

# Teaching Programming with Tidyverse Koans

A Journey of Successes and Failures

Colleen O'Briant

## Vision

Today, software engineers are not the only people who program. People in just about every field write programs to help them do their tasks.

This means that computer scientists are not the only people who *teach* programming anymore.

**If you're tasked with teaching people to program, building koans is the best place to start.**

Koans are gentle, self-paced, easy to create, and you'll have great results.

## My First R Class



Reading in different types of data as tibbles:

- csv: comma-separated values use `readr::read_csv()`
- excel files: use `readxl::read_excel()`
- .dta from STATA: use `haven::read_dta()`



Try these for yourself:

- `View(gapminder)`
- `?gapminder`
- `colnames(____)`
- `head(____)`
- `tail(____, n = 15)`
- `class(____)`
- `str(____)`
- `dim(____)`
- `nrow(____)`
- `summary(____)`
- `skimr::skim(____)`



### 3.1 Select values

```
# Select the population of Afghanistan in 1952
# How?
# Use `head(gapminder)` to find the row __, column __ of the data.
# To select the element in the 3rd row, 2nd column of a matrix, use my_matrix[3, 2]. As a
# n aside, to select the 10th element of a vector, use my_vector[10].
```

```
gapminder[1, 5]
```

**Note:** select value ranges with `gapminder[1:10, 3]`.

### 3.2 Select columns

```
# Select the entire first column 'country' in gapminder
# How?
# my_matrix[, 10] selects all rows of the 10th column
# my_matrix[10, ] selects the 10th row of all columns
```

```
gapminder[, 1]
```

```
# R allows you to extract a named column with the `$` operator
gapminder$country
```

## Avoid

- square brackets
- \$
- `for` and `if` and `while`

## The Issue with Live Coding

Engagement comes from TERA<sup>1</sup>

Tribe

Expectations

Rank

Autonomy

1. The Coaching Habit, Michael Bungay Stanier

## Version 2: Koans



```
# 1. Make the first element of this numeric vector '6'. Remember to always  
# un-comment the line of code (Cmd-Shift-C), execute it (Cmd-Return), and then  
# test this file to make sure you passed (Cmd-Shift-T).
```

```
#1@
```

```
# c(__, 4, 5, 2, 3)
```

```
#@1
```

```
#:-----
```

```
# 2. Make the third element of this character vector 'economics'. -----  
# Note that quotes shouldn't be used with numbers, but should be used with  
# character strings.
```

```
#2@
```

```
# c("apple", "banana", __)
```

```
#@2
```



# 3d. Multiply a scalar and a vector.

#3d@

# 100 \* c(6, 3, 2) == c(\_\_, \_\_, \_\_)

#@3d



```
# To create a vector that does random sampling, use 'sample':  
?qelp::sample  
  
# This randomly draws 0's or 1's to create a random vector of length 10.  
sample(c(0, 1), size = 10, replace = TRUE)
```

```
# 6. Create a random character vector that draws "heads" or "tails". -----
```

```
#6@
```

```
# sample(__, size = 5, replace = TRUE)
```

```
#@6
```





# Outline for instructors

K01_vector.R	Data in R is held in vectors. Did you know that even a single number like 4 is a vector in R, of length 1? Starting with vectors gets learners familiar with the concept of functions being vectorized (many functions work on vectors of any length). This is a powerful first building block for the tidyverse's functional, declarative approach. It's a cardinal sin to let learners believe they need to do something as complicated as looping or mapping over a vector when the function they're applying is already vectorized.	
K02_tibble.R	The next building block to learn about are tibbles: the tidyverse's spreadsheet. Data is still being held in vectors (column vectors specifically). The <b>tidied data format</b> is where columns are variables and rows are observations.	
K03_pipe.R	The most commonly used function in the tidyverse. Use it to chain together a sequence of operations.	
K04_dplyr1.R	The first 3 dplyr functions to learn are filter(), select(), and mutate(). Tidyverse literature talks about using dplyr to "manipulate" data. I don't like that. I think it's a lot more useful for learners to think about using dplyr for writing <i>queries</i> on	

	their data, like with SQL. That way, they remember that if they have a question about their data, they should reach for dplyr.	
K05_dplyr2.R	The next two dplyr functions are summarize() and group_by().	
K06_dplyr3.R	The final two dplyr functions are arrange() and slice().	
K07_left_join.R	One more function from dplyr: use left_join() to combine two tibbles when you have a variable that acts as a key to match them. This koan sets learners up to solve <a href="#">the dplyr murder mystery I adapted from here</a> . This project has historically been a favorite: to quote my teaching evaluations, "The murder mystery proj was even enjoyable."	
K08_qplot_to_ggplot.R	Koans 8-11 shift the focus from dplyr to ggplot2. In this koan, I've given learners some exposure to qplot in their homework, and I introduce some ggplot2 syntax by drawing comparisons between qplot and ggplot2.	
K09_ggplot_aes.R	aes(): aesthetic mappings between variables in the data and visual properties in the plot like axis, color, and size.	
K10_ggplot_geom.R	Histograms, density plots, vertical and horizontal lines and annotations, scatterplots, boxplots, and animations!	
	This koan is a refresher and a reminder for	

