



Week 1: *Getting Started with R*

EMSE 4574: Intro to Programming for Analytics

John Paul Helveston

September 01, 2020

Week 1: *Getting Started with R*

1. Course Introduction
2. Break: Install Course Tools
3. Getting started with R & RStudio
4. Operators & data types
5. Preview of HW 1

Week 1: *Getting Started with R*

1. Course Introduction
2. Break: Install Course Tools
3. Getting started with R & RStudio
4. Operators & data types
5. 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!



Saurav Pantha (aka "The Firefighter")

- Graduate Assistant (GA)
- Masters student in EMSE

Meet your tutors!



Jennifer Kim (aka "The Monitor")

- Learning Assistant (LA)
- EMSE Junior & P4A alumni

Course orientation

🌐 Everything you need will be on the course website:

<https://p4a.seas.gwu.edu/2020-Fall/>

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

1. Programming
2. Analytics

Homeworks (48% of grade)

🕒 ~Every week (12 total)

⌚ Soft due dates (11pm Monday before class)

⚠ Don't abuse this flexibility

Two hard deadlines on homework submissions:

1. Oct. 20 (HWs 1-6)

2. Dec. 08 (HWs 7-12)

Quizzes (15% of grade)

- 🕒 In class every other week-ish (7 total, drop lowest 2)
- ⌚ 5 minutes (3-5 questions)

☰ Example quiz

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 (32% of grade)

- CALENDAR Midterm (weeks 1 - 6) on Oct. 20
- CALENDAR Final (weeks 1 - 13) on Dec. 15

Grading: Standard

Course Component	Weight	Notes
Homeworks	48%	12 x 4% each
Quizzes	15%	5 x 3% each
Midterm Exam	12%	
Final Exam	20%	
Participation	5%	

Grading: Alternative Minimum Grade (AMG)

- Students who struggle early on, 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%

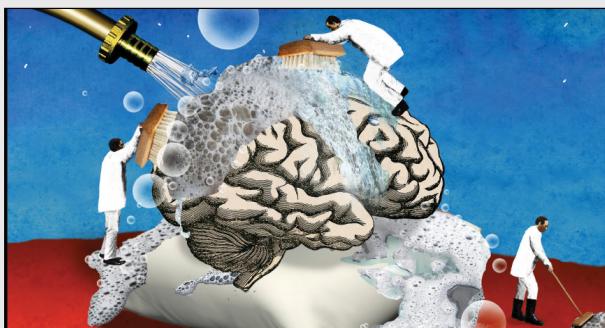
Course policies

- **BE NICE. BE HONEST. DON'T CHEAT.**
- Write your own code (even in "collaborative" assignments)
- Don't cheat

How to succeed in this class

Take care of your brain

- Sleep!
- Exercise!
- Eat good food!



Start HW early!

Take breaks often!

Ask for help!

Getting Help

❖ Use [Slack](#) to ask questions.

❑ Meet with your tutors

❑ Schedule a meeting w/[Prof. Helveston](#):

- Tuesdays from 3:30-4:30pm
- Wednesdays from 2:00-4:30pm
- Fridays from 12:00-2:00pm

🛠 Course Tools (see [course prep lesson](#))

># Slack

- [Link to join](#) (also posted on Blackboard announcement).
- Install Slack on your phone and **turn notifications on!**

Course Tools (see [course prep lesson](#))

R & RStudio (Install both)

After installed:

Open this:



Not this:



R: Engine



RStudio: Dashboard



🛠 Course Tools (see [course prep lesson](#))

WiFi [GWU VPN](#) (Install Cisco AnyConnect VPN Client)

WiFi +  = [RStudio](#) online!

Week 1: *Getting Started with R*

1. Course Introduction
2. Break: Install Course Tools
3. Getting started with R & RStudio
4. Operators & data types
5. Preview of HW 1

Install Course Tools (see [course prep lesson](#))

Slack

- [Link to join](#) (also posted on Blackboard announcement).
- Install Slack on your phone and **turn notifications on!**

R & [RStudio](#) (Install both)

