Welcome!

An overview of the course

Daniel Anderson

Week 1, Class 1

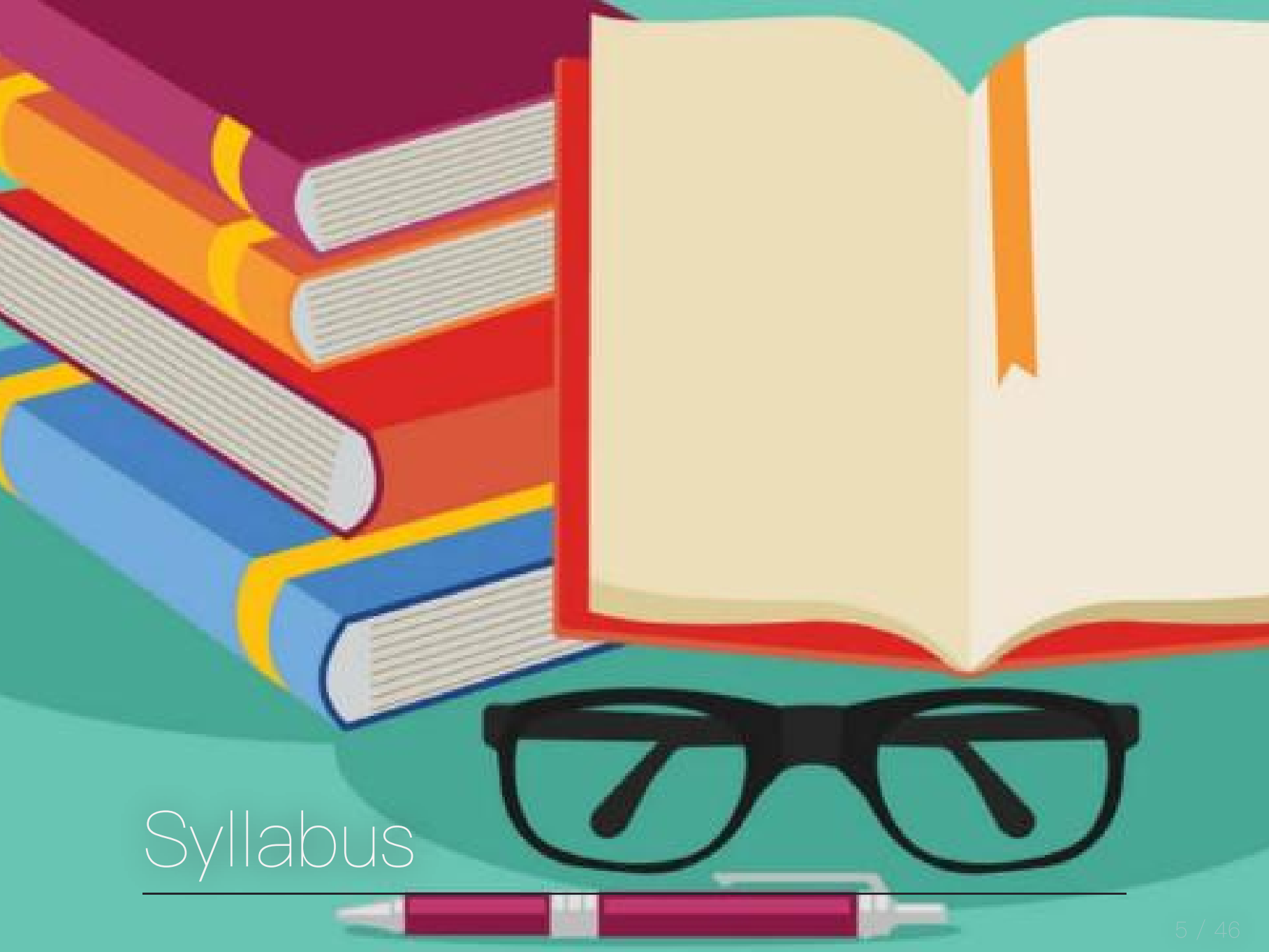
Agenda

- Getting on the same page
- Syllabus
- Intro to data types

Getting on the same page

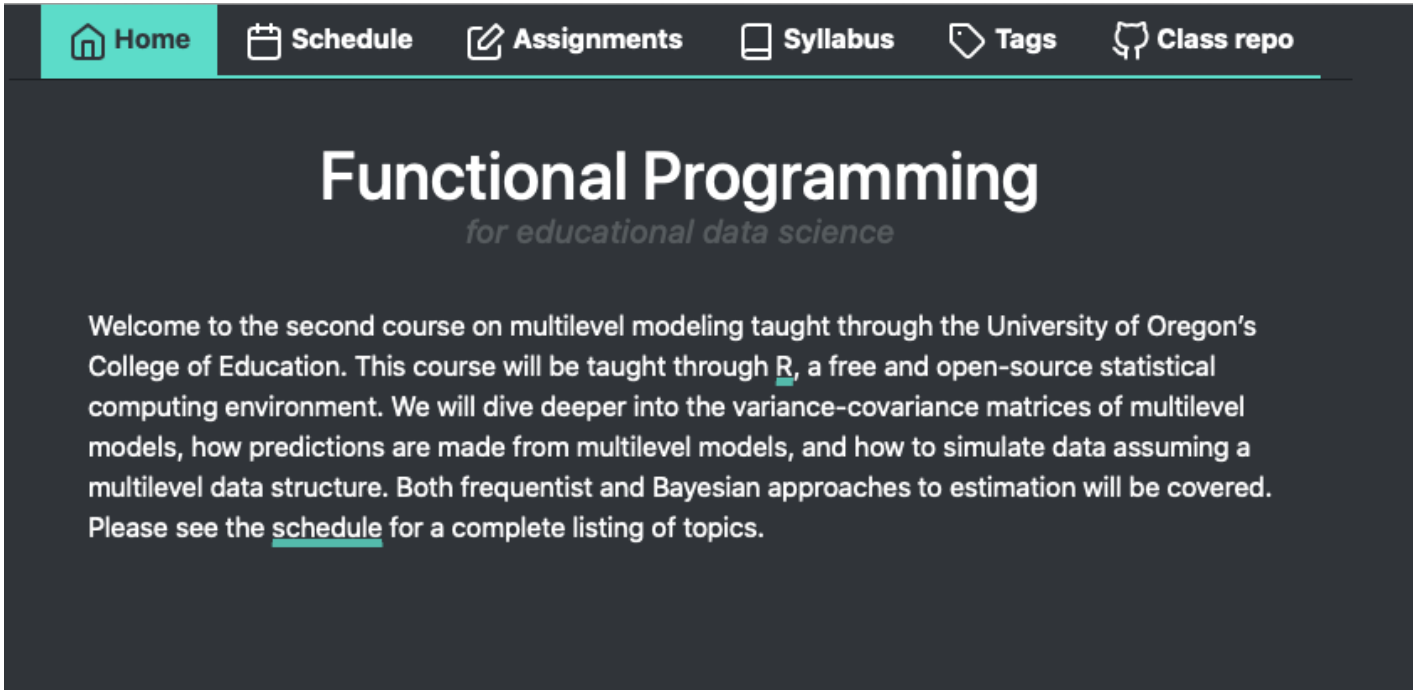
Introduce yourself!

- Most of us know each other, but a few do not.
- Tell us why you're taking the class
- What's one fun thing you've done recently outside of school stuff?



Syllabus

Course Website(s)



Course learning objectives

- Understand and be able to describe the differences in R's data structures and when each is most appropriate for a given task
- Explore `purrr::map` and its variants, how they relate to base R functions, and why the {purrr} variants are often preferable.
- Work with lists and list columns using `purrr::nest` and `purrr:unnest`
- Understand how the new `dplyr::rowwise()` can help you avoid some of the above

