

Welcome to Data Wrangling!

- Check the BANA 7025 HW Groups in Canvas, to see which group number you belong to. Find your group's number and sit there. (Please introduce yourself to your group.)
- Download the material for today's class on the course website (<https://xiaoruizhu.github.io/data-wrangling/week-1>).

DATA WRANGLING WITH R

Welcome!

INTRODUCTIONS



FIRST THINGS FIRST...

Please call me Xiaorui or Jeremy.

~~Doctor~~

~~Professor~~

~~Mister~~





In a previous life...



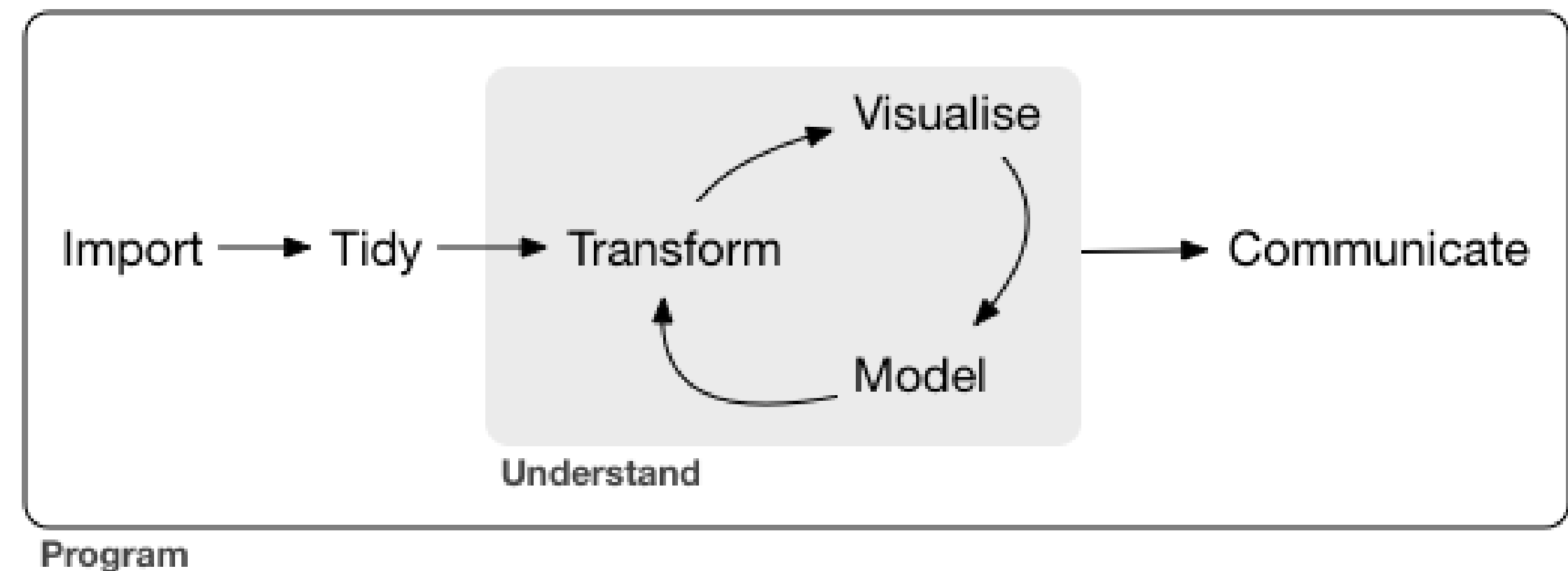
PennState

IMS Health & Quintiles are now



COURSE OBJECTIVES

- Perform your data analysis in a literate programming environment
 - Import and manage structured and unstructured data
 - Manipulate, transform, and summarize your data
 - Join disparate data sources
 - Methodically explore and visualize your data
 - Perform iterative functions
 - Write your own functions
 - Get an introduction to machine learning
- ...all with R!



Lots of hands-on coding exercises



You will be overwhelmed!

Strong proponent of collaborative work!



CLASS MATERIAL

<https://xiaoruizhu.github.io/data-wrangling/>

these modules will also prepare you for your final project.

3. The in-class small group work will teach you to work on a coding task collaboratively and within a constrained time limit and also teach you to assess other people's code.

Material

All required classroom material will be provided in class or online. Any recommended yet optional material will also be provided in the classroom notes.

Schedule

Session	Description
1	Introduction 📁 Intro to data wrangling, R, and course outline Managing your workflow and reproducibility
2	First Date Guidelines for Data 📁 Importing data Understanding the basics of your data
3	Data Structures & Cleaning 📁 Understanding data structures Tidying & preparing data for analysis

- All material online
- Tutorials, resources, & exercises
- Any info regarding class prep

Module 1

Welcome to the first session! This first module will focus on making sure everyone is on the same page regarding the syllabus, project deliverables, and other administrative details. We will also make sure you are up and running with R, RStudio, and Slack.

Class Prep

Please read & work through the following prior to our first class.

Syllabus

If you have not already done so, be sure to read through the [syllabus](#) so that you understand the structure of my classes, the tentative schedule, grading policies, and other pertinent details.

Communication

[Slack](#) will replace e-mail and Blackboard for our course. You will receive an invitation to the [WFU R slack team](#). You may wish to install one of the [apps](#). If you have any questions or concerns your first step should be to go to Slack and post your issue. You and your classmates should be monitoring slack to help each other out. In addition, I will also be watching slack and will chime in when necessary but my hope is that this will be a social process where everyone contributes to knowledge advancement.

- Watch and read the introductory material to get started with Slack [here](#).
- You can also read this [introduction to Slack](#) from one of Kris Shaffer's courses (although this is a completely different course and slack team it provides a nice introduction that you might find useful).
- Sign into the Slack team and post a witty comment in the *Random* channel.

In-Class Material

You can download the materials for class here: 📁

CLASS GRADING

Engagement 10%

- In class discussion
- Canvas discussions
- Small group activities

Homework Assignments 20%

- Must be completed by Mon @ 9am
- Group homeworks!
- Submit via Canvas

Mid-term Project Eval 20%

- HTML report via R Markdown
- Completes items 1.1-3.5 of the grading rubric
- Helps to build your final project

Final Project 50%

- Fully reproducible HTML report
- Imports, cleans, prepares, explores publicly available data
- Helps to build your portfolio

TENTATIVE SCHEDULE

<u>Week</u>	<u>Date</u>	<u>Topic</u>	<u>Readings to complete BEFORE class</u>	<u>Due</u>
1	Oct 18	Introduction & Base R <ul style="list-style-type: none"> • Intro to the course, R, and RStudio • Base R and Data Cleaning 		
2	Oct 25	First Date Guidelines for Data <ul style="list-style-type: none"> • Reproducible documents and workflow management • Importing data and getting to know it 	<ul style="list-style-type: none"> • Chapter 27 of R for Data Science (R4DS), sections 27.1 through 27.5 	Homework #1 (Due Nov 1 @ 9AM)
3	Nov 1	Tidy Data and Data Manipulation <ul style="list-style-type: none"> • Tidying & preparing data for analysis • Data manipulation 	<ul style="list-style-type: none"> • Chapter 12 of R4DS, sections 12.1 through 12.5 • Chapter 5 of R4DS, sections 5.1 through 5.4 	Homework #2 (Due Nov 8 @ 9AM)
4	Nov 8	Data Transformation <ul style="list-style-type: none"> • Relational data • Leveraging the Tidyverse to simplify data wrangling 	<ul style="list-style-type: none"> • Chapter 5 of R4DS, sections 5.5 through 5.7 	Mid-term Project Evaluation (Due Nov 15 @ 12:50PM)
5	Nov 15	Data Visualization <ul style="list-style-type: none"> • Data visualizations 	<ul style="list-style-type: none"> • Chapter 3 of R4DS, all sections 	Homework #3 (Due Nov 22 @ 9AM)
6	Nov 22	Creating Efficient Code <ul style="list-style-type: none"> • Control statements and iterations • Writing functions 	<ul style="list-style-type: none"> • Chapter 19 of R4DS, sections 19.1 through 19.6 • Chapter 21 of R4DS, sections 21.1 through 21.5 	Homework #4 (Due Nov 29 @ 9AM)
7	Nov 29	Introduction to Applied Modeling <ul style="list-style-type: none"> • Unsupervised learning • Supervised learning 	<ul style="list-style-type: none"> • Chapter 22 of R4DS, all sections • Chapter 23 of R4DS, all sections 	Final Project Due (Due Dec 04 @ 11:59PM)

