Basic Syntax: A Very Short Introduction

Dr. Aline Coutinho

Exploring R

When we are talking programming, *syntax* means any command that is *valid*. A valid command is not necessarily the most efficient command, but is one that works.

At the most basic level, R is a very powerful calculator. You insert certain commands and R provides a valid response.

```
1 + 1
```

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

```
5 * 2
```

```
## [1] 10
```

```
5 ^ 2
```

```
## [1] 25
```

Logical operators

You can use logical operators to assess the validity of logical tests. Below are some common logical operators:

- `==` means equal;
- `!=` means not equal;
- `>` means greater than;
- `<` means less than;
- `>=` means greater than or equal to;
- `<=` means less than or equal to;
- `|` means the Boolean operator OR;
- `&` means the Boolean operator AND.

```
10 == 10
```

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

```
10 != 100
```

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

```
10 > 100
```

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

```
10 < 100
```

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

```
10 >= 100
```

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

```
10 <= 100
```

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

```
10 == 10 | 100 == 100
```

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

```
10 == 10 & 10 == 100
```

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

Character strings, factors, and vectors

R works with **character strings**. The string must be inside quotation marks. Note that character strings allow the use of spaces:

```
"Aline is an amazing instructor"
```

```
## [1] "Aline is an amazing instructor"
```

You can also use logical operators with character strings:

```
"Aline is the most amazing instructor" == "Aline is an okay instructor"
```

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

```
"Aline is the most amazing instructor" != "Aline is an okay instructor"
```

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

Factors represent categorical data. They are treated specially by linear modelling functions such as `lm()` or `glm()`, which we will talk more about it on another tutorial. For now, note how factors can be created with the `factor()` function:

```
Best_shows <- factor(c("Mandalorian", "Schitts Creek", "Cobra Kai", "One Day at a Time"))

Best_shows
```

```
## [1] Mandalorian       Schitts Creek     Cobra Kai         One Day at a Time
## Levels: Cobra Kai Mandalorian One Day at a Time Schitts Creek
```

Vectors are a series of numbers or strings. You combine these values with the concatenate `c()` function. You can also create a vector with the `vector()` function.

```
c("Aline is awesome", "Aline is okay", "This is boring") == "Aline is awesome"
```

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

```
c(10, 100, 1000) != 100
```

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

```
# To create vectors with the vector() function.
# Note how the vector() function returns an empty vector.

x <- vector("numeric", length=5)
x
```

```
## [1] 0 0 0 0 0
```

> **Note:** A `vector` only contains objects of the same class. If a vector contains objects of different classes, R will *coerce* the elements in this vector into the same class. A `list` can contain objects of different classes such as numeric and character data.

Objects

R stores information in what we call *objects*. These are shortcuts (Imai, 2017) that include the list of all the data you are working with. We assign a value to an object with the *assignment operator* `<-`.

For example:

```
result <- 5 + 3
result
```

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

The following are basic classes of objects and data types:

1. Character;
2. Numeric;
3. Interger;

4. Logical (either TRUE or FALSE).

It is important to keep a few things in mind:

- Object names are case sentisive;
- We can store a string of characters by using quotation marks:

```
Aline <- "the most amazing instructor"
Aline
```

```
## [1] "the most amazing instructor"
```

- Numbers can be transformed into strings if you put them in quotation:

```
Result <- "10"
Result
```

```
## [1] "10"
```

- We cannot perform arithmetic operations for character strings. Think about it. Why?
- There is a function to tell us to which class an object belongs to: `class()`.

```
class(result)
```

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

```
class(Result)
```

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

```
class(Aline)
```

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

> **Note:** If we capitalize the function, an error message appears: `Error in Class(Coutinho) : could not find function "Class"`.

Functions

Certain commands are **functions**. A function is an object that performs a task.

```
# Suppose you want R to give you a sequence from 1 to 10.
# You type the function seq(1,10) and R responds to your command if it is a valid one:
seq(1, 10)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

We might want to add some **arguments** to certain functions. For example, we might want to draw the odd numbers of a certain sequence. A valid function is:

```r
seq(1,10, by =2)
```

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

```r
# A function to take the log of 3
log(3)
```

```
## [1] 1.098612
```

```r
# A function to take the square root of 9
sqrt(9)
```

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

A function has a body and arguments. Everything between {} are arguments or parameters we add to the function. Note how the function below is stored as an R object.