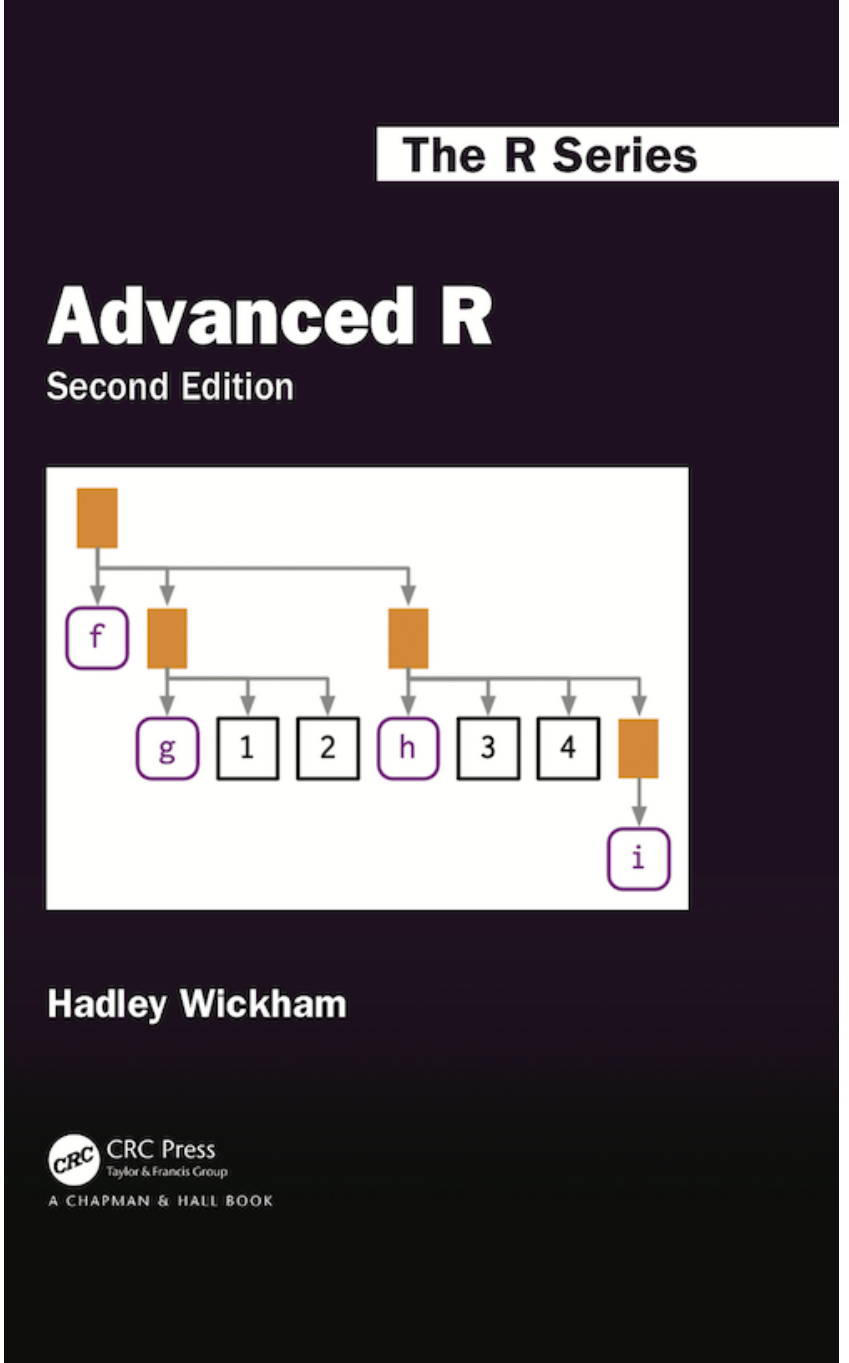
Course learning objectives

- Convert repetitive tasks into functions
- Understand elements of good functions, and things to avoid
- Write effective and clear functions with the mantra of "Don't Repeat Yourself"

This Week's learning objectives

- Understand the requirements of the course
- Understand the requirements of the final project

Textbooks

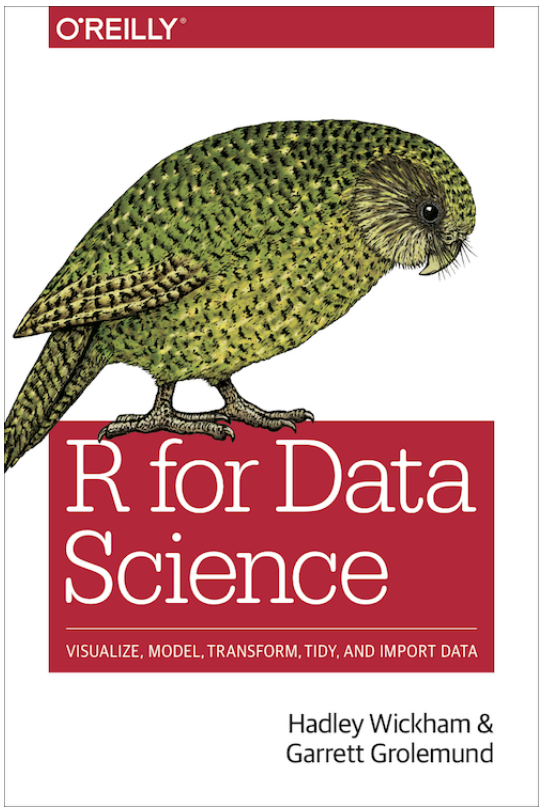

The R Series
Advanced R
Second Edition
f
g
1
2
h
3
4
i
Hadley Wickham
CRC Press
Taylor & Francis Group
A CHAPMAN & HALL BOOK

Link

Other books (also free)

