Introduction to R

INFO 201

Joel Ross

Winter 2017

https://slides.com/joelross/info201w17-r-basics/live

Today's Objectives

By the end of class, you should

- Feel confident with version control basics.
- Understand how to to **write**, **execute**, and **debug** computer programs in the R language
- Store data in **variables**
- Utilize **functions** to manipulate data

Warmup: Command-Line

Use the command-line to perform the following steps to organize your code for this class:

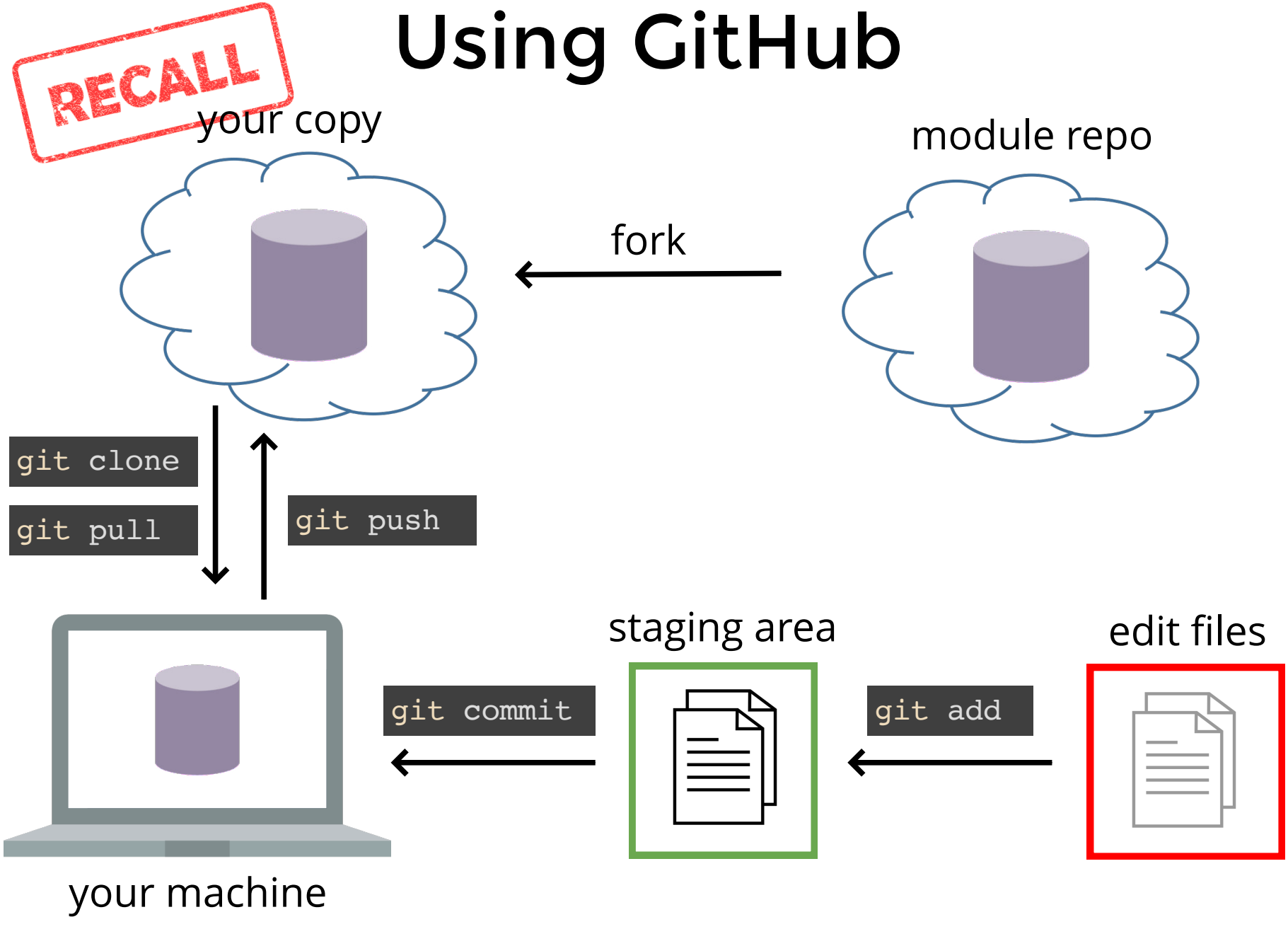
1. Change directory into a general folder where you'll keep all your class work (`Documents`, `Desktop`, `~` (Home), etc). Pick something you'll remember!

