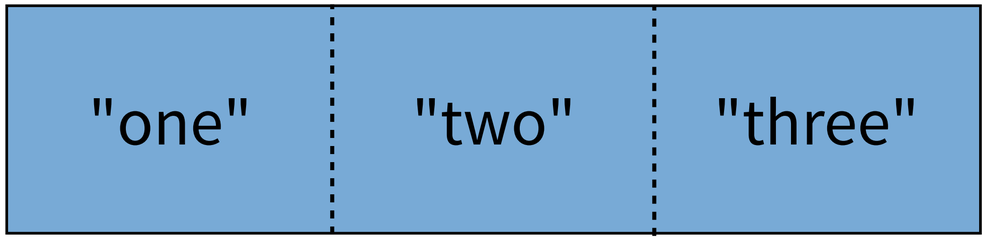
Iteration with

Quiz

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

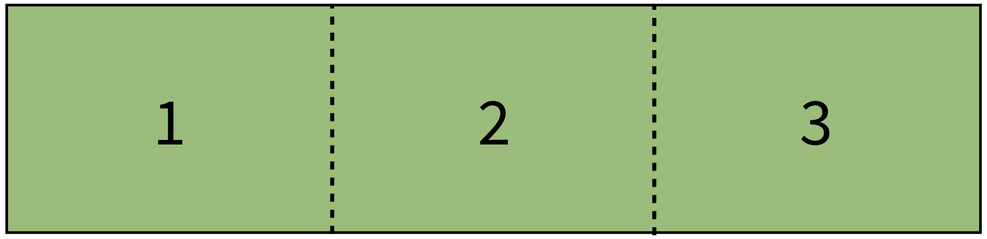
Atomic Vector  type

Atomic Vector



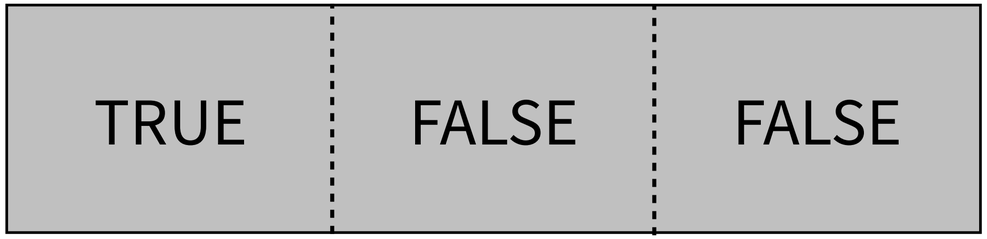
character

Atomic Vector



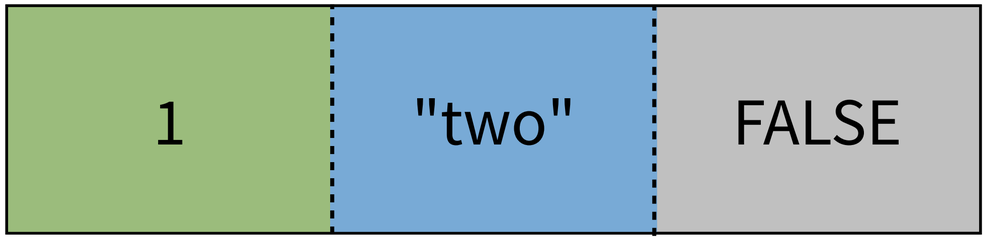
double

Atomic Vector



logical

Atomic Vector

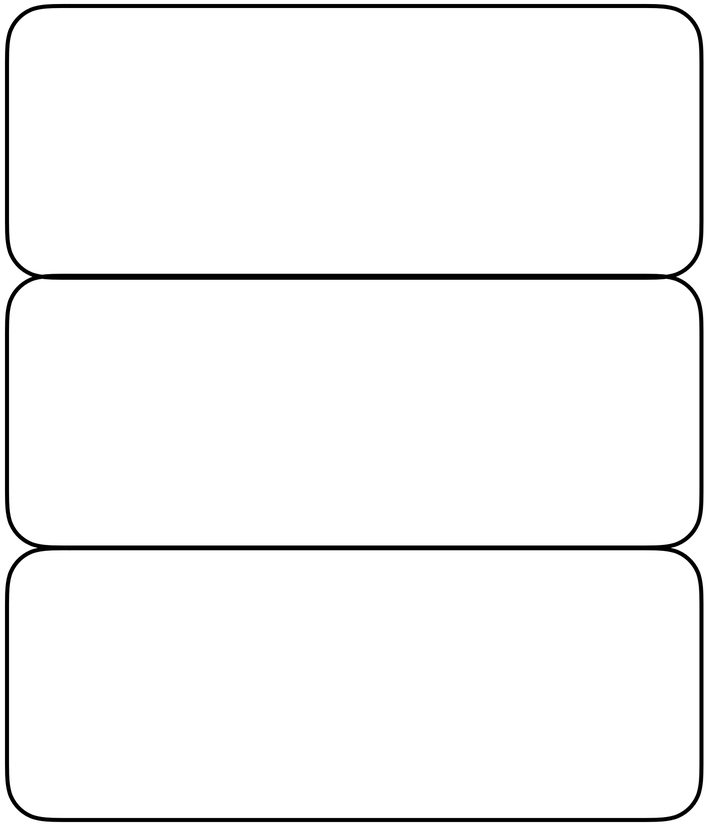


?

