Iterate

Jake Thompson
wjakethompson.com
@wjakethompson

ATLAS
Accessible Teaching, Learning, and Assessment Systems

Your Turn 0

- Open **09-Iterate.Rmd**
- Run the setup chunk

Your Turn 1

```
mod <- lm(price ~ carat + cut + color + clarity,
          data = diamonds)
View(mod)
```

What kind of object is **mod**? What are models stored as this kind of object?

02 : 00

Lists

In R4DS

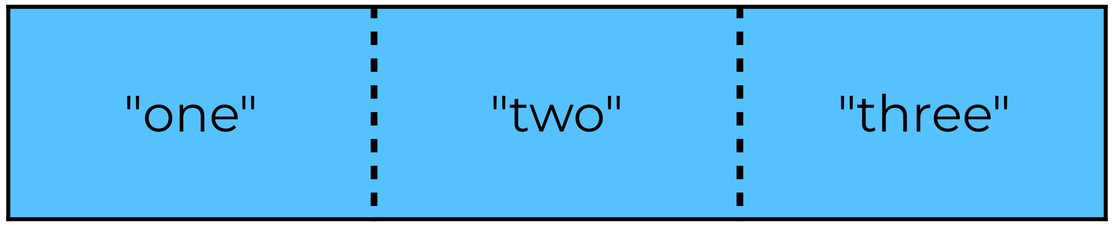
Lists

Quiz

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

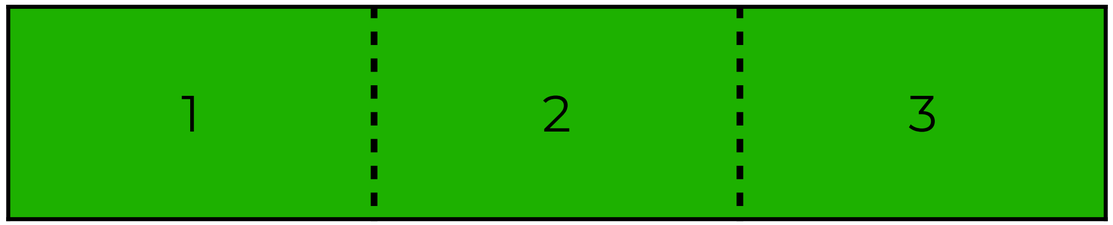
Atomic Vector  type

Atomic Vector



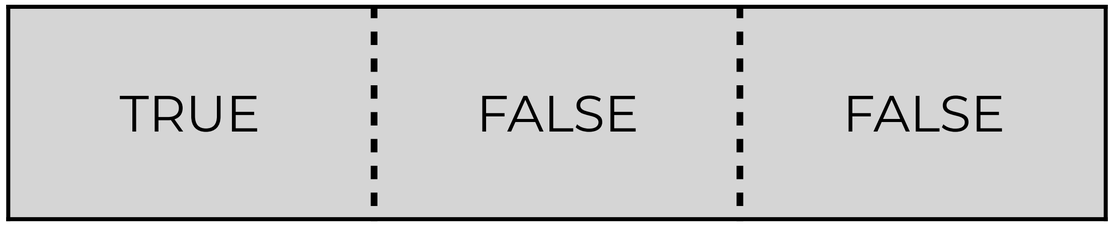
character

Atomic Vector



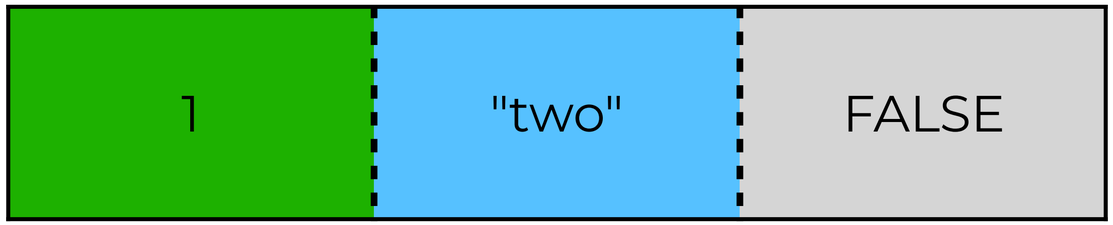
character

Atomic Vector



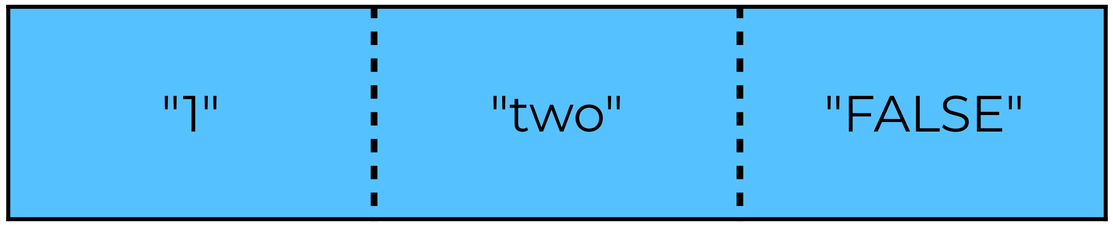
character

Atomic Vector



?

