

Getting started with R

Back to Basics



Steffi LaZerte <https://steffilazerte.ca>

Online workshops can be challenging

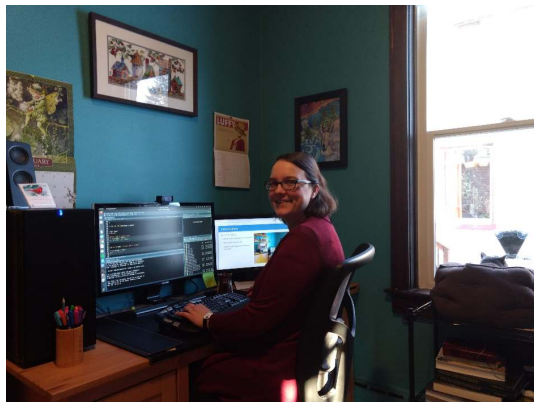
- **Keep your video on** (if possible)
 - We're here together!
 - Kids? Pets? Spouses? No problem!
- **Interrupt me!**
 - Generally keep yourself muted but un-mute anytime to ask questions
- **Ask Questions!**
 - Group trouble-shooting is really valuable
 - If you have a problem, others may also (or may have it in the future)
- **Screen-sharing**
 - I may ask you to share your screen with the group
 - For privacy, close your email etc. Or just share your RStudio window

2 / 49

Introductions

Dr. Steffi LaZerte

- Background in Biology (Animal Behaviour)
- Working with R since 2007
- Professional R programmer/consultant since 2017
- Third year giving BU R Workshop!



3 / 49

What about you?

- Name
- Pets? (share on camera!)
- Background (Student/Faculty/Staff, Area of study, etc.)
- Familiarity with Computer Programming (C+, Java, HTML, PHP, python, SAS)
- Familiarity with R
 - I've heard of R
 - I've installed R (before this class)
 - I've used R
 - I've used R a lot
 - I use R all the time

4 / 49

About this Workshop

Format

- I will provide you tools and workflow to get started with R
- We'll have hands-on, lecture, and demonstrations

R is hard: But have no fear!

- Don't expect to remember everything!
- Copy/Paste is your friend (never apologize for using it!)
- Consider this workshop a resource to return to

5 / 49

About this Workshop

Format

- I will provide you tools and workflow to get started with R
- We'll have hands-on, lecture, and demonstrations

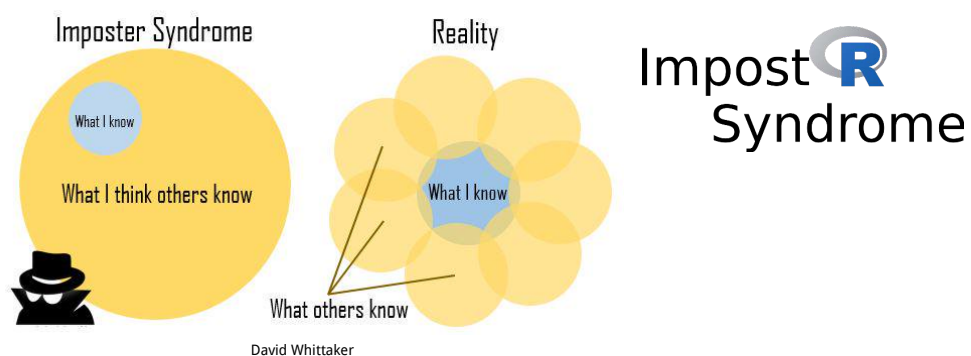
R is hard: But have no fear!

- **Don't expect to remember everything!**
- Copy/Paste is your friend (never apologize for using it!)
- Consider this workshop a resource to return to