K11_lm.R	learners about running regressions in R, especially using <code>lm()</code> to get an R-squared, the fitted values and residuals, doing log or squared transformations of variables inside the <code>lm()</code> call, interactions, etc.
K12_stats.R	Exploring the normal distribution, Student's t distribution, and F distribution in R. Randomly generate numbers from each of these distributions. Visualize a hypothesis test from each of these distributions using <code>ggplot</code> .
K13_functions.R	Learn to write your own functions to make your code more readable and organized.
K14_functions_dplyr.R	It's fun and valuable to be able to write your own functions on top of tidyverse code that use env-variables just like in <code>dplyr</code> or <code>ggplot2</code> . The error is cryptic ( <code>object not found</code> ), but the solution is simple ( <code>{{ vars }}</code> ). <code>dplyr</code> vignettes call this "indirection".
K15_map.R	Map: what to do when the functions you want to apply aren't vectorized in the way you want: <code>map(.x, .f)</code> applies the function <code>.f</code> to each element of <code>.x</code> . <code>map(.x, .f)</code> is declarative. The imperative analog is a <code>for</code> loop. I don't think it's useful to teach students both <code>map</code> and <code>for</code> loops when they're learning the tidyverse. The first reason is that we want to stay as declarative as possible for consistency. The second reason is that this concept can be difficult. so being



	given two ways to solve the problem can be really confusing. I'd much rather have learners gain confidence with one way first.	
K16_more_map.R	Use <code>map(.x, .f)</code> to do monte carlo simulations with elegance.	
K17_lag.R	By this time in my class, we've transitioned to time series models, where lags are important to be able to compute. A lag is just a previous observation for a variable.	
K18_first_differences.R	Use lags to "first difference" variables in your data and visualize the results in a ggplot with two sets of vertical axis labels.	
K19_reduce.R	Reduce is a powerful functional programming concept. I like to go over the <a href="#">pictures in Adv-R</a> in lecture before learners attempt these last two koans. This concept isn't difficult, but does require a little focused attention.	
K20_accumulate.R	I provide an example of how to use <code>reduce()</code> (and its relative <code>accumulate()</code> ) to generate a random walk and then an autocorrelated series.	

## Course Evaluations

- *The fastest and best way I've ever learned to code.*
- *I also like the koan exercises very much. They are fun and very useful for learning the R language. Being able to test the answers of the koans before handing in also reduces the stress of the work.*
- *The most impressive aspect of the class was the koans and your approach to coding.*
- *I want more koans!*

# New developments: video and zine

The image is a composite screenshot of a computer screen. On the left, a GitHub README page for "EC320 Lab #1" is displayed. It contains instructions for the lab, a "Setting up your Workspace" section, and a "Grab a Computer" section. In the center, there is a video call interface showing a person's face. On the right, a list of recorded sessions titled "EC320 labs" is shown, each with a thumbnail, title, and duration.

**EC320 Lab #1**

Lab is on Wednesday 4/5 from 4-5:20pm. Find the link to join the Zoom meeting on Canvas. I'll also have a Zoom office hour on Friday 4/7 from 8-9am to help with any last minute questions about the lab assignment for the week.

Prep: Setting up your Workspace (see below)

Lab assignment: `K01_vectors.R` and `K02_pipe.R` are due Friday 4/7 at 5pm. Upload your knitted html files to Canvas under Lab 1. Collaboration is welcome, but every individual must type up and compile their own solutions on their own computers.

**Setting up your Workspace**

Follow these instructions before lab on Wednesday so that you can jump in! I'll help troubleshoot during lab if you run into problems with your installations, but please come to lab prepared by trying it yourself first.

**Grab a Computer**

First things first, you should decide which computer you'd like to do your programming assignments on. It can be a Mac, Windows, or Linux machine; all are equally valid. I do 0:01 / 33:50 anything on my

**EC320 labs**

- Colleen O'Briant - 1 / 5
  - EC320 Lab 1 Colleen O'Briant 33:51
  - EC320 lab2 Colleen O'Briant 8:47
  - Lab 3 Colleen O'Briant 23:37
  - EC320 Lab 4 Colleen O'Briant 16:44
  - EC320 lab 7 and 8 Colleen O'Briant 33:46

All From Colleen O'Briant Computer programs >

# LEARNING THE TIDYVERSE



BY COLLEEN O'BRIANT

## FILTER

FILTER OUT ROWS THAT DON'T MEET YOUR CONDITIONS.  
FOR EXAMPLE, TO QUERY FOR ONLY THE FEMALE STUDENTS:

<b>GENERAL FORMAT</b> <pre>students %&gt;%   filter(condition)</pre>	<b>SQL</b> <pre>SELECT * FROM dataset WHERE condition</pre>
<b>EXAMPLE</b> <pre>students %&gt;%   filter(sex == "female")</pre>	<pre>SELECT * FROM students WHERE sex = "female"</pre>

**Students**

sex	study_time	failures	romantic	grade	final_grade
female	<2h	0	no	44.2	44.4
female	<2h	0	yes	45.0	45.7
male	<2h	1	yes	46.1	66.3
female	<2h	0	no	46.2	46.7
male	<2h	0	no	77.1	77.7

**Students**

sex	study_time	failures	romantic	grade	final_grade
female	<2h	0	no	44.2	44.4
female	2-5h	0	yes	45.0	45.7
female	2-5h	0	no	46.2	46.7
female	>10h	0	no	76.7	76.7

---

## left\_join()

ALSO TAKES AN X AND A Y TABLE, BUT IT FILLS OUT THE LEFT HAND TABLE WITH THE VARIABLES ON THE RIGHT, MATCHING BY COMMON KEYS (VARIABLES).