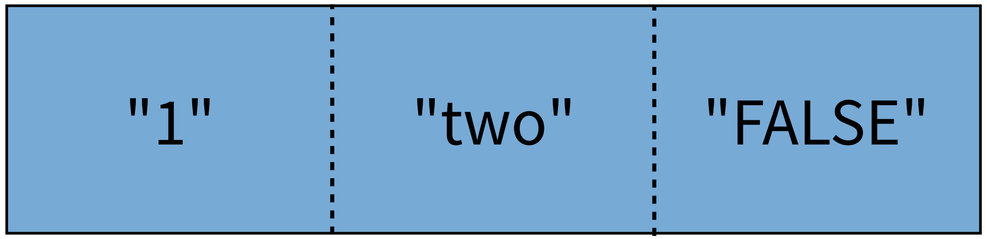
Atomic Vector

type

List

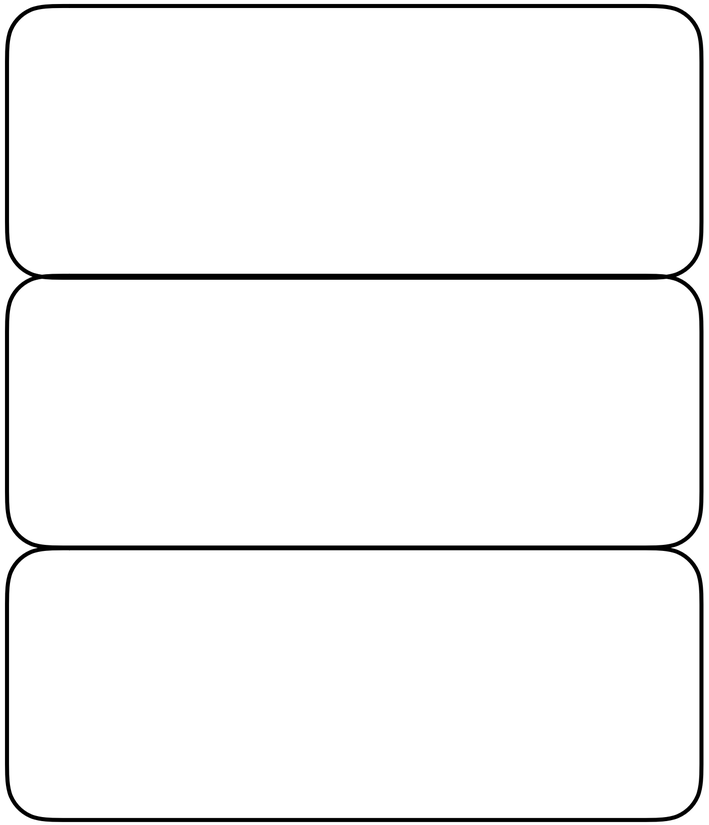


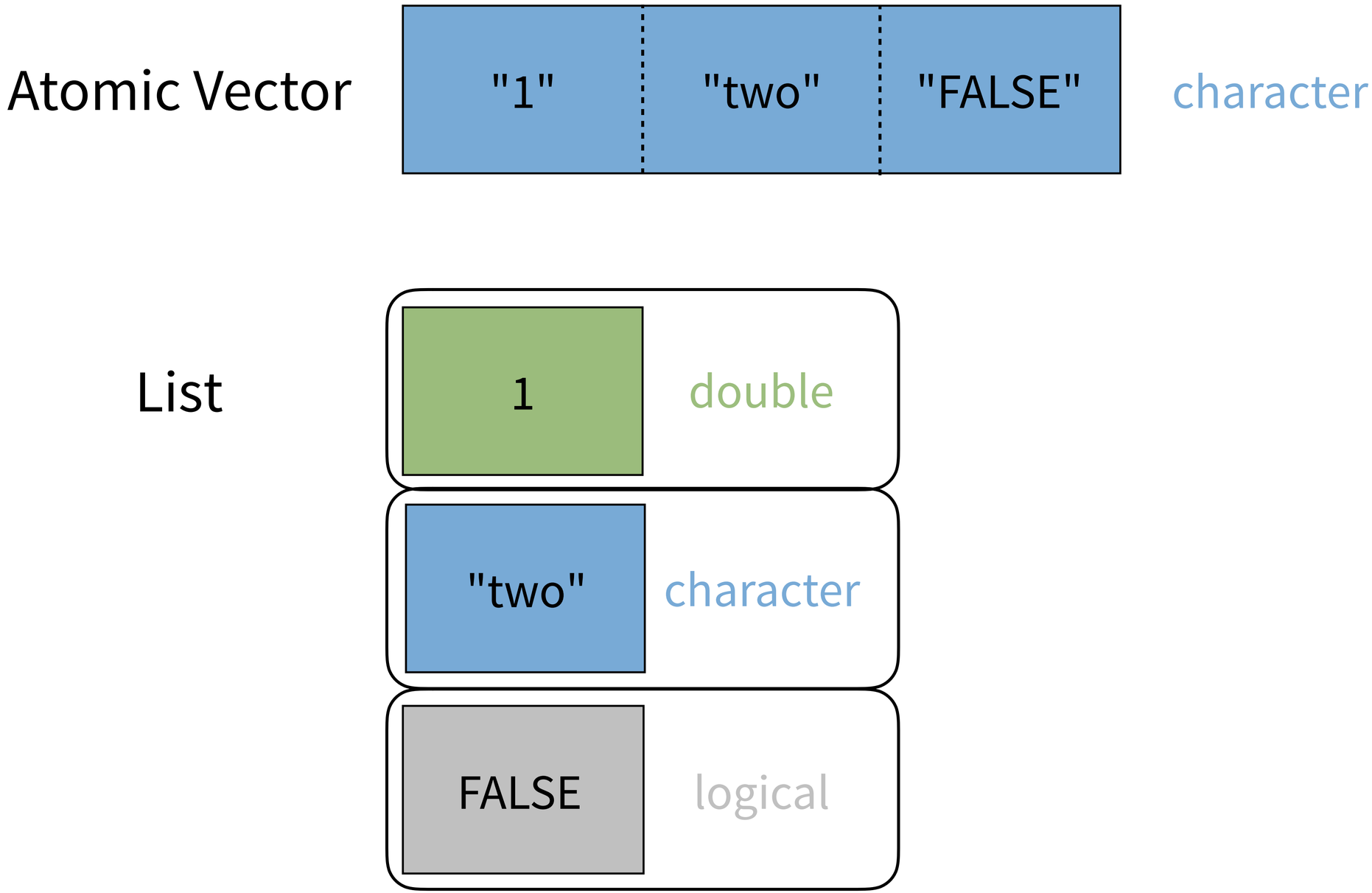
Atomic Vector

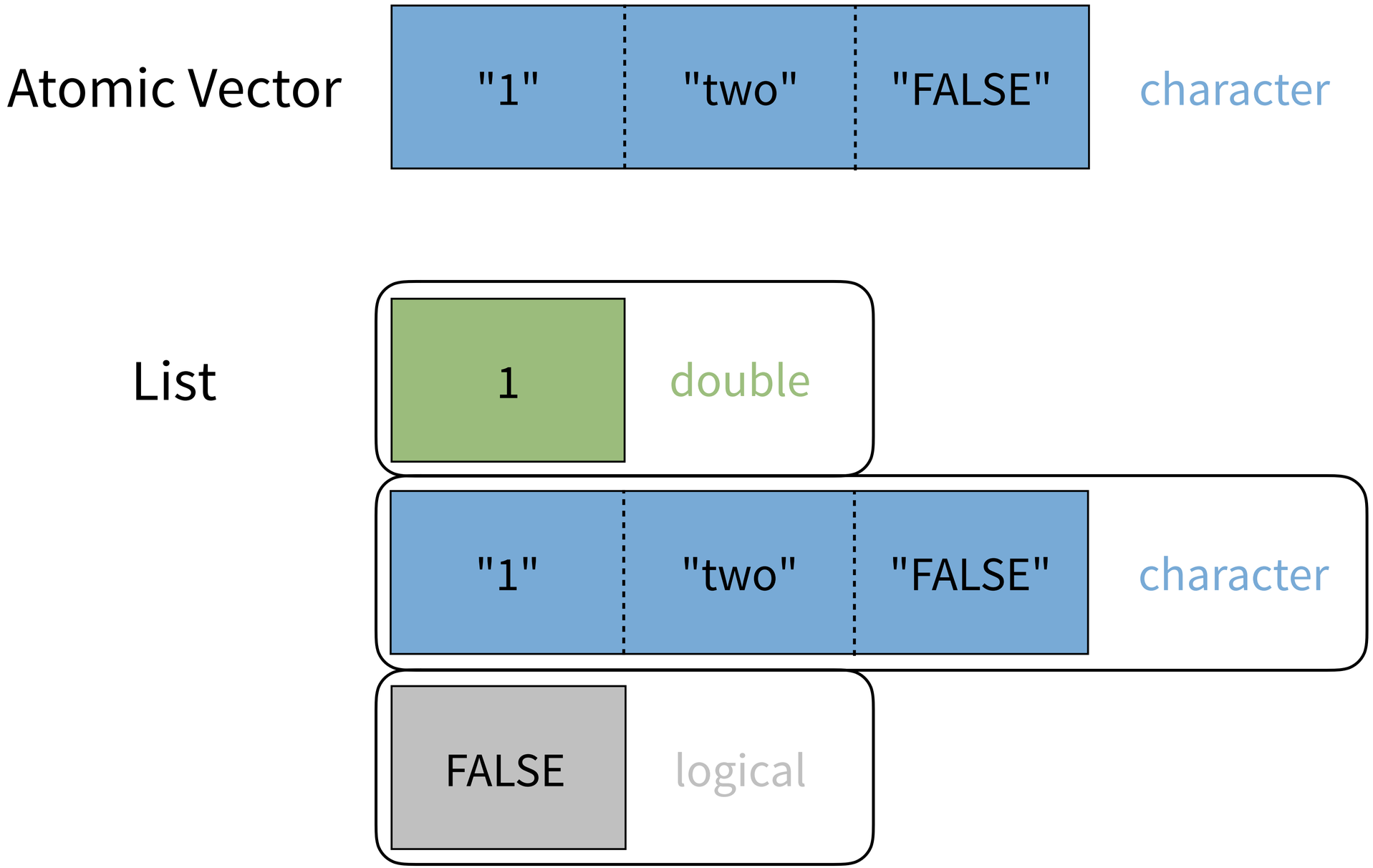