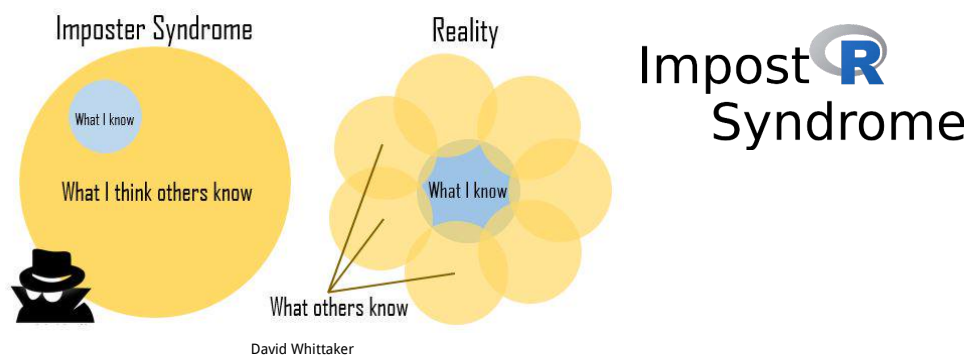
6 / 49

ImpostR Syndrome

7 / 49



8 / 49



Moral of the story?

Make friends, code in groups, learn together and don't beat yourself up

8 / 49



Artwork by [@allison_horst](#)

9 / 49

All about R

Why R?

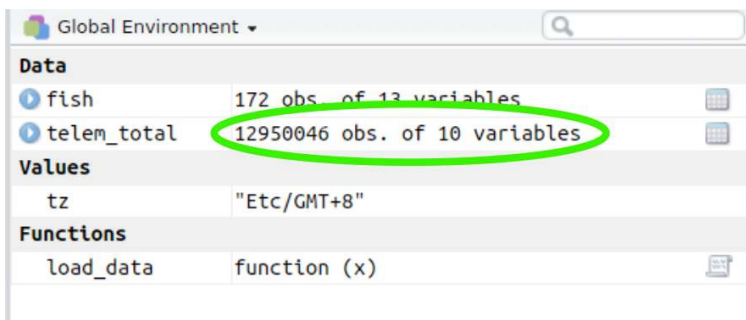
R is hard

```
# Get in circle around city
circle <- data.frame()
cutoff <- 10
for(i in unique(gps$region)) {
  n <- nrow(gps[gps$region == i,]) ##number of IDs
  if(i == "wil") tmp <- geocode("Williams Lake, Canada")
  if(i == "kam") tmp <- geocode("Kamloops, Canada")
  if(i == "kel") tmp <- geocode("Kelowna, Canada")
  temp <- data.frame()
  for(a in 1:n){
    if(a <= cutoff) temp <- rbind(temp, gcDestination(lon = tmp$lon,
                                                       lat = tmp$lat,
                                                       bearing = (a*(360/(cutoff))-360/(cutoff)),
                                                       dist = 20,
                                                       dist.units = "km",
                                                       model = "WGS84"))
    if(a > cutoff) temp <- rbind(temp, gcDestination(lon = tmp$lon,
                                                       lat = tmp$lat,
                                                       bearing = ((a-cutoff)*(360/(max(table(gps$region
                                                       ))-10))-360/(max(table(gps$region))-cutoff)),
                                                       dist = 35,
                                                       dist.units = "km",
                                                       model = "WGS84"))
  }
  circle <- rbind(circle, cbind(temp,
                                region = i,
                                hab = gps$hab[gps$region == i],
                                spl = gps$spl.orig[gps$region == i],
```

11 / 49

Why R?

But R is powerful (and reproducible)!

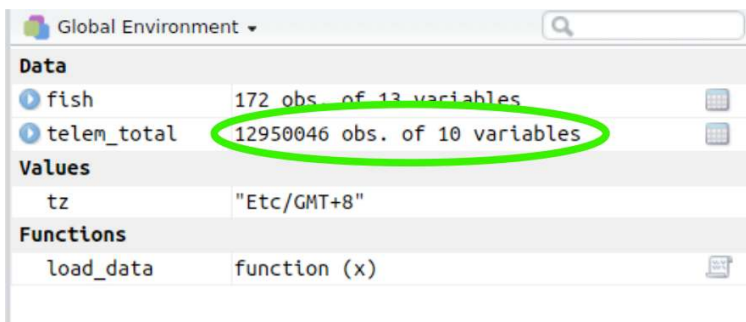


Global Environment	
Data	
fish	172 obs. of 13 variables
telem_total	12950046 obs. of 10 variables
Values	
tz	"Etc/GMT+8"
Functions	
load_data	function (x)

12 / 49

Why R?