TENTATIVE SCHEDULE

<u>Week</u>	<u>Date</u>	<u>Topic</u>	<u>Readings to complete BEFORE class</u>	<u>Due</u>	
1	Oct 18	Introduction & Base R <ul style="list-style-type: none"> • Intro to the course, R, and RStudio 			
2					Homework #1 (Due Nov 1 @ 9AM)
3					Homework #2 (Due Nov 8 @ 9AM)
4					Midterm Project Evaluation (Due Nov 15 @ 12:50PM)
5					Homework #3 (Due Nov 22 @ 9AM)
6					Homework #4 (Due Nov 29 @ 9AM)
6	Nov 22	Program <ul style="list-style-type: none"> • Control statements and iterations • Writing functions 	<ul style="list-style-type: none"> • Chapter 19 of R4DS, sections 19.1 through 19.6 • Chapter 21 of R4DS, sections 21.1 through 21.5 	Homework #4 (Due Nov 29 @ 9AM)	
7	Nov 29	Introduction to Applied Modeling <ul style="list-style-type: none"> • Unsupervised learning • Supervised learning 	<ul style="list-style-type: none"> • Chapter 22 of R4DS, all sections • Chapter 23 of R4DS, all sections 	Final Project Due (Due Dec 04 @ 11:59PM)	

COMMUNICATION

Teams (and Canvas)

- Different channels for different weeks
- Teams is the first place to go to ask questions
- Share code, scripts, files, resources
- ~~Talk bad about the instructor~~
- I check multiple times per day but am not on continuously

PROGRAMMING & ANALYSIS



Download and install R, a free software environment for statistical computing and graphics from CRAN, the Comprehensive R Archive Network. It is highly recommended to install a precompiled binary distribution for your operating system; follow these instructions:

1. Go to <https://cran.r-project.org/>
2. Click “Download R for Mac/Windows”
3. Download the appropriate file:
 - Windows users click Base, and download the installer for the latest R version
 - Mac users select the file R-3.X.X.pkg that aligns with your OS version
4. Follow the instructions of the installer.

PROGRAMMING & ANALYSIS



Install RStudio's IDE (stands for integrated development environment), a powerful user interface for R. RStudio includes a text editor, so you do not have to install another stand-alone editor. Follow these instructions:

1. Go to RStudio for desktop
<https://www.rstudio.com/products/rstudio/download/>
2. Select the install file for your OS
3. Follow the instructions of the installer.

There are other R IDE's available: Emacs, Microsoft R Open, Notepad++, etc; however, I have found RStudio to be my preferred route. When you are done installing RStudio click on the icon.

QUESTIONS ABOUT THE CLASS?



FUNDAMENTALS



OVERVIEW OF THE RSTUDIO IDE

Script files

- Saves your script
- Allows code & comments
- Can have multiple files open at a time

Console/Command line

- Can use as calculator
- Does not save code
- This is where your output is displayed

The screenshot displays the RStudio IDE interface with four main panels highlighted by colored boxes:

- Script files (green box):** Shows two open R script files: `initial_functions.R` and `plot_functions.R`. The active file, `initial_functions.R`, contains R code for a function `block_summary` that takes parameters `t`, `m`, `n`, and `r`, and returns a summary of production units.
- Workspace environment (blue box):** Shows the current environment with variables `m` (10), `n` (100), `r` (0.77), and `t` (5). It also lists several functions defined in the workspace, including `block_summary`, `cum_appx`, `cum_exact`, `lc_rate`, `midpoint`, `natural_slope`, and `unit_curve`.
- Console/Command line (red box):** Shows the output of the `block_summary` function call, displaying the number of block hours, midpoint unit, and midpoint hours. It also shows the output of a simple calculator-like operation: `> m + n` resulting in `[1] 110` and `> m + n * t^r` resulting in `[1] 355.3082`.
- Misc (grey box):** Shows the R Documentation for the `mean` function, including its description, usage, and arguments.

Workspace environment

- Holds your objects
- Can review history

Misc - Displays:

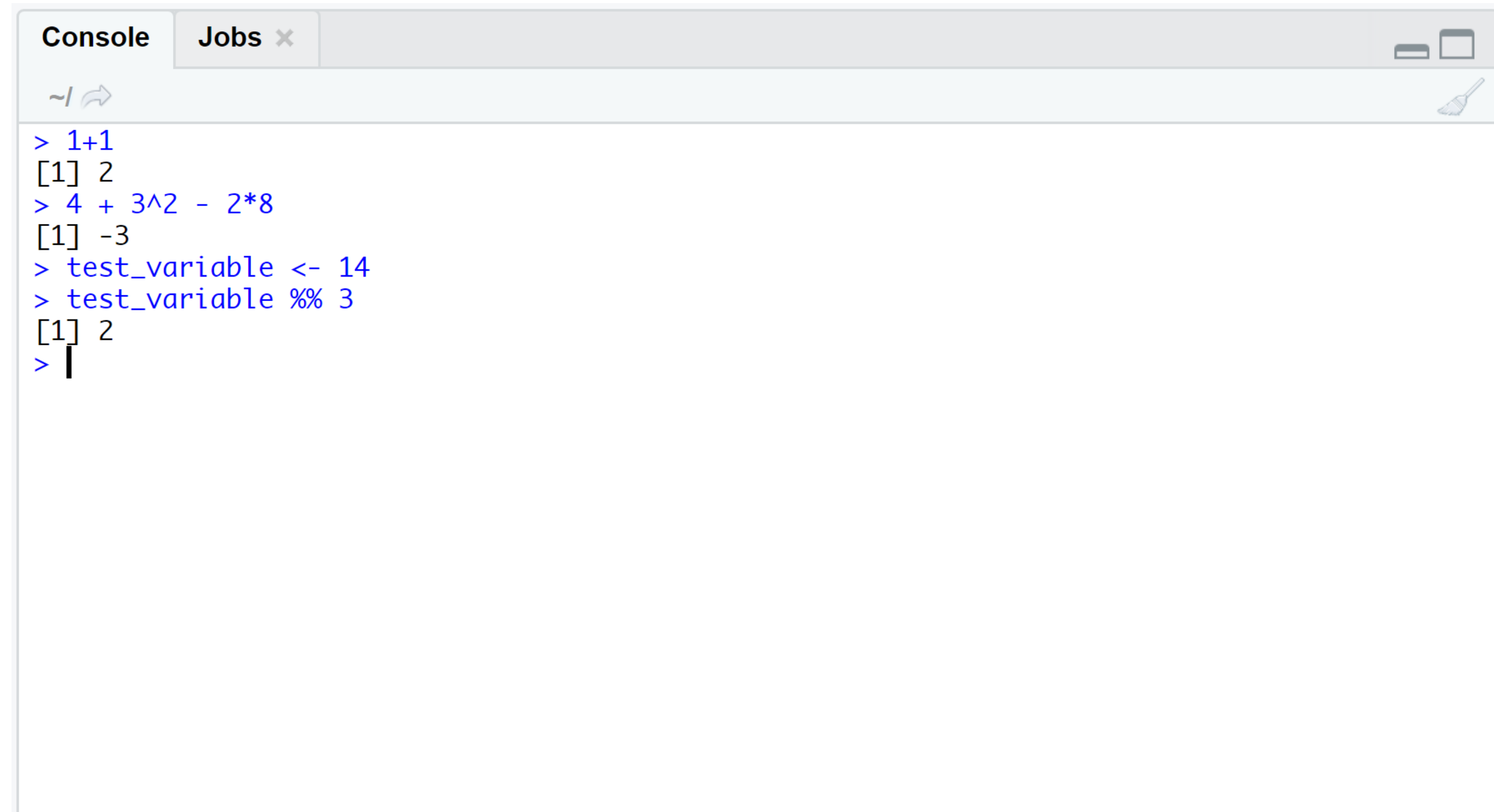
- files in working directory
- plots when produced
- help files/search

Thorough tutorial regarding the RStudio console:

<http://dss.princeton.edu/training/RStudio101.pdf>

THE RSTUDIO CONSOLE

- Can type commands, arithmetic, functions, and other things here
- DOES NOT save code for future RStudio sessions
- `>` symbol means the command you typed begins here
- The `[1]` means the first value of the command's results begins here



```
Console Jobs x
~| ↩
> 1+1
[1] 2
> 4 + 3^2 - 2*8
[1] -3
> test_variable <- 14
> test_variable %% 3
[1] 2
> |
```

YOUR TURN!