Atomic Vector



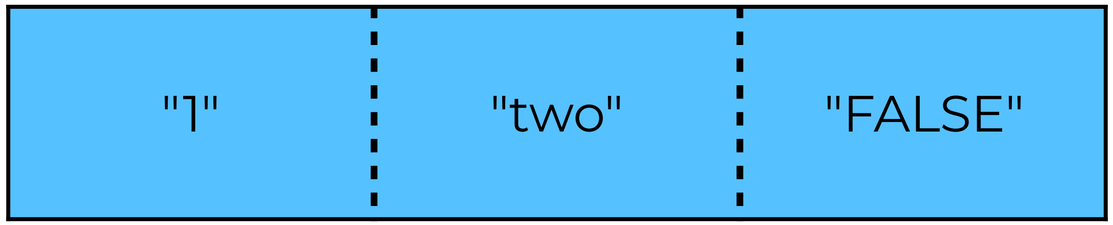
character

Atomic Vector

type

List

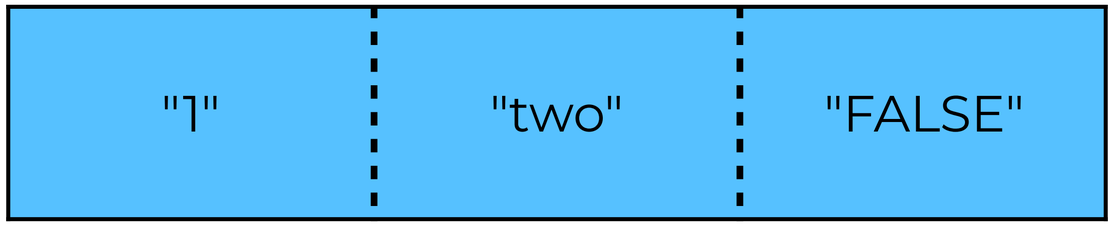
Atomic Vector



character

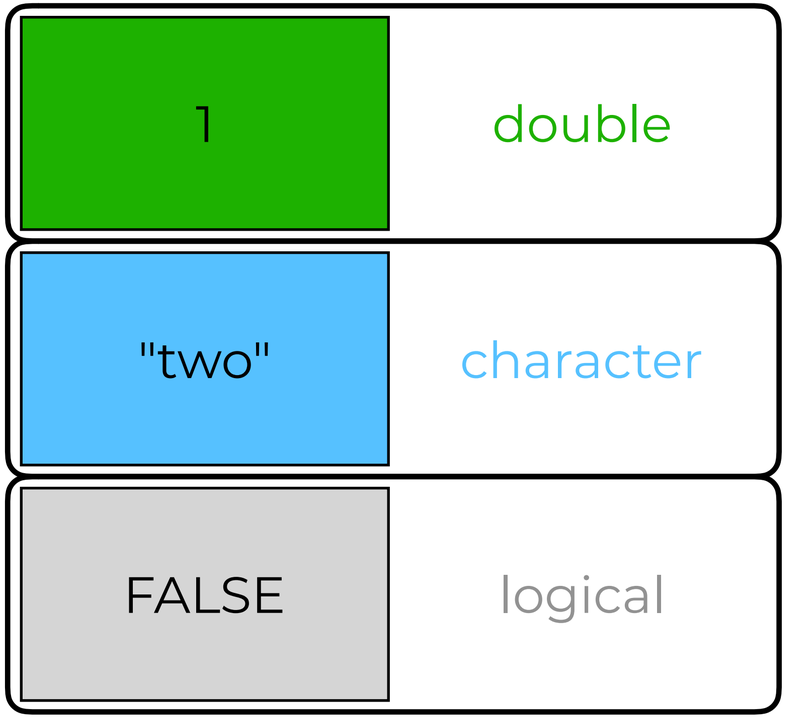
List

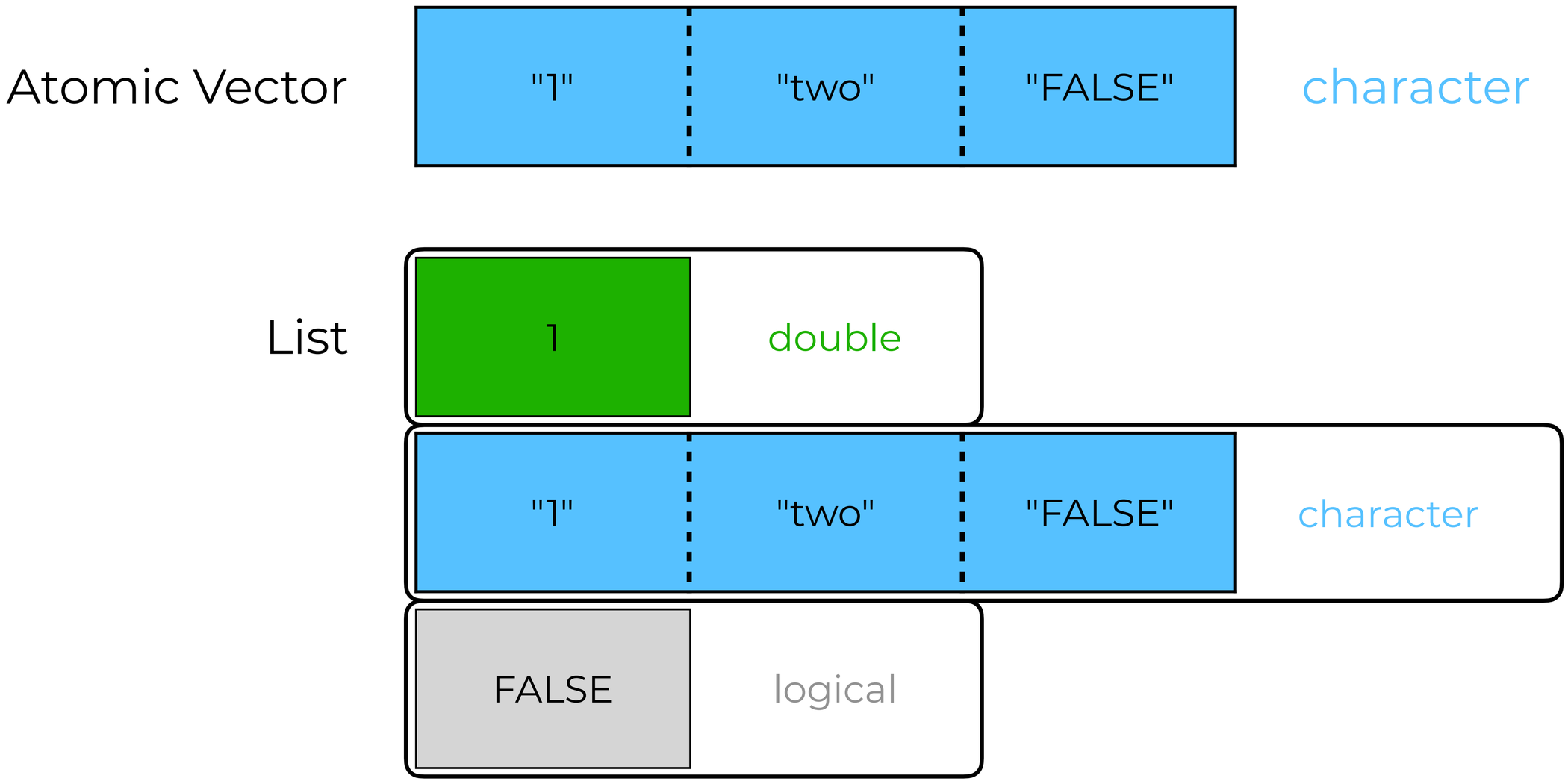
Atomic Vector



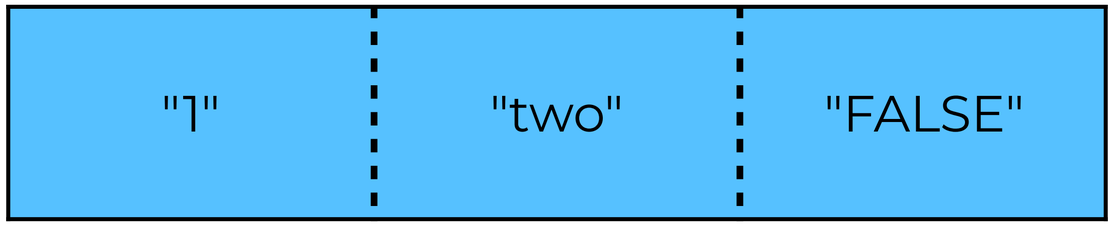
character

List



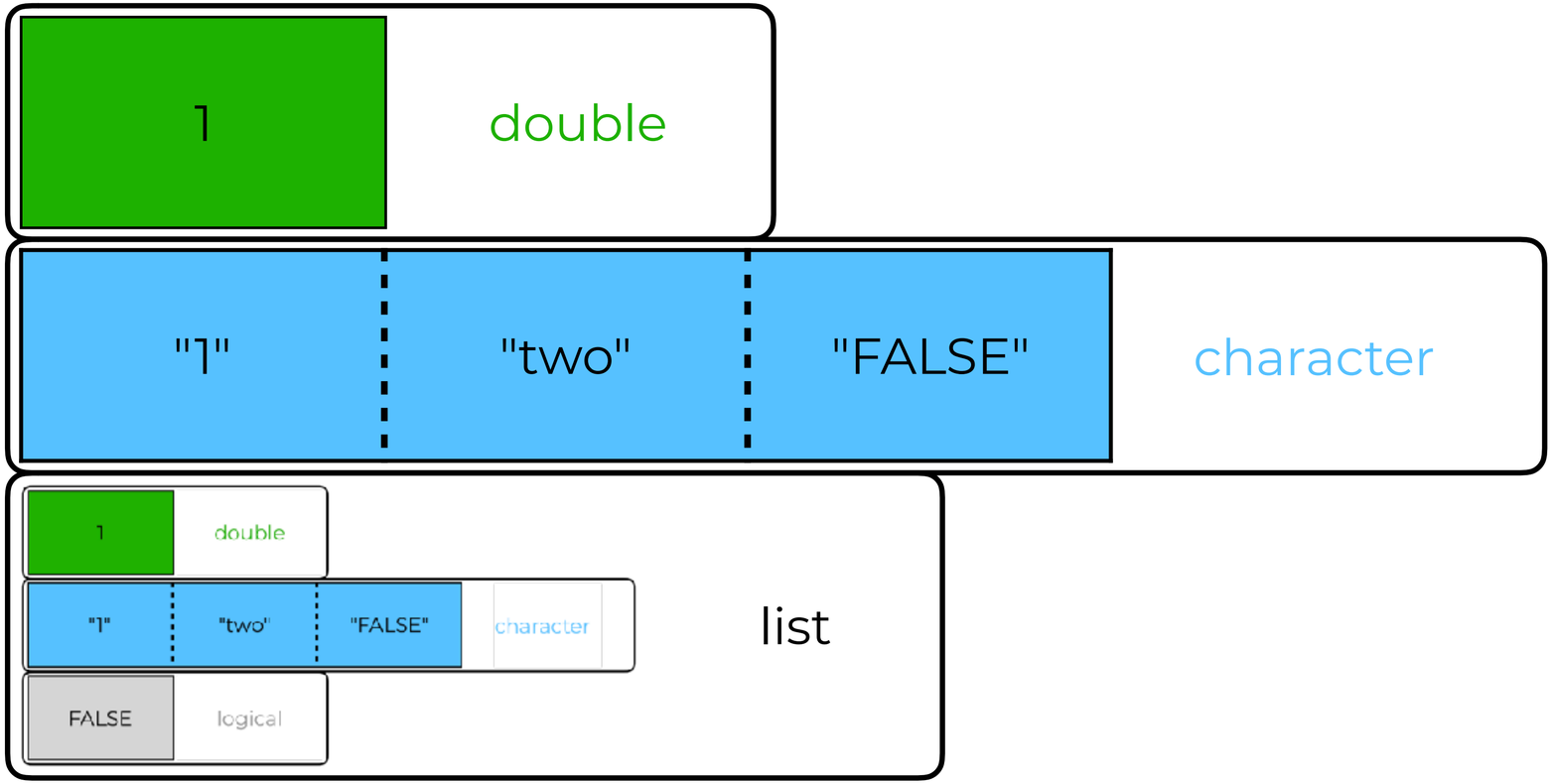


Atomic Vector



character

List



Your Turn 2

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
```

02 : 00

