R Programming: An Introduction

Stacey Borrego (edited by Kristina Riemer)

Helpful Links

- Data Carpentry: Data Analysis and Visualization in R for Ecologists
- Tidyverse: R packages for Data Science
- R for Data Science 2nd Edition by Hadley Wickham and Garret Grolemund
- Advanced R by Hadley Wickham
- ggplot2: Elegant graphics for data analysis by Hadley Wickham
- Posit Cheatsheets by Posit
- The R Gallery by Kyle W. Brown
- Introduction to R by Douglas, Roos, Mancini, Cuoto, & Lusseau

Notes

- Sections marked with $^{\$}$ may have been skipped due to time limits

Overview of RStudio

- (Lower) left hand side: *console* which runs R code
- Create a new script
  - Hit file button in upper left hand corner
  - Select first option "R Script"
- Upper left hand side: *scripts* where R code can be saved
- Upper right hand side: where objects are
- Lower right hand side: files and folders on computer

Simple math

In the *console*, type in the following expressions
Hit `ENTER` after each expression

```
3 + 5
12 / 7
3 * 4
8 ^ 2
3 + 5 / 2
(3 + 5) / 2
```

Creating objects in R

Working with variables/objects

Do this in the *script* that was created

Assign values to a variable/object

```
weight_kg <- 55
```

Running code in a script

- Select `Run` button at the top of the script window
- Select code or move cursor to the line to be run and type `Ctrl` + `ENTER`

How to name variables:

- Assignment operator (<-)
- Cannot start with numbers
- Case sensitive
- Avoid function names
- Avoid dots (.)
- Be consistent in the styling of your code
  - R Style Guide
  - Styles can include "lower_snake", "UPPER_SNAKE", "lowerCamelCase", "UpperCamelCase", etc
  - We will use "lower_snake" for readibility during this workshop

Print the value of an object

```
weight_kg
```

Simple math with objects

```
2.2 * weight_kg
```

Reassign object's value

```
weight_kg

weight_kg <- 65
weight_kg

2.2 * weight_kg
```

Assign a new object a value containing another object

```
weight_lb <- 2.2 * weight_kg
weight_lb
```

Exercises

1. Create a variable for the length of a line and store the value 15.3
2. Multiply that object by 3 to get volume
3. Create a variable that stores that volume value

```
length <- 15.3
length * 3
volume <- length * 3
```

What is the value of volume?

```
print(volume)
```

```
## [1] 45.9
```

Saving code in a script

- Save the file
  - File > Save (`Ctrl` + `S`)

Add comments in a script

- The comment character in R is `#`. Anything to the right of a `#` in a script will be ignored.
- Comment out a section by selecting and pressing `Ctrl` + `Shift` + `C`
  - press the same keys again to remove the comment

```
# This is a comment.
# The sum of 2 + 2
2 + 2
```

Functions and arguments

- Functions are scripts that automate more commands
- Functions are available in base R, by importing packages, or by making them yourself
- Usually require one or more inputs called **arguments**
- Functions typically return a **value**

```
# This is a function call
sqrt(10)
weight_kg <- sqrt(10)
weight_kg
```

- Information about functions can be found in the `Help` window by typing `?` and the function name. Example: `?round`
- You can also use the function `args()` to see the arguments of a function.

```
round(3.14159)

?round
args(round)

round(3.14159, digits = 2)
```

Exercises

1. Get the absolute value of -253 (hint: look at at the `Help` page for the `sqrt` function)
2. Take the logarithm of 10 (hint: the function name is the first three letters of the word)
3. Take the logarithm of 10, but change the base to `exp(3)`

```r
abs(-253)
```

```
## [1] 253
```

```r
log(10)
```

```
## [1] 2.302585
```

```r
log(10, base = exp(3))
```

```
## [1] 0.7675284
```

Vectors and data types

- A vector is composed by a series of values, which can be either numbers or characters.
  - Quotes must surround characters.
  - In a vector, all of the elements are the same type of data.
- We can assign a series of values to a vector using the `c()` function.

```r
# Numerical vector
weight_g <- c(50, 60, 65, 82)
weight_g

# Character vector
animals <- c("mouse", "rat", "dog")
animals
```

Explore the content of vector

- `length()` tells you how many elements are in a particular vector
- `class()` indicates what kind of object you are working with
- `str()` provides an overview of the structure of an object and its elements.

```r
# Length
length(weight_g)
length(animals)

# Class
class(weight_g)
```

```r
class(animals)

# Structure
str(weight_g)
str(animals)
```

