Module 1: Intro to Coding in R

Ellen Bledsoe

2023-01-26

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
mass <- 47.5            # this is the mass in kg
age <- 122
mass <- mass * 2        # multiply
age <- age - 20         # subtract
mass_index <- mass/age  # divide
mass_sq <- mass^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 a basic data structure in R.

```
mass_kg <- c(3, 2, 4, 9, 7, 3, 6)
mass_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
mass_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 [ordinal data]
```

```
## [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.

```
vec <- (1, 1.000, "1", TRUE)
```

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).

```
# Use square brackets
mass_kg[2]
```

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

```
mass_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.

```
## Functions
mass_kg_mean <- mean(mass_kg)   # average of the mass_kg vector from above
mass_kg_mean
```

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

```
round(mass_kg_mean)             # rounding
```

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

```
round(mass_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`.

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

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

Group Challenge

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

- create a vector with a number from 6 to 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)

```
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

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

```r
# 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.

```r
# 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
```

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

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

```r
# We can also choose specific columns using `$`
plants$height
```

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

Discussion Point

This is a simple data set, but let's come up with some questions.

Example: height of plants treated with nitrogen vs. those not treated.

```r
# Example: height of plants treated with nitrogen vs. those not treated.

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

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

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

```
## [1] 56.66667
```

Group Challenge (5 min)

Using help files on functions

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?

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

```
## [1] 10.59874
```

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

```
## [1] 13.6504
```

Come up with a definition of standard deviation (Google is your friend!), use the help file to find out how the `sd()` function works, and be prepared to show the code you used.

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"
```

Rmarkdown and Code Chunks (10 minutes)

Rmarkdown (.Rmd) is a file format that lets us incorporate text and code into one document seamlessly. In fact, it is the file format for this document!

- For writing text, you can type as you would normally
- Code chunks are a bit different.
  - Type your R code in the lines under the `{r}`.
  - To include text in them, you will need to put a `#` in front. R will not read anything after the `#` as code. Chunks look like this:

A quick shortcut for adding a code chunk is `Ctrl + Atl + i` (`Cmd + Opt + i` on a Mac). Alternatively, you can go to Code > Insert Chunk.

To run a chunk of code, click the green arrow on the far right side of the chunk. You can also run one or a few lines of code at a time by having your cursor on the line or highlighting multiple lines and hitting `Ctrl` + `Enter` (or `Cmd` + `Enter` on a Mac).

Let's work with an example code chunk. Practice entering a new code chunk below.

```
# My first code chunk!
# Ellen Bledsoe
# Aug 2022

# Examining a pre-built data set in R
Orange
```

```
##    Tree  age circumference
## 1     1  118            30
## 2     1  484            58
## 3     1  664            87
## 4     1 1004           115
## 5     1 1231           120
## 6     1 1372           142
## 7     1 1582           145
## 8     2  118            33
## 9     2  484            69
## 10    2  664           111
## 11    2 1004           156
## 12    2 1231           172
## 13    2 1372           203
## 14    2 1582           203
```

```
## 15    3  118            30
## 16    3  484            51
## 17    3  664            75
## 18    3 1004           108
## 19    3 1231           115
## 20    3 1372           139
## 21    3 1582           140
## 22    4  118            32
## 23    4  484            62
## 24    4  664           112
## 25    4 1004           167
## 26    4 1231           179
## 27    4 1372           209
## 28    4 1582           214
## 29    5  118            30
## 30    5  484            49
## 31    5  664            81
## 32    5 1004           125
## 33    5 1231           142
## 34    5 1372           174
## 35    5 1582           177
```

```r
# printing maximum circumference of the biggest tree
max(Orange$circumference)
```

```
## [1] 214
```

If you run code in the console (below) rather than the code chunk in a .Rmd file, you don't need to add the `{r}` to tell it that you are typing R code. The console only understand R code, so we don't need to tell it!

Group challenge

Construct a new Rmarkdown script (with all of the appropriate formatting we discussed) that calculates the average (mean) circumference of trees in the orange data set. Save the mean as an object called "avg_circ."

Save the script and be ready to share it.

If you're already finished, try this challenge. Find the average circumference of trees aged 484 days.

```r
head(Orange)
```

```
##   Tree  age circumference
## 1    1  118            30
## 2    1  484            58
## 3    1  664            87
## 4    1 1004           115
## 5    1 1231           120
## 6    1 1372           142
```

```r
avg_circ <- mean(Orange$circumference, na.rm = TRUE)
avg_circ
```

```
## [1] 115.8571
```

```r
trees484 <- Orange[Orange$age == 484, 3]
mean(trees484)
```

```
## [1] 57.8
```