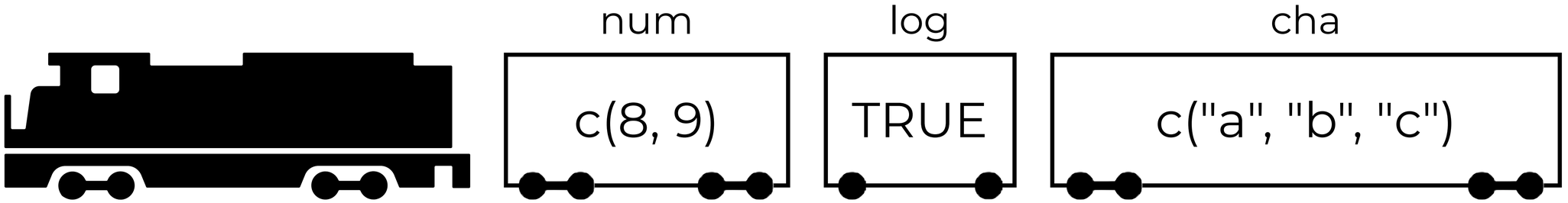
ATLAS

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

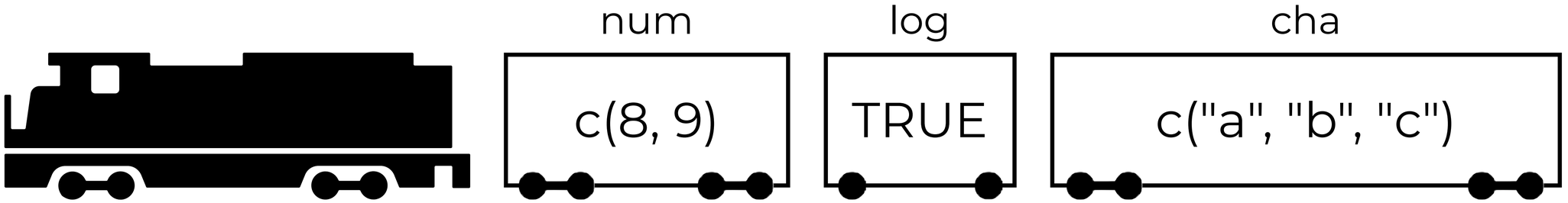
A list
(with one element named `num`
that contains an atomic vector

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

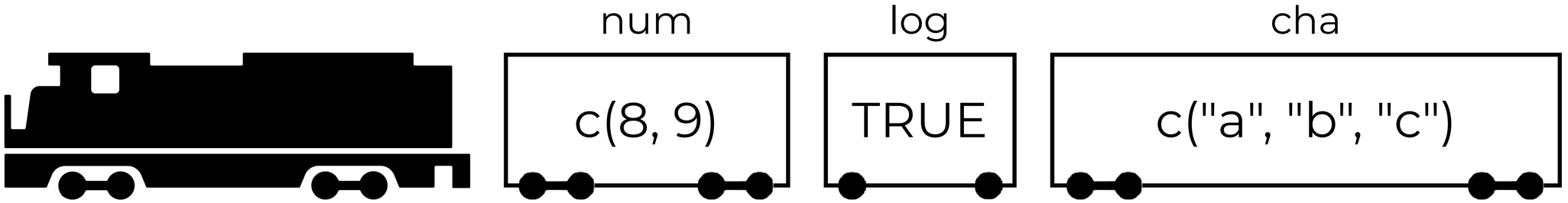
An atomic vector




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




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




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