2. Make a new directory called `info201`

3. **Copy** any code repositories you've created (assignments, modules, etc) into this folder.
   - It's easiest to navigate to the parent of those folders and then `cp` them into the `info201` folder

4. Once you're done, delete the old copies of the repos (use Finder/File Explorer---it's safer!)

Using GitHub



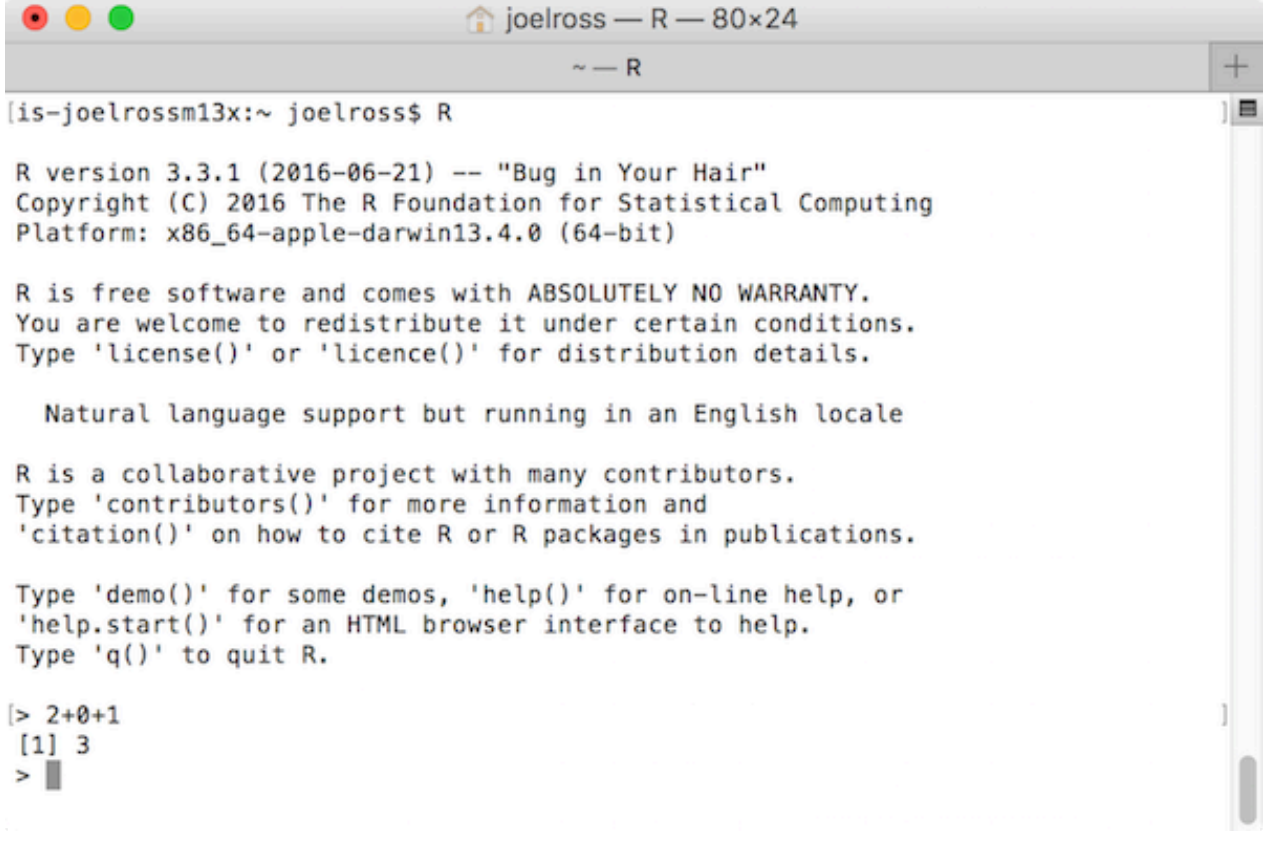
Fork and clone **module-5**

(into your new `info201` folder!)

A statistical programming language

Command-Line R

It is possible to run the **R interpreter** from the command-line. This will let you specify commands in the **R** language for the computer to *interpret* and *execute*.