1. Type the following command into the console. What does the `:` operator in R do?

2:173

2. Type *exactly* what you see below in the console and then press Return.

`table(iris$Species`

3. Then type a right parenthesis and press Return. What's happening?

SOLUTION

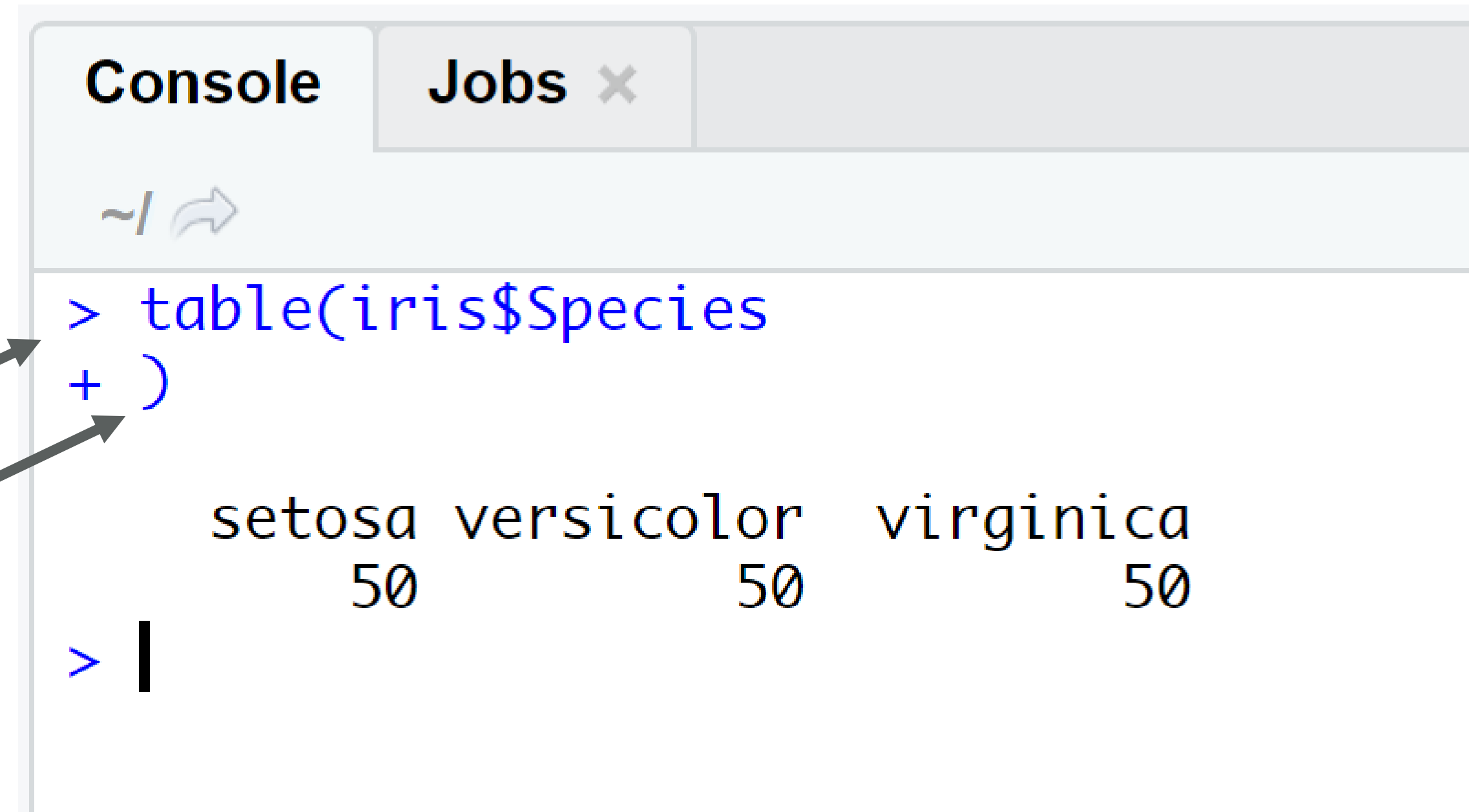
1. *Create a vector of integers starting at 2 and ending with 173. (What do the numbers in brackets mean?)*

```
Console Jobs x
~/
> 2:173
 [1]  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21
[21] 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41
[41] 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61
[61] 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81
[81] 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101
[101] 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121
[121] 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141
[141] 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161
[161] 162 163 164 165 166 167 168 169 170 171 172 173
> |
```

SOLUTION

2. *R displays a + prompt in the console if you type an incomplete command.*

3. *No big deal—Complete the command or press Escape to start a new command.*



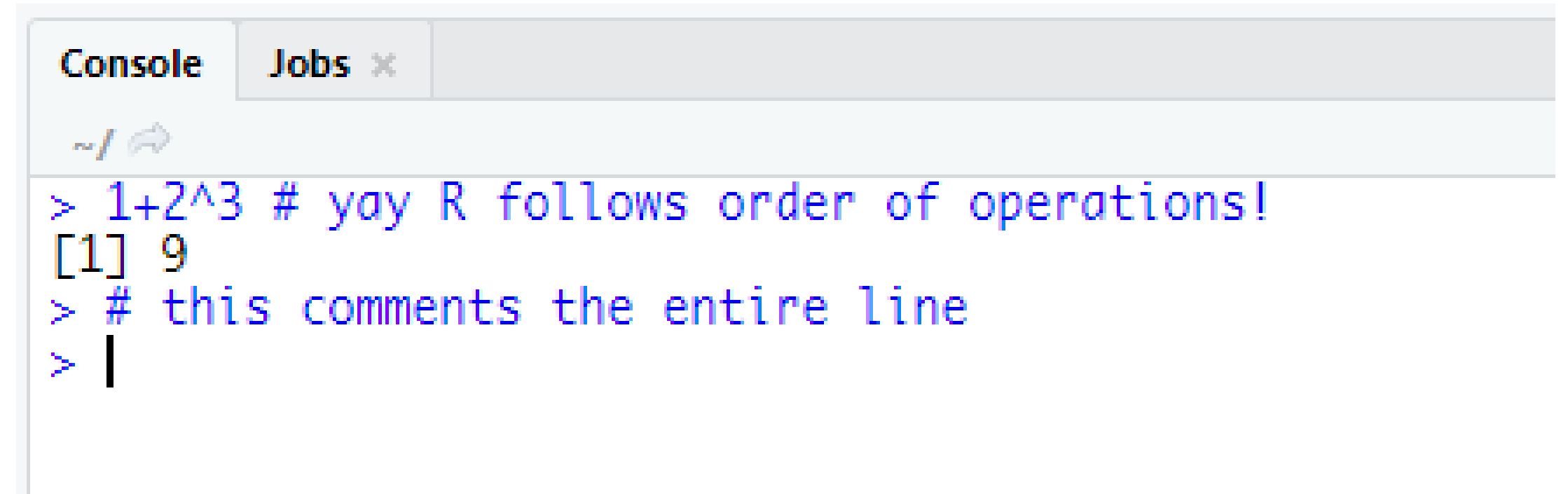
The screenshot shows an R console window with two tabs: "Console" and "Jobs x". The console prompt is "~/" with a right-pointing arrow. The command entered is `> table(iris$Species`. Below this, a `+)` prompt indicates the command is incomplete. The output of the command is displayed as a table:

setosa	versicolor	virginica
50	50	50

Below the output, the prompt `> |` is shown, indicating the command has been completed.

OTHER FUN CONSOLE STUFF

- Use # to comment part of a line or the entire line (get ready to comment way too much this semester)
- Use the STOP button on top of the console (or CTRL + c) to cancel a command
- You can change preferences by going to Tools → Global Options

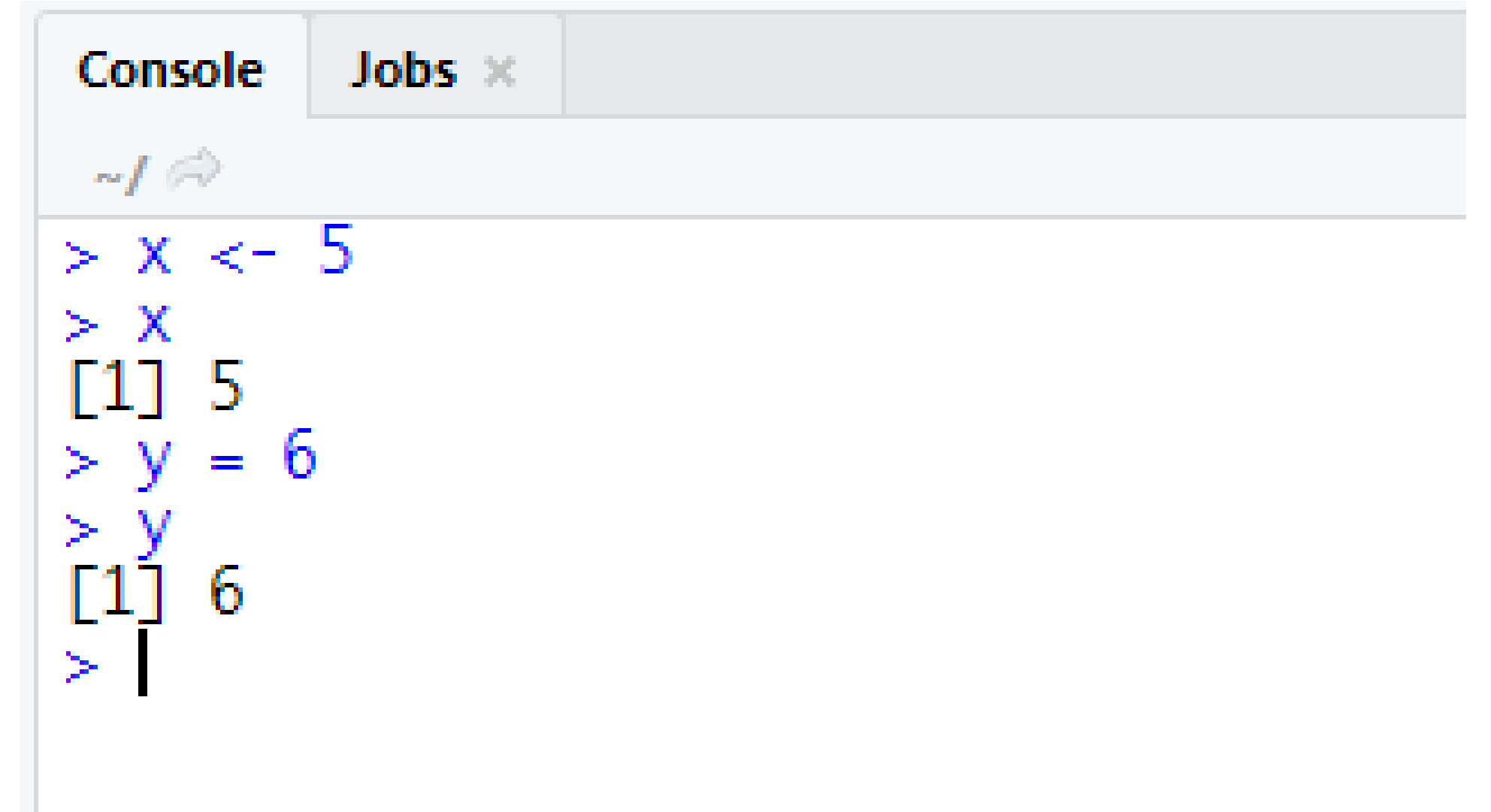


```
Console Jobs x  
~/ ↵  
> 1+2^3 # yay R follows order of operations!  
[1] 9  
> # this comments the entire line  
> |
```