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

c(8, 9)

```
lst$num
```

c(8, 9)

Your Turn 3

What will each of these return? Run the code chunks to confirm.

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

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

02 : 00

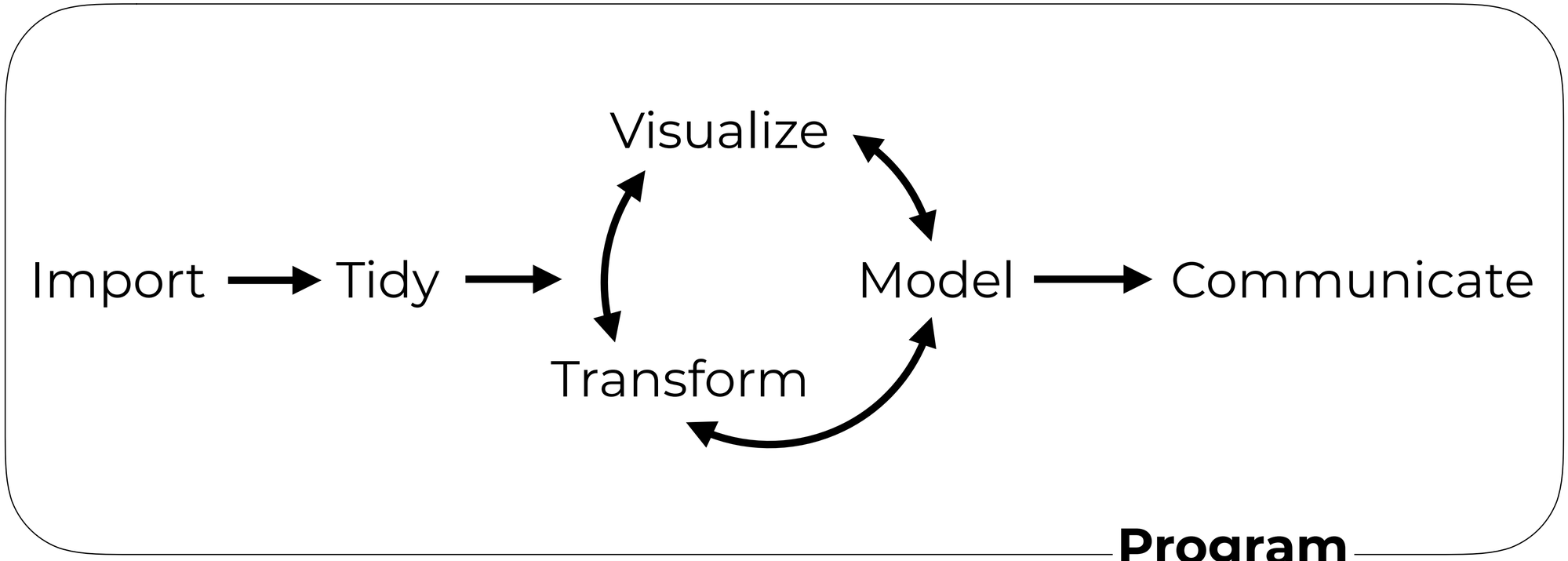
ATLAS

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

lst <- c(-2, -1, 0, 1, 2)
abs(lst)
Error in abs(lst) : non-numeric argument to mathematical
function
```

