Week 1: *Getting Started*

EMSE 4571: Intro to Programming for Analytics

John Paul Helveston

January 19, 2023

Week 1: *Getting Started*

1. Course orientation

BREAK

2. Getting started with R & RStudio

3. Operators & data types

4. Preview of HW 1

Week 1: *Getting Started*

1. Course orientation

BREAK

2. Getting started with R & RStudio

3. Operators & data types

4. Preview of HW 1

Meet your instructor!

John Helveston, Ph.D.

Assistant Professor, Engineering Management & Systems Engineering

- 2016-2018 Postdoc at Institute for Sustainable Energy, Boston University
- 2016 PhD in Engineering & Public Policy at Carnegie Mellon University
- 2015 MS in Engineering & Public Policy at Carnegie Mellon University
- 2010 BS in Engineering Science & Mechanics at Virginia Tech
- Website: www.jhelvy.com

Meet your tutors!

Michael Rossetti

- Graduate Assistant (GA)
- PhD student in EMSE
- Website: https://prof-rossetti.org/

Meet your tutors!

Ben Buechner

- Learning Assistant (LA)
- EMSE Sophomore & P4A / EDA alumni

Course orientation

🌐 Everything you need will be on the course website:

https://p4a.seas.gwu.edu/2023-Spring/

🧩 Course is broken into **two chunks**:

1. Programming (before Spring Break)

2. Analytics (after Spring Break)

Learning Objectives

After this class, you will know how to...

...write R code to solve medium-sized tasks.

...pro-actively test and debug code.

...reproducibly import, export, manipulate, and visualize data.

Attendance / Participation (7%)

Attendance will be taken and will be part of your participation grade

Homeworks (48% of grade)

- ~Every week (13 total, lowest dropped)
- Due 11:59pm Wed. before class

Late submissions

- **3** late days - use them anytime, no questions asked
- After that, 50% off for up to 24 hours after deadline, 0% afterwards
- Contact me for special cases

Quizzes (15% of grade)

In class every other week-ish (7 total, drop lowest 2)

~10-15 minutes (1-3 questions)

Why quiz at all? There's a phenomenon called the "retrieval effect" - basically, you have to *practice* remembering things, otherwise your brain won't remember them (details in the book "Make It Stick: The Science of Successful Learning").

Exams (30% of grade)

- Midterm (weeks 1 - 7) on March 09
- Final (weeks 1 - 14) on May 11

Grades

Component	Weight	Notes
Participation / Attendance	7%	
Homeworks & Readings (13x)	48%	Lowest 1 dropped
Quizzes (7x)	15%	Lowest 2 dropped
Midterm Exam	10%	
Final Exam	20%	

Alternative Minimum Grade (AMG)

- Designed for those who struggle early but work hard to succeed in 2nd half.
- Highest possible grade is "C"

Course Component	Weight
Best 10 Homeworks	40%
Best 4 Quizzes	10%
Midterm Exam	10%
Final Exam	40%

Course policies

- BE NICE
- BE HONEST
- DON'T CHEAT

Don't copy-paste others' code!

On chatGPT

Using an AI to do your assignments is the same as using an external expert.

It's cheating.

Don't do it.

How to succeed in this class

 Participate during class!

 Start assignments early and **read carefully**!

 Get sleep and take breaks often!

 Ask for help!

Getting Help

Use Slack to ask questions.

Meet with your tutors

Schedule a call w/Prof. Helveston

GW Coders

Course Software

Slack: Install app & **turn notifications on**!

R & RStudio: Install both.

RStudio Cloud: A (free) web-based version of RStudio.