But R is powerful (and reproducible)!



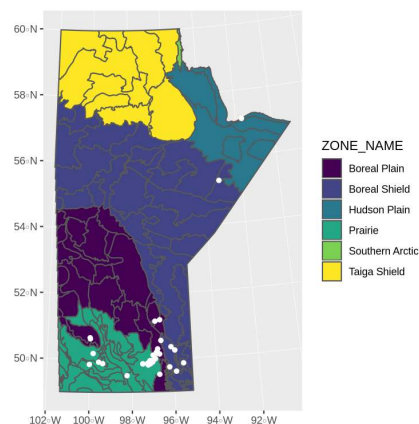
Global Environment	
Data	
fish	172 obs. of 13 variables
telem_total	12950046 obs. of 10 variables
Values	
tz	"Etc/GMT+8"
Functions	
load_data	function (x)

(I made these slides with Rmarkdown)

12 / 49

Why R?

R is also beautiful



13 / 49

Why R?

R is affordable (i.e., free!)

R is available as Free Software under the terms of the [Free Software Foundation's GNU General Public License](#) in source code form. It compiles and runs on a wide variety of UNIX platforms and similar systems (including FreeBSD and Linux), Windows and MacOS.

14 / 49

What is R?

R is Programming language

A programming **language** is a way to give instructions in order to get a computer to do something

- You need to know the language (i.e., the code)
- Computers don't know what you mean, only what you type (unfortunately)
- Spelling, punctuation, and capitalization all matter!

For example

R, what is 56 times 5.8?

```
56 * 5.8
```

```
## [1] 324.8
```

16 / 49

Use code to tell R what to do

R, what is the average of numbers 1, 2, 3, 4?

```
mean(c(1, 2, 3, 4))
```

```
## [1] 2.5
```

17 / 49

Use code to tell R what to do

R, what is the average of numbers 1, 2, 3, 4?

```
mean(c(1, 2, 3, 4))
```

```
## [1] 2.5
```

R, save this value for later

```
steffis_mean <- mean(c(1, 2, 3, 4))
```

17 / 49

Use code to tell R what to do

R, what is the average of numbers 1, 2, 3, 4?

```
mean(c(1, 2, 3, 4))
```

```
## [1] 2.5
```

R, save this value for later

```
steffis_mean <- mean(c(1, 2, 3, 4))
```

R, multiply this value by 6

```
steffis_mean * 6
```

```
## [1] 15
```

17 / 49

Code, Output, Scripts

Code

- The actual commands

Output

- The result of running code or a script

Script

- A text file full of code that you want to run
- You should always keep your code in a script

18 / 49

Code, Output, Scripts

Code

- The actual commands

For example:

```
mean(c(1, 2, 3, 4))
```

Code

Output

- The result of running code or a script

```
## [1] 2.5
```

Output

Script

- A text file full of code that you want to run
- You should always keep your code in a script

```
# myscript.R
15 # Setup
16 # RStudio setup
17 library(tidyverse)
18 library(stringr)
19 library(ggfortify)
20 library(grid)
21 library(boot)
22
23 theme_cust <- theme_bw() +
24   theme(panel.grid = element_blank())
25
26 # Load data
27 d <- read_csv("../Data/datasets/pca.csv") %>%
28   mutate(hab_c = ifelse(hab == 0, "urban", "rural"))
29
30 summary(d$hab)
31
32 # Plotting
33 d_sml <- d %>%
34   group_by(hab_c) %>%
35   summarise(prop = sum(atypical_c) / length(atypical_c))
36
37 d_n <- count(d, atypical_c, hab_c)
38
39 # Sample sizes
40 # RStudio sample_size
41 count(d, hab_c)
42 count(d, atypical_c)
43 count(d, timeup, monthname, freq_sweep)
44
45 count(d, region)
46 count(d, project = ifelse(is.na(project), "No[NC]([0-9][2?]", "Staff1",
47   checkmate = )
```

18 / 49

RStudio vs. R



RStudio



R

- **RStudio** is not **R**
- RStudio is a User Interface or IDE (integrated development environment)
 - (i.e., Makes coding simpler)
- But sometimes tries to be **too** helpful

19 / 49

RStudio Features

Changing Options: Tools > Global Options

- General > Restore RData into workspace at startup (NO!)
- General > Save workspace to on exit (NEVER!)
- Code > Insert matching parens/quotes (Personal preference)

Projects

- Handles working directories
- Organizes your work

Packages

- Can use the package manager to install packages
- Can use the manager to load them as well, but not recommended

20 / 49

Let's take a look at RStudio

Your first *real* code!