Take aways

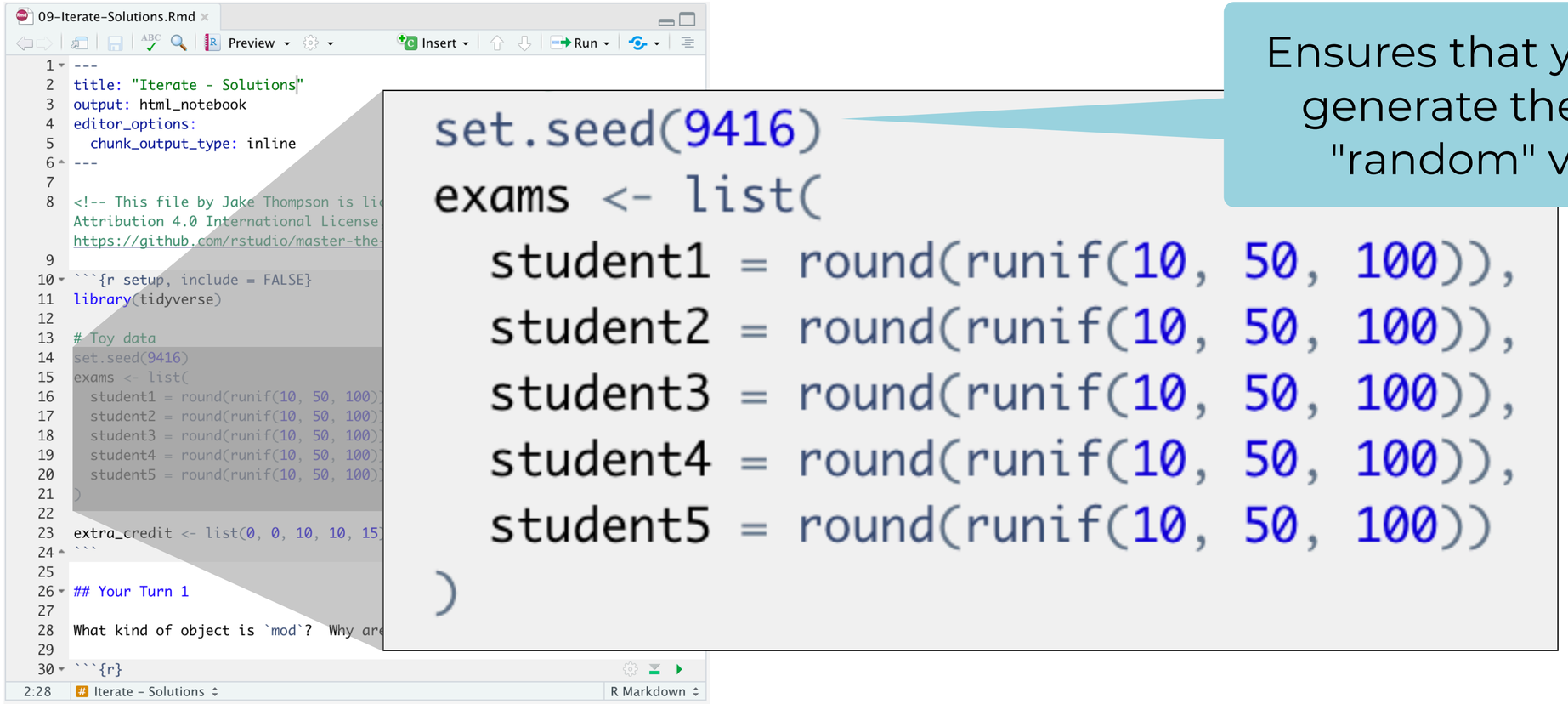
- Lists are a useful way to organize data
- But you need to arrange manually for functions to iterate over the elements of a list.

Iteration



Toy data for practice

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



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

ATLAS

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

```
exams
$student1
 [1] 61 88 79 64 62 79 81 91 78 76

$student2
 [1] 55 88 59 75 97 87 96 62 63 72

$student3
 [1] 52 85 97 76 80 63 76 95 82 71

$student4
 [1] 92 85 91 90 97 85 79 53 62 68

$student5
 [1] 82 97 97 67 66 69 76 82 61 87
```

How can we compute the mean grade for each student?