character

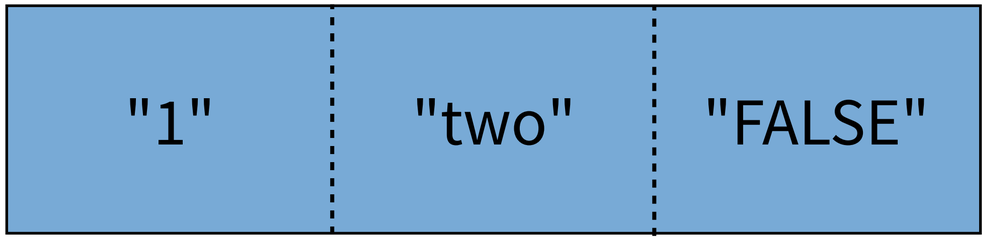
List



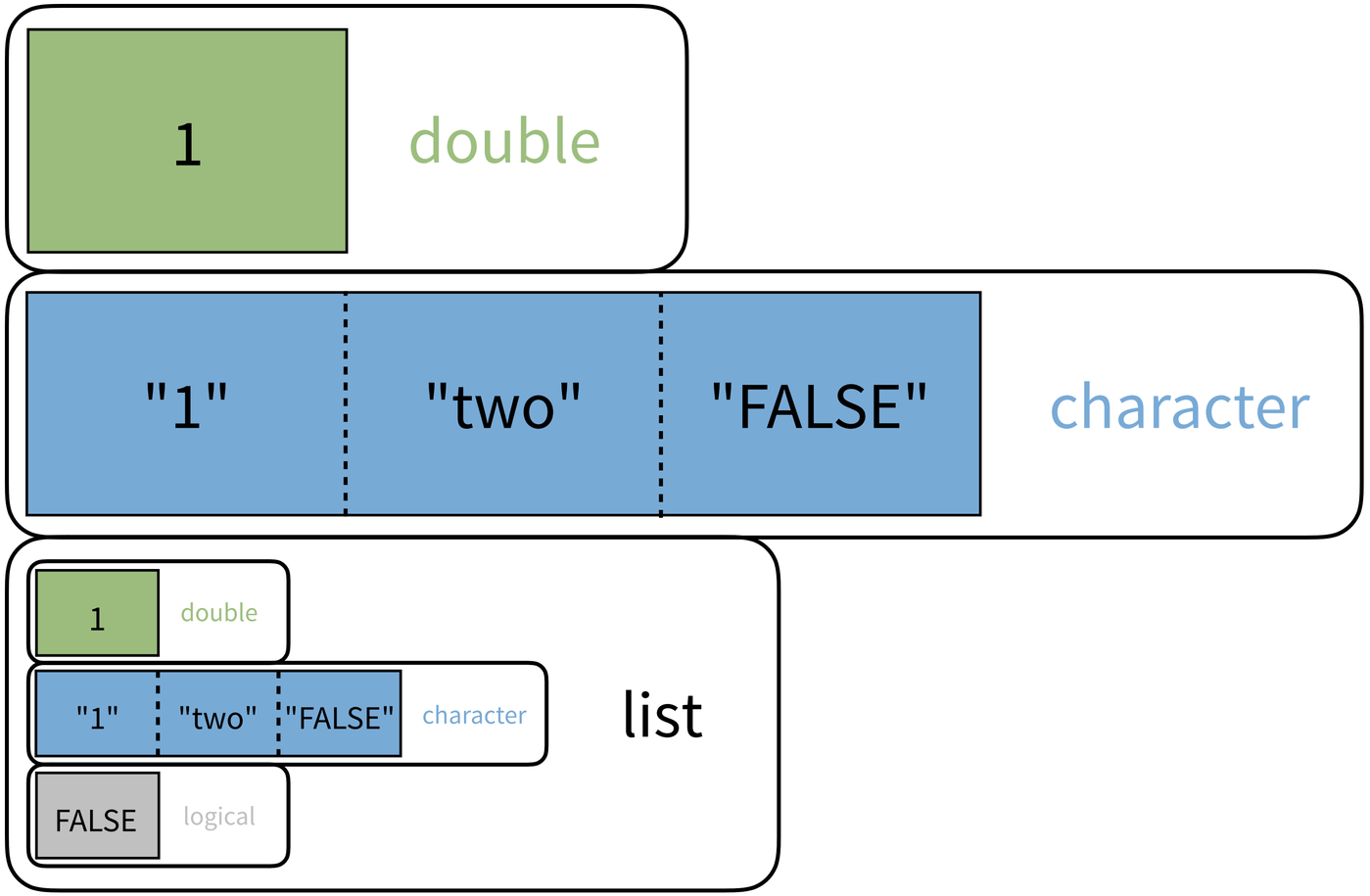

Atomic Vector
"1"
"two"
"FALSE"
character
List
1
double
"two"
character
FALSE