First Code

```
# First load the packages
library(tidyverse)
library(palmerpenguins)

# Now create the figure
ggplot(data = penguins, aes(x = body_mass_g, y = flipper_length_mm, colour = species)) +
  geom_point()
```

- Copy/paste or type this into the script window in RStudio
 - You may have to go to File > New File > R Script
- Click anywhere on the first line of code
- Use the 'Run' button to run this code, **or** use the short-cut **Ctrl-Enter**
 - Repeat until all the code has run

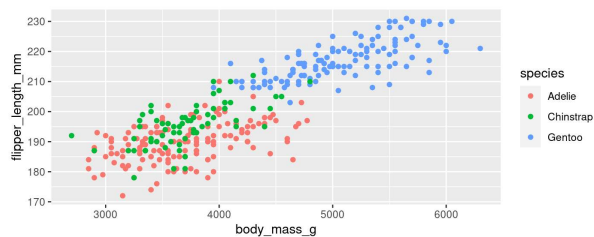
23 / 49

First Code

```
# First load the packages
library(tidyverse)
library(palmerpenguins)

# Now create the figure
ggplot(data = penguins, aes(x = body_mass_g, y = flipper_length_mm, colour = species)) +
  geom_point()
```

```
## Warning: Removed 2 rows containing missing values (geom_point).
```



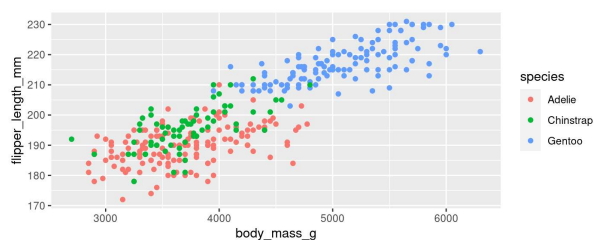
24 / 49

First Code

```
# First load the packages
library(tidyverse)
library(palmerpenguins)

# Now create the figure
ggplot(data = penguins, aes(x = body_mass_g, y = flipper_length_mm, colour = species)) +
  geom_point()
```

```
## Warning: Removed 2 rows containing missing values (geom_point).
```



25 / 49

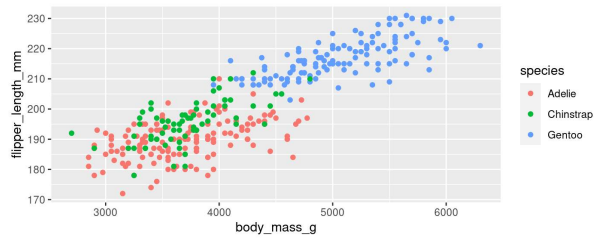
First Code

```
# First load the packages
library(tidyverse)
library(palmerpenguins)

# Now create the figure
ggplot(data = penguins, aes(x = body_mass_g, y = flipper_length_mm, colour = species)) +
  geom_point()

## Warning: Removed 2 rows containing missing values (geom_point).
```

Functions:
library(), **ggplot()**
aes(), and **geom_point()**



26 / 49

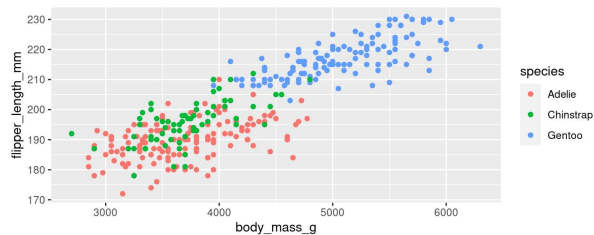
First Code

```
# First load the packages
library(tidyverse)
library(palmerpenguins)

# Now create the figure
ggplot(data = penguins, aes(x = body_mass_g, y = flipper_length_mm, colour = species)) +
  geom_point()

## Warning: Removed 2 rows containing missing values (geom_point).
```