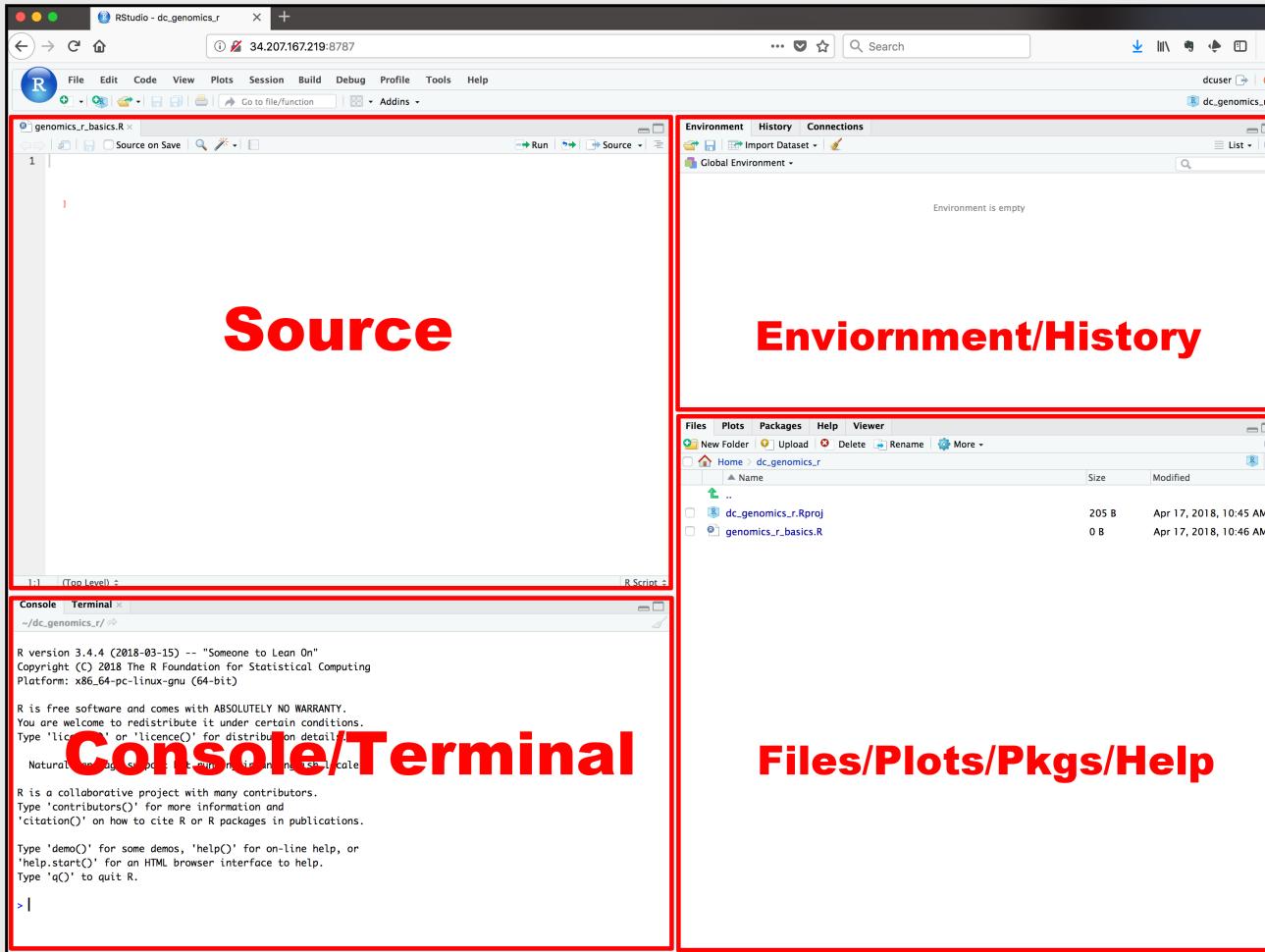
[GWU VPN](#) (Install Cisco AnyConnect VPN Client)

 +  = [RStudio](#) online!

Week 1: *Getting Started with R*

1. Course Introduction
2. Break: Install Course Tools
3. Getting started with R & RStudio
4. Operators & data types
5. Preview of HW 1

RStudio Orientation



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

Your first conversation

Write stuff in the console, then press "enter"

Example: **addition**

```
3 + 4
```

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

Example: **error**

```
3 + "4"
```

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

Storing values

Use the "`<-`" symbol to assign *values* to *objects*

Example:

```
x <- 40  
x
```

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

```
x + 2
```

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

Storing values

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

Example:

```
x <- 42  
x
```

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

```
x <- 50  
x
```

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

Storing values

You can also use the "`=`" symbol to assign values
(but you really should use "`<-`")