STORING OBJECTS

Use the `<-` assignment operator to store objects as variables.

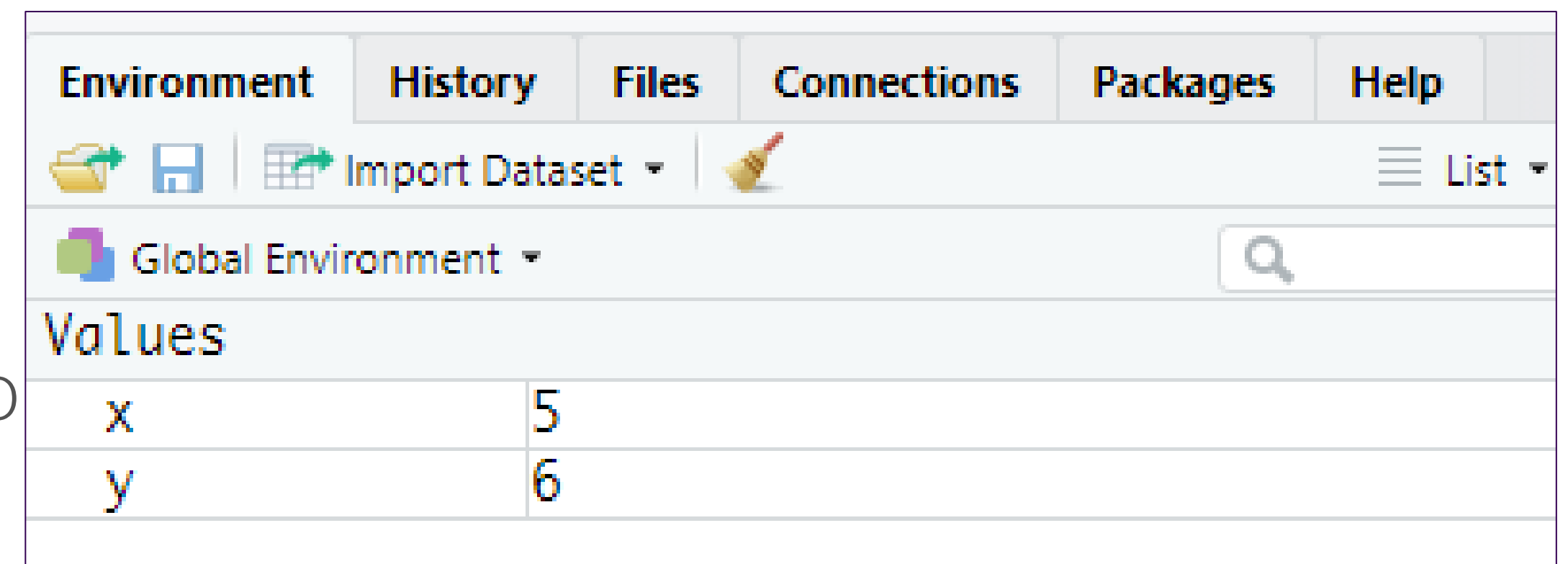
- The [Tidyverse Style Guide](#) suggests using `<-` for the assignment operator instead of `=`
 - Yes, you can use either. But...
 - `<-` is typically used for assigning values to variables
 - `=` is typically used for specifying parameter values in functions
- Type the variable's name and press Return to display the variable's value after assignment



```
Console Jobs x
~/
> x <- 5
> x
[1] 5
> y = 6
> y
[1] 6
> |
```

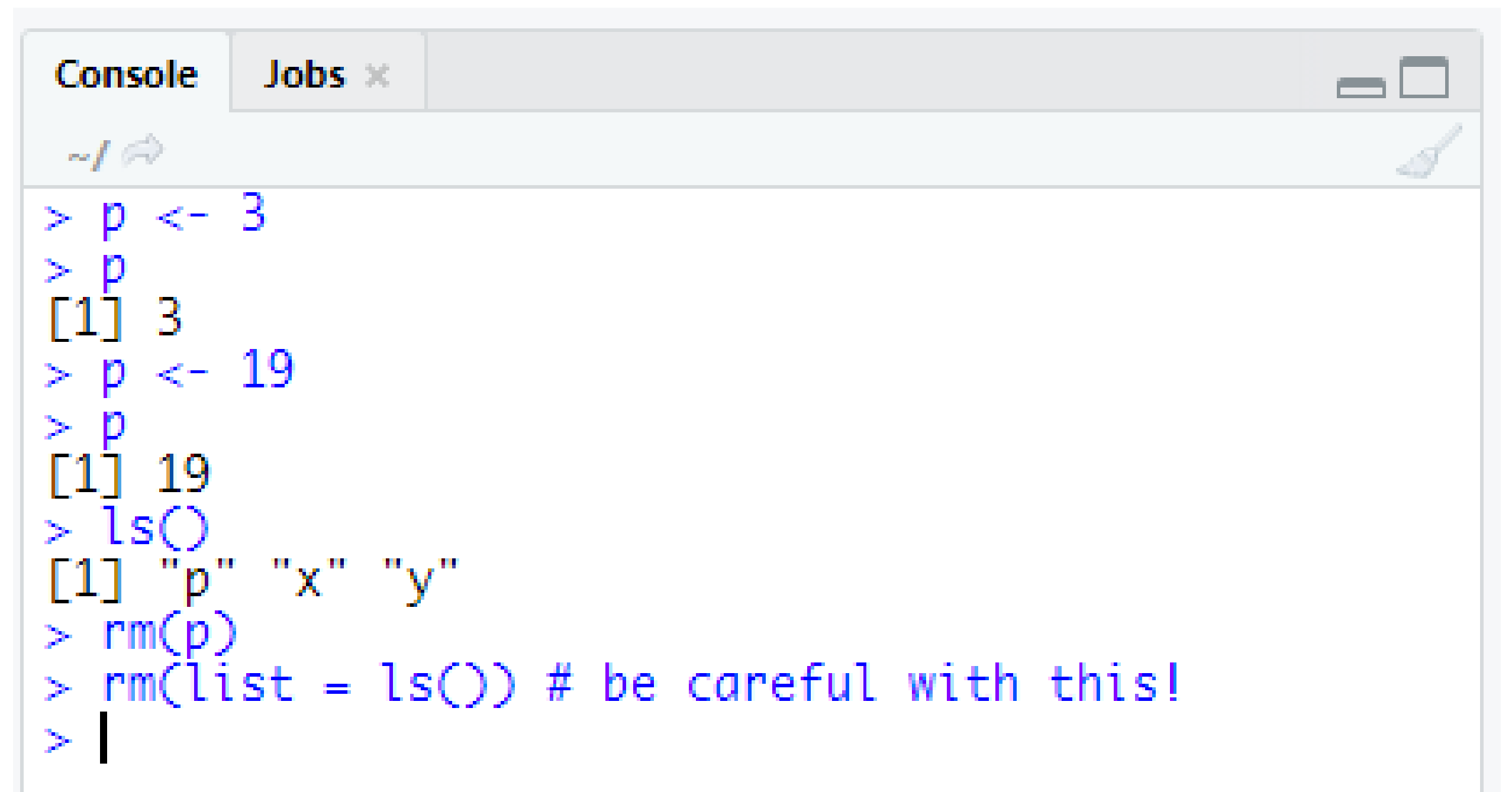
MORE ON STORING OBJECTS

- Variables appear in the Environment pane after assignment
- Variable names must:
 - Begin with a letter or a period (there's more to this)
 - Only contain letters, digits, periods, and underscores
 - Reserved words in R cannot be used (e.g., TRUE)
- Tidyverse Style Guide: use [snake case](#) for variable names



EVEN MORE ON STORING OBJECTS

- Keyboard shortcut for the `<-` assignment operator: ALT + -
- R is case sensitive!
- R overwrites variable names
- Use `ls()` to list all object names
- Use `rm()` to remove objects



```
Console Jobs x
~/ →
> p <- 3
> p
[1] 3
> p <- 19
> p
[1] 19
> ls()
[1] "p" "x" "y"
> rm(p)
> rm(list = ls()) # be careful with this!
> |
```

YOUR TURN!

