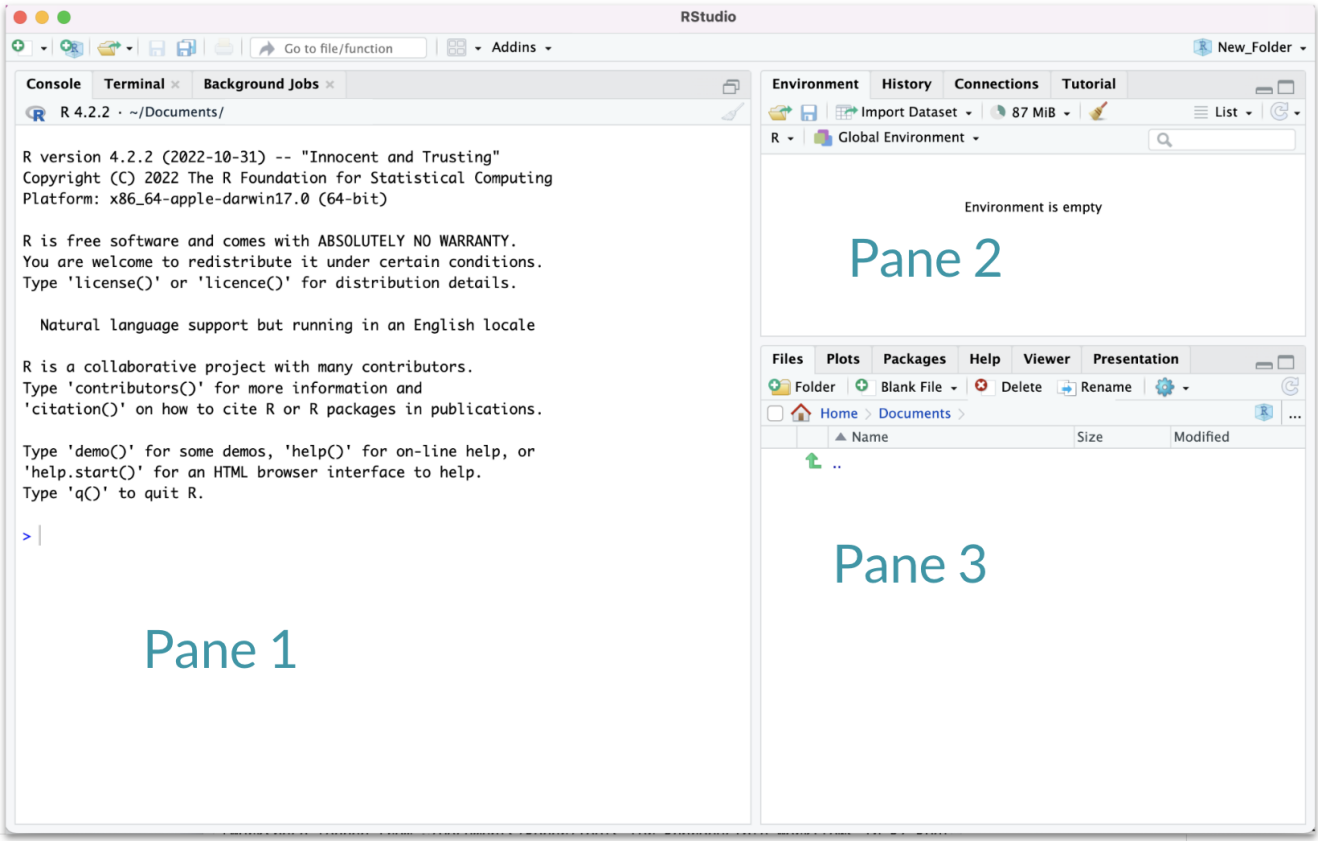
Basic R

Working in R

For now, we will be working in the **Console** (Pane 1)