How could we compute the average grade?

```
means(exams)
[1] NA
Warning message:
In mean.default(exams) : argument is not numeric or logical: returning NA
```

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       $student2       $student3
[1] 75.9        [1] 75.4        [1] 77.7

$student4       $student5
[1] 80.2        [1] 78.4
```

Is there a better way?

In R4DS

Iteration

ATLAS

Your Turn 4

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

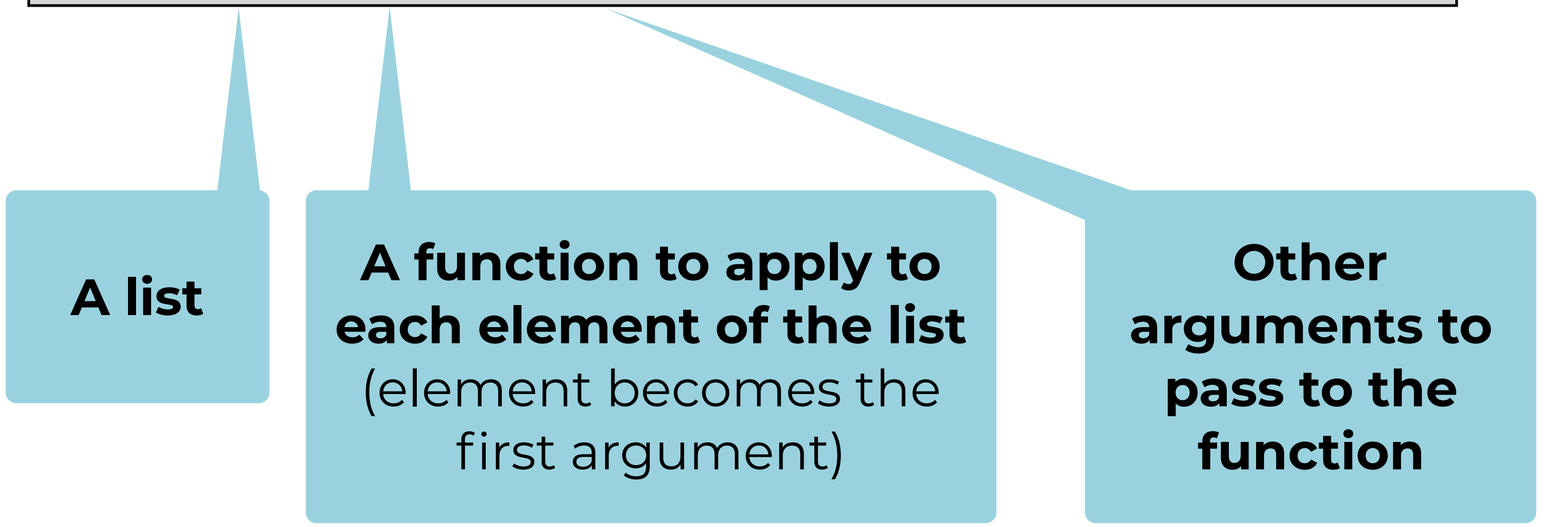
```
map(exams, mean)
```

```
exams %>% map(mean)
$student1
[1] 75.9

$student2
[1] 75.4

$student3
[1] 77.7

$student4
[1] 80.2

$student5
[1] 78.4
```

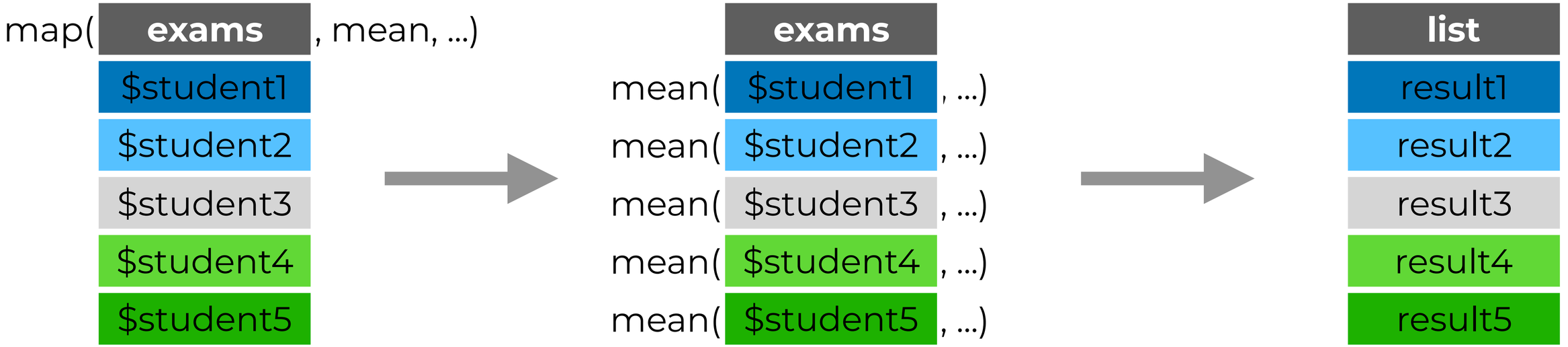

map()

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

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



map()



Your Turn 5

Calculate the variance (**var()**) of each student's exam grades.

02 : 00

```
exams %>% map(var)
$student1
[1] 108.9889

$student2
[1] 246.0444

$student3
[1] 186.2333

$student4
[1] 211.2889

$student5
[1] 163.6
```


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
    75.9     75.4     77.7     80.2     78.4
```


Extra arguments

What if the grade was the 90th percentile score?

```
exams %>% map_dbl(quantile, prob = 0.9)
student1 student2 student3 student4 student5
    88.3     96.1     95.2     92.5     97.0
```

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 6

Calculate the max grade (**max()**) for each student. Return the result as a vector.

```
exams %>% map_dbl(max)
student1 student2 student3 student4 student5
      91       97       97       97       97
```


Consider

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**?