Bryan

Wickham & Grolemund



Structure of the course

- First 5 weeks – mostly iteration
    - Data types
    - Base R iterations
    - {purrr}
    - Batch processes and working with list columns
    - Parallel iterations (and a few extras)
- Second 5 weeks – Writing functions and shiny
    - Writing functions 1–3
    - Shiny 1–3
    - Packages (briefly)

Labs

15%

3 @ 10 points each

Two labs on iteration and one on functions

Lab	Date Assigned	Date Due	Topic
1	Wed, April 07	Wed, April 14	Subsetting lists and base R `for()` loops
2	Wed, April 14	Wed, April 21	Multiple models and API calls with **{purrr}**
3	Mon, May 10	Mon, May 17	Create and apply functions

Midterm

70 points total (35%)

Two parts:

- Small quiz on canvas to demonstrate knowledge (4/21; 10 points)
    - Identifying bugs in code
    - Multiple choice/fill–in the blank questions
    - Free response
- Take–home portion to demonstrate ability to write the correct code (assigned 4/21; 60 points)
    - Write loops to solve problems

Take home portion

- Group project: 3–5 people
- Shared GitHub repo
- Fairly long – but the only homework assignment you have
- Divide it up to ease the workload, then just check each other's work

Already posted!

Final Project

150 points total (60%)

5 parts

Component	Points	Due
Groups Finalized	0	4/07/21
Outline	5	4/21/21
Draft	10	5/19/21
Peer review	15	5/26/21
Product	70	6/09/21 (11:59 pm)

What is it?

Two basic options

Data Product

Similar to first class

- Brief research manuscript (can be APA or not, I don't really care 🤷)
- Shiny app
- Dashboard
    - Probably unlikely to work well though, unless you make it a shiny dashboard
- Blog post
- For the ambitious – a documented R package

Tutorial

- Probably best done through a blog post or series of blog posts
- Approach as if you're teaching others about the content I'll ask you to cover
- BONUS: You can actually release the blog post(s) and may get some traffic 🎉👏🥳

Make it your own

What you have to have

- Everything on GitHub
- Publicly available dataset
- Team of 2–4

What you have to cover

Unfortunately, this still is a class assignment. I have to be able to evaluate that you can actually apply the content within a messy, real–world setting.

The grading criteria (which follow) may force you into some use cases that are a bit artificial. This is okay.

Grading criteria

- No code is used repetitively (no more than twice) 10 points
- More than one variant of `purrr::map` is used 5 points
- At least one {purrr} function outside the basic `map` family (`walk_*`, `reduce`, `modify_*`, etc.) 5 points
- At least one instance of parallel iteration (e.g., `map2_*`, `pmap_*`) 5 points
- At least one use case of `purrr::nest %>% mutate()` 5 points

Grading criteria

- At least two custom functions 20 points; 10 points each
    - Each function must do exactly one thing
    - The functions **may** replicate the behavior of a base function – as noted above this is about practicing the skills you learn in class
- Code is fully reproducible and housed on GitHub 10 points
- No obvious errors in chosen output format 5 points
- Deployed on the web and shareable through a link 5 points

Outline

Due 4/21/21

Four components:

- Description of data source (must be publicly available)
- Purpose (tutorial or substantive)
- Chosen format
- Lingering questions
    - How can I help?

Please include all components – including the question(s) section!

Draft

Due 5/19/21, before class

- Expected to still be a work in progress
    - This means some of your code may be rough and/or incomplete. However:
- Direction should be obvious
- Most, if not all, grading elements should be present
- Provided to your peers so they can learn from you as much as you can learn from their feedback

Peer Review

- Exact same process we've used before
- If, during your peer review, you find grading elements not present, definitely note them

Utilizing GitHub (required)

- You'll be assigned two groups to review
- Fork their repo
- Embed comments, suggest changes to their code
    - Please do both of these
- Submit a PR
    - Summarize your overall review in the PR

Grading

200 points total

- 3 labs at 10 points each (30 points; 15%)
- Midterm in–class (10 points; 5%)
- Midterm take–home (60 points; 30%)
- Final Project (100 points; 50%)
    - Outline (5 points; 2.5%)
    - Draft (10 points; 5%)
    - Peer review (15 points; 7.5%)
    - Product (70 points; 35%)

Grading

Lower percent	Lower point range	Grade	Upper point range	Upper percent
0.97	(194 pts)	A+		
0.93	(186 pts)	A	(194 pts)	0.97
0.90	(180 pts)	A–	(186 pts)	0.93
0.87	(174 pts)	B+	(180 pts)	0.90
0.83	(166 pts)	B	(174 pts)	0.87
0.80	(160 pts)	B–	(166 pts)	0.83
0.77	(154 pts)	C+	(160 pts)	0.80
0.73	(146 pts)	C	(154 pts)	0.77
0.70	(140 pts)	C–	(146 pts)	0.73
		F	(140 pts)	0.70


COMMUNICATION

Any time left?

