# Introduction to R

*Data Carpentry contributors*

---

### Learning Objectives

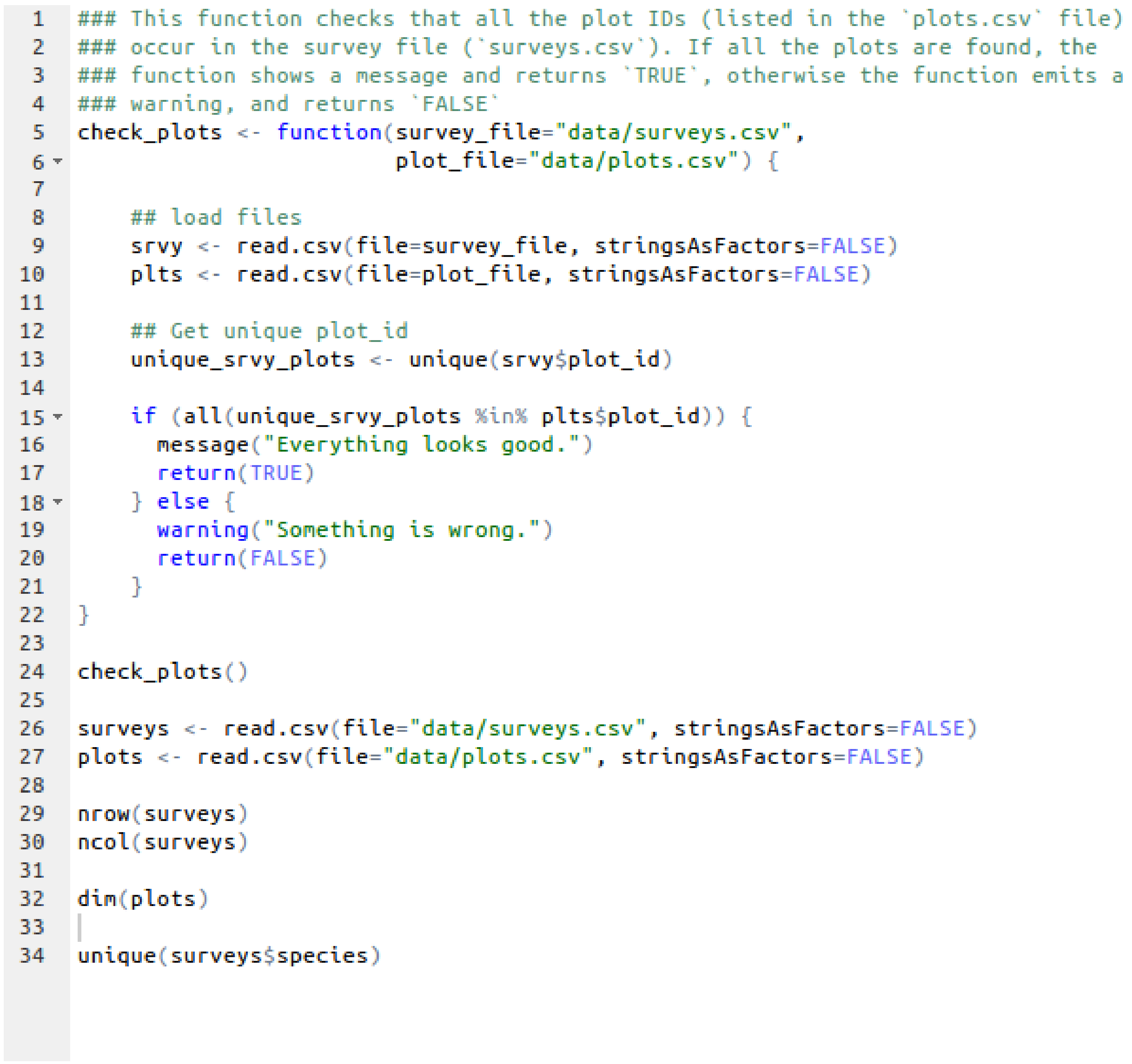
- Familiarize participants with R syntax
- Understand the concepts of objects and assignment
- Understand the concepts of vector and data types
- Get exposed to a few functions