logical


Atomic Vector
"1"
"two"
"FALSE"
character
List
1
double
"1"
"two"
"FALSE"
character
FALSE
logical

Atomic Vector



List



Quiz

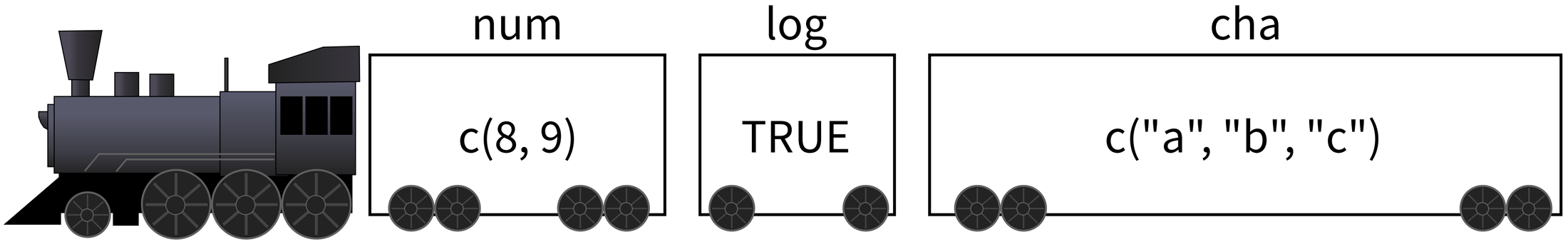
Here is a list:

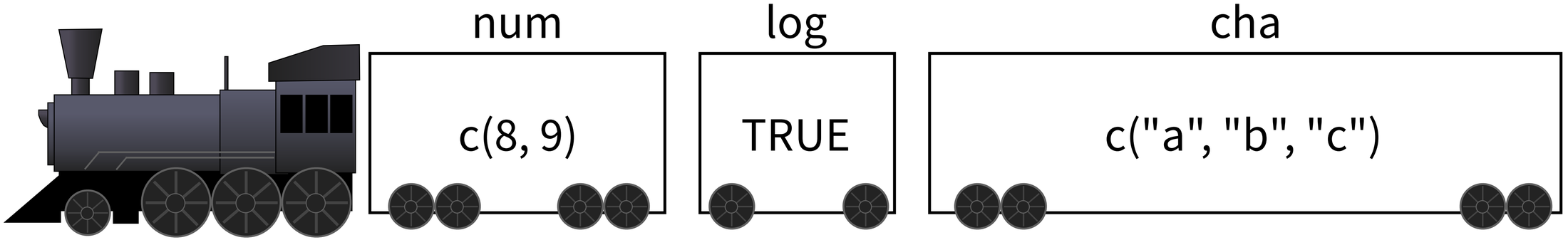
```
lst <- list(num = c(8, 9),
            log = TRUE,
            cha = c("a", "b", "c"))
```

Here is a subsetting command.

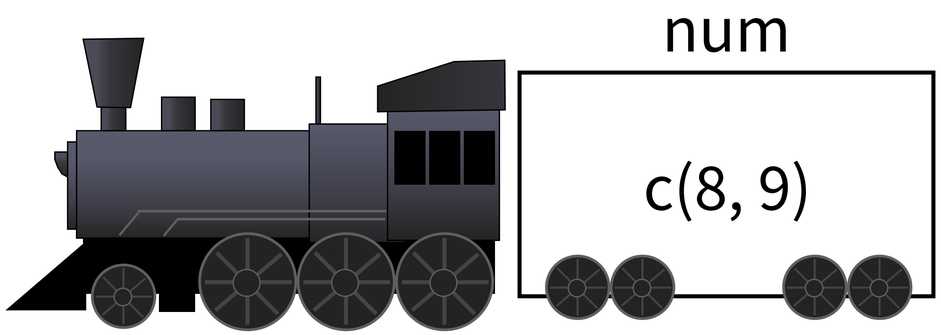
What type of object does it return?

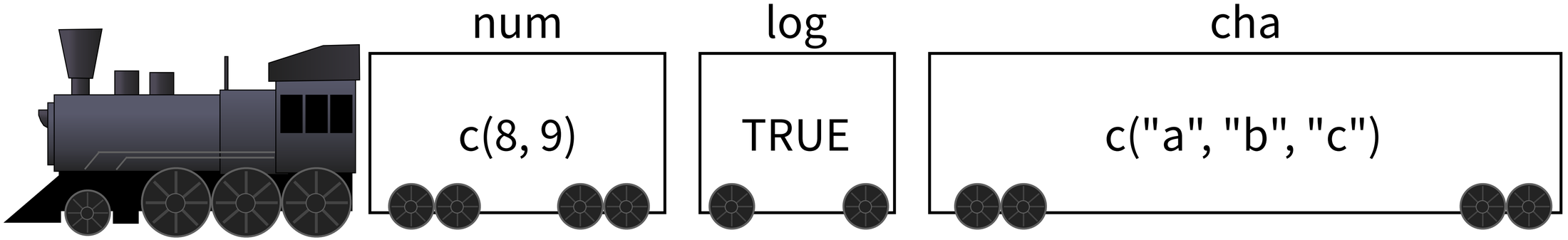
```
lst["num"]
```


num
log
cha
c(8, 9)
TRUE
c("a", "b", "c")