Basic data types

Vectors

Pop quiz

Discuss with your neighbor

- What are the four basic types of atomic vectors?
- What function creates a vector?
- **T**/**F**: A list (an R list) is not a vector.
- What is the fundamental difference between a matrix and a data frame?
- What does *coercion* mean, and when do
play?

05:00

Vector types

4 basic types

Note there are two others (complex and raw), but we don't care about them (I've never even seen them used).

- Integer
- Double
- Logical
- Character

Integer and double vectors are both numeric.

Creating vectors

Vectors are created with `c`. Below are examples of each of the four main types of vectors.

```
# L explicitly an integer, not double
integer <- c(5L, 7L, 3L, 94L)
double <- c(3.27, 8.41, Inf, -Inf)
logical <- c(TRUE, TRUE, TRUE, FALSE, TRUE, FALSE, FALSE)
character <- c("red", "orange", "yellow", "green", "blue",
               "violet", "rainbow")
```

Coercion

- Vectors **must** be of the same type.
- If you try to mix types, implicit coercion will occur
- Implicit coercion defaults to the most flexible type
    - which is... ?

```
c(7L, 3.25)
```

```
## [1] 7.00 3.25
```

```
c(3.24, TRUE, "April")
```

```
## [1] "3.24"  "TRUE"  "April"
```

```
c(TRUE, 5)
```

```
## [1] 1 5
```

Explicit coercion

- You can alternatively define the coercion to occur

```
as.integer(c(7L, 3.25))
```

```
## [1] 7 3
```

```
as.logical(c(3.24, TRUE, "April"))
```

```
## [1]   NA TRUE   NA
```

```
as.character(c(TRUE, 5)) # still maybe a bit unexpected?
```

```
## [1] "1" "5"
```

Checking types

- Use `typeof` to verify the type of vector

```
typeof(c(7L, 3.25))
```

```
## [1] "double"
```

```
typeof(as.integer(c(7L, 3.25)))
```

```
## [1] "integer"
```

Piping

- Although traditionally used within the tidyverse (not what we're doing here), it can still be useful. The following are equivalent

```
library(magrittr)

typeof(as.integer(c(7L, 3.25)))
```

```
## [1] "integer"
```

```
c(7L, 3.25) %>%
  as.integer() %>%
  typeof()
```

```
## [1] "integer"
```

Pop quiz

Without actually running the code, predict which type each of the following will coerce to.

```
c(1.25, TRUE, 4L)
c(1L, FALSE)
c(7L, 6.23, "eight")
c(TRUE, 1L, 0L, "False")
```

01:00

Answers

```
typeof(c(1.25, TRUE, 4L))
```

```
## [1] "double"
```

```
typeof(c(1L, FALSE))
```

```
## [1] "integer"
```

```
typeof(c(7L, 6.23, "eight"))
```

```
## [1] "character"
```

```
typeof(c(TRUE, 1L, 0L, "False"))
```

```
## [1] "character"
```

Lists

- Lists are vectors, but not *atomic* vectors
- Fundamental difference – each element can be a different type

```
list("a", 7L, 3.25, TRUE)
```

```
## [[1]]
## [1] "a"
##
## [[2]]
## [1] 7
##
## [[3]]
## [1] 3.25
##
## [[4]]
## [1] TRUE
```

Lists

- Each element of the list is another vector, possibly atomic, possibly not
- The prior example included all *scalar* vectors
- Lists do not require all elements to be the same length

```
list(
  c("a", "b", "c"),
  rnorm(5),
  c(7L, 2L),
  c(TRUE, TRUE, FALSE, TRUE)
)
```

```
## [[1]]
## [1] "a" "b" "c"
##
## [[2]]
## [1] -1.4730979 -1.1787801 -0.8040743
##
## [[3]]
## [1] 7 2
##
## [[4]]
## [1]  TRUE  TRUE FALSE  TRUE
```

Summary

- Atomic vectors must all be the same type
    - implicit coercion occurs if not (and you haven't specified the coercion explicitly)
- Lists are also vectors, but not atomic vectors
    - Each element can be of a different type and length
    - Incredibly flexible, but often a little more difficult to get the hang of

Next time

- More on data types
  - Missing values
  - Subsetting
  - Attributes
  - More on coercion (good to be fluent)
- Lists

Any lingering questions?