Break

Install course software if you haven't

05:00

Week 1: *Getting Started*

1. Course orientation

BREAK

2. Getting started with R & RStudio

3. Operators & data types

4. Preview of HW 1

What is R? (Read a brief history here)

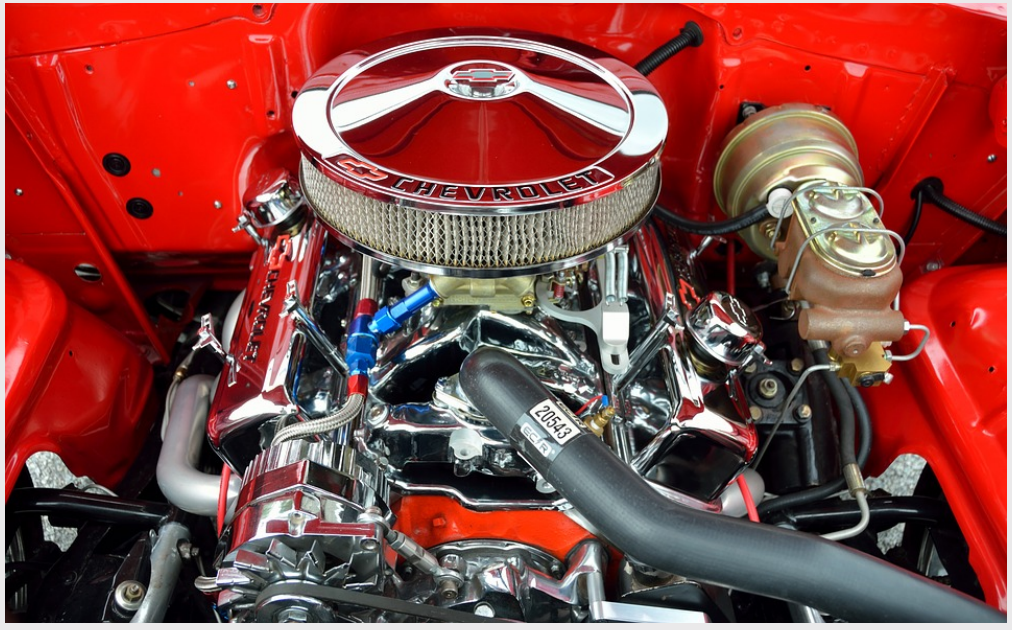
Chambers creates "S" (1976, Bell Labs)
Ross & Robert create "R" (1991, U. of Auckland)

John Chambers

Ross Ihaka

Robert Gentleman

What is RStudio?