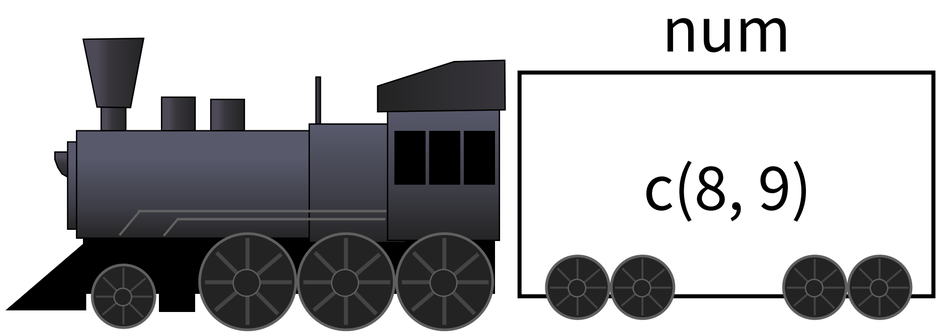


`lst["num"]`



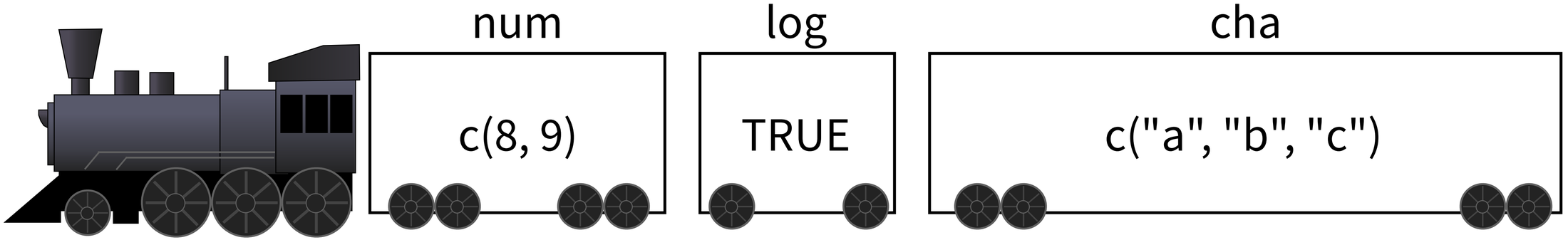


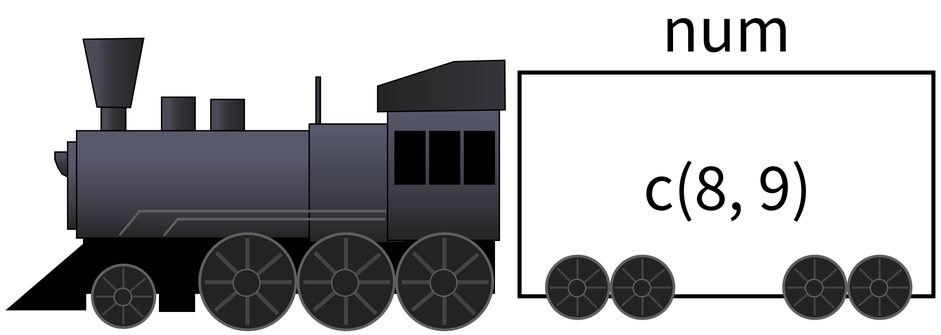
`lst["num"]`



`lst[["num"]]`

c(8, 9)

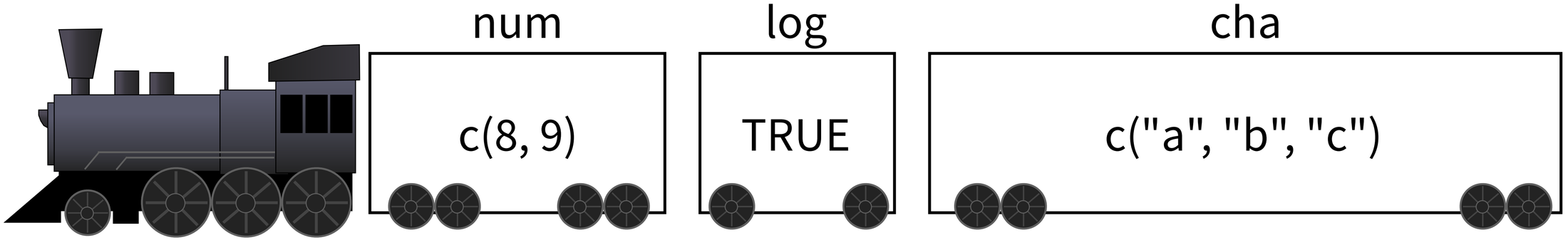


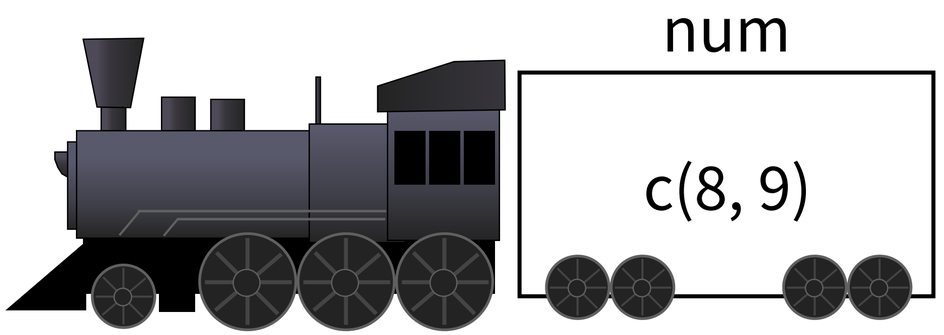



```
lst["num"]
```

```
lst[["num"]]
```

c(8, 9)





`lst["num"]`

`lst[["num"]]` c(8, 9)

`lst$num` c(8, 9)

x

x

x[1]

x

x[1]

x[[1]]

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

Iteration

Quiz

What will this return?

```
vec <- c(-2, -1, 0, 1, 2)
abs(vec)
```

```
# 2 1 0 1 2
```

What will this return?

```
lst <- list(-2, -1, 0, 1, 2)
abs(lst)
```

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

Take home

Lists are a useful way to organize data.
But you need to arrange manually for
functions to iterate over the elements of a list.

purrr

purrr

Functions for working with functions.

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

map()

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

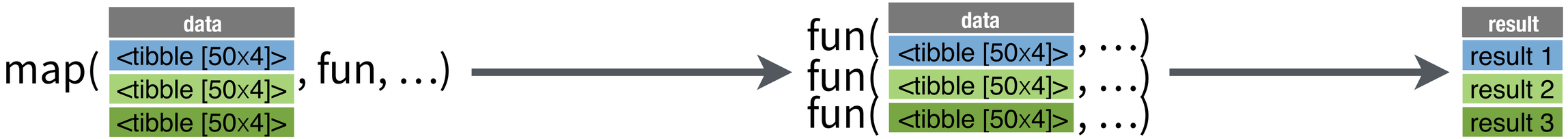
```
map(.x, .f, …)
```

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(
data
<tibble [50×4]>
<tibble [50×4]>
<tibble [50×4]>
, fun, …)
fun(
data
<tibble [50×4]>
, …)
fun(
<tibble [50×4]>
, …)
fun(
<tibble [50×4]>
, …)
result
result 1
result 2
result 3


purrr

Toy data

Suppose we have the exam scores of five students…

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

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

If the final grade is the mean, we can compute it for each with:

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

```
$student1
[1] 71.3485

$student2
[1] 74.6095

$student3
[1] 70.21575

$student4
[1] 75.30758

$student5
[1] 79.06386
```

map functions

function	returns results as
map()	list
map_chr()	character vector
map_dbl()	double vector (numeric)
map_int()	integer vector
map_lgl()	logical vector
map_df()	data frame

map_dbl()

If we want the output as a vector:

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

```
## student1  student2  student3  student4  student5
## 71.34850  74.60950  70.21575  75.30758  79.06386
```

extra arguments

What if the grade was the 90th percentile score?

```
exams %>%
  map_dbl(quantile, prob = 0.9)
```

```
## student1  student2  student3  student4  student5
## 87.03640  88.71630  90.34335  90.09150  90.88785
```

extra argument for quantile

map_lgl()

How about a participation grade?

```
exams %>%
  map(length) %>%
  map_lgl(all.equal, 10)
```

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

Your Turn

Calculate the standard deviation (**sd()**) of each student's exams. Return the result as a vector.

```
exams %>%
  map_dbl(sd)
```

```
## student1  student2  student3  student4  student5
## 13.12410  13.98773  14.84878  15.08786  12.78509
```

Quiz

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

For example, what if the final grade is the mean exam score after we drop the lowest score?

A: Write a function.

Functions (very basics)

1. Write code that solves the problem for a real object

```
vec <- exams[[1]]
```

To write a function (very basics)

1. Write code that solves the problem for a real object

```
vec <- exams[[1]]
(sum(vec) - min(vec)) / (length(vec) - 1)
# 73.34424
```

Note: this code does the same thing no matter what vec is.
But it is a bother redefine vec overtime we use the code.

```
vec <- exams[[1]]
  (sum(vec) - min(vec)) / (length(vec) - 1)
vec <- exams[[2]]
  (sum(vec) - min(vec)) / (length(vec) - 1)
vec <- exams[[3]]
  (sum(vec) - min(vec)) / (length(vec) - 1)
vec <- exams[[4]]
  (sum(vec) - min(vec)) / (length(vec) - 1)
vec <- exams[[5]]
  (sum(vec) - min(vec)) / (length(vec) - 1)
```

To write a function (very basics)

1. Write code that solves the problem for a real object
2. Wrap the code in **function(){}** to save it

```
vec <- exams[[1]]
grade <- function() {
  (sum(vec) - min(vec)) / (length(vec) - 1)
}
```

To write a function (very basics)

1. Write code that solves the problem for a real object
2. Wrap the code in function(){} to save it
3. **Add the name of the real object as the function argument**

```
vec <- exams[[1]]
grade <- function(vec) {
  (sum(vec) - min(vec)) / (length(vec) - 1)
}
```

To write a function (very basics)

1. Write code that solves the problem for a real object
2. Wrap the code in function(){} to save it
3. Add the name of the real object as the function argument
4. To run the function ,call the object followed by parentheses.
   Supply new values to use for each of the arguments.

```
vec <- exams[[1]]
grade <- function(vec) {
  (sum(vec) - min(vec)) / (length(vec) - 1)
}
grade(exams[[2]]) # 76.93898
```

```
grade <- function(vec) {
  (sum(vec) - min(vec)) / (length(vec) - 1)
}


exams %>%
  map_dbl(grade)
## student1  student2  student3  student4  student5
## 73.34424  76.93898  72.06320  78.00649  81.68257
```

```
grade <- function(x) {
  (sum(x) - min(x)) / (length(x) - 1)
}

exams %>%
  map_dbl(grade)
## student1  student2  student3  student4  student5
## 73.34424  76.93898  72.06320  78.00649  81.68257
```

```
grade <- function(x) (sum(x) - min(x)) / (length(x) - 1)

exams %>%
  map_dbl(grade)
## student1  student2  student3  student4  student5
## 73.34424  76.93898  72.06320  78.00649  81.68257
```

```r
grade <- function(x) (sum(x) - min(x)) / (length(x) - 1)


exams %>%
  map_dbl(function(x) (sum(x) - min(x)) / (length(x) - 1))
## student1  student2  student3  student4  student5
## 73.34424  76.93898  72.06320  78.00649  81.68257
```

Your Turn

Write a function that counts the best exam twice and then takes the average. Use it to grade all of the students.

```
exams %>%
  map_dbl(function(x) (sum(x) + max(x)) / (length(x) + 1))
## student1   student2   student3   student4   student5
## 73.34424   76.93898   72.06320   78.00649   81.68257
```

map2()

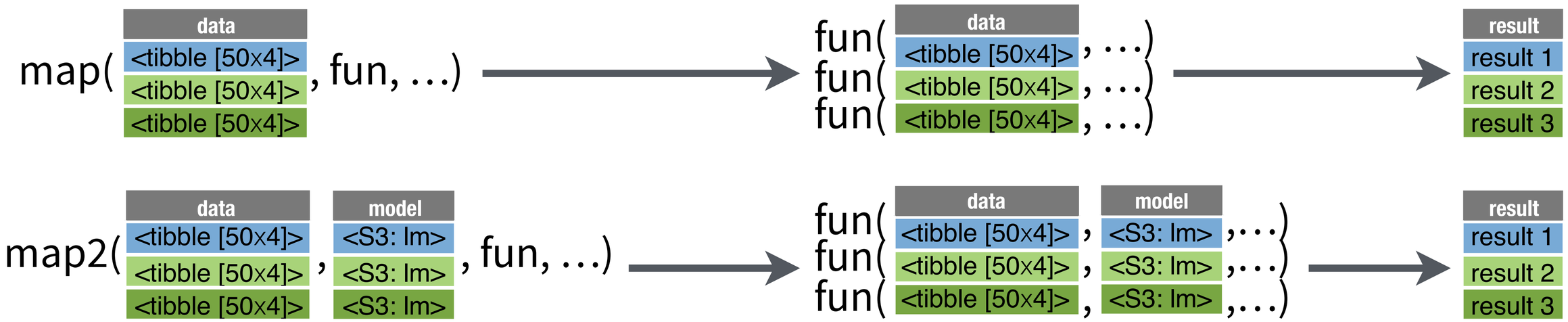
Applies a function to every element of two lists.
Returns the results as a list.

```
map2(.x, .y, .f, …)
```