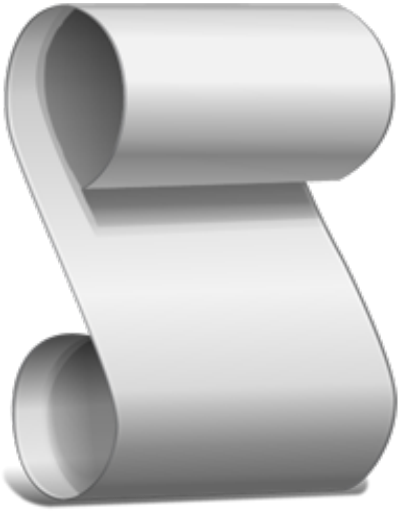


Script

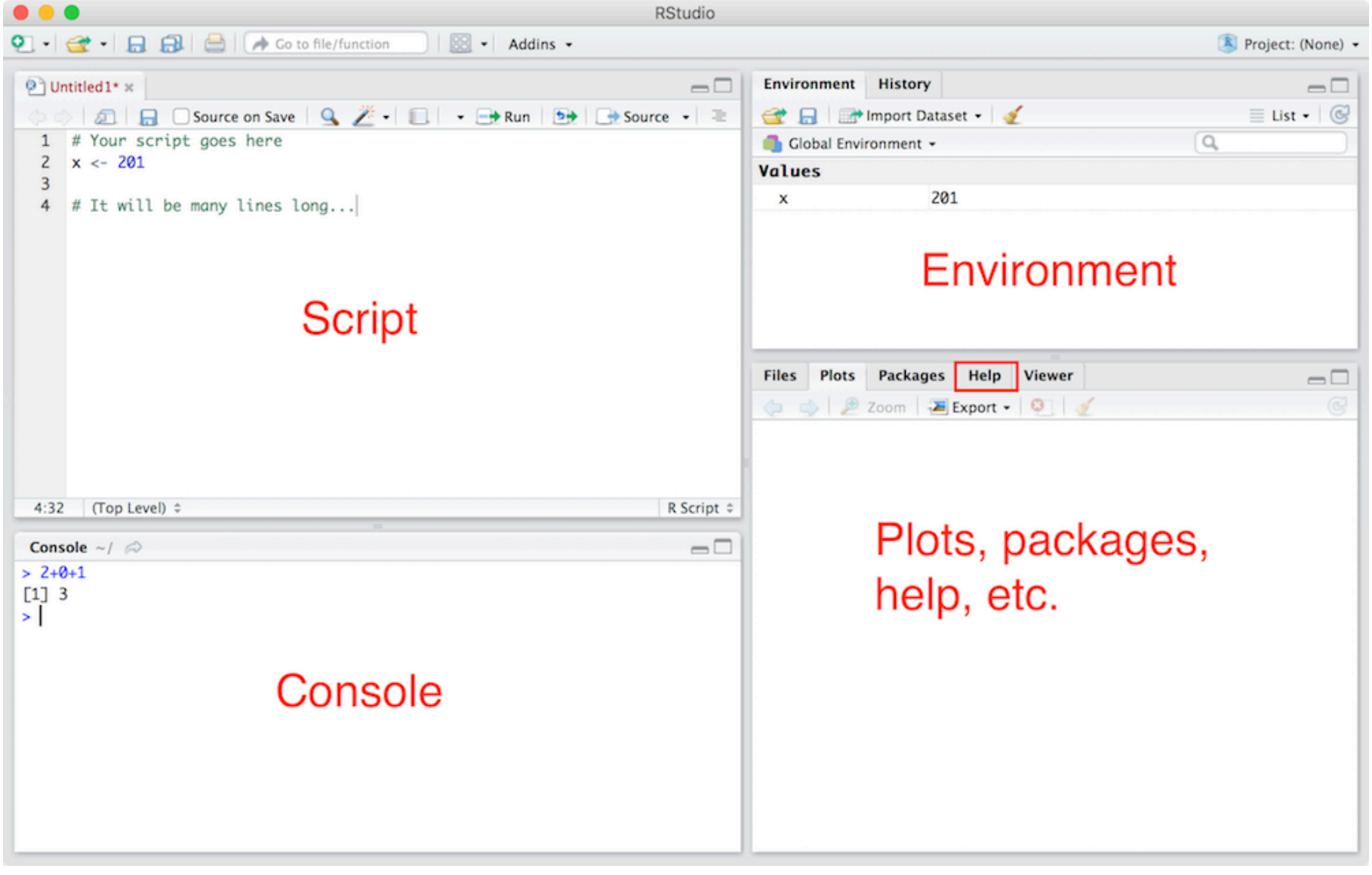
A list of instructions (in order) for the computer to **execute**; a "script" that our program should follow.