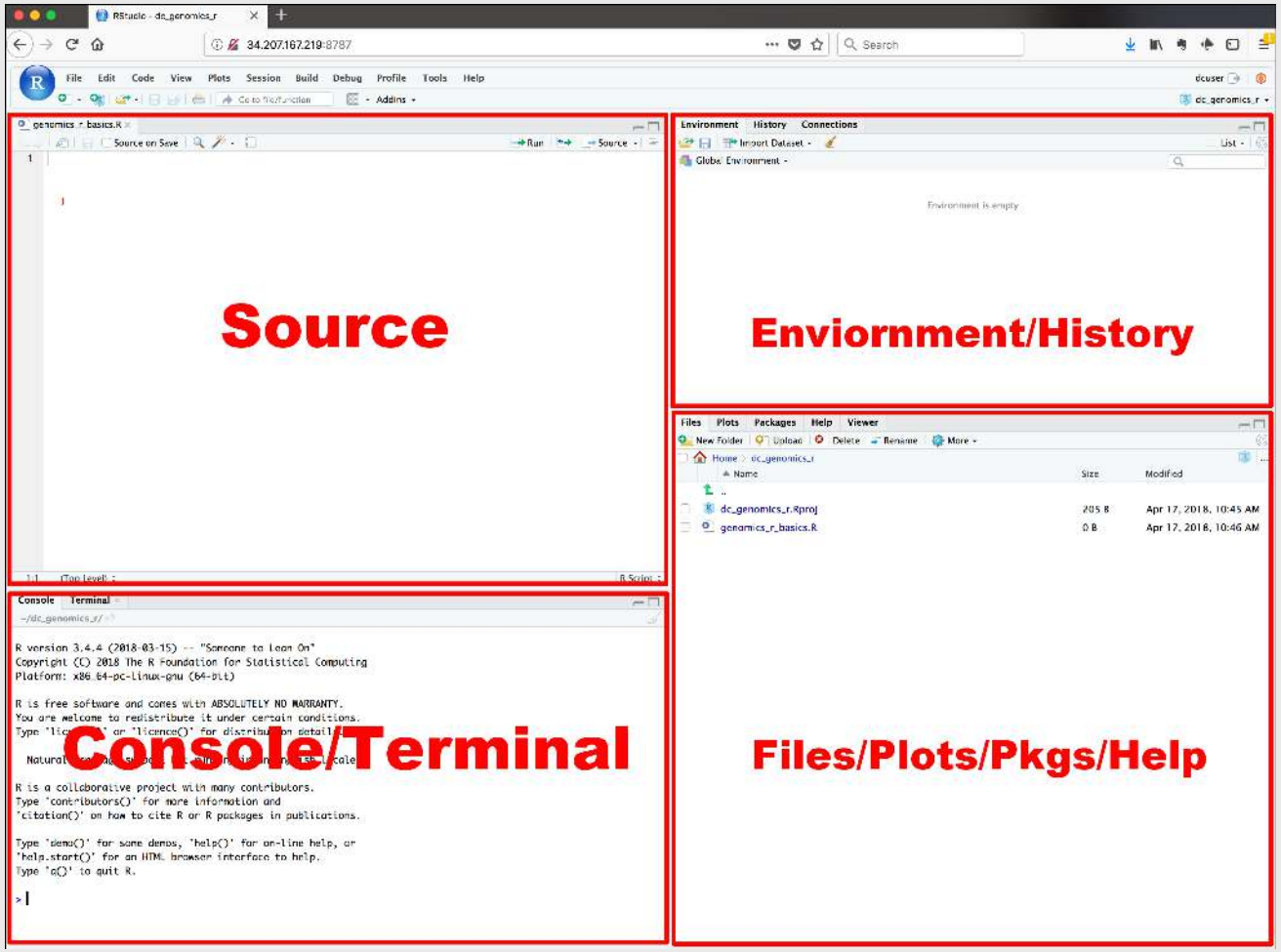


RStudio Orientation

Open this

Not this

RStudio Orientation



- Know the boxes
- Customize the layout
- Customize the look
- Extra themes

Your first conveRsation

Write stuff in the console, then press "enter"

```
3 + 4
```

```
#> [1] 7
```

```
3 + "4"
```

```
#> Error in 3 + "4": non-numeric argument to binary operator
```

Storing values

Use the "<–" symbol to assign *values* to *objects*

```
x <- 40
x
```

```
#> [1] 40
```

```
x + 2
```

```
#> [1] 42
```

Storing values

If you overwrite an object, R "forgets" the old value

```
x <- 42
x
```

```
#> [1] 42
```

```
x <- 50
x
```

```
#> [1] 50
```

Storing values

You can also use the = symbol to assign values

```
x = 50
x
```

```
#> [1] 50
```

...but you should use <-

Storing values

You can store more than just numbers

```
x <- "If you want to view paradise"
y <- "simply look around and view it"
```

```
x
```

```
#> [1] "If you want to view paradise"
```

```
y
```

```
#> [1] "simply look around and view it"
```

Storing values

Pro tip 1:

Shortcut for `<-` symbol

OS	Shortcut
mac	`option` + `-`
windows	`alt` + `-`

(see here for more shortcuts)

Pro tip 2:

Always surround `<-` with spaces

Example:

```
x<-2
```

Does this mean `x <- 2` or `x < -2`?