Answer: Write a function

Functions

In R4DS

Functions

Functions (very basics)

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

```
vec <- exams$student 1
```

To write a function (very basics)

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

```
vec <- exams$student1
(sum(vec) - min(vec)) / (length(vec) - 1)
[1] 77.55556
```


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

```
vec <- exams$student1
  (sum(vec) - min(vec)) / (length(vec) - 1)
vec <- exams$student2
  (sum(vec) - min(vec)) / (length(vec) - 1)
vec <- exams$student3
  (sum(vec) - min(vec)) / (length(vec) - 1)
vec <- exams$student3
  (sum(vec) - min(vec)) / (length(vec) - 1)
vec <- exams$student5
  (sum(vec) - min(vec)) / (length(vec) - 1)
```

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

```
vec <- exams$student1
  (sum(vec) - min(vec)) / (length(vec) - 1)
vec <- exams$student2
  (sum(vec) - min(vec)) / (length(vec) - 1)
vec <- exams$student3
  (sum(vec) - min(vec)) / (length(vec) - 1)
vec <- exams$student3
  (sum(vec) - min(vec)) / (length(vec) - 1)
vec <- exams$student5
  (sum(vec) - min(vec)) / (length(vec) - 1)
```

"When you've written the same code three times, write a function."

–Hadley Wickham

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)
}
```

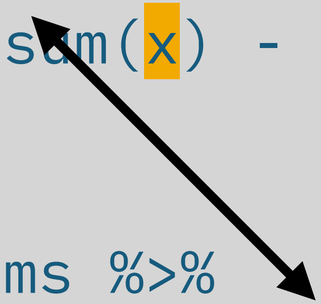

To write a function (very basics)

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 argument.

```
vec <- exams[[1]]
grade <- function(vec) {
  (sum(vec) - min(vec)) / (length(vec) - 1)
}
grade(exams[[2]])
[1] 77.66667
```


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

exams %>%
  map_dbl(grade)
student1 student2 student3 student4 student5
77.55556 77.66667 80.55556 83.22222 80.33333
```


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

exams %>%
  map_dbl(grade)
student1 student2 student3 student4 student5
77.55556 77.66667 80.55556 83.22222 80.33333
```



Your Turn 7

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

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

05 : 00

Define a new function, and pass in its name

```
double_best <- function(x) {
  (sum(x) + max(x)) / (length(x) + 1)
}

exams %>%
  map_dbl(double_best)
student1 student2 student3 student4 student5
77.27273 77.36364 79.45455 81.72727 80.09091
```


Use anonymous function

```
exams %>%
  map_dbl(function(x) (sum(x) + max(x)) / (length(x) + 1))
student1 student2 student3 student4 student5
77.27273 77.36364 79.45455 81.72727 80.09091
```


Use a purrr ~(formula) shortcut

```
exams %>%
  map_dbl(~ (sum(.x) + max(.x)) / (length(.x) + 1))
student1 student2 student3 student4 student5
77.27273 77.36364 79.45455 81.72727 80.09091
```

More on this approach at: https://github.com/cwickham/purrr-tutorial

Consider

What does this return?

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

```
#> 2
```


Consider

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()

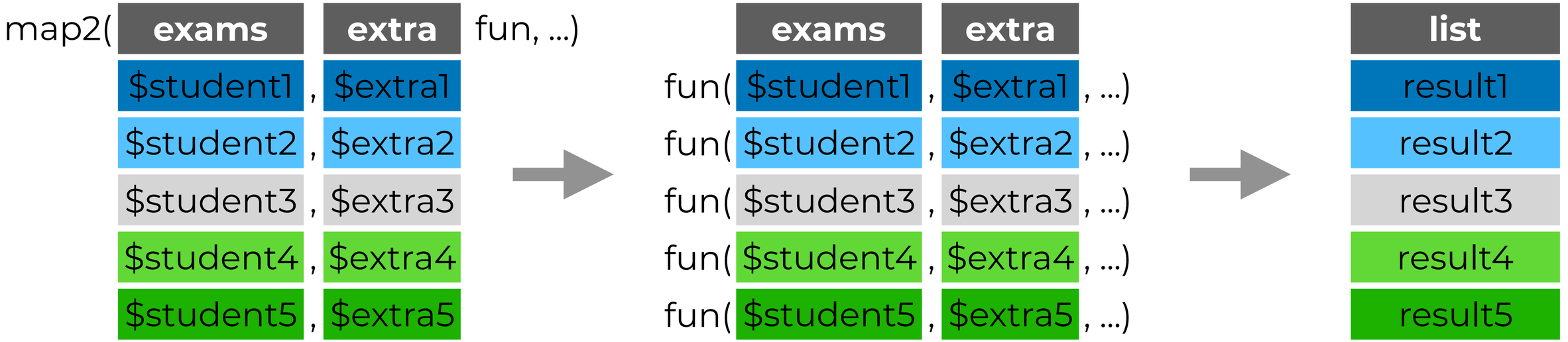
Applies a function to every element of two lists.
Returns the results as a list.