+
(Specific to **ggplot**)



27 / 49

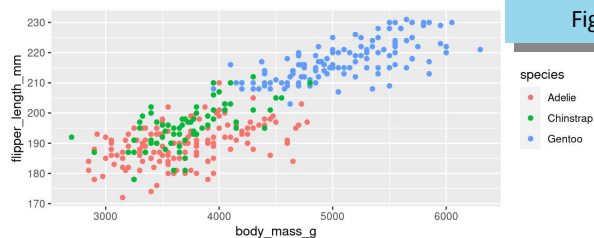
First Code

```
# First load the packages
library(tidyverse)
library(palmerpenguins)

# Now create the figure
ggplot(data = penguins, aes(x = body_mass_g, y = flipper_length_mm, colour = species)) +
  geom_point()

## Warning: Removed 2 rows containing missing values (geom_point).
```

Figure!



28 / 49

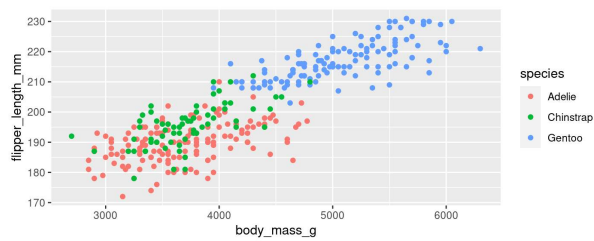
First Code

```
# First load the packages
library(tidyverse)
library(palmerpenguins)

# Now create the figure
ggplot(data = penguins, aes(x = body_mass_g, y = flipper_length_mm, colour = species)) +
  geom_point()
```

```
## Warning: Removed 2 rows containing missing values (geom_point).
```

Warning



29 / 49

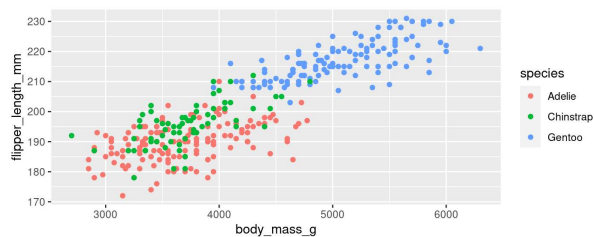
First Code

```
# First load the packages
library(tidyverse)
library(palmerpenguins)

# Now create the figure
ggplot(data = penguins, aes(x = body_mass_g, y = flipper_length_mm, colour = species)) +
  geom_point()
```

Comments
(Start with #)

```
## Warning: Removed 2 rows containing missing values (geom_point).
```



30 / 49

R Basics: Objects

Objects are *things* in the environment
(Check out the **Environment** pane in RStudio)

functions()

Do things, Return things

Does something but returns nothing

e.g., `write_csv()` - Saves the `mtcars` data frame as a csv file

```
write_csv(mtcars, path = "mtcars.csv")
```

Does something and returns something

e.g., `sd()` - returns the standard deviation of a vector

```
sd(c(4, 10, 21, 55))
```

```
## [1] 22.78157
```

32 / 49

functions()

- Functions can take **arguments** (think 'options')
- **data, x, y, colour**

```
ggplot(data = msleep, aes(x = sleep_total, y = sleep_rem, colour = vore)) +  
  geom_point()
```

33 / 49

functions()

- Functions can take **arguments** (think 'options')
- **data, x, y, colour**

```
ggplot(data = msleep, aes(x = sleep_total, y = sleep_rem, colour = vore)) +  
  geom_point()
```

- Arguments defined by **name** or by **position**
- With correct position, do not need to specify by name

By name:

```
mean(x = c(1, 5, 10))
```

```
## [1] 5.333333
```

By order:

```
mean(c(1, 5, 10))
```

```
## [1] 5.333333
```

33 / 49

functions()

Watch out for 'hidden' arguments

By name:

```
mean(x = c(1, 5, 10, NA),  
      na.rm = TRUE)
```

```
## [1] 5.333333
```

34 / 49

functions()

Watch out for 'hidden' arguments

By name:

```
mean(x = c(1, 5, 10, NA),  
      na.rm = TRUE)
```

```
## [1] 5.333333
```