**Diagram:** A small icon of two overlapping rectangles, one white and one grey, with an arrow pointing from the white rectangle to the grey one.

**Students A**

name	study_time	failures	final_grade
Celine	<2h	0	44.4
Thomas	<2h	0	44.4
Carry	<2h	0	45.7
Carry	<2h	1	66.3
Carry	<2h	0	46.7
Mandy	<2h	0	77.7

**Students B**

name	failures	romantic	grade
Mandy	0	no	77.7
Carry	0	yes	45.7
Carry	0	no	46.7
Carry	1	yes	66.3
Carry	0	no	44.4
Celine	0	no	44.4
Thomas	0	no	44.4

**left\_join(StudentsA, StudentsB, join\_by(name))**

name	study_time	failures	final_grade	failures	romantic	grade
Celine	<2h	0	44.4	0	no	44.4
Thomas	<2h	0	44.4	0	no	44.4
Carry	<2h	0	45.7	0	yes	45.7
Carry	<2h	1	66.3	1	yes	66.3
Carry	<2h	0	46.7	0	no	46.7
Mandy	<2h	0	77.7	0	no	77.7

HERE, THE KEY **STUDENTS A** AND **STUDENTS B** HAVE IN COMMON IS **NAME**.

## RECIPE 1: BAR PLOTS

### geom\_bar()

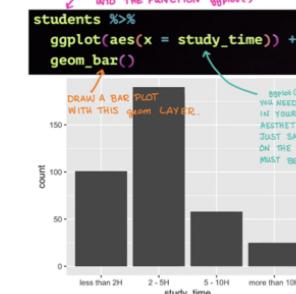
HOW MANY HOURS DO STUDENTS SPEND STUDYING?  
→ DISTRIBUTION OF A SINGLE DISCRETE VARIABLE: USE A BAR PLOT

**PIPE YOUR TIBBLE INTO THE FUNCTION ggplot()**

```
students %>%
  ggplot(aes(x = study_time)) +
  geom_bar()
```

AFTER THE `ggplot()` CALL, LAYERS ARE ADDED WITH `+`.

**geom\_bar()** TELLS R WHICH VARIABLES IN YOUR DATASET MAP TO WHICH AESTHETICS IN THE PLOT. HERE, WE'RE JUST SAYING TO DRAW STUDY\_TIME ON THE X-AXIS. AESTHETIC MAPPINGS MUST BE WRAPPED IN A `aes()`.



**GGPLOT BASICS: DECIDING BETWEEN :**  
`geom_bar()`, `geom_histogram()`, `geom_point()`, `geom_boxplot()`

```

graph TD
    Q1[HOW MANY VARS ARE YOU TRYING TO VISUALIZE?]
    Q1 -- 1 --> Q1_Discrete[IS THE VARIABLE DISCRETE OR CONTINUOUS?]
    Q1 -- 2 --> Q1_Continuous[ARE THE VARIABLES DISCRETE OR CONTINUOUS?]
    
    Q1_Discrete -- DISCRETE --> Bar_Plot[BAR PLOT  
geom_bar()]
    Q1_Discrete -- CONTINUOUS --> Histogram[HISTOGRAM  
geom_histogram()]
    
    Q1_Continuous -- ONE IS DISCRETE, THE OTHER IS CONTINUOUS --> Box_Plot[BOXPLOT  
geom_boxplot()]
    Q1_Continuous -- BOTH ARE DISCRETE --> Scatter_Plot[SCATTERPLOT  
geom_point()]
    Q1_Continuous -- BOTH ARE CONTINUOUS --> Scatter_Plot
    
    Bar_Plot --> Note["WARNING: THIS ONLY WORKS ON CATEGORICAL DATA! BUT I'M NOT GUARANTEE THIS WILL LEAD TO THE BEST POSSIBLE PLOT. THERE IS LOTS MORE IN LATER."]
  
```

WARNING: THIS ONLY WORKS ON CATEGORICAL DATA! BUT I'M NOT GUARANTEE THIS WILL LEAD TO THE BEST POSSIBLE PLOT. THERE IS LOTS MORE IN LATER.

<https://koans-cascadia.netlify.app/#/section-5>

18/19

## Final Thoughts

- Keep it as simple as possible (a natural law of the universe is that students will discover fancy stuff when they're ready)
- With koans, test coverage becomes a fundamental first building block
- Don't forget to tell your students they're doing well