R ignores **extra space**

```
x      <-         2
y  <-               3
z          <- 4
```

Check:

```
x
```

```
#> [1] 2
```

```
y
```

```
#> [1] 3
```

```
z
```

```
#> [1] 4
```

R cares about **casing**

```
number <- 2
Number <- 3
numbeR <- 4
```

Check:

```
number
```

```
#> [1] 2
```

```
Number
```

```
#> [1] 3
```

```
numbeR
```

```
#> [1] 4
```

Use # for comments

R ignores everything after the # symbol

Example:

```
speed <- 42 # This is mph, not km/h
speed
```

```
#> [1] 42
```

Use meaningful variable names

Example: You are recording the speed of a car in mph

Poor variable name:

```
x <- 42
```

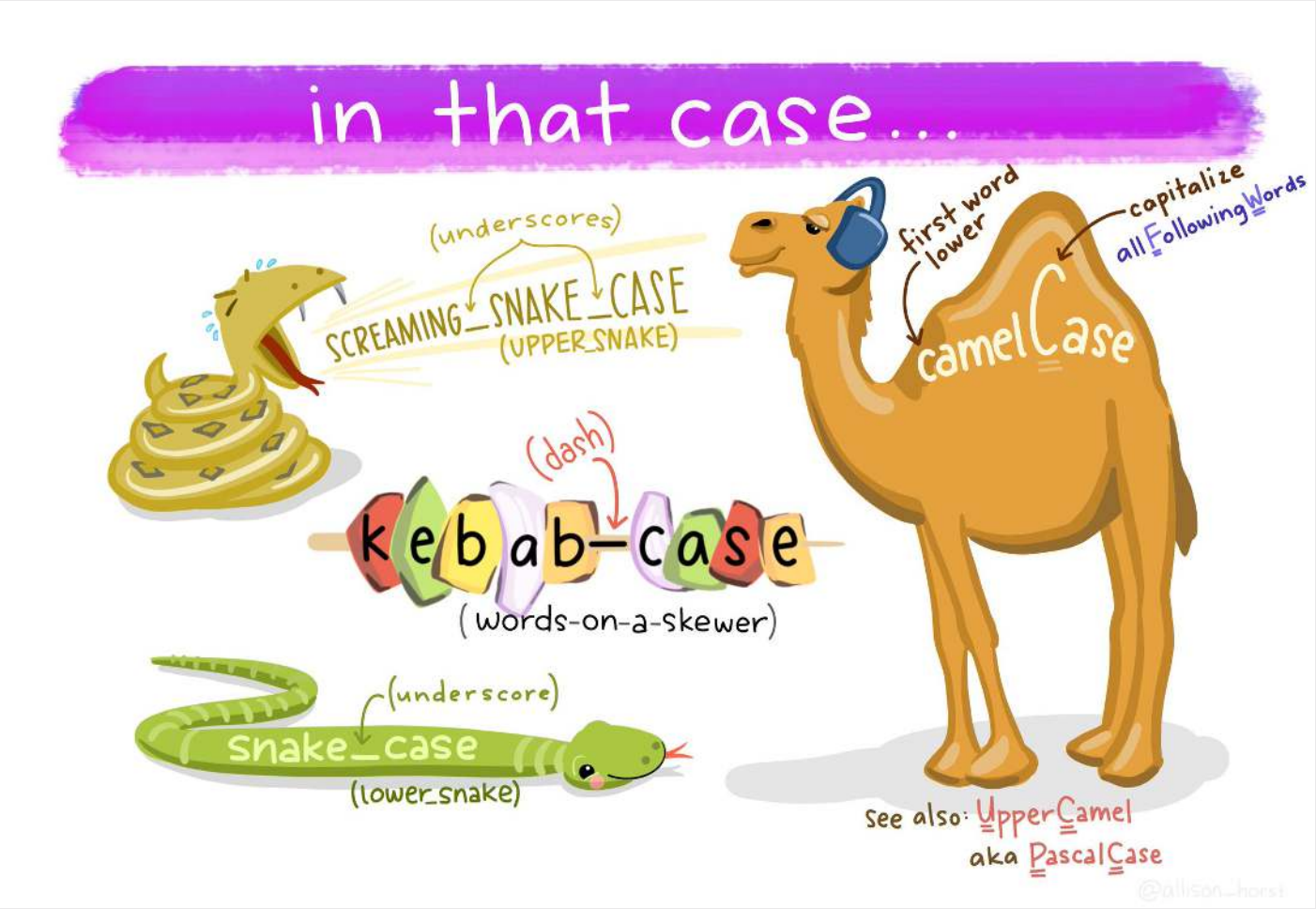
Good variable name:

```
speed <- 42
```

