Module 1: Intro to Coding in R

Ellen Bledsoe

2026-01-27

Introduction to Coding

Learning Outcomes

- Students will be able to define the following terms: object, assignment, vector, function, data frame
- Students will be able to run code line-by-line and as code chunks from an Rmarkdown file.
- Students will be able to write code assigning values to variables and use these variables to perform various operations.
- Students will be able to recall and explain how functions operate, and the basic syntax around functions (arguments, auto-completion, parentheses).
- Students will be able to differentiate different data classes in R.

Assigning Objects

Assignments are really key to almost everything we do in R. This is how we create permanence in R. Anything can be saved to an object, and we do this with the assignment operator, `<-`.

The short-cut for `<-` is `Alt + -` (or `Option + -` on a Mac)

```
# Assigning Objects
height <- 47.5
age <- 122

# We can do math with objects
height <- height * 2    # multiply
age <- age - 20         # subtract
height_index <- height/age  # divide
height_sq <- height^2       # raise to an exponent

# This is simplistic and you'll rarely do it in real-world scenarios.
```

1-Dimensional Data: Vectors

We can also assign more complex group of elements of the same type to a particular object. This is called a **vector**, a basic data structure in R.

```
weight_kg <- c(3, 2, 4, 9, 7, 3, 6)
weight_kg
```

```
## [1] 3 2 4 9 7 3 6
```

```
animals <- c("cat", "rat", "bat", "rat")
animals
```

```
## [1] "cat" "rat" "bat" "rat"
```

```r
# R does everything in vectors
```

Data classes

There are a few main types of data in R, and they behave differently. We call these types of data "classes."

- numeric / double (numbers, decimals allowed)
- integer (no decimals allowed)
- character (letters or mixture)
- logical (True or False; T or F)
- factors (best used for data that need to be in a specific order; levels indicate the order)

```r
# Examples of different data classes
weight_kg   # numeric, integer, double
```

```
## [1] 3 2 4 9 7 3 6
```

```r
animals     # character
```

```
## [1] "cat" "rat" "bat" "rat"
```

```r
animal_size <- c("medium", "large", "small", "medium")
animal_size <- factor(animal_size, levels = c("small", "medium", "large"))
animal_size  # factor, put in order
```

```
## [1] medium large  small  medium
## Levels: small medium large
```

```r
logic <- c(T, F, F, T)  # logical
logic
```

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

Vectors have to contain elements that are all of the same class. What happens if we put data of different classes into one vector?

```r
vec <- c(1, 1.000, "1")
```

Subsetting Vectors

Sometimes we want to keep specific values from a vector. This is called subsetting (taking a smaller set of the original).

We can subset vectors in two different ways:

- by index (position)
- by condition

Regardless of which type of subsetting we choose, we indicate that we want to subset by using square brackets: `[]`.

Subsetting by Index

When we subset by index, we are subsetting based on the position of an element in the vector.

```r
# Use square brackets
weight_kg[2]    # returns the 2nd element in the vector
```

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

```r
weight_kg[2:4]  # returns the 2nd, 3rd, and 4th elements in the vector
```

```
## [1] 2 4 9
```

Subsetting by Condition

Sometimes we don't know or don't want to list out all of the locations for the data we need. Instead, we might want to subset based on a quality of the data itself.

To do this, we set a "condition" that must be met in order for the data to be returned.

```r
# let's start with a condition
weight_kg > 5
```

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

```r
# we now put that condition inside the square brackets
weight_kg[weight_kg > 5]
```

```
## [1] 9 7 6
```

```r
# we can also do this with characters
animals == "cat"
```

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

```r
animals[animals == "cat"]
```

```
## [1] "cat"
```

Functions

Functions are pre-written bits of codes that perform specific tasks for us. Functions are always followed by parentheses.