Exercises

1. Create a character vector of names of colors
2. Put the vector elements into alphabetical order using the `sort` function

```r
colors <- c("red", "blue", "purple", "black")
sort(colors)
```

```
## [1] "black"  "blue"   "purple" "red"
```

$Add elements to a vector

```r
# Add to the end of the vector
weight_g <- c(weight_g, 90)

# Add to the beginning of the vector
weight_g <- c(30, weight_g)

# Inspect the modified vector
weight_g
```

Exercises

1. Add another color to your colors vector
2. Put the vector into reverse alphabetical order (hint: use `?`)

```r
colors <- c(colors, "magenta")
sort(colors, decreasing = TRUE)
```

```
## [1] "red"     "purple"  "magenta" "blue"    "black"
```

Data Types

An atomic vector is the simplest R data type and is a linear vector of a single type.

There are four common types of atomic vectors: character, numeric or double, logical, and integer. There are two rare types that will not be discussed: complex and raw.

- `character`
- `numeric` or `double`
- `logical` for TRUE and FALSE (the boolean data type)
- `integer` for integer numbers (e.g., 2L, the L indicates to R that it's an integer)

- `complex` to represent complex numbers with real and imaginary parts (e.g., 1 + 4i)
- `raw` for bitstreams

```
typeof(weight_g)
typeof(animals)
```

Coercion

All elements of an atomic vector must be the same type, so when you attempt to combine different types they will be coerced to the most flexible type. Types from least to most flexible are: logical, integer, double, and character.

```
num_char <- c(1, 2, 3, "a")
typeof(num_char)
str(num_char)

num_logical <- c(1, 2, 3, TRUE, FALSE)
typeof(num_logical)
str(num_logical)

char_logical <- c("a", "b", "c", TRUE)
typeof(char_logical)
str(char_logical)

tricky <- c(1, 2, 3, "4")
typeof(tricky)
str(tricky)
```

Exercises

1. Create a logical vector with any number and order of `TRUE` and `FALSE`

2. Add a numeric value to this vector
3. What kind of vector will this be?

```
some_booleans <- c(TRUE, FALSE, FALSE, FALSE)
some_booleans <- c(3, some_booleans)
typeof(some_booleans)
```

```
## [1] "double"
```

$Conditional subsetting

To extract one or several values from a vector, provide one or several indices in square brackets.

```
animals <- c("mouse", "rat", "dog", "cat")

animals[2]

animals[c(3, 1)]

animals[1:3]

more_animals <- animals[c(1, 4, 1, 3, 1, 2)]
more_animals
```

Subset by using a logical vector. TRUE will select the element with the same index, while FALSE will not.

```
weight_g <- c(21, 34, 55)

weight_g[c(TRUE, FALSE, TRUE)]

weight_g > 50

weight_g[weight_g > 50]
```

Combine multiple tests using `&` (both conditions are true, **AND**) or `|` (at least one of the conditions is true, **OR**).

Helpful operators

- > greater than
- < less than

- `<=` less than or equal to
- `==` equal to, test for numerical equality between the left and right hand sides

```
weight_g

weight_g[weight_g > 20 & weight_g < 30]

weight_g[weight_g <= 30 | weight_g == 55]

weight_g[weight_g >= 30 & weight_g == 21]
```

Search for strings in a vector

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

animals[animals == "cat" | animals == "rat"]

animals %in% c("rat", "cat", "duck", "fish")

animals[animals %in% c("rat", "cat", "duck", "fish")]
```

Missing data

- Missing data are represented in vectors as NA.
- You can add the argument `na.rm = TRUE` to calculate the result as if the missing values were removed (rm stands for **rem**ove)
- `is.na()` identifies missing elements
- `na.omit()` returns the object with incomplete cases removed
- `complete.cases()` returns a logical vector indicating which cases are complete

```
heights <- c(1, 2, 3, NA, 5, NA)

# Performing an operation on a vector with NA will usually return NA
mean(heights)
max(heights)

# Using `na.rm = TRUE` will exclude NA values and perform the operation
mean(heights, na.rm = TRUE)
max(heights, na.rm = TRUE)
```

$Function examples

```
# Returns vector of boolean values indicating if the value NA or not
is.na(heights)

# Sum of all the NAs present
sum(is.na(heights))

# Identifies the index containing the NA value
which(is.na(heights))

# Returns the TRUE values for `is.na()`
heights[is.na(heights)]

# Returns the FALSE values for `is.na()`
heights[!is.na(heights)]

# Returns values with incomplete cases (NA) removed
na.omit(heights)
mean(na.omit(heights))

# Returns values with complete cases only - NA removed
complete.cases(heights)
heights[complete.cases(heights)]
```

Data Structures

Vectors are one of the many data structures that R uses.

- lists (`list`)
- matrices (`matrix`)
- data frames (`data.frame`)
- factors (`factor`)
- arrays (`array`)

Data Frames

When we load data into R, it may be stored as an object of class data frame.

A data frame is the representation of data in the format of a table where the columns are vectors that all have the same length. Because columns are vectors, each column must contain a single type of data (e.g., characters, integers, factors, logical values).

For example, here is an example data frame comprising a numeric, character, and logical vector.