Even better variable name:

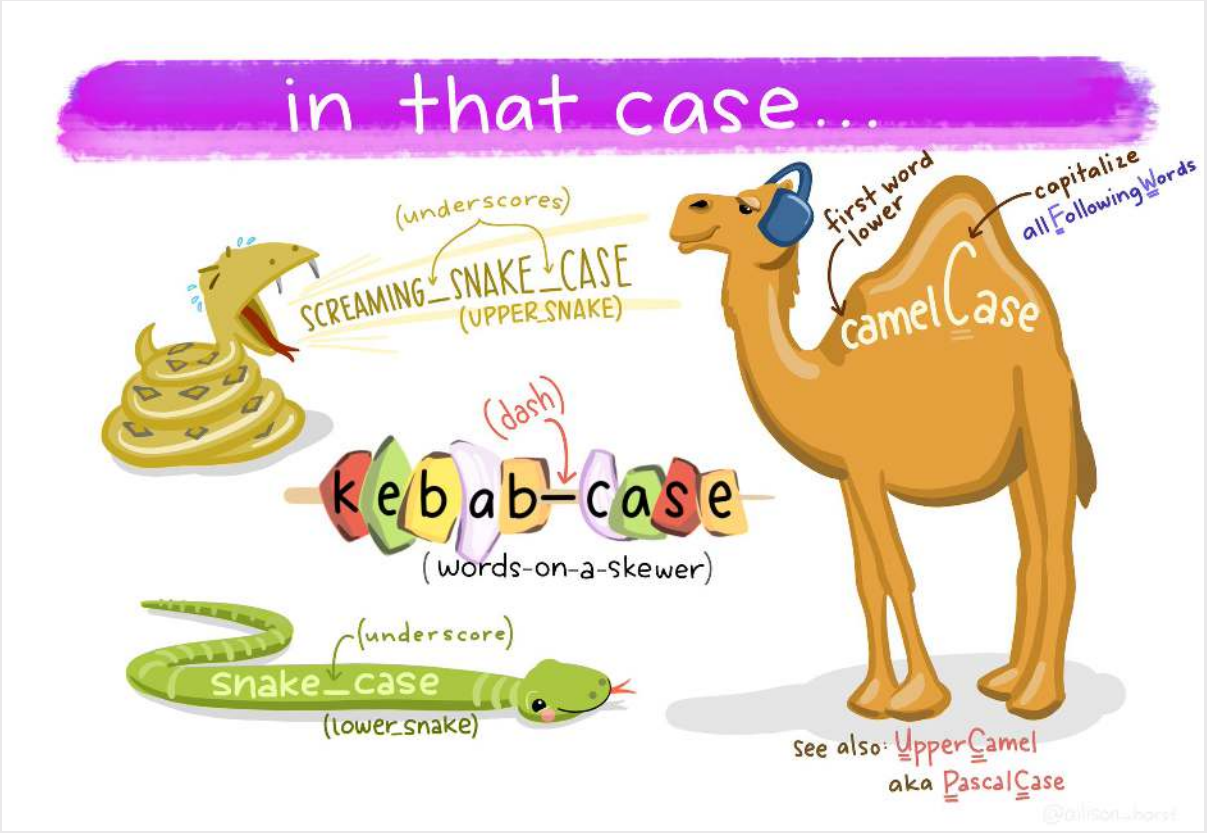
```
speed_mph <- 42
```

Use standard casing styles



Art by Allison Horst

Use standard casing styles



Art by Allison Horst

I recommend using one of these:

- `snake_case_uses_underscores`
- `camelCaseUsesCaps`

Example:

```
days_in_week <- 7
monthsInYear <- 12
```

The workspace

View all the current objects:

```
objects()
```

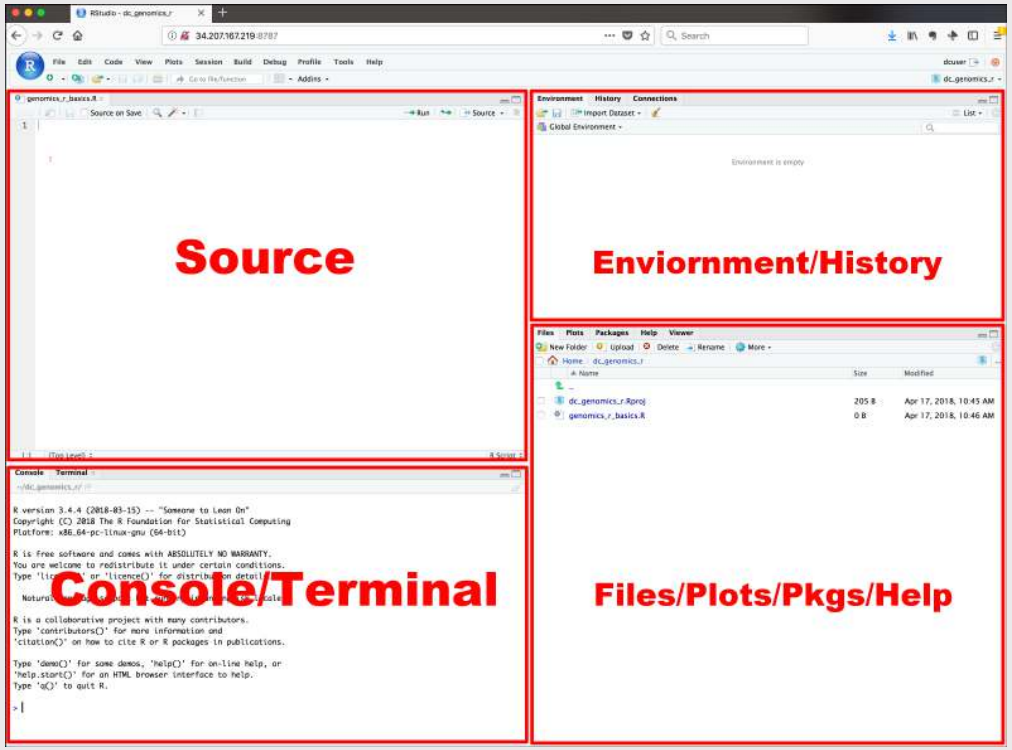
```
#>  [1] "class"          "days_in_week"
"from"           "input"
"monthsInYear"   "number"         "numbeR"
"Number"         "output_file"
"path_notes"     "path_pdf"
"path_slides"    "proc"
"rmd_args"
#> [15] "root"           "self_contained"
"speed"          "speed_mph"      "to"
"x"              "y"              "z"
```

Remove an object by name:

```
rm(number)
objects()
```

```
#>  [1] "class"          "days_in_week"
"from"           "input"
"monthsInYear"   "numbeR"         "Number"
"output_file"    "path_notes"
"path_pdf"       "path_slides"    "proc"
"rmd_args"       "root"
#> [15] "self_contained" "speed"
"speed_mph"      "to"             "x"
"y"              "z"
```

View prior code in history pane



Use "up" arrow see previous code

Staying organized

1) Save your code in .R files

File > New File > R Script

2) Keep work in R Project files

File > New Project...

Your turn

A. Practice getting organized

1. Open RStudio and create a new R project called `week1`.
2. Create a new R script and save it as `practice.R`.
3. Open the `practice.R` file and write your answers to these questions in it.

10:00

B. Creating & working with objects

1) Create objects to store the values in this table:

City	Area (sq mi)	Population (thousands)
San Francisco, CA	47	884
Chicago, IL	228	2,716
Washington, DC	61	694

2) Using the objects you created, answer the following questions:

- Which city has the highest density?
- How many *more* people would need to live in DC for it to have the same population density as San Francisco?

