Module 1: Intro to Coding in R

Ellen Bledsoe

2024-01-23

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 comment their code effectively.
- Students will be able to write code assign values to variables and use these variables to perform various operations.
- Students will be able to use help files to learn how to use functions.
- 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.
- Students will learn how to create their own data structures (vectors, data frames).

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 simple 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")
animals
```

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

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

Data classes

There are a few main types in R, and they behave differently.

- numeric (numbers)
  - integer (no decimals allowed)
  - double (decimals allowed—interchangeable with numeric)
- 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)

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

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

```
animals     # character
```

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

```
animal_size <- as.factor(c("small", "medium", "large"))
animal_size  # factor, put in order
```

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

```
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.

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

Sub-setting Vectors

Sometimes we want to pull out and work with specific values from a vector. This is called sub-setting (taking a smaller set of the original).

```r
# Use square brackets
weight_kg[2]
```

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

```r
weight_kg[2:4]
```

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

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**.

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

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

```r
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"
```

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 `vec`
- subset `vec` to include the last 3 numbers (should include 3, 2, 1)
- find the sum of the numbers (hint: use the `sum()` function)

Answer: 6

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

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

```
vec <- vec[4:6]
vec
```

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

```
sum(vec)
```

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

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

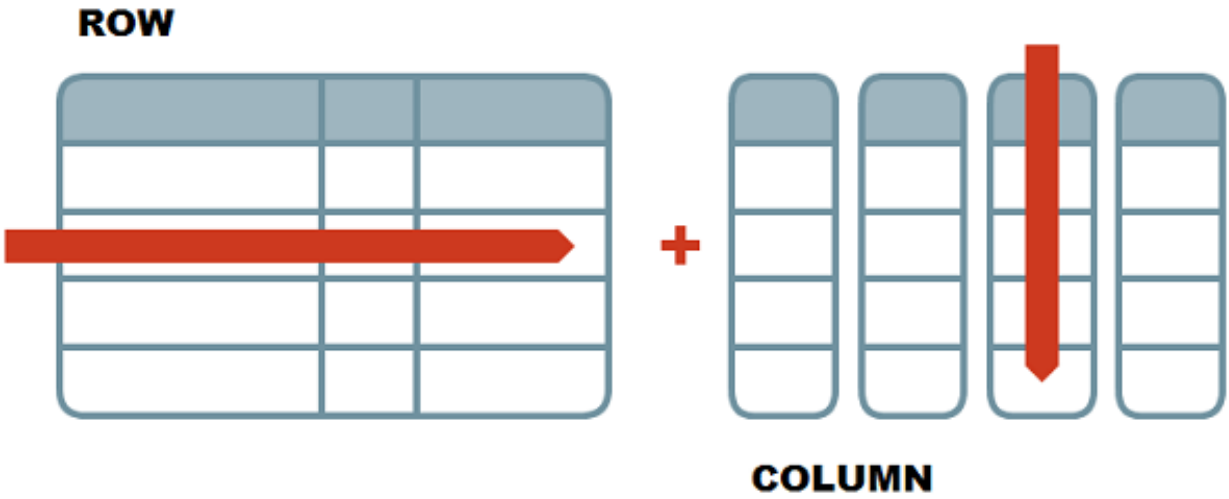
```
vec <- seq(6, 1)
sum(vec[4:6])
```

```
## [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.

- **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.

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

plants
```

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

Sub-setting Data Frames

Because data frames are two-dimensional, we can subset data in different ways. We can select specific columns, specific rows, or filter rows by values.

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

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

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

```
plants[,2]
```

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

```
plants$height
```

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

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

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

```
## [1] 44.66667
```

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()`

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

```
## 'data.frame':    6 obs. of  2 variables:
##  $ height  : num  55 17 42 47 68 39
##  $ nitrogen: chr  "Y" "N" "N" "Y" ...
```

```
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
```

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

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

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

```
##   height nitrogen
## 3     42        N
## 4     47        Y
## 5     68        Y
## 6     39        N
```

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

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

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

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

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

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

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

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

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

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