Anything you type into the parentheses are called arguments. Arguments are pieces of information that we give to a function so it performs its task the way we want it to. To add more than one argument, you separate them with a comma.

```r
## Functions
weight_kg_mean <- mean(weight_kg)   # average of the mass_kg vector from above
weight_kg_mean
```

```
## [1] 4.857143
```

```r
# separate arguments with commas
round(weight_kg_mean)             # rounding
```

```
## [1] 5
```

```r
round(weight_kg_mean, digits = 2) # round to 2 digits past 0
```

```
## [1] 4.86
```

To get more information about a function, use the `help()` function or `?name_of_function`.

```r
help(round) # or type ?help
```

We can use a function called `class()` to figure out the data type of a vector.

```r
class(weight_kg)
```

```
## [1] "numeric"
```

Small Group Challenge

Let's practice! Write a few lines of code that do the following:

- create a vector with numbers from 6 to 1 (6, 5, 4, 3, 2, 1)
- assign the vector to an object named `six_to_one`
- subset `six_to_one` to include the last 3 numbers (should include 3, 2, 1)
- find the sum of the numbers (hint: use the `sum()` function)

Answer: 6

```
six_to_one <- c(6, 5, 4, 3, 2, 1)
six_to_one
```

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

```
# subsetting by index (position)
last_three <- six_to_one[4:6]
last_three
```

```
## [1] 3 2 1
```

```
# alternate: subsetting by condition
last_three <- six_to_one[six_to_one < 4]
last_three
```

```
## [1] 3 2 1
```

```
sum(last_three)
```

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

Already finished? See if you can condense your code down any further or turn around and help out a neighbor.

```
six_to_one <- seq(6, 1)
sum(six_to_one[4:6])
```

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

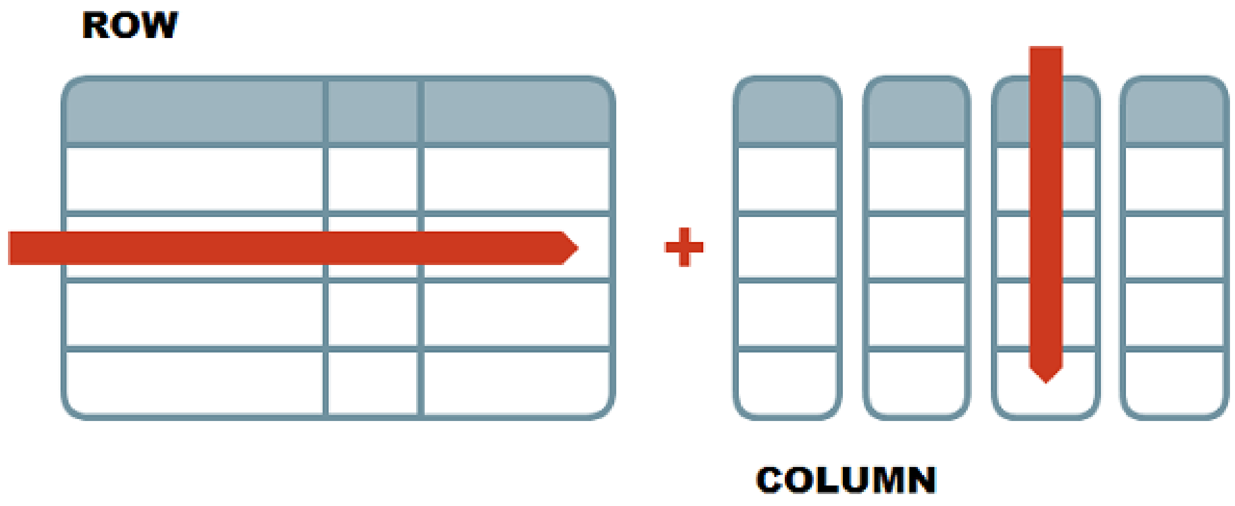
2-Dimensional Data: Data Frames

Most of the data you will encounter is two-dimensional (i.e., it has columns and rows). Its structure resembles a spreadsheet. R is really good with these types of data. We call these 2D object data frames.

- **rows** go side-to-side
- **columns** go up-and-down

Columns typically represent variables (a factor, trait, or condition) we are interested in.