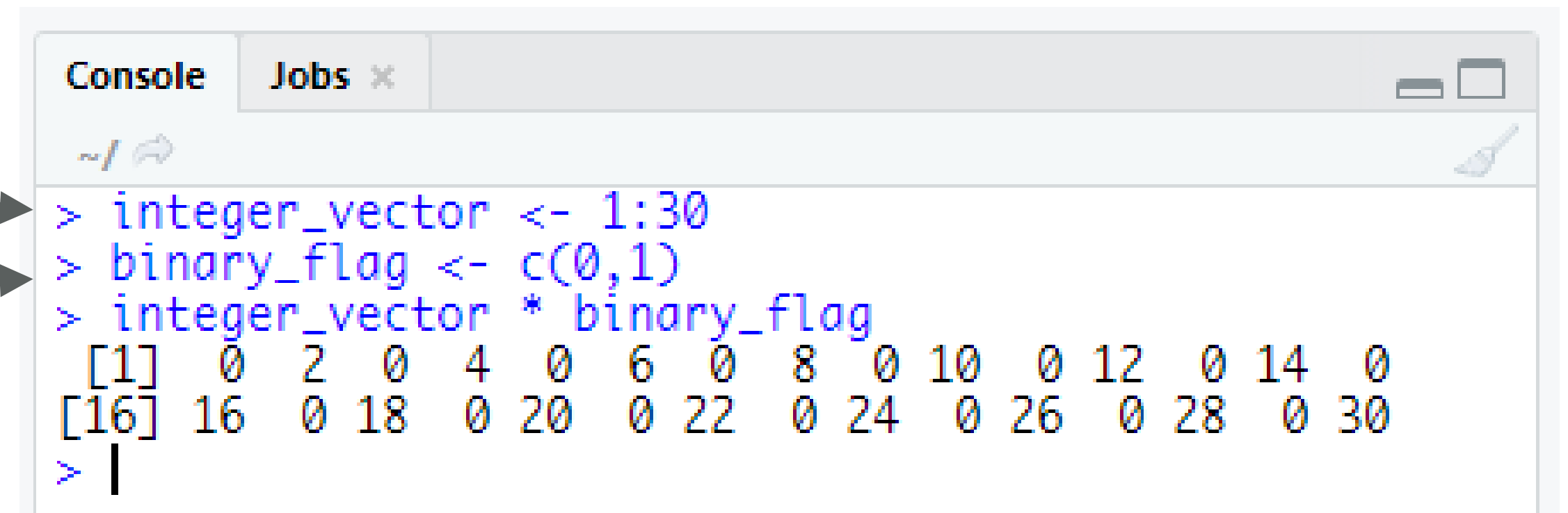
1. Create a vector of integers from 1 to 30 (inclusive). Call it *integer_vector*.
2. Use the command `c(0,1)` to create a separate two-element vector consisting of 0 and 1. Call it *binary_flag*. (What does the `c()` function do?)
3. Multiply your two vectors. What do you notice?

SOLUTION

1.

2.

3. *R performs element-wise execution, recycling vectors as needed.*



```
Console Jobs x
~/
> integer_vector <- 1:30
> binary_flag <- c(0,1)
> integer_vector * binary_flag
[1] 0 2 0 4 0 6 0 8 0 10 0 12 0 14 0
[16] 16 0 18 0 20 0 22 0 24 0 26 0 28 0 30
> |
```

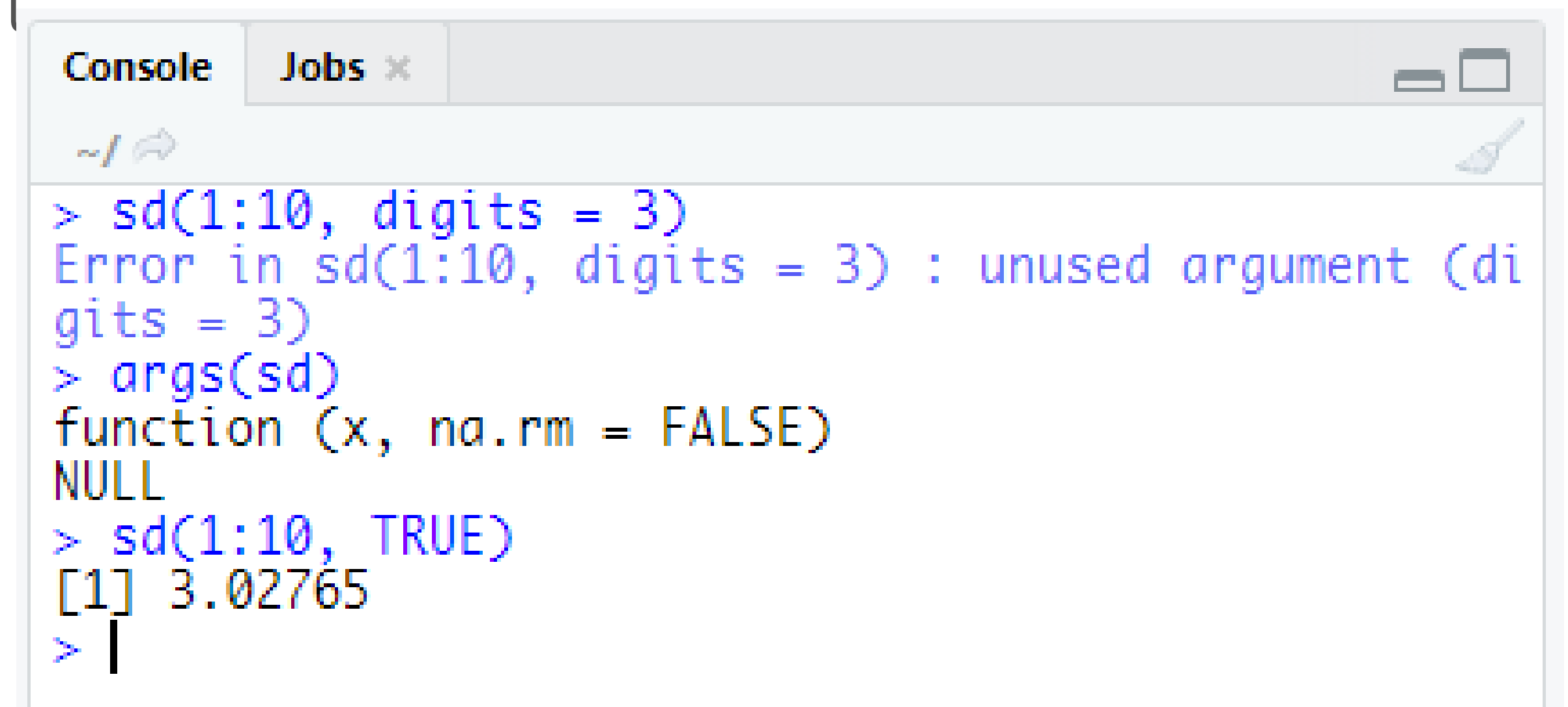
FUNCTIONS

- Type the function name and enter value(s)/data in the parentheses
- Data passed into a function is called the function's **argument**
 - Arguments can be results from another function
 - Use = to specify names of arguments, especially with multiple arguments for readability/QA

```
Console Jobs x
~/ →
> round(2.71828)
[1] 3
> sd(1:10)
[1] 3.02765
> round(sd(1:10))
[1] 3
> round(2.71828, digits = 2)
[1] 2.72
> |
```

MORE ON FUNCTION ARGUMENTS

- R ~~yells at you~~ throws an error when you specify an unknown argument
- Use `args()` to look up a functions arguments
- Unless you specify arguments by name, R matches values to arguments in the function by order



```
Console Jobs x
~/ ↩
> sd(1:10, digits = 3)
Error in sd(1:10, digits = 3) : unused argument (digits = 3)
> args(sd)
function (x, na.rm = FALSE)
NULL
> sd(1:10, TRUE)
[1] 3.02765
> |
```

GETTING HELP

provides details for specific function

help(sqrt) OR ?sqrt

provides examples for said function

example(sqrt)

search help documentation if you forget the function's name

??sqr

External to R:

[Google](#): just add "with R" at the end of any search.