- Instructions are written in a programming language (like R) and **interpreted** by the computer. We save **R** scripts in files ending with `.R`

- A script represents an **algorithm** for solving a problem!



RStudio


RStudio
Go to file/function
Addins
Project: (None)
Untitled1*
Source on Save
Run
Source
# Your script goes here
x <- 201
# It will be many lines long...
Script
4:32
(Top Level)
R Script
Console ~/
> 2+0+1
[1] 3
>
Console
Environment
History
Import Dataset
List
Global Environment
Values
x
201
Environment
Files
Plots
Packages
Help
Viewer
Zoom
Export
Plots, packages, help, etc.

Using RStudio

- Type your script into the **Script** pane
  - Press `cmd/ctrl-enter` to run selected statements/code (`cmd/ctrl-a` to select all)

- See output of commands in the **Console** pane

- View data information in the **Environment** pane

- Find additional information in the **Files/Packages/etc.** pane.

R Syntax

Comments

Text that is **not** read by the interpreter.

Anything after a `#` is skipped (until the end of the line)

Used to give more information to the **human**

```
# Add some numbers
2+0+1  # 3
```

Comments should include information that is **not** otherwise in the program.

Include lots of comments!

Printing

Use `print()` to print whatever is in the parentheses to the console. This is an example of a **function.**

```
# My first program
print("Hello world!")
```

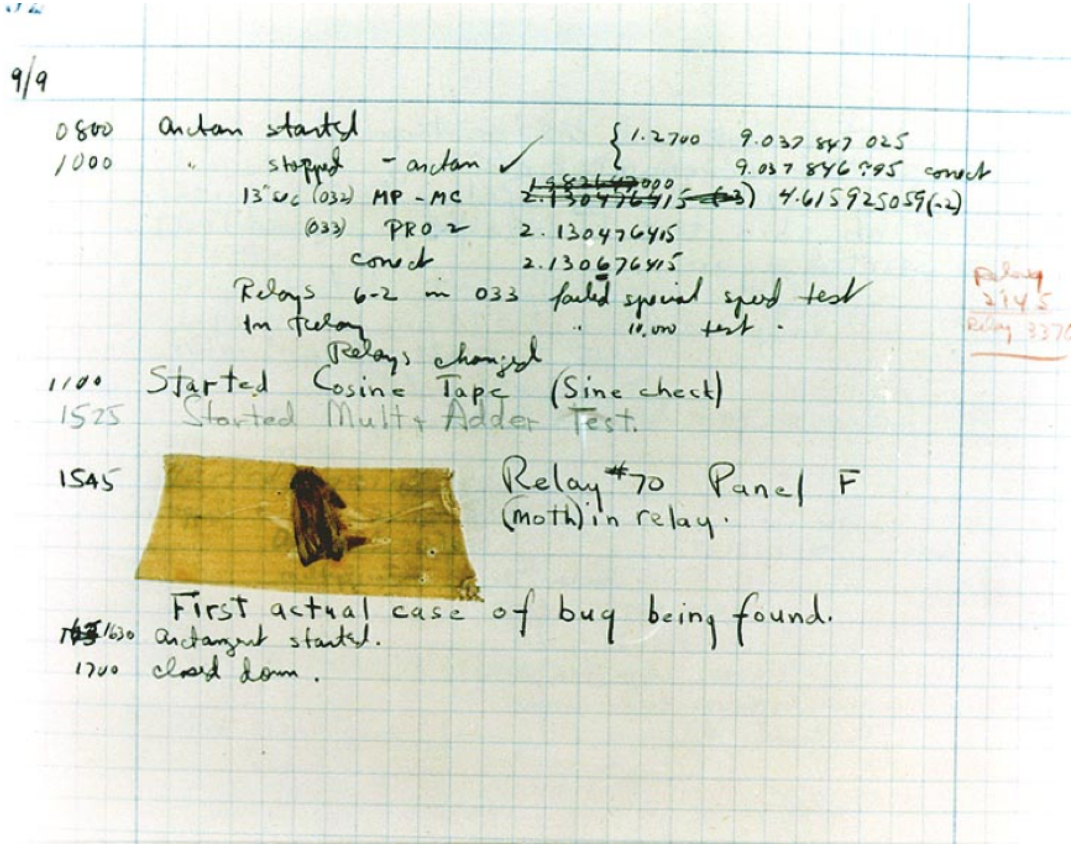
What could go wrong?

```
# My first program
print "Hello world!"
```

```
# My first program
print("Hello world!)
```

```
# My first program
print("Hello wold!")
```

