Introduction to Coding in R

EKB

2026-02-09

Introduction to R and RStudio

Student Learning Outcomes

- Students will be able to explain the benefit of each of the four main panels in RStudio
- Students will be able to do the following in the R languages:
  - perform mathematics
  - assign objects
  - use functions
  - explore and describe vectors (1D data)

RStudio Cloud Tour

Let's first explore what each of the panels in RStudio do.

1. *Source* (upper left): This is where documents which have data or code in them are opened. You can save all the code you type here for future (re)use, which is a big reason coding in R is reproducible.
2. *Console* (bottom left): This is where code from the source is "run" and you see the outputs. You can also execute lines of code which you type into the console, but they will not be saved. You can think of this section as where RStudio really interfaces with R–it is where R actually evaluates code within RStudio.
3. *Environment* (upper right): This panel becomes more helpful as you get familiar with R and RStudio. It keeps track of data objects and other items you have created and gives a bit of information about them.
4. *Files/Help/etc.* (bottom right): This panel is (clearly) very multifaceted. The Files tab lets you see all the files in your current workspace, and we will mostly use this tab in the panel. The Help tab can also be useful, as it contains the R documentation for information about functions we use.

Basics of Coding in R

How do you "run" code? Running or executing code means that you are sending a line of code to the console for R to interpret it.

In the source, there are a few different ways to run code.

- Sometimes you want to run only one line of code at a time. You can do this by putting your cursor on the line you want to run and either hitting the "Run" button in the upper right-hand corner of the source panel or holding down `Ctrl` + `Enter` (`Cmd` + `Enter` on Macs).
- If you want to run a couple lines of code, you can highlight them and do the same thing as above.
- If you want to run an entire code chunk (see below), you can click on the green arrow on the right side of the code chunk.

In the console, you hit `Enter`.

Using R as a calculator

For example, we can perform mathematical calculations using R.

```
# R can perform mathematical calculations for us

3 + 5   # addition
```

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

```
15 / 5  # division
```

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

```
48 * 17 # multiplication
```

```
## [1] 816
```

By the way, you can add notes or information into code chunks.

```
# we can add comments to our code by using the # sign
# R doesn't read anything past a #, so you can either put one at the beginning of a whole line or after

4^2     # exponents
```

```
## [1] 16
```

Let's practice in the console, too.

You can also do basic math in the console.

The console only understands R code, so we don't need to use the `{r}` notation; we can type numbers and mathematical symbols (among other things).

Try multiplying 5 and 3 (hint: `*` means multiply) in the console.

Assigning Objects

Assignments are really key to almost everything we do in R. We create permanent objects in R by assignment, meaning we can reference these objects later.

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). We will be typing this *a lot*.

```
# save as objects (in the environment)
height <- 47.5
age <- 122
```

You will notice 2 things.

1. Any time we write a line of code which creates an object, the output of the code is not shown below the code chunk. To show the output, type the name of the object on a new line of code.

   ```
   obj <- 24
   obj
   ```

   ```
   ## [1] 24
   ```

2. Each new object will show up in your environment (upper right panel). This is particularly helpful when you want to check if your code ran successfully or to confirm that the object represents what you *think* it represents.

We can reference objects in other parts of the code. For example, we can perform mathematical operations using numeric objects.

```
# we can add comments to our code by using the # sign
# R doesn't read anything past a #, so you can either put one at the beginning of a whole line or after

height <- height * 2        # multiply
age <- age - 20             # subtract
height_index <- height/age  # divide
```

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 allow us to give the function additional information about how we want it to perform its task.

```
sqrt(10)
```

```
## [1] 3.162278
```

```
weight_kg <- sqrt(10) # square root

round(weight_kg) # rounding
```

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

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

```
## [1] 3.16
```

To get more information about a function, use the help function.

```
# these two lines of code do the same thing
help(mean)
?mean
```

Let's Practice!