[Stack Overflow](#): a searchable Q&A site oriented toward programming issues. 75% of my answers come from SO

[Cross Validated](#): a searchable Q&A site oriented toward statistical analysis.

[R-bloggers](#): a central hub of content from over 500 bloggers who provide news and tutorials about R.

YOUR TURN!

1. Look at documentation for the **quantile** function. What does the `na.rm` argument do?
2. Use the `quantile` function to find the 25th and 75th percentiles of the integers 1 through 297.

SOLUTION

```
Console Jobs x [min] [max] [close]
~ / ↩ [brush]
> ?quantile
> quantile(1:297, probs = c(0.25, 0.75))
25% 75%
 75 223
> |
```

SCRIPTS

- Place code in a script to save for later use/editing/additions
- Open a new script by going to File → New File → R Script
- Different ways to run code in a script
 - Clicking the Run button to execute the line of code the cursor is on (or the line(s) of code that you highlighted)
 - Ctrl + Enter to run the line/block of code that's highlighted or that the cursor is on
 - Run the entire script by pressing the Source button

SET YOUR WORKING DIRECTORY

```
# get your current working directory
```

```
getwd()
```

```
[1] "/Users/Xiaorui/Dropbox/UC/BANA7025"
```

```
# set your working directory
```

```
setwd("/Users/ Xiaorui/Dropbox/UC/BANA7025")
```

```
getwd()
```

```
[1] "/Users/ Xiaorui/Dropbox/UC/BANA7025"
```

Keeping your files organized is critical

YOUR TURN!

*Set your working directory to the “Session 01” folder
you downloaded for this class.*

PACKAGES

- Everything we've discussed so far is considered "base R", which means we do not require additional capabilities to use these functions
- Packages are collections of custom functions and objects that extend the capabilities of base R
- Examples of packages you may have seen already:
 - dplyr (useful for data manipulation)
 - ggplot2 (useful for visualization)

INSTALLING PACKAGES

The fundamental unit of shareable code is the package.

CRAN: 10,000+

Bioconductor: 1,000+

GitHub: Many more plus beta versions for updated packages not yet published

So how do we install these packages?

```
# install packages from CRAN
install.packages("packagename")
```

```
# install packages from Bioconductor
source("http://bioconductor.org/biocLite.R") # only required the first time
biocLite() # only required the first time
biocLite("packagename")
```

```
# install packages from GitHub
install.packages("devtools") # only required the first time
devtools::install_github("username/packagename")
```

YOUR TURN!

Install these packages from CRAN:

dplyr

nycflights13

SOLUTION

```
install.packages("tidyverse")  
install.packages("nycflights13")
```

```
# alternative  
install.packages(c("tidyverse", "nycflights13"))
```

For a full list of useful packages see this guide: <http://bit.ly/1x9vkzV>

I INSTALLED A PACKAGE. NOW WHAT?

- Installing packages simply downloads them onto your hard drive.
- Now you need to load these packages in order to leverage their capabilities.
- Important difference!
 - You only need to install a package once (assuming the package hasn't changed).
 - You need to load a package every time you start an RStudio session.

LOADING PACKAGES

Loading packages:

```
# load the package to use in the current R session
library(tidyverse)

# use a particular function within a package without loading the package
stringr::str_replace()
```

Getting help on packages:

```
# provides details regarding contents of a package
help(package = "tidyr")

# list vignettes available for a specific package
vignette(package = "tidyr")
browseVignettes("KraljicMatrix")

# view specific vignette
vignette("tidy-data")
```

READ WARNINGS WHEN LOADING PACKAGES!

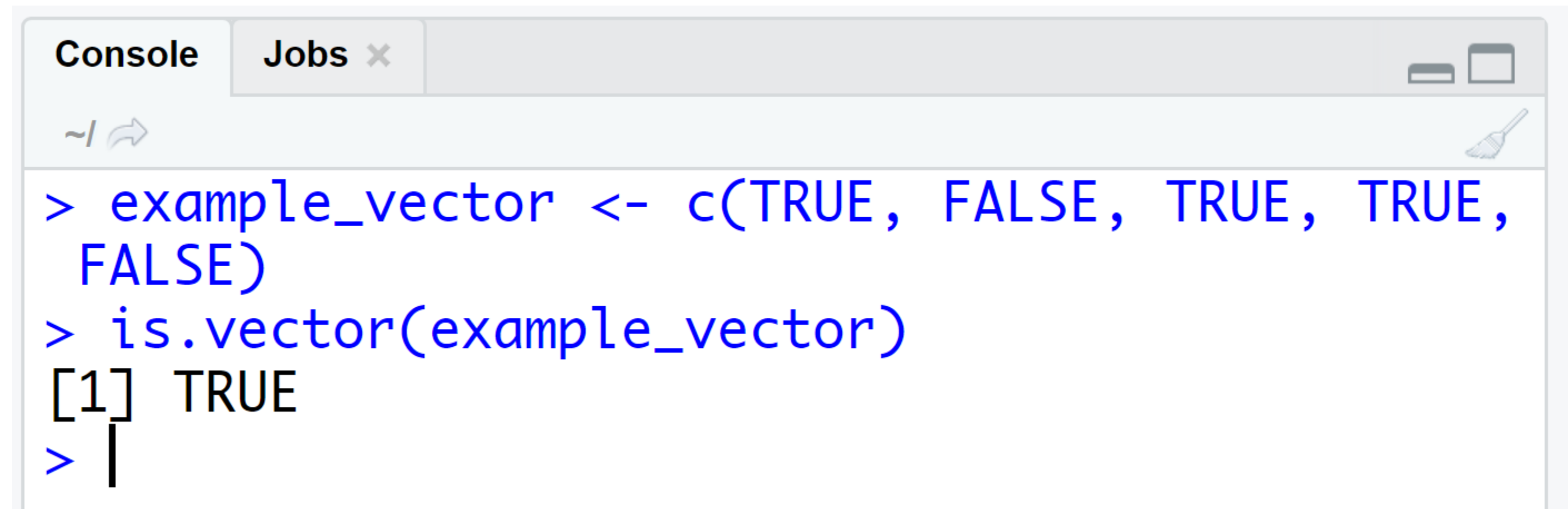
```
Console Jobs x
~/
> library(tidyverse)
-- Attaching packages ----- tidyverse 1.2.1 --
v ggplot2 3.2.1      v purrr  0.3.2
v tibble  2.1.3      v dplyr  0.8.3
v tidyr   0.8.3      v stringr 1.4.0
v readr   1.3.1      v forcats 0.4.0
-- Conflicts ----- tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
> |
```

R OBJECTS



WHAT ARE ATOMIC VECTORS?

- The simplest data structure in R
- Linear vectors of a single data type
- Use `is.vector()` to test if an object is an atomic vector

