

Intro to R

Basic R

Common new users frustrations

1. **Different versions of software**
2. **Data type problems (is that a character or a number?)**
3. Working directory problems: trying to read files that R “can’t find”
 - RStudio can help, and so do RStudio Projects
 - discuss in Data Input/Output lecture
4. Typos (R is **case sensitive**, x and X are different)
 - RStudio helps with “tab completion”
 - discussed throughout

Explaining output on slides

In slides, a command (we'll also call them code or a code chunk) will look like this

```
print("I'm code")
```

```
[1] "I'm code"
```

And then directly after it, will be the output of the code.

So `print("I'm code")` is the code chunk and `[1] "I'm code"` is the output.

R as a calculator

```
2 + 2
```

```
[1] 4
```

```
2 * 4
```

```
[1] 8
```

```
2^3
```

```
[1] 8
```

Note: when you type your command, R inherently thinks you want to print the result.

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

$$2 + (2 * 3)^2$$

[1] 38

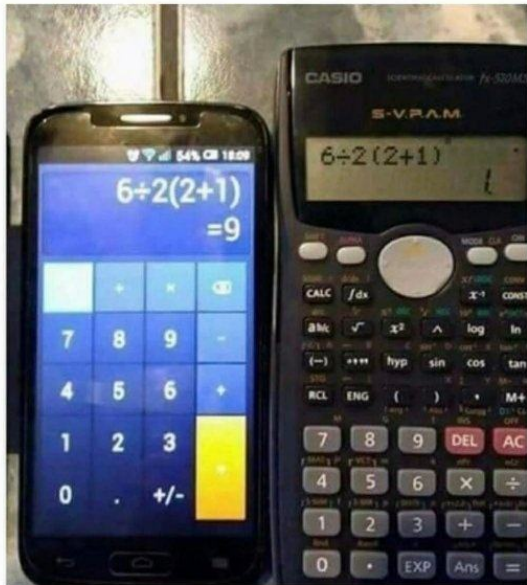
$$(1 + 3) / 2 + 45$$

[1] 47

$$6 / 2 * (1 + 2)$$

[1] 9

Why I have trust issues



#BEDMAS #PEMDAS

Math Prof answers $6 \div 2(1+2) = ?$ once and for all ***Viral Math Problem***

114,629 views • Nov 14, 2020



Dr. Trefor Bazett

159K subscribers

lol, am I really doing this? Ok, fine. There is a ***viral math problem*** about, uh, order of operations. You know, #BEDMAS or #PEMDAS. The most common form is $6/2(1+2)$ but it also shows up as $60/5(7-5)$ and other equivalent forms. What is the correct answer explained by a math

SHOW MORE

R as a calculator

Try evaluating the following:

- $2 + 2 * 3 / 4 - 3$

- $2 * 3 / 4 * 2$

- $2^4 - 1$

Commenting in Scripts

creates a comment in R code

```
# this is a comment
```

```
# nothing to its right is evaluated
```

```
# this # is still a comment
```

```
### you can use many #'s as you want
```

```
1 + 2 # Can be the right of code
```

```
[1] 3
```

In an `.Rmd` file, you can write notes outside the R chunks.

Assigning values to objects

- You can create objects from within the R environment and from files on your computer
- R uses `<-` to assign values to an object name (you might also see `=` used, but this is not best practice)
- Object names are case-sensitive, i.e. `X` and `x` are different

```
x <- 2  
x
```

```
[1] 2
```

```
x * 4
```

```
[1] 8
```

```
x + 2
```

```
[1] 4
```

Assigning values to objects

- The most comfortable and familiar class/data type for many of you will be `data.frame`
- You can think of these as essentially spreadsheets with rows (usually subjects or observations) and columns (usually variables)
- `data.frames` are somewhat advanced objects in R; we will start with simpler objects

Assigning values to objects

- Here we introduce “1 dimensional” classes; often referred to as ‘vectors’
- Vectors can have multiple sets of observations, but each observation has to be the same class.
- Use the `class()` function to check the class of an object.

```
class(x)
```

```
[1] "numeric"
```

```
y <- "hello world!"  
print(y)
```

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

```
class(y)
```

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

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"
```

The 'combine' function `c()`

The function `c()` collects/combines/joins single R objects into a vector of R objects. It is mostly used for creating vectors of numbers, character strings, and other data types.

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

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

```
class(x)
```

```
[1] "numeric"
```

The 'combine' function `c()`

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

The 'combine' function `c()`

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

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

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


Arguments inside R functions

- The contents you give to an R function are called “arguments”
- Here, R assumes all arguments should be objects contained in the vector
- We will talk more about arguments as we use more complicated functions!

```
name2 <- c("Ava", "Hoffman")  
# Arg 1      ^^^^^
```

```
name2 <- c("Ava", "Hoffman")  
# Arg 2      ^^^^^^^^^
```

length of R objects

`length()`: Get or set the length of vectors (including lists) and factors, and of any other R object for which a method has been defined.

```
length(x)
```

```
[1] 4
```

```
y
```

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

```
length(y)
```

```
[1] 1
```

length of R objects

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

What are the lengths of each?

length of R objects

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

What are the lengths of each?

```
length(name)
```

```
[1] 1
```

```
length(name2)
```

```
[1] 2
```

Math + vector objects

You can perform functions to entire vectors of numbers very easily.

```
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
```

Lab Part 1

Lab

Math + vector objects

But things like algebra 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.

R objects

This is handy when we start dealing with bigger / more complex objects.

```
str(z)
```

```
num [1:100] 9 12 9 9 2 6 2 6 6 9 ...
```

Lab Part 2

Lab

Useful functions to create vectors `seq()`

For numeric: `seq()` can be very useful- both integer and double.

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] 84 84 31 76 16
```

Summary

- R functions as a calculator
- Use `<-` to save (assign) values to objects
- Use `c()` to **combine** 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

[Class Website](#)

[Basic R Lab](#)