By order:

```
mean(c(1, 5, 10, NA),  
      TRUE)
```

```
## Error in mean.default(c(1, 5, 10, NA), TRUE): 'trim' must  
be numeric of length one
```

34 / 49

functions()

Watch out for 'hidden' arguments

By name:

```
mean(x = c(1, 5, 10, NA),  
      na.rm = TRUE)
```

```
## [1] 5.333333
```

By order:

```
mean(c(1, 5, 10, NA),  
      TRUE)
```

```
## Error in mean.default(c(1, 5, 10, NA), TRUE): 'trim' must  
be numeric of length one
```

This error states that we've assigned the argument **trim** to a non-valid argument

Where did **trim** come from?

34 / 49

R documentation

?mean

35 / 49

R documentation

?mean

mean {base}

R Documentation

Arithmetic Mean

Description

Generic function for the (trimmed) arithmetic mean.

Usage

```
mean(x, ...)
```

Default S3 method:

```
mean(x, trim = 0, na.rm = FALSE, ...)
```

Arguments

x An R object. Currently there are methods for numeric/logical vectors and [date](#), [date-time](#) and [time interval](#) objects. Complex vectors are allowed for `trim = 0`, only.

trim the fraction (0 to 0.5) of observations to be trimmed from each end of `x` before the mean is computed. Values of `trim` outside that range are taken as the nearest endpoint.

na.rm a logical value indicating whether NA values should be stripped before the computation proceeds.

... further arguments passed to or from other methods.

35 / 49

Data

Generally kept in **vectors** or **data.frames**

- These are objects with names (like functions)
- We can use `<-` to assign values to objects (assignment)

Vector (1 dimension)

```
my_letters <- c("a", "b", "c")
my_letters
```

```
## [1] "a" "b" "c"
```

Data frame (2 dimensions)

rows x columns

```
my_data <- data.frame(x = c("s1", "s2", "s3", "s4"),
  y = c(101, 102, 103, 104),
  z = c("a", "b", "c", "d"))
```

```
my_data
```

```
##   x   y z
## 1 s1 101 a
## 2 s2 102 b
## 3 s3 103 c
## 4 s4 104 d
```

36 / 49

Vectors

Use `c()` to create a vector

```
a <- c("apples", 12, "bananas")
```

Use `x[index]` to access part of a vector

```
a[3] # [1] "bananas"
```

Vectors contain one type of variable

(Even if you try to make it with more)

```
class(a) # [1] "character"
```

37 / 49

Data frames (also tibbles)

```
my_data
```

```
##   x   y z
## 1 s1 101 a
## 2 s2 102 b
## 3 s3 103 c
## 4 s4 104 d
```

- Columns have different types of variables
- `x$colname` to pull columns out as vector
- `x[row, col]` to access rows and columns of a data frame

38 / 49

Your Turn: Vectors and Data frames

Try out the following code...

1. What is the output in your console?
2. How does your **environment** change (upper right panel)?

Vectors

```
a <- c("apples", 12, "bananas")
a
```

Data frames

```
my_data <- data.frame(x = c("s1", "s2", "s3", "s4"),
  y = c(101, 102, 103, 104),
  z = c("a", "b", "c", "d"))
my_data
```

39 / 49

Your Turn: Vectors and Data frames

Try out the following code...

Vectors

```
a[2]
a[c(1, 3)]
a[3:5]
```

Data frames

```
my_data[3, ]
my_data[3, 1]
my_data[, 1:2]
```

40 / 49

Your Turn: Vectors and Data frames

Try out the following code...