Computer Bugs



First computer bug (1946)

Admiral Grace Hopper

Errors

Syntax Error

- An error in the use of the R language. A problem with how you said something.
- Interpreter will error at the site of the problem.

Logical Error

- An error in the algorithm you used. A problem with what you said to do.
- Interpreter will error, but possibly <u>after</u> the problem.

Semantic Error

- An error in your approach to solve a problem.
- Interpreter will not error, but will not do what you want.

Variables

RECALL

$$y = x^2 + 3x + 7$$

Variables

A label that refers to a **value** (data)

assignment

```
my.num <- 201
```

variable
(label)

value
(data)

Using Variables

Once we have a variable, we can use that label to **refer** to a value (in place of a value)

```
x <- 4  # x refers to 4
y <- x  # y refers to the value of x (4)
z <- y + 7  # y refers to sum of y and 4 (11)

z <- z + 1  # take z, add 1, and store result
            # back in z
```

*When a label is on the **left**, it means the **variable**.*

*When a label is on the **right**, it means the **value**.*

Good variable names describe the data

```
# valid program, but what does it mean?
a <- 35.0
b <- 12.50
c <- a*b

# much better!
hours <- 35.0
pay.rate <- 12.50
earnings <- hours * rate

# Get out.
x1q3z9ahd <- 35.0
x1q3z9afd <- 12.50
x1q3p9afd <- x1q3z9ahd * x1q3z9afd
```

Data Types

Numeric Data

Used to store numbers (whole or decimal). The default computational data type.

```
x <- 2  # whole number
y <- 3.5  # decimal (floating point) number

# can perform mathematical operations
z <- x + y
```

operation operands

```
# Use parentheses to enforce order of operations
z <- 3*(x-y)^2
```

Character Data

Used to store ***strings*** of characters (letters, punctuation, symbols, etc). Written in single or double quotes.

```
my.name <- "Joel Ross"  # 'Joel' would be equivalent

# Can include any keyboard symbol (and more!)
course <- "Info 201: Technical Foundations!"
```

Logical (Boolean) Data

Used to store "yes or no" data: `TRUE` or `FALSE`

```
is.lunch.time <- TRUE  # Not the string "TRUE"
is.monday <- FALSE
```

Can produce logical values using relational operators (comparisons):

```
x <- 3
y <- 3.15

# compare numbers
x > y  # FALSE (x IS NOT bigger than y)
x != y  # TRUE (x IS not-equal to y)
x <= y  # TRUE (x IS less-or-equal to y)
```

not an assignment!

written as read: "less than or equal to"

```
# compare strings (based on alphabetical ordering)
"cat" > "dog"  # returns FALSE, "cat" is first
```

Logical (Boolean) Data

Can apply **logical (boolean)** operators to **logical data:**

`&` (and), `|` (or), `!` (not)

```
pet <- "dog"
weather <- "rain"

# pet is "cat" AND weather is "rain"
pet == "cat" & weather == "rain"  # TRUE

# pet is "cat" OR "dog"
pet == "cat" | pet == "dog"  # TRUE

# weather IS NOT "rain"
weather != "rain"  # FALSE; also !(weather == "rain")

# pet is "dog" AND NOT weather is "rain"
pet == "dog" & !(weather == "rain")  # FALSE
```

Module 5 exercise-1

Functions

RECALL

$$f(x) = 4x - 3$$

function name

input (domain)

result (range)

Functions

A named sequence of instructions (lines of code). We **call** a function to do those steps.

```
print("Hello world")
```

function name argument (value)