Write some code that calculates the sum of the `weight_kg` object and the `height` object. You'll want to use the `sum()` function. Save the result as an object called `weight_and_height`.

```
weight_and_height <- sum(weight_kg, height)
```

When you've finished that, start working on Questions 5 & 6 in the assignment.

Vectors

Vectors are the most common and basic data type in R. They make up most of the other data types we will work with in R. They are composed of series of values (called elements), which can either be numbers, characters, or other types of data.

We use the `c()` function (stands for combine) to create a vector.

```
# Let's create a vector of animal weights (numeric)
weight_g <- c(60, 65, 82)
weight_g
```

```
## [1] 60 65 82
```

```
# A vector can also contain character strings (character)
animals <- c("mouse", "rat", "penguin", "rat")
animals
```

```
## [1] "mouse"   "rat"     "penguin" "rat"
```

```
# Logical vector
mammal <- c(TRUE, TRUE, FALSE, TRUE)
mammal
```

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

There are many functions we can use to look at vectors and learn more about them.

```
# how many elements
length(weight_g)
```

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

```
length(animals)
```

```
## [1] 4
```

```
# type of data we are working with
class(weight_g)
```

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

```
class(animals)
```

```
## [1] "character"
```

```
# structure of an object
str(weight_g)
```

```
##  num [1:3] 60 65 82
```

```
str(animals)
```

```
##  chr [1:4] "mouse" "rat" "penguin" "rat"
```

```
# find the unique values
unique(animals)
```

```
## [1] "mouse"   "rat"     "penguin"
```

Vectors can only contain data that is one class. Let's experiment with that.

```
test_vec <- c(weight_g, animals, mammal)
test_vec
```

```
##  [1] "60"      "65"      "82"      "mouse"   "rat"     "penguin" "rat"
##  [8] "TRUE"    "TRUE"    "FALSE"   "TRUE"
```

```
class(test_vec) # coerced everything into character (don't know how to make words numeric)
```

```
## [1] "character"
```

Let's Practice!

Start working on Questions 7-10 in the assignment.

Sub-setting Vectors

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

In R, there are 2 main ways to subset:

- by index (position of the element)
- by condition (meeting certain requirements)

Let's explore both.

Sub-setting by Index

One way to subset is with an index, meaning the position of the value or element in the vector. To do this, we use square brackets and one or more numbers to indicate the position(s) to return.

```
weight_g[2]
```

```
## [1] 65
```

```
weight_g[c(2,4)]
```

```
## [1] 65 NA
```

```
weight_g[c(1:4)]
```

```
## [1] 60 65 82 NA
```

Sub-setting by Condition

Another way we can subset data is through conditions. The vector will only return data which meet certain requirements we have set. We set these requirements through "conditional statements."

```
# conditional statement
weight_g > 60
```

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

```
# sub-setting
weight_g[weight_g > 55]
```

```
## [1] 60 65 82
```

```
animals[animals == "rat"]
```

```
## [1] "rat" "rat"
```

Conditions can be set using the following options:

- `>`: greater than
- `<`: lesser than
- `>=`: greater than or equal to
- `<=`: lesser than or equal to
- `==`: equal to (note the double `=`)
- `!=`: not equal to

Let's Practice!

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

- create a vector with even numbers from 1 to 10 (including 10)
- assign the vector to an object named `vec`

- using the *index* method, subset `vec` to include the last 3 numbers (should include 6, 8, 10)
- find the sum of the numbers (hint: use the `sum()` function)

```
# the answer you should get out is 24
vec <- c(2, 4, 6, 8, 10)
vec
```

```
## [1]  2  4  6  8 10
```

```
last3 <- vec[3:5]
last3
```

```
## [1]  6  8 10
```

```
sum(last3)
```

```
## [1] 24
```

If you've finished that task, try to get the same result using *conditional* sub-setting.

```
sum(vec[vec > 5])
```

```
## [1] 24
```

Once you've completed this task, work on Questions 11-20 on the assignment. Don't forget to go back to Questions 1-4 before you turn it in!