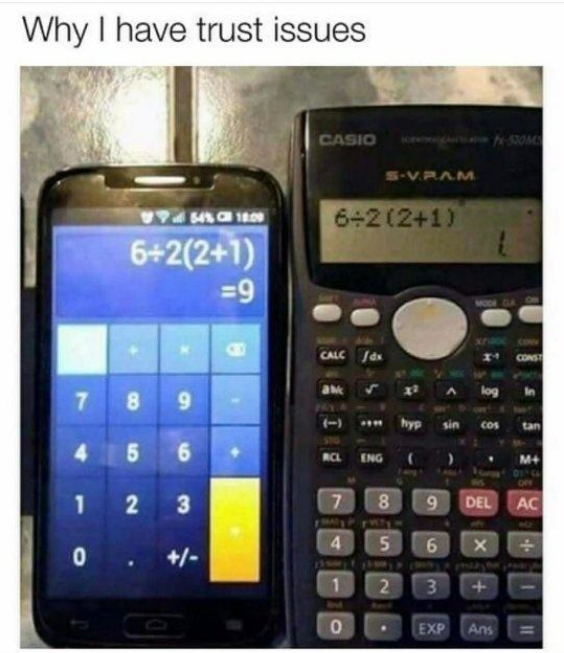
R as a calculator

- The R console is a full calculator
- Try to play around with it:
  - +, -, /, * are add, subtract, divide and multiply
  - ^ or ** is power
  - parentheses – (and) – work with order of operations
  - %% finds the remainder

R as a calculator

Try evaluating the following. Type these in the Console and press *return* to evaluate:

- `2 + 2`
- `2 * 3 / 4`
- `2^4 - 1`



Basic terms: "object"

Object - an object is something that can be worked with or on in R - can be lots of different things!

You can think of objects as **nouns** in R.

- a variable
- a dataset
- a plot

... many more

Assigning values to objects

- You can create objects within the R environment and from files on your computer
- R uses `<-` to create objects (you might also see `=` used, but this is not best practice)

```
x <- 2
x
```

```
[1] 2
```

```
x * 4
```

```
[1] 8
```

```
x + 2
```

```
[1] 4
```

GUT CHECK: What is an "object"?

A. Something I can touch

B. Something that can be worked with in R

C. A software version

Objects with text

Create objects with text using quotation marks:

```
y <- "hello world!"
y
```

```
[1] "hello world!"
```

numeric vs. character classes?

We will talk in-depth about classes. For now:

numeric

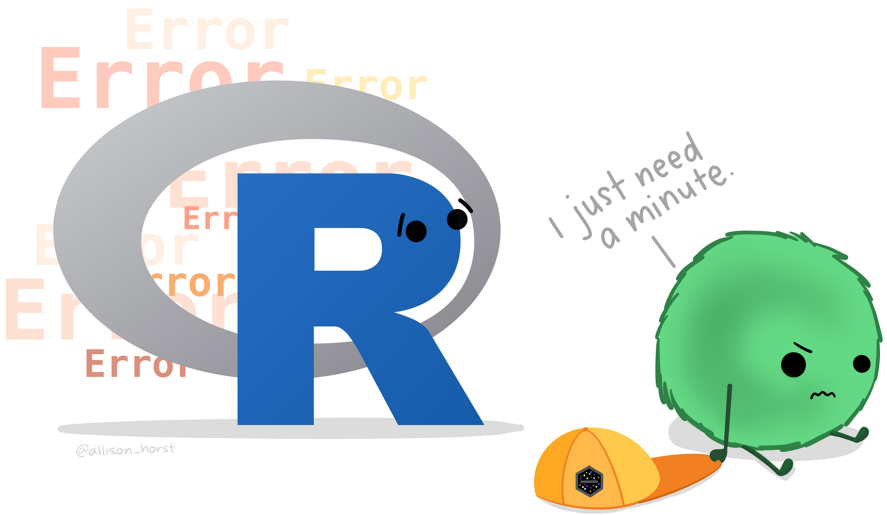
- Numbers
- No quotation marks

```
2
```

character

- Text with quotation marks
- Green lettering (default)

```
"hello!"
```



Common issues

TROUBLESHOOTING: R is case sensitive

Object names are case-sensitive, i.e., X and x are different

```
x
```

```
[1] 2
```

```
X
```

```
Error: object 'X' not found
```

TROUBLESHOOTING: No commas in big numbers

Commas separate objects in R, so they shouldn't be used when entering big numbers.

```
z <- 3,000
```