```
df <- data.frame(values = c(1, 3, 5, 7, 9, 11),
                 some_characters = c("F", "O", "O", "B", "A", "R"),
                 true_false = c(TRUE, FALSE, TRUE, FALSE, FALSE, FALSE))
df
```

```
##   values some_characters true_false
## 1      1               F       TRUE
## 2      3               O      FALSE
## 3      5               O       TRUE
## 4      7               B      FALSE
## 5      9               A      FALSE
## 6     11               R      FALSE
```

Simple column subsetting

Data frames can also be subset using square brackets as demonstrated with the single vectors or using `$` to specifically name a column.

Using single square brackets will return a data frame containing the values of the vector, whereas using double square brackets or `$` will return a simplified vector of the type of class as the values contained with in it.

```
# Column 1 data frame
df[1]
```

```
##   values
## 1      1
## 2      3
## 3      5
## 4      7
## 5      9
## 6     11
```

```
# Column 1 simplified vector
df[[1]]
```

```
## [1]  1  3  5  7  9 11
```

```
df$values
```

```
## [1]  1  3  5  7  9 11
```

Data can be further subset by providing the indices for the rows and columns you want. The notation is to first name the data frame followed by single square brackets listing the rows and columns.

```
# Column 2
df[2]
```

```
##   some_characters
## 1               F
## 2               O
## 3               O
## 4               B
## 5               A
## 6               R
```

$ Row and column subsetting

```
# Rows 4 through 6 in column 2
df[4:6, 2]
```

```
## [1] "B" "A" "R"
```

```
# Rows 1, 5, and 4 in column 2
df[c(1, 5, 4), 2]
```

```
## [1] "F" "A" "B"
```

```
# Rows 1, 5, and 4 in all columns
df[c(1, 5, 4), ]
```

```
##   values some_characters true_false
## 1      1               F       TRUE
## 5      9               A      FALSE
## 4      7               B      FALSE
```

```
# Rows 1, 5, and 4 in columns 1 through 2
df[c(1, 5, 4), 1:2]
```

```
##   values some_characters
## 1      1               F
## 5      9               A
## 4      7               B
```

```
# Rows 1, 5, and 4 in columns 1 and 3
df[c(1, 5, 4), c(1:3)]
```

```
##   values some_characters true_false
## 1      1               F       TRUE
## 5      9               A      FALSE
## 4      7               B      FALSE
```

$Column name subsetting

The subset data frame can be further subset using single square brackets and the specific names of the columns. HERE is a pretty thorough article on subsetting.

```
df["some_characters"]
```

```
##   some_characters
## 1               F
## 2               O
## 3               O
## 4               B
## 5               A
## 6               R
```

```
df[1, "some_characters"]
```

```
## [1] "F"
```

```
df[1:3, "some_characters"]
```

```
## [1] "F" "O" "O"
```

```r
df[c(1, 5, 6), c("values", "some_characters")]
```

```
##   values some_characters
## 1      1               F
## 5      9               A
## 6     11               R
```

Explore the content of dataframe

- `dim()` returns number of rows and columns
- `colnames()` returns names of all columns
- `str()` can be used with dataframe as with vectors to show structure
- `summary()` returns summary statistics of columns

```r
dim(df)
```

```
## [1] 6 3
```

```r
colnames(df)
```

```
## [1] "values"          "some_characters" "true_false"
```

```r
str(df)
```

```
## 'data.frame':    6 obs. of  3 variables:
##  $ values         : num  1 3 5 7 9 11
##  $ some_characters: chr  "F" "O" "O" "B" ...
##  $ true_false     : logi  TRUE FALSE TRUE FALSE FALSE FALSE
```

```r
summary(df)
```

```
##      values     some_characters    true_false
##  Min.   : 1.0   Length:6           Mode :logical
##  1st Qu.: 3.5   Class :character   FALSE:4
##  Median : 6.0   Mode  :character   TRUE :2
##  Mean   : 6.0
##  3rd Qu.: 8.5
##  Max.   :11.0
```