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

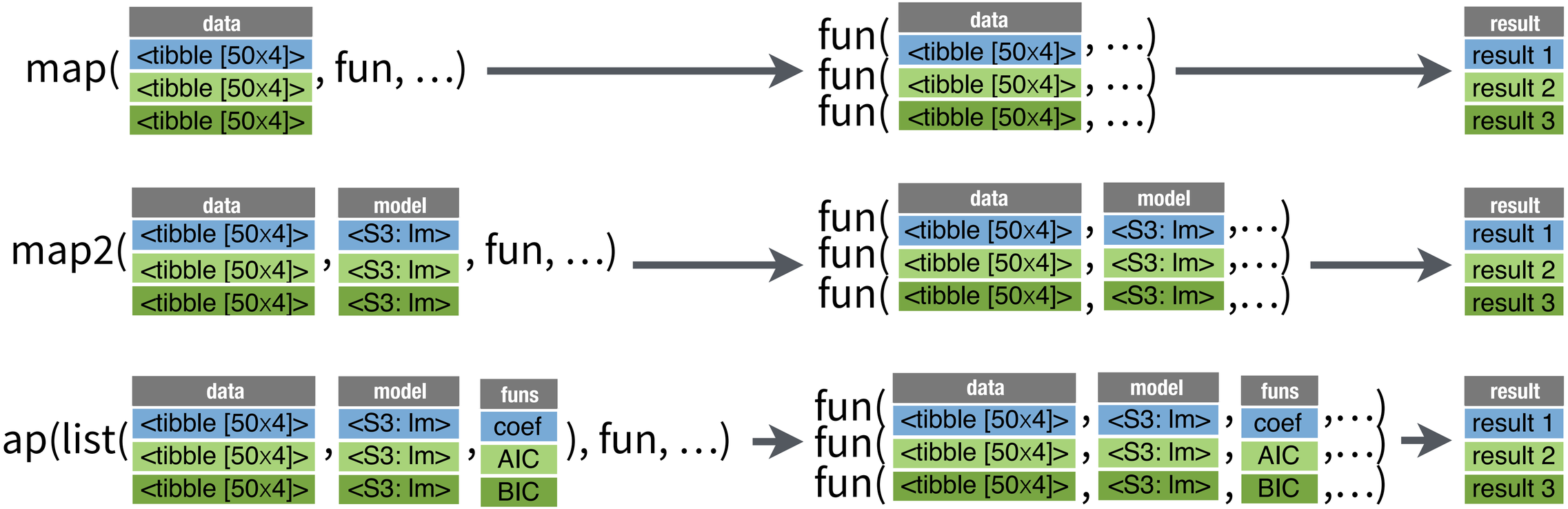
A list of elements to pass to the second argument of .f

map2()


map(
data
<tibble [50x4]>
<tibble [50x4]>
<tibble [50x4]>
, fun, ...)
fun(data <tibble [50x4]> , ...)
fun(<tibble [50x4]> , ...)
fun(<tibble [50x4]> , ...)
result
result 1
result 2
result 3
map2(
data
<tibble [50x4]>
<tibble [50x4]>
<tibble [50x4]>
,
model
<S3: lm>
<S3: lm>
<S3: lm>
, fun, ...)
fun(data <tibble [50x4]> , model <S3: lm> ,...)
fun(<tibble [50x4]> , <S3: lm> ,...)
fun(<tibble [50x4]> , <S3: lm> ,...)
result
result 1
result 2
result 3


purrr

pmap()


map(data <tibble [50x4]> <tibble [50x4]> <tibble [50x4]> , fun, ...)
fun(data <tibble [50x4]> , ...)
fun(<tibble [50x4]> , ...)
fun(<tibble [50x4]> , ...)
result result 1 result 2 result 3
map2(data <tibble [50x4]> <tibble [50x4]> <tibble [50x4]> , model <S3: lm> <S3: lm> <S3: lm> , fun, ...)
fun(data <tibble [50x4]> , model <S3: lm> ,...)
fun(<tibble [50x4]> , <S3: lm> ,...)
fun(<tibble [50x4]> , <S3: lm> ,...)
result result 1 result 2 result 3
pmap(list(data <tibble [50x4]> <tibble [50x4]> <tibble [50x4]> , model <S3: lm> <S3: lm> <S3: lm> , funs coef AIC BIC), fun, ...)
fun(data <tibble [50x4]> , model <S3: lm> , funs coef ,...)
fun(<tibble [50x4]> , <S3: lm> , AIC ,...)
fun(<tibble [50x4]> , <S3: lm> , BIC ,...)
result result 1 result 2 result 3


purrr

walk()

A version of map for functions that do not return values, but have side effects (e.g. write_csv(), plot(), print())

```
walk(.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 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

Iteration with