```
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()

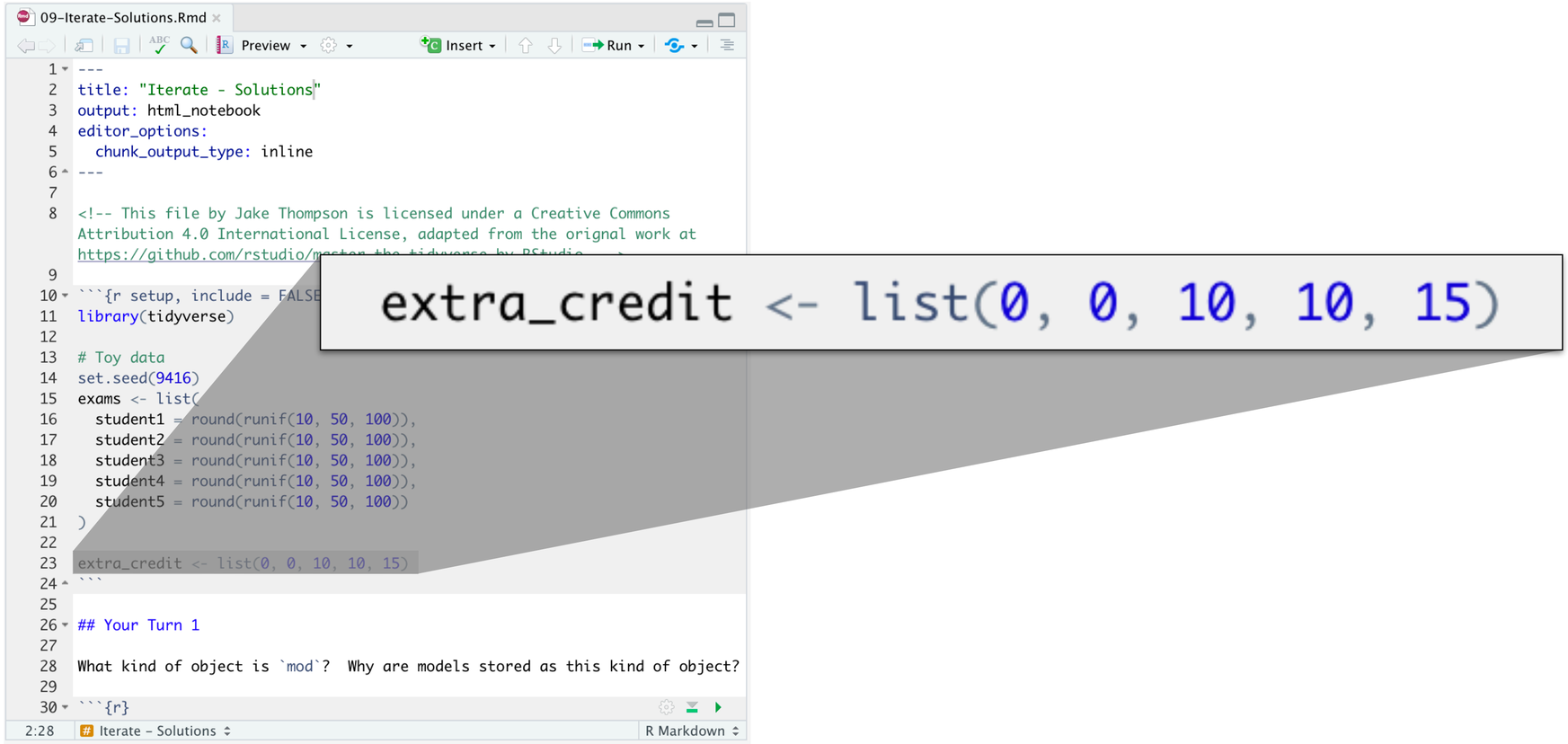


map functions

single list	two lists	returns results as
`map()`	`map2()`	list
`map_chr()`	`map2_chr()`	character vector
`map_dbl()`	`map2_dbl()`	double vector
`map_int()`	`map2_int()`	integer vector
`map_lgl()`	`map2_lgl()`	logical vector
`map_df()`	`map2_df()`	data frame

Toy data for practice

Suppose we have extra credit for the five students...



Your Turn 8

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.

05 : 00

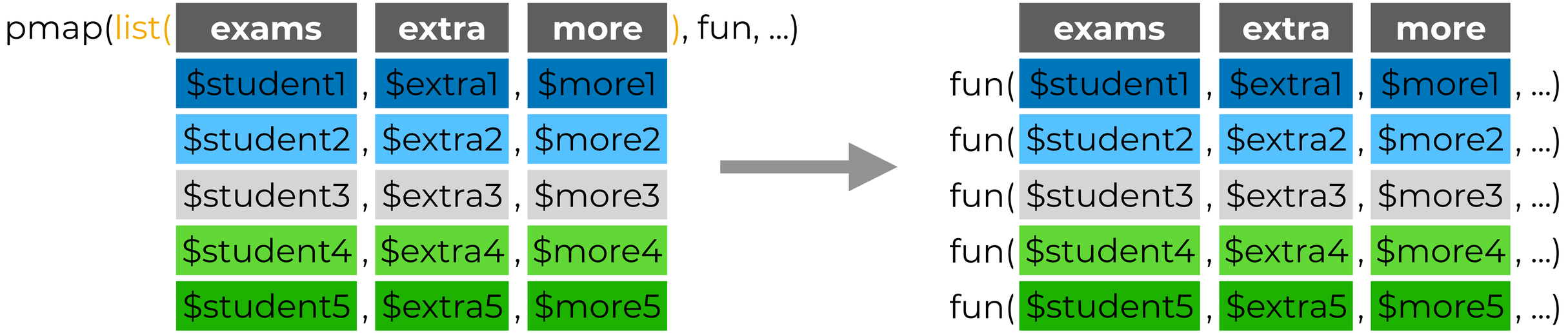
The grades with extra credit...

```
exams %>%
  map2_dbl(extra_credit, function(x, y) mean(x) + y)
student1 student2 student3 student4 student5
    75.9     75.4     87.7     90.2     93.4
```


Other mapping functions

pmap()

Map over three or more lists. Put the lists into a list of lists whose names match argument names in the function.



walk(), walk2(), and pwalk()

Versions of **map()**, **map2()**, and **pmap()** that do not return results. These are for triggering side effects (like writing files or saving graphs).

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 (numeric)
`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

Iterate

ATLAS
Accessible Teaching, Learning, and Assessment Systems

wjakethompson.com wjakethompson@ku.edu @wjakethompson @wjakethompson