Rows represent observations. Each row will be one set of observations.



Data frames are made up of multiple vectors. Each vector becomes a column.

```r
# Create a simple data frame from scratch
plants <- data.frame(height = c(55, 17, 42, 47, 68, 39, 51, 23),
                     nitrogen = c("Y", "N", "N", "Y", "Y", "N", "Y", "N"))

plants
```

```
##   height nitrogen
## 1     55        Y
## 2     17        N
## 3     42        N
## 4     47        Y
## 5     68        Y
## 6     39        N
## 7     51        Y
## 8     23        N
```

Subsetting Data Frames

Because data frames are two-dimensional, we can subset the data in a data frame by selecting specific columns, specific rows, or both!

R *always* takes information for the row first, then the column.

Just like with vectors, we can subset data frames by index or by condition using square brackets.

The pattern is `dataframe[rows, columns]`.

Subsetting by Index

```r
# Sub-setting data frames
# 2-dimensional, so you need to specify row and then column
# plants[3] # doesn't work
# row then column
plants[4,1]
```

```
## [1] 47
```

```r
plants[,2]
```

```
## [1] "Y" "N" "N" "Y" "Y" "N" "Y" "N"
```

Another way to pull out a single column from a data frame is with the `$` operator. This can really come in handy when you know the name of the column but not the position.

```r
plants$height
```

```
## [1] 55 17 42 47 68 39 51 23
```

Regardless of how you specify the column, you can put that code inside of a function, such as the `mean()`.

```r
mean(plants$height)
```

```
## [1] 42.75
```

Subsetting by Condition

This is a simple data set, but we can use it to ask a question.

Example: Are the heights of plants treated with nitrogen different from those not treated?

First, we will need to keep only the plants that were treated with nitrogen.

```r
# filter rows based on values in the nitrogen column
plants[plants$nitrogen == "Y", ]
```

```
##   height nitrogen
## 1     55        Y
## 4     47        Y
## 5     68        Y
## 7     51        Y
```

```r
# calculate the mean
mean(plants[plants$nitrogen == "Y", 1])
```

```
## [1] 55.25
```

We can create a new data frame by saving the subset data frame to a new object.

```r
plants_no_nitrogen <- plants[plants$nitrogen == "N", ]
```

Small Group Challenge (5 min)

As a group, find the standard deviation (`sd()`) of the height of plants treated with nitrogen and those not treated with nitrogen. Which group has the larger standard deviation? Any ideas what that means?

```r
sd(plants[plants$nitrogen == "Y", 1])
```

```
## [1] 9.105859
```

```r
sd(plants[plants$nitrogen == "N", 1])
```

```
## [1] 12.14839
```

Helpful Functions

Below are some functions that I often find very helpful when working with vectors and data frames:

- `str()`
- `head()` and `tail()`
- `length()`
- `ncol()` and `nrow()`
- `names()`

```r
str(plants) # structure of the object
```

```
## 'data.frame':    8 obs. of  2 variables:
##  $ height  : num  55 17 42 47 68 39 51 23
##  $ nitrogen: chr  "Y" "N" "N" "Y" ...
```

```r
head(plants) # first 6 values or rows
```

```
##   height nitrogen
## 1     55        Y
## 2     17        N
## 3     42        N
## 4     47        Y
## 5     68        Y
## 6     39        N
```

```r
head(plants, n = 4) # first n values or rows
```

```
##   height nitrogen
## 1     55        Y
## 2     17        N
## 3     42        N
## 4     47        Y
```

```r
tail(plants, n = 4) # last n values or rows
```

```
##   height nitrogen
## 5     68        Y
## 6     39        N
## 7     51        Y
## 8     23        N
```

```r
length(plants)  # for a dataframe, number of columns
```

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

```r
length(plants$height) # for a column, number of rows
```

```
## [1] 8
```

```r
ncol(plants)  # number of columns
```

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

```r
nrow(plants)  # number of rows
```

```
## [1] 8
```

```r
names(plants) # list of column or object names
```

```
## [1] "height"   "nitrogen"
```