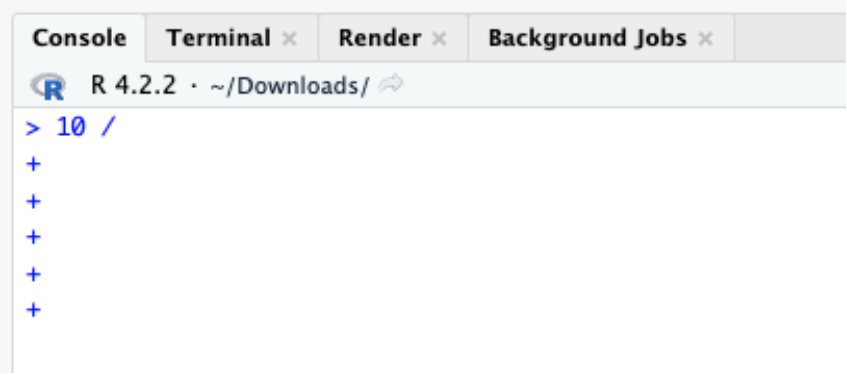
```
Error in parse(text = input): <text>:1:7: unexpected ','
1: z <- 3,
          ^
```

TROUBLESHOOTING: Complete the statement

```
10 /

Error in parse(text = input): <text>:2:0: unexpected end of input
1: 10 /
   ^
```

+ indicates an incomplete statement. Hit “esc” to clear and bring back the **>**.



Simple object practice

Try assigning your full name to an R object called `name`

Simple object practice

Try assigning your full name to an R object called `name`

```
name <- "Ava Hoffman"
name
```

```
[1] "Ava Hoffman"
```

Combining objects with `c()`

Use `c()` to collect/combine single R objects into a vector of R objects. It is mostly used for creating vectors of numbers and character strings.

```
x <- c(1, 4, 6, 8)
x
```

```
[1] 1 4 6 8
```

Combining objects with `c()`

Try assigning your first and last name as 2 separate character strings into a vector called `name2`

Combining objects with `c()`

Try assigning your first and last name as 2 separate character strings into a vector called `name2`

```
name2 <- c("Ava", "Hoffman")
name2
```

```
[1] "Ava"     "Hoffman"
```

Basic terms: "function"

Function - a function is a piece of code that allows you to do something in R. You can write your own, use functions that come directly from installing R, or use functions from additional packages.

You can think of a function as **verb** in R.

A function might help you add numbers together, create a plot, or organize your data.

Using functions on our vector

- `class()` tells us what kind of values the object contains (numeric, character, etc)
- `length()` tells us how many elements.

```
name
```

```
[1] "Ava Hoffman"
```

```
class(name)
```

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

```
x
```

```
[1] 1 4 6 8
```

```
length(x)
```

```
[1] 4
```

GUT CHECK: What is a “function”?

A. a number or text

B. a button inside RStudio

C. code that does something

Combining vectors

It's fine to combine vectors, but all values will end up with the same class!

```
vect <- c(name, x)
vect
```

```
[1] "Ava Hoffman" "1"           "4"           "6"           "8"
```

```
class(vect)
```

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

Practicing functions

What do you expect for the length of the `name2` object?

What is the class?

Practicing functions

What do you expect for the length of the `name2` object?

What is the class?

```
length(name2)
```

```
[1] 2
```

```
class(name2)
```

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

Commenting in code

creates a comment in R code

```
# 1 + 2 <- this does not get run
```

```
1 + 2 # <- this does
```

```
[1] 3
```

Lab Part 1

- Assign values to objects with `<-` (new name on left side)
- Use the `c()` function to combine text/numbers/etc. into a vector
- `class()` tells you the class (kind) of object
- Use the `length()` function to determine number of elements
- # for comments or to deactivate a line of code

Just open up the file to see the questions for lab. More about the file type soon!

 Lab

Math + vector objects

You can perform math with vectors.

```
x + 2
```

```
[1]  3  6  8 10
```

```
x * 3
```

```
[1]  3 12 18 24
```

```
x + c(1, 2, 3, 4)
```

```
[1]  2  6  9 12
```

Math + vector objects

But math can only be performed on numbers.

```
name2 + 4
```

```
Error in name2 + 4: non-numeric argument to binary operator
```

Reassigning to a new object

Save these modified vectors as a new vector called `y`.

```
y <- x + c(1, 2, 3, 4)
y
```

```
[1]  2  6  9 12
```

Note that the R object `y` is no longer “hello world!” - It has been overwritten by assigning new data to the same name.