Vectors

```
a[2]
## [1] "12"
a[c(1, 3)]
## [1] "apples" "bananas"
a[3:5]
## [1] "bananas" NA      NA
```

Data frames

```
my_data[3, ]
##      x    y z
## 3 s3 103 c
my_data[3, 1]
## [1] "s3"
my_data[, 1:2]
##      x    y
## 1 s1 101
## 2 s2 102
## 3 s3 103
## 4 s4 104
```

41 / 49

Miscellaneous

R has spelling and punctuation

- R cares about spelling
- R is also case sensitive! (**Apple** is not the same as **apple**)
- Comma's are used to separate arguments in functions

For example

This is correct:

```
mean(c(5, 7, 10)) # [1] 7.333333
```

This is **not** correct:

```
mean(c(5 7 10))
```

```
## Error: <text>:1:10: unexpected numeric constant
## 1: mean(c(5 7
##                ^
```

43 / 49

R has spelling and punctuation

- R cares about spelling
- R is also case sensitive! (**Apple** is not the same as **apple**)
- Comma's are used to separate arguments in functions

For example

This is correct:

```
mean(c(5, 7, 10)) # [1] 7.333333
```

>80% of learning R is learning to
troubleshoot

This is **not** correct:

```
mean(c(5 7 10))
```

```
## Error: <text>:1:10: unexpected numeric constant
## 1: mean(c(5 7
##                ^
```

43 / 49

R has spelling and punctuation

Spaces usually don't matter unless they change meanings

```
5>=6 # [1] FALSE
5 >=6 # [1] FALSE
5 >= 6 # [1] FALSE
5 > = 6 # Error: unexpected '=' in "5 > ="
```

Periods don't matter either, but can be used in the same way as letters

(But don't)

```
apple.oranges <- "fruit"
```

44 / 49

Assignments and Equal signs

Use <- to assign values to objects

```
a <- "hello"
```

Use = to set function arguments

```
mean(x = c(4, 9, 10))
```

Use == to determine equivalence (logical)

```
10 == 10 # [1] TRUE  
10 == 9  # [1] FALSE
```

45 / 49

Braces/Brackets

Round brackets: ()

- Identify functions (even if there are no arguments)

```
Sys.Date() # Get the Current Date
```

```
## [1] "2021-01-15"
```

46 / 49

Braces/Brackets

Round brackets: ()

- Identify functions (even if there are no arguments)

```
Sys.Date() # Get the Current Date
```

```
## [1] "2021-01-15"
```

- Without the (), R spits out information on the function:

```
Sys.Date
```

```
## function (  
## as.Date(as.POSIXlt(Sys.time()))  
## <bytecode: 0x55cfa98bfbe0>  
## <environment: namespace:base>
```

46 / 49

Braces/Brackets

Round brackets: ()

- Identify functions (even if there are no arguments)

```
Sys.Date() # Get the Current Date
```

```
## [1] "2021-01-15"
```

- Without the (), R spits out information on the function:

```
Sys.Date
```

```
## function (  
## as.Date(as.POSIXlt(Sys.time()))  
## <bytecode: 0x55cfa98bfbe0>  
## <environment: namespace:base>
```

() must be associated with a function

(Well, *almost* always)

46 / 49

Braces/Brackets

Square brackets: []

- Extract parts of objects

```
LETTERS
```

```
## [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S"  
## [20] "T" "U" "V" "W" "X" "Y" "Z"
```

```
LETTERS[1]
```

```
## [1] "A"
```

```
LETTERS[26]
```

```
## [1] "Z"
```

47 / 49

Braces/Brackets

Square brackets: []

- Extract parts of objects

```
LETTERS
```

```
## [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S"  
## [20] "T" "U" "V" "W" "X" "Y" "Z"
```

```
LETTERS[1]
```

```
## [1] "A"
```

```
LETTERS[26]
```

```
## [1] "Z"
```

[] have to be associated with an object that has dimensions

(Always)

47 / 49

Improving code readability

Use spaces like you would in sentences:

```
a <- mean(c(4, 10, 13))
```

is easier to read than

```
a<-mean(c(4,10,13))
```

(But the same, coding-wise)

48 / 49

Improving code readability

Don't be afraid to use line breaks ('Enters') to make the code more readable

Hard to read

```
a <- data.frame(exp = c("A", "B", "A", "B", "A", "B"), sub = c("A1", "A1", "A2", "A2", "A3", "A3"),  
res = c(10, 12, 45, 12, 12, 13))
```

Easier to read

```
a <- data.frame(exp = c("A", "B", "A", "B", "A", "B"),  
  sub = c("A1", "A1", "A2", "A2", "A3", "A3"),  
  res = c(10, 12, 45, 12, 12, 13))
```

(But the same, coding-wise)

49 / 49