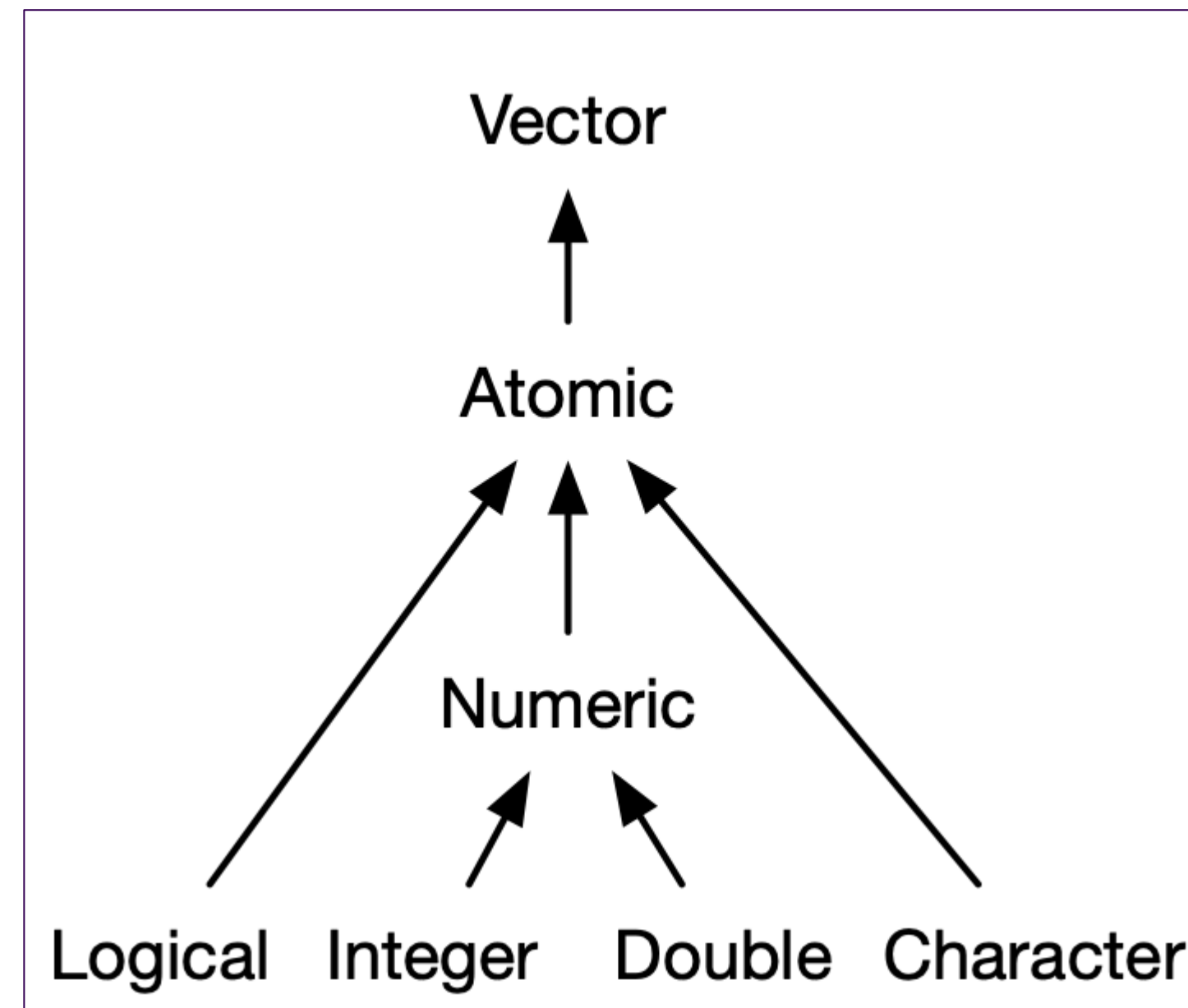


```
Console Jobs x
~/ →
> example_vector <- c(TRUE, FALSE, TRUE, TRUE,
  FALSE)
> is.vector(example_vector)
[1] TRUE
> |
```

WHAT ARE ATOMIC VECTORS?

- Four (actually there are six) types of atomic vectors
- Some R functions (and most people) refer to doubles as numerics
- Use `is.*()` to test if an atomic vector is a certain data type

```
Console Jobs x  
~/ →  
> example_vector <- c(TRUE, FALSE, TRUE, TRUE, FALSE)  
> is.vector(example_vector)  
[1] TRUE  
> is.character(example_vector)  
[1] FALSE
```



CREATING VECTORS

- Use the `c()` function to combine values into a vector (or a list)
- Can use many operators or functions to create numeric vectors
- Integers are stored as numerics by default. Use an uppercase `L` after each integer to store as an integer.

```
# character vector
character_vector <- c("Hello", "how", "are", "you?")

# numeric vector
numeric_vector <- seq(from = 1,
                       to = 39,
                       by = 2)

# a slick way to ensure
# you create an integer vector
# use L after a number
int_vector <- c(1L, 1e4L, -5L)

# logical vector
logical_vector <- c(TRUE, FALSE, TRUE)
```


QA EVERY DAY! CHECK VECTORS

- Use `length()` to find the number of elements a vector has
- Use `typeof()` or `class()` to identify the type of atomic vector

```
Console Jobs x  
~/ →  
> length(numeric_vector)  
[1] 20  
> typeof(int_vector)  
[1] "integer"  
> class(logical_vector)  
[1] "logical"
```

YOUR TURN!

1. Look at documentation for the `runif()` function.
2. In your script for today's class, create a vector with 75 observations that come from a uniform distribution with a minimum value of -3 and a maximum value of 14. (Don't forget to assign this vector to a variable. You pick the name!)
3. Write functions to examine this vector:
 - Is it an atomic vector?
 - What kind of vector is it?
 - How long is the vector?

SOLUTION

```
# look at documentation
```

```
?runif
```

```
help(runif)
```

```
args(runif)
```

```
# create vector
```

```
runif_vector <- runif(n = 75,  
                     min = -3,  
                     max = 14)
```

```
# examine the vector
```

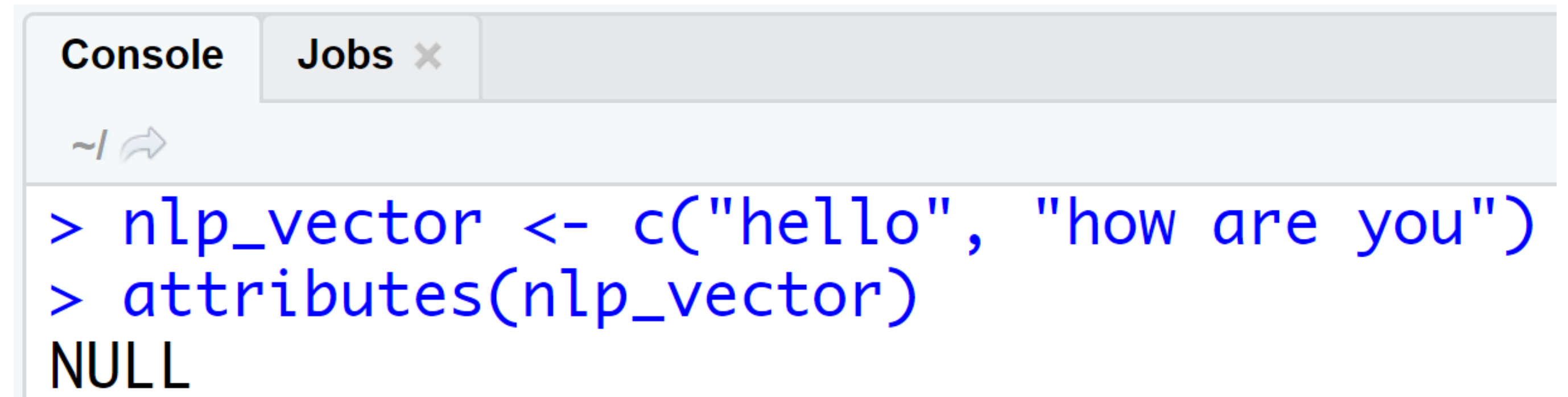
```
is.vector(runif_vector)
```

```
typeof(runif_vector) OR class(runif_vector)
```

```
length(runif_vector)
```

ATTRIBUTES

- Attributes are information you can attach to an R object
- Attributes will not appear when displaying an object or affect object values
- Check for attributes with `attributes()`



```
Console Jobs x  
~/ ↩  
> nlp_vector <- c("hello", "how are you")  
> attributes(nlp_vector)  
NULL
```

THE NAMES ATTRIBUTE

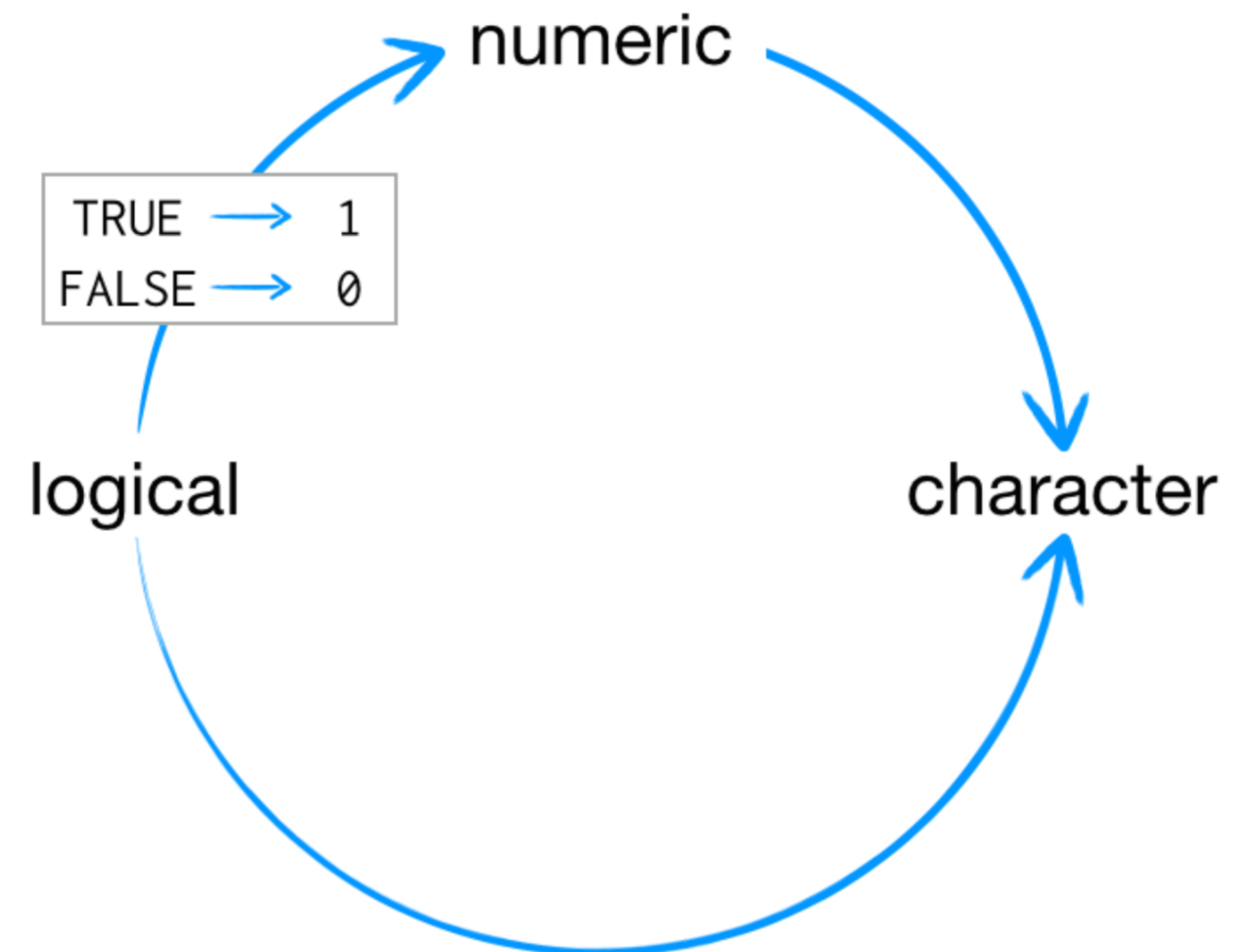
- View the names of each element with `names()`
- Assign or change names with `names()` (usually) a character vector
- Remove value names by assigning a `NULL` (missing) value to `names`

```
Console Jobs x
~/ ↩
> names(nlp_vector)
NULL
> names(nlp_vector) <- c("greeting", "followup question")
> attributes(nlp_vector)
$names
[1] "greeting"          "followup question"

> names(nlp_vector) <- NULL
> names(nlp_vector)
```