Reassigning to a new object

Reassigning allows you to make changes “in place”

```
# results not stored:
x + c(1, 2, 3, 4)

# x remains unchanged, results stored in `y`:
y <- x + c(1, 2, 3, 4)

# replace `x` in place
x <- x + c(1, 2, 3, 4)
```

R objects

You can get more attributes than just class. The function `str()` gives you the structure of the object.

```
str(x)
```

```
 num [1:4] 1 4 6 8
```

```
str(y)
```

```
 num [1:4] 2 6 9 12
```

This tells you that `x` is a numeric vector and tells you the length.

Basic terms: “argument”

Argument - what you pass to a function

- can be data like the number 1 or 20234
- can be options about how you want the function to work
- separated by commas

Like an **adverb**.

Create vectors with **seq()**

For numeric: `seq()`

- The `from` argument says what number to start on.
- The `to` argument says what number to not go above.
- The `by` argument says how much to increment by.
- The `length.out` argument says how long the vector should be overall.

```
seq(from = 0, to = 1, by = 0.2)
```

```
[1] 0.0 0.2 0.4 0.6 0.8 1.0
```

```
seq(from = 0, to = 10, by = 1)
```

```
 [1]  0  1  2  3  4  5  6  7  8  9 10
```

```
seq(from = -5, to = 5, length.out = 10)
```

```
 [1] -5.0000000 -3.8888889 -2.7777778 -1.6666667 -0.5555556  0.5555556
 [7]  1.6666667  2.7777778  3.8888889  5.0000000
```

Useful functions to create vectors `rep()`

For character: `rep()` can create very long vectors. Works for creating character and numeric vectors.

The `each` argument specifies how many of each item you want repeated. The `times` argument specifies how many times you want the vector repeated.

```
rep(WHAT_TO_REPEAT, arguments)
```

```
rep(c("black", "white"), each = 3)
```

```
[1] "black" "black" "black" "white" "white" "white"
```

```
rep(c("black", "white"), times = 3)
```

```
[1] "black" "white" "black" "white" "black" "white"
```

```
rep(c("black", "white"), each = 2, times = 2)
```

```
[1] "black" "black" "white" "white" "black" "black" "white" "white"
```

Creating numeric vectors `sample()`

You can use the `sample()` function to make a random sequence. The `x` argument specifies what you are sampling from. The `size` argument specifies how many values there should be. The `replace` argument specifies if values should be replaced or not.

```
seq_hun <- seq(from = 0, to = 100, by = 1)
seq_hun
```

```
  [1]   0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
 [19]  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35
 [37]  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53
 [55]  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71
 [73]  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89
 [91]  90  91  92  93  94  95  96  97  98  99 100
```

```
y <- sample(x = seq_hun, size = 5, replace = TRUE)
y
```

```
[1] 89 69 78 77 13
```

Installing packages to do more!

Some functions and data come with R right out of the box ("base R"). We will add more functionality with packages. Think of these like "expansion packs" for R.

Must be done **once** for each installation of R (e.g., version 4.2 >> 4.3).

An important package we will use is `tidyverse`. It is a mega-package great for data import, wrangling, and visualization.

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

Loading packages

After installing packages, you will need to "load" them into memory so that you can use them.

This must be done **every time** you start R.

We use a function called `library` to load packages.

```
library(tidyverse)
```

Installing + Loading packages

Images sourced from https://www.wikihow.com/Change-a-Light-Bulb

Summary

- R functions as a calculator
- Use `<-` to save (assign) values to objects. Reassigning allows you to make changes "in place".
- Use `c()` to **combine** into vectors
- `length()`, `class()`, and `str()` tell you information about an object
- The sequence `seq()` function helps you create numeric vectors (`from`,`to`, `by`, and `length.out` arguments)
- The repeat `rep()` function helps you create vectors with the `each` and `times` arguments
- `sample()` makes random vectors
- `install.packages()` and `library()` install and load packages, respectively.

Summary

- Class Website
- Basic R Lab
- Day 1 Cheatsheet

Image by Gerd Altmann from Pixabay