Week 1: *Getting Started*

1. Course orientation

BREAK

2. Getting started with R & RStudio

3. Operators & data types

4. Preview of HW 1

R as a calculator

Basic operators:

- Addition: `+`
- Subtraction: `-`
- Multiplication: `*`
- Division: `/`

Other important operators:

- Power: `^`
- Integer Division: `%/%`
- Modulus: `%%`

Integer division: %/%

Integer division drops the remainder from regular division

```
4 / 3 # Regular division
```

```
#> [1] 1.333333
```

```
4 %/% 3 # Integer division
```

```
#> [1] 1
```

Integer division: %/%

Integer division drops the remainder from regular division

What will this return?

```
4 %/% 4
```

```
#> [1] 1
```

What will this return?

```
4 %/% 5
```

```
#> [1] 0
```

Modulus operator: %%

Modulus returns the *remainder* after doing division

```
5 %% 3
```

```
#> [1] 2
```

```
3.1415 %% 3
```

```
#> [1] 0.1415
```

Modulus operator: %%

Modulus returns the *remainder* after doing division

What will this return?

```
4 %% 4
```

```
#> [1] 0
```

What will this return?

```
4 %% 5
```

```
#> [1] 4
```

Odds and evens with `n %% 2`

If `n %% 2` is `0`, `n` is **EVEN**

```
10 %% 2
```

```
#> [1] 0
```

```
12 %% 2
```

```
#> [1] 0
```

Also works with negative numbers!

```
-42 %% 2
```

```
#> [1] 0
```

If `n %% 2` is `1`, `n` is **ODD**

```
1 %% 2
```

```
#> [1] 1
```

```
13 %% 2
```

```
#> [1] 1
```

Also works with negative numbers!

```
-47 %% 2
```

```
#> [1] 1
```

Number "chopping" with 10s

The mod operator (%%) "chops" a number and returns everything to the *right*

```
123456 %% 1
```

```
#> [1] 0
```

```
123456 %% 10
```

```
#> [1] 6
```

```
123456 %% 100
```

```
#> [1] 56
```

Integer division (%/%) "chops" a number and returns everything to the *left*

```
123456 %/% 1
```

```
#> [1] 123456
```

```
123456 %/% 10
```

```
#> [1] 12345
```

```
123456 %/% 100
```

```
#> [1] 1234
```

Number "chopping" with 10s

- `%%` returns everything to the *right* (`"chop" –>`)
- `%/%` returns everything to the *left* (`<– "chop"`)
- The "chop" point is always just to the *right* of the chopping digit

Example	“Chop” point	
`1234 %% 1`	`1234 |`	Right of the `1`’s digit
`1234 %% 10`	`123 | 4`	Right of the `10`’s digit
`1234 %% 100`	`12 | 34`	Right of the `100`’s digit
`1234 %% 1000`	`1 | 234`	Right of the `1,000`’s digit
`1234 %% 10000`	`| 1234`	Right of the `10,000`’s digit

Comparing things: **Relational operators**

Compare if condition is TRUE or FALSE using:

- Less than: <
- Less than or equal to : <=
- Greater than or equal to: >=
- Greater than: >
- Equal: ==
- Not equal: !=

```
2 < 2
```

```
#> [1] FALSE
```

```
2 <= 2
```

```
#> [1] TRUE
```

```
(2 + 2) == 4
```

```
#> [1] TRUE
```

```
(2 + 2) != 4
```

```
#> [1] FALSE
```

```
"penguin" == "penguin"
```

```
#> [1] TRUE
```

Comparing things: **Logical operators**

Make multiple comparisons with:

- And: &

- Or: |

- Not: !

Comparing things: **Logical operators**

With "and" (&), every part must be TRUE, otherwise the whole statement is FALSE:

```
(2 == 2) & (3 == 3)
```

```
#> [1] TRUE
```

```
(2 == 2) & (2 == 3)
```

```
#> [1] FALSE
```

With "or" (|), if *any* part is TRUE, the whole statement is TRUE:

```
(2 == 2) | (3 == 3)
```

```
#> [1] TRUE
```

```
(2 == 2) | (2 == 3)
```

```
#> [1] TRUE
```