Example:

```
x = 42
x
```

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

```
y <- 42
y
```

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

Storing values

You can store more than just numbers

Example:

```
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 **case**

```
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
```

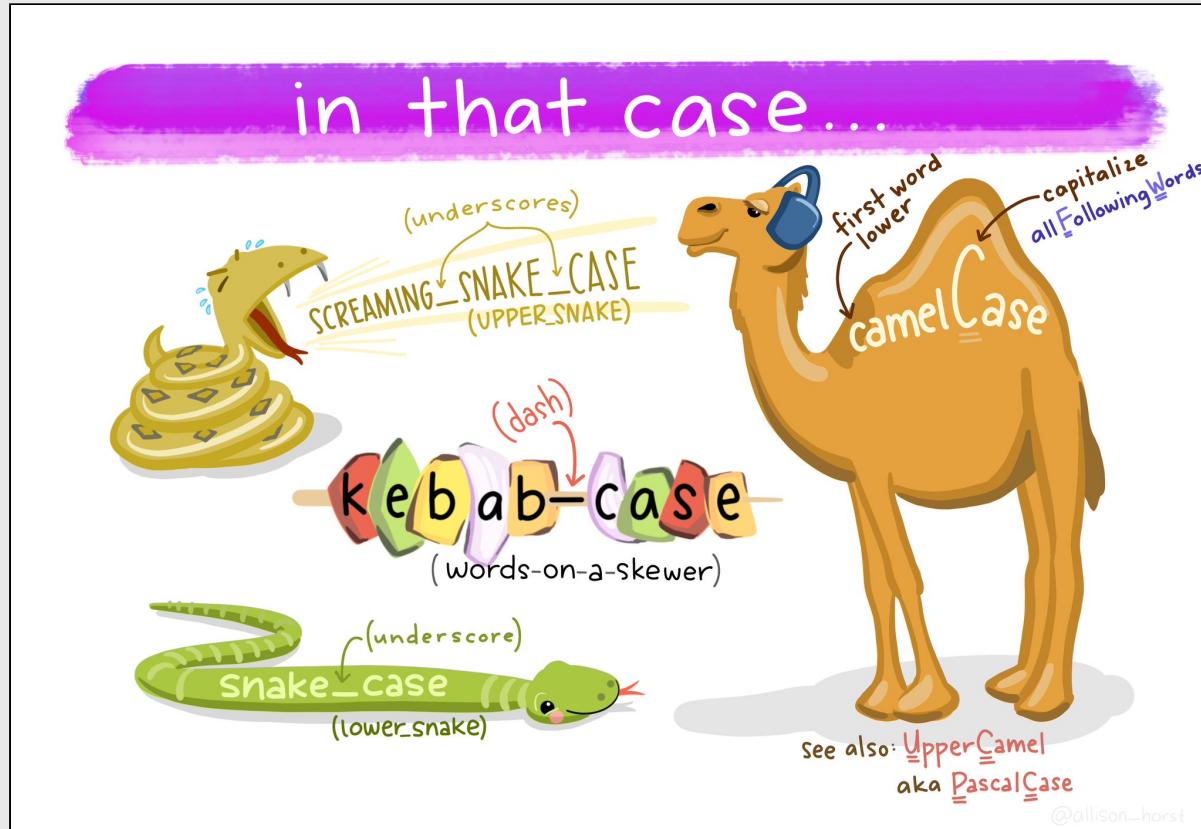
Good variable name:

```
speed <- 42
```

Even better variable name:

```
car_speed_mph <- 42
```

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()
```