```r
subtract_and_divide<- function(x,y){
x-y
x/y
}

subtract_and_divide(100,10)
```

```
## [1] 10
```

> **Note:**
>
> The output only represents the last expression within {}.

The following are useful functions to calculate descriptive statistics:

- `min()`, for the minimum value
- `max()`, for the maximum value
- `mean()`, for the average value
- `range()`, for the range of the data
- `sum()`, for the sum of data

The argument inside the parenthesis is usually the name of variable to be summarized.

Some common functions to draw information about a specific vector are:

- `sort(name of vector)`
- `rev(name of vector)`
- `table(name of vector)`
- `unique(name of vector)`
- `lenght(name of vector)`

Tip: Some functions only work with some kinds of data, so it is important to know the data you're working with.

What is a data frame?

Data frames are the spreadsheets stored in the R environment once you import a dataset. They store tabular data in R, where every column has the same length as one another.

Data frames can be creates by calling the `read.table()` or the `read.csv()` functions. It is also possible to create a data frame by combining vectors together. So, for example:

```
# Use the data.frame() function

snacks <- data.frame(Number_of_Snacks_Today = 1:4, Snacks_Ive_Eaten = c(F, F, F, T))

snacks
```

```
##   Number_of_Snacks_Today Snacks_Ive_Eaten
## 1                      1            FALSE
## 2                      2            FALSE
## 3                      3            FALSE
## 4                      4             TRUE
```

```
# Suppose you have two vectors:

This_class_is_OK <- c(1,2,3,4,5)
This_class_is_awesome <- c(6,7, 8, 9, 10)

# You can combine two vectors in R:

data.frame (This_class_is_OK, This_class_is_awesome)
```

```
##   This_class_is_OK This_class_is_awesome
## 1                1                     6
## 2                2                     7
## 3                3                     8
## 4                4                     9
## 5                5                    10
```

```
# And even save that combination under a third object

hours_study <- data.frame("class1" = This_class_is_OK, "class2" = This_class_is_awesome)

# Note how you can change the name of variables with the 'data>frame()' function

hours_study
```

```
##   class1 class2
## 1      1      6
## 2      2      7
## 3      3      8
## 4      4      9
## 5      5     10
```

In R, when you are doing your analysis or creating visualizations, you will use what we call tibble. Tible are data frames, but in a format that facilitates your analysis.

I highly recommend you installing the package `tibble` with the following code: `install.packages("tibble", dependencies = TRUE)`.

You can get more information about tibbles by writing the following code on your console:`vignette("tibble")`

When data is in a tibble format, you can call a specific column by using the dollar sign `$`. To illustrate with the data frame above:

```
# Suppose you want to take the mean of hours students studied in class 2

mean(hours_study$class2)
```

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

Here are some important functions to explore the characteristics of a specific dataframe:

- `str(name of data frame)`: the structure function will give you information about number of rows and columns
- `nrow(name of data frame)`: just the number of rows
- `ncol(name of data frame)`: just the number of columns
- `names(name of data frame)`: to see the name of the variables
- `head(name of data frame)`: to see the first few rows of the dataframe. Important if you have a huge dataset.
- `tail(name of data frame)`: to see the last few rows of dataframe
- `view(name of data frame)`: allows you to see the dataframe in a spreadsheet format

Combining data frames

Suppose you have two different data frames and you want to combine them. It is possible to do so with the `cbind()` function. But before doing so, make sure your two data frames have the same number of rows otherwise they will not align well. Your code should look similar to this:

```
combined_dataset <- cbind(data frame 1, data frame 2)
```

However, I advise using the `_join` function family. The following are four functions, all allowing to merge data frames in different ways:

- `left_join`: includes all observations in the left data frame
- `right_join`: includes all observations in the right data frame;
- `inner_join`: includes only some observations that are in the data frames you are merging;
- `full_join`: includes all observations from both the data frames you are merging.

Here is how your code will probably look like:

```
left_join(data frame 1, data frame 2, by ="name of variable)
```

Additional resources

R Cheat Sheet that goes over R basic syntax

R for Data Science (Wickham and Grolemund), Chapter 4: Workflow: scripts

R for Data Science (Wickham and Grolemund), Chapter 10: Tibbles

Read more about tibbles here

References

Imai, K. (2017). *Quantitative Social Science: An Introduction.* Princeton University Press.