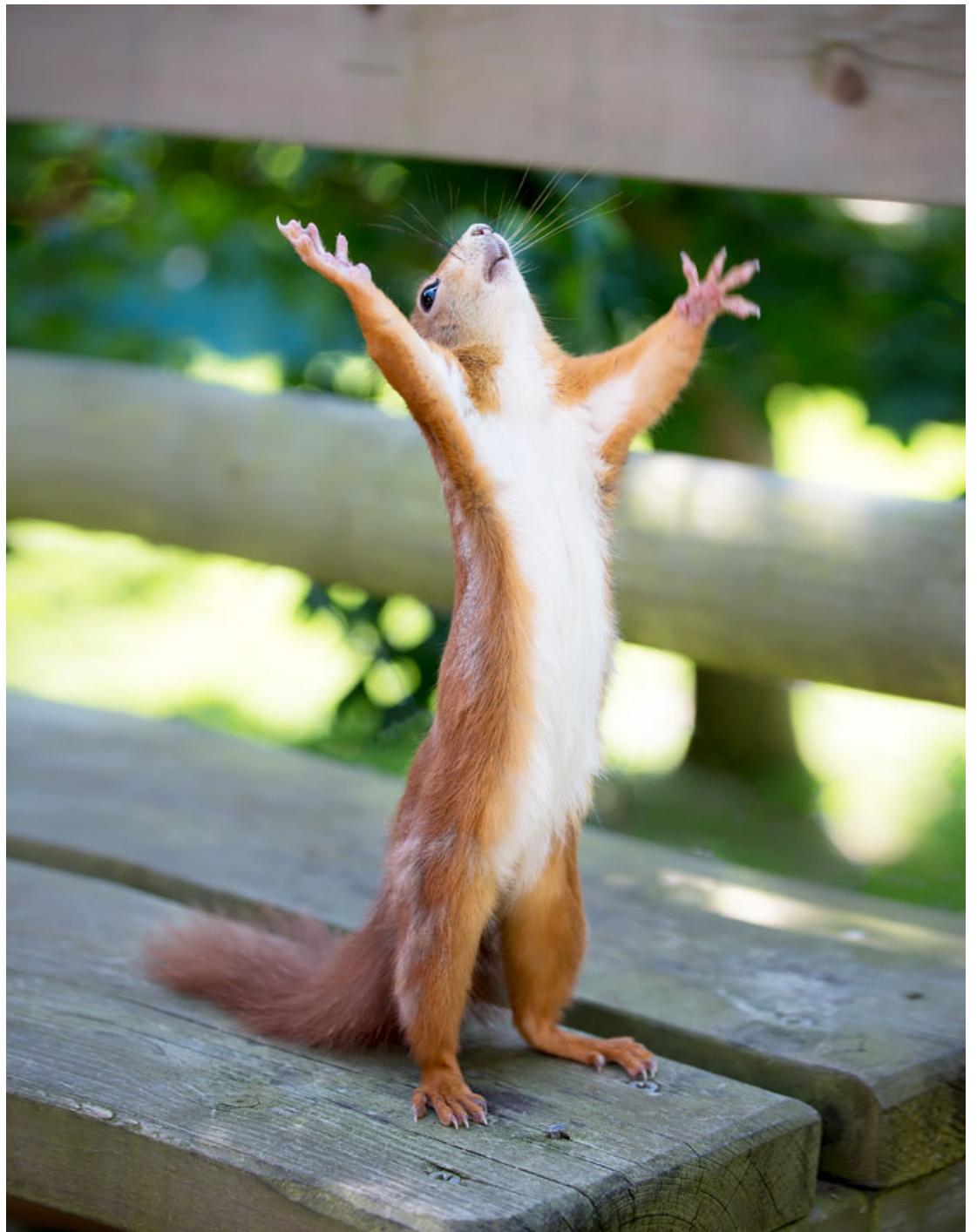
Reduce Lists

Modify function behavior



Lists with





Thank you

Please leave feedback on
your red and green papers