Comparing things: **Logical operators**

The "not" (`!`) symbol produces the *opposite* statement:

```
! (2 == 2)
```

```
#> [1] FALSE
```

```
! (2 == 2) | (3 == 3)
```

```
#> [1] TRUE
```

```
! ((2 == 2) | (3 == 3))
```

```
#> [1] FALSE
```

Comparing things: **Logical operators**

Order precedence for logical operators: `!` > `&` > `|`

```
TRUE | FALSE & FALSE
```

```
#> [1] TRUE
```

```
(TRUE | FALSE) & FALSE
```

```
#> [1] FALSE
```

```
! TRUE | TRUE
```

```
#> [1] TRUE
```

```
! (TRUE | TRUE)
```

```
#> [1] FALSE
```

Comparing things: **Logical operators**

Pro tip: Use parentheses

```
! 3 == 5   # Confusing
```

```
#> [1] TRUE
```

```
! (3 == 5) # Less confusing
```

```
#> [1] TRUE
```

Other important points

R follows BEDMAS:

1. **B**rackets
2. **E**xponents
3. **D**ivision
4. **M**ultiplication
5. **A**ddition
6. **S**ubtraction

Pro tip: Use parentheses

```
1 + 2 * 4   # Confusing
```

```
#> [1] 9
```

```
1 + (2 * 4) # Less confusing
```

```
#> [1] 9
```

Your turn

08:00

Consider the following objects:

```
w <- TRUE
x <- FALSE
y <- TRUE
```

Write code to answer the following questions:

1. Fill in *relational* operators to make the following statement return TRUE:

   ```
   ! (w __ x) & ! (y __ x)
   ```

2. Fill in *logical* operators to make this statement return FALSE:

   ```
   ! (w __ x) | (y __ x)
   ```

Data Types

Type	Description	Example
`double`	Numbers w/decimals (aka "float")	`3.14`
`integer`	Numbers w/out decimals	`42`
`character`	Text (aka "string")	`"this is some text"`
`logical`	Used for comparing objects	`TRUE`, `FALSE`

Use `typeof()` to find the type

```
typeof(2)
```

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

```
typeof("hello")
```

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

```
typeof(TRUE)
```

```
#> [1] "logical"
```

Numeric types (there are 2)

Integers

No decimals (e.g. `7`)

Doubles (aka "float")

Decimals (e.g. `7.0`)

In R, numbers are "doubles" by default

```
typeof(3)
```

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

R assumes that `3` is really `3.0`

Make it an integer by adding `L`:

```
typeof(3L)
```

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

Character types

Use single or double quotes around anything:

```
typeof('hello')
```

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

```
typeof("3")
```

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

Use single / double quotes if the string *contains* a quote symbol:

```
typeof("don't")
```

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

Logical types

Logical data only have two values: TRUE or FALSE

```
typeof(TRUE)
```

```
#> [1] "logical"
```

```
typeof(FALSE)
```

```
#> [1] "logical"
```

Note that these have to be in all caps, and **not** in quotes:

```
typeof('TRUE')
```

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

```
typeof(True)
```

```
#> Error in typeof(True): object 'True'
not found
```

Logical types

Use to answer questions about logical statements.

Example: Is 1 greater than 2?

```
1 > 2
```

```
#> [1] FALSE
```

```
1 < 2
```

```
#> [1] TRUE
```

Special values

`Inf`: Infinity (*or really big numbers*)

```
1/0
```

```
#> [1] Inf
```

`NaN`: Not a Number

```
0/0
```

```
#> [1] NaN
```

`NA`: Not available (*value is missing*)

`NULL`: no value whatsoever

Your turn

05:00

Will these return TRUE or FALSE?

(try to answer first, then run the code to check)

- `! typeof('3') == typeof(3)`
- `(typeof(7) != typeof("FALSE")) | FALSE`
- `! (typeof(TRUE) == typeof(FALSE)) & FALSE`

Week 1: *Getting Started*

1. Course orientation

BREAK

2. Getting started with R & RStudio

3. Operators & data types

4. Preview of HW 1

Go to the schedule

...and read carefully!