VECTOR COERCION

- Combining different data types or forcing functions on certain data types results in **coercion**
- Vector coercion in R always follows a predetermined procedure



VECTOR COERCION EXAMPLES

Forcing functions on data types coerces vectors

```
Console Jobs x  
~/ ↩  
> as.character(43)  
[1] "43"  
> as.logical(0)  
[1] FALSE  
> as.numeric(TRUE)  
[1] 1
```

Vector coercion isn't always a bad thing!

```
Console Jobs x  
~/ ↩  
> # math with logicals  
> logical_vector  
[1] TRUE FALSE TRUE  
> sum(logical_vector)  
[1] 2  
> mean(logical_vector)  
[1] 0.6666667
```

FUNDAMENTAL DATA STRUCTURE

Matrix

- Two-dimensional array
- Numeric data only
- Use the `matrix()` function to create a matrix

```
Console ~/ ↵
> VADeaths
      Rural Male Rural Female Urban Male Urban Female
50-54      11.7      8.7      15.4      8.4
55-59      18.1     11.7     24.3     13.6
60-64      26.9     20.3     37.0     19.3
65-69      41.0     30.9     54.6     35.1
70-74      66.0     54.3     71.1     50.0
> class(VADeaths)
[1] "matrix"
> nrow(VADeaths)
[1] 5
> ncol(VADeaths)
[1] 4
> dim(VADeaths)
[1] 5 4
```


YOUR TURN!

1. *Look at documentation for the `matrix()` function.*
2. *Create a 5x4 matrix of the integers from 1 to 20, filling by rows. Save the matrix as a variable called `example_matrix`.*
3. *Examine the names, class, and dimensions of this matrix.*

SOLUTION

```
# documentation for matrix()  
?matrix
```

```
# create the matrix  
example_matrix <- matrix(  
  data = 1:20,  
  nrow = 5,  
  ncol = 4,  
  byrow = TRUE  
)
```

```
# examine attributes  
names(example_matrix)  
class(example_matrix)  
dim(example_matrix)
```

FUNDAMENTAL DATA STRUCTURE

List

- One dimension
- Each element can be its own object (a vector, matrix, data frame, or even a list)
- Use the list() function to create a list
- You'll see lists frequently with linear models

Console ~/ ↗

```
> # example of list
> list(
+   seq(3, 30, 3),
+   letters,
+   matrix(1:6, nrow = 2)
+ )
```

```
[[1]]
 [1]  3  6  9 12 15 18 21 24 27 30
```

```
[[2]]
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l"
[13] "m" "n" "o" "p" "q" "r" "s" "t" "u" "v" "w" "x"
[25] "y" "z"
```

```
[[3]]
      [,1] [,2] [,3]
 [1,]    1    3    5
 [2,]    2    4    6
```

FUNDAMENTAL DATA STRUCTURE

Data Frame

- Two-dimension version of a list (think of a spreadsheet)
- Named list of vectors with specific attributes
- Each vector become a column
- Vectors (i.e., columns) must be the same length in a data frame

```
Console ~/ ↵  
> iris_abbreviated  
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species  
1          5.1         3.5         1.4         0.2   setosa  
2          4.9         3.0         1.4         0.2   setosa  
3          4.7         3.2         1.3         0.2   setosa  
4          4.6         3.1         1.5         0.2   setosa  
5          5.0         3.6         1.4         0.2   setosa  
6          5.4         3.9         1.7         0.4   setosa  
7          4.6         3.4         1.4         0.3   setosa  
8          5.0         3.4         1.5         0.2   setosa  
9          4.4         2.9         1.4         0.2   setosa  
10         4.9         3.1         1.5         0.1   setosa
```

FUNDAMENTAL DATA STRUCTURE

Data Frame

- Use the `data.frame()` function to create a data frame
- Data frames have a special `data.frame` class
- Use `str()` to examine the structure of a data frame

```
# example of data frame
cool_df <- data.frame(
  # columns of data frame
  observation = c("a", "b", "c", "d"),
  rand_norm_value = rnorm(4),
  exclude_flag = c(TRUE, FALSE, FALSE, TRUE),
  # other options for data frame
  row.names = NULL,
  stringsAsFactors = FALSE
)
```

```
# examine data frame
```

```
cool_df
class(cool_df)
str(cool_df)
```

```
> cool_df
  observation rand_norm_value exclude_flag
1          a      0.2200598          TRUE
2          b      0.4772280          FALSE
3          c     -1.5238578          FALSE
4          d      2.2312713          TRUE

> class(cool_df)
[1] "data.frame"

> str(cool_df)
'data.frame':  4 obs. of  3 variables:
 $ observation      : chr  "a" "b" "c" "d"
 $ rand_norm_value: num  0.22 0.477 -1.524 2.231
 $ exclude_flag    : logi  TRUE FALSE FALSE TRUE
```

IMPORTING AND EXPORTING DATA

Importing Data

- Use the `read.csv()` function to read in `.csv` files
- Other `read.*` functions exist for various file formats
- Watch the `stringsAsFactors` argument!

Exporting Data

- Use the `write.csv()` function to save `.csv` files to your hard drive
- Watch the `row.names` argument!

STYLE GUIDE

I am a stickler for nicely formatted code

STYLE GUIDE

Naming scripts:

basic-stuff.r
detail.r

weather-analysis.R
emerson-text-analysis.R

Which is good, which is bad?

STYLE GUIDE

Naming scripts:

Bad

```
basic-stuff.r  
detail.r
```

Good

```
weather-analysis.R  
emerson-text-analysis.R
```

Which is good, which is bad?

STYLE GUIDE

Naming objects:

naming_convention
naming.convention

namingconvention
namingConvention
NamingConvention

Which is good, which is bad?

STYLE GUIDE

Naming objects:

Good

```
naming_convention  
naming.convention
```

Bad

```
namingconvention  
namingConvention  
NamingConvention
```

Which is good, which is bad?

STYLE GUIDE

Code spacing:

```
average<-mean(feet/12+inches,na.rm=TRUE)
```

```
average <- mean(feet / 12 + inches, na.rm = TRUE)
```

Which is good, which is bad?

STYLE GUIDE

Code spacing:

Bad

```
average<-mean(feet/12+inches,na.rm=TRUE)
```

Good

```
average <- mean(feet / 12 + inches, na.rm = TRUE)
```

Which is good, which is bad?

WHAT TO REMEMBER



FUNCTIONS TO REMEMBER

Operator/Function	Description
<code>help()</code> , <code>?</code> , <code>example()</code>	Get help on functions and provide examples
<code>getwd()</code> , <code>setwd()</code>	Get and set your working directory
<code>+</code> , <code>-</code> , <code>*</code> , <code>/</code> , <code>^</code>	Arithmetic
<code><-</code>	Assignment operator
<code>ls()</code> , <code>rm()</code>	list and remove objects in your global environment
<code>install.packages()</code> , <code>library()</code>	Install and load packages
<code>vignette()</code>	View/list package vignette