Functions ***abstract*** *computer programs!*

Functions

Function **arguments** are the "inputs".

```
# prints "Hello+World"
paste("Hello", "World", sep="+")
```

multiple arguments are separated by commas

named args (more later)

```
# rounds 5/7 to the nearest .01
round(5/7, 2)  # 0.71
```

expressions in arguments are evaluated **before** function is executed

Functions

Functions may **return** a value (the "output"). This value must be *stored in a variable* for the machine to use later!

```
# store min value in smallest.number variable
smallest.number <- min(1, 6/8, 4/3)  # 0.75

# use the variable as normal, i.e., for a comparison
min.is.big <- smallest.number > 1  # FALSE

# use functions in expressions
phi <- .5 + sqrt(5)/2  # 1.618...

# pass the result of a function as another argument
# watch out for where the parentheses close!
print(min(1.5, sqrt(3)))  # prints 1.5
```

Built-in Functions

R comes with a number of "built-in" functions

Function Name	Description	Example
`sum(a,b,...)`	Calculates the sum of all input values	`sum(1, 5)` returns `6`
`round(x,digits)`	Rounds the first argument to the given number of digits	`round(3.1415, 3)` returns `3.142`
`toupper(str)`	Returns the characters in lowercase	`toupper("hi there")` returns `"HI THERE"`
`paste(a,b,...)`	*Concatenate* (combine) characters into one value	`paste("hi", "there")` returns `"hi there"`
`nchar(str)`	Counts the number of characters in a string	`nchar("hi there")` returns `8` (space is a character!)
`c(a,b,...)`	*Concatenate* (combine) multiple items into a *vector* (see module-7)	`c(1, 2)` returns `1, 2`
`seq(a,b)`	Return a sequence of numbers from a to b	`seq(1, 5)` returns `1, 2, 3, 4, 5`

Use `?FunctionName` to look up a function!

https://www.rdocumentation.org/

Module 6 exercise-3

Fork and clone!

Loading Functions

We can download and *load* **packages** (a.k.a. "**libraries**") of additional functions to call.

```
# Install `stringr` package (for string funcs)
# Only needs to be done once per machine!
install.packages("stringr")

# Load the package (tell R funcs available for use)
library("stringr") # quotes optional here

sentence <- "The quick brown fox jumped over the lazy dog"

# Get words 2 through 4 of the sentence
word(sentence, 2, 4)  # "quick brown fox"
```

https://www.rdocumentation.org/packages/stringr/versions/1.1.0

Writing Functions

By writing our own functions we can:

- Easily reuse algorithms (write less code!)
- Debug one piece of a program at a time
- Abstract an algorithm to focus on the bigger picture

Defining a Function

optional, comma-separated

```
# general syntax
FunctionName <- function(name) {
   # body: instructions (code) go here
}
```

block

CamelCase, without periods!

```
# A function that says hello to someone
SayHello <- function(name) {
   greeting <- paste("Hello", name)
   print(greeting)
}

SayHello("Joel")
```

Function Arguments

Arguments are **variables** (labels) that are assigned values when the function is called.

```
SayHello <- function(name) {
   greeting <- paste("Hello", name)
   print(greeting)
}
                    name <- "Joel" #implicit
SayHello("Joel")
```

Scope

Variables created inside a function (including the arguments) are **local variables**, and so are only available **inside** the function.

```
MakeFullName <- function(first.name, last.name) {
   full.name <- paste(first.name, last.name)
}

MakeFullName("Joel", "Ross")
print(full.name)  #Error! variable not found
```

Read the error message!

Return Values

Functions can **return** a single value as a result. This is different than printing an output.

```
MakeFullName <- function(first.name, last.name) {
   full.name <- paste(first.name, last.name)

   return(full.name)
}


my.full.name <- MakeFullName("Joel","Ross")
```

func to "return" value
This ends our function.

Remember to give the result
a label to use it later!

Module 6 exercise-1

Module 6 exercise-2

Participation!

To get participation credit:

- **add** and **commit** your exercise work from today (module 6 primarily)
- **push** your changes to GitHub
- **Submit** an assignment on Canvas with a link to your forked GitHub repo
  - If worked with a partner, submit a link to their repo
  - If you forgot to fork & clone, do so now then use:

```
# cd into the cloned repo
cd module6-functions

# change remote bookmark and push
git remote set-url origin https://github.com/USER_NAME/module6-functions
git push origin master
```

Action Items!

- Be comfortable with **module 5 & 6** by Thu
- Assignment 1 due Wednesday night!
- Assignment 2 due next Tuesday

Tuesday: Vectors (pre-read: **module 7**)