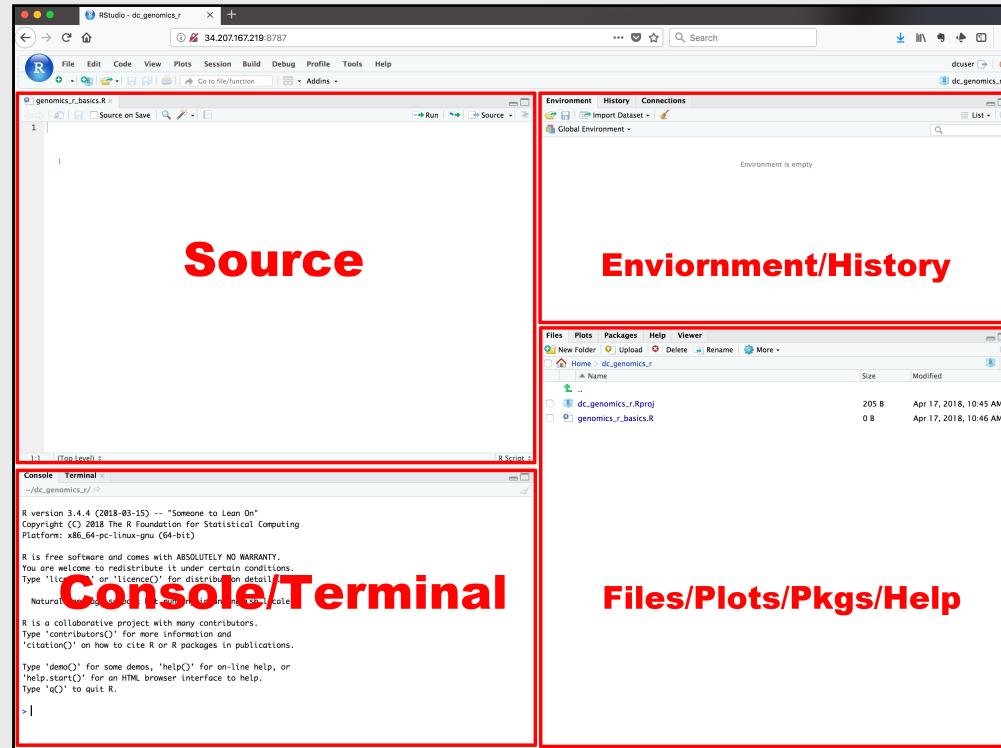
```
## [1] "car_speed_mph"  "days_in_week"   "monthsIn
## [5] "numbeR"          "Number"        "speed"
## [9] "y"               "z"
```

Remove an object by name:

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

```
## [1] "car_speed_mph"  "days_in_week"   "monthsIn
## [5] "Number"         "speed"         "x"
## [9] "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 the question below in it.

10:00

B. Creating & working with objects

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

City	Area (sq. mi.)	Population
San Francisco, CA	46.87	884,363
Chicago, IL	227.63	2,716,450
Washington, DC	61.05	693,972

1. Use the objects you created to 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 with R*

1. Course Introduction
2. Break: Install Course Tools
3. Getting started with R & RStudio
4. Operators & data types
5. 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

Example:

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

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

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

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

Integer division: $\%/\%$

Integer division drops the remainder

What will this return?

```
4 %/% 4
```

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

What will this return?

```
4 %/% 5
```

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

Modulus operator: `%%`

Modulus returns the remainder *after* doing integer division

Example:

```
5 %% 3
```

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

```
3.1415 %% 3
```

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

Modulus operator: `%%`

Modulus returns the remainder *after* doing integer 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" \rightarrow)
- `/%%` returns everything to the *left* (\leftarrow "chop")
- The "chop" point is always just to the *right* of the chopping digit:

Example	"Chop" point	"Chop" point description
<code>1234 %% 1</code>	<code>1234 </code>	Right of the <code>1</code> 's digit
<code>1234 %% 10</code>	<code>123 4</code>	Right of the <code>10</code> 's digit
<code>1234 %% 100</code>	<code>12 34</code>	Right of the <code>100</code> 's digit
<code>1234 %% 1000</code>	<code>1 234</code>	Right of the <code>1,000</code> 's digit
<code>1234 %% 10000</code>	<code> 1234</code>	Right of the <code>10,000</code> '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: !

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
```

Comparing things: Logical operators

Make multiple comparisons with:

- And: &
- Or: |
- Not: !

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

Make multiple comparisons with:

- And: &
- Or: |
- Not: !

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

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

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

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

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

Comparing things: Logical operators

And: &

"Are any of the statements
FALSE"?

```
(2 == 2) & (2 == 3) & (4 == 4)
```

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

Or: |

"Are any of the statements
TRUE"?

```
(2 == 2) | (2 == 3) | (4 == 7)
```

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

& > |

The & operator takes
precedence over |

```
(2 == 2) | (2 == 3) & (4 == 7)
```

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

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
```

10:00

Your turn

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 assess the type of any variable:

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

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

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

Example:

```
typeof(3)
```

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

Even though it *looks* like an integer, 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
```

Example: Is 2 greater than 1?

```
1 < 2
```

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

Special values

Infinity: Inf

*really big
numbers*

```
1/0
```

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

Not available: NA

value is "missing"

```
NA
```

Not a Number:

NaN

"not a number"

```
0/0
```

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

No value: NULL

no value

whatsoever

05 : 00

Your turn

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

Final points

- 1) HW 1 Preview
- ! Read carefully!
- 2) Please take this survey