---

## The R syntax

*Start by showing an example of a script*

- Point to the different parts:
- a function
- the assignment operator `<-`
- the `=` for arguments
- the comments `#` and how they are used to document function and its content
- the `$` operator
- Point to indentation and consistency in spacing to improve clarity



```r
### This function checks that all the plot IDs (listed in the `plots.csv` file)
### occur in the survey file (`surveys.csv`). If all the plots are found, the
### function shows a message and returns `TRUE`, otherwise the function emits a
### warning, and returns `FALSE`
check_plots <- function(survey_file="data/surveys.csv",
                        plot_file="data/plots.csv") {

    ## load files
    srvy <- read.csv(file=survey_file, stringsAsFactors=FALSE)
    plts <- read.csv(file=plot_file, stringsAsFactors=FALSE)

    ## Get unique plot_id
    unique_srvy_plots <- unique(srvy$plot_id)

    if (all(unique_srvy_plots %in% plts$plot_id)) {
      message("Everything looks good.")
      return(TRUE)
    } else {
      warning("Something is wrong.")
      return(FALSE)
    }
}

check_plots()

surveys <- read.csv(file="data/surveys.csv", stringsAsFactors=FALSE)
plots <- read.csv(file="data/plots.csv", stringsAsFactors=FALSE)

nrow(surveys)
ncol(surveys)

dim(plots)

unique(surveys$species)
```

## Creating objects

You can get output from R simply by typing in math in the console

```r
3 + 5
12/7
```

However, to do useful and interesting things, we need to assign *values* to *objects*. To create an object, we need to give it a name followed by the assignment operator `<-`, and the value we want to give it:

```r
weight_kg <- 55
```

Objects can be given any name such as `x`, `current_temperature`, or `subject_id`. You want your object names to be explicit and not too long. They cannot start with a number (`2x` is not valid, but `x2` is). R is case sensitive (e.g., `weight_kg` is different from `Weight_kg`). There are some names that cannot be used because they are the names of fundamental functions in R (e.g., `if`, `else`, `for`, see here for a complete list).

In general, even if it's allowed, it's best to not use other function names (e.g., `c`, `T`, `mean`, `data`, `df`, `weights`). In doubt check the help to see if the name is already in use. It's also best to avoid dots (`.`) within a variable name as in `my.dataset`. There are many functions in R with dots in their names for historical reasons, but because dots have a special meaning in R (for methods) and other programming languages, it's best to avoid them. It is also recommended to use nouns for variable names, and verbs for function names. It's important to be consistent in the styling of your code (where you put spaces, how you name variable, etc.). In R, two popular style guides are Hadley Wickham's and Google's.

When assigning a value to an object, R does not print anything. You can force to print the value by using parentheses or by typing the name:

```
weight_kg <- 55    # doesn't print anything
(weight_kg <- 55)  # but putting parenthesis around the call prints the value of `weight_kg`
weight_kg          # and so does typing the name of the object
```

Now that R has `weight_kg` in memory, we can do arithmetic with it. For instance, we may want to convert this weight in pounds (weight in pounds is 2.2 times the weight in kg):

```
2.2 * weight_kg
```

We can also change a variable's value by assigning it a new one:

```
weight_kg <- 57.5
2.2 * weight_kg
```

This means that assigning a value to one variable does not change the values of other variables. For example, let's store the animal's weight in pounds in a new variable, `weight_lb`:

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

and then change `weight_kg` to 100.

```
weight_kg <- 100
```

What do you think is the current content of the object `weight_lb`? 126.5 or 200?

### Challenge

What are the values after each statement in the following?

```
mass <- 47.5            # mass?
age  <- 122             # age?
mass <- mass * 2.0      # mass?
age  <- age - 20        # age?
mass_index <- mass/age  # mass_index?
```

## Vectors and data types

A vector is the most common and basic data structure in R, and is pretty much the workhorse of R. It's a group of values, mainly either numbers or characters. You can assign this list of values to a variable, just like you would for one item. For example we can create a vector of animal weights:

```r
weights <- c(50, 60, 65, 82)
weights
```

A vector can also contain characters:

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

There are many functions that allow you to inspect the content of a vector. `length()` tells you how many elements are in a particular vector:

```r
length(weights)
length(animals)
```

An important feature of a vector, is that all of the elements are the same type of data. The function `class()` indicates the class (the type of element) of an object:

```r
class(weights)
class(animals)
```

The function `str()` provides an overview of the object and the elements it contains. It is a really useful function when working with large and complex objects:

```r
str(weights)
str(animals)
```

You can add elements to your vector by using the `c()` function:

```r
weights <- c(weights, 90) # adding at the end of the vector
weights <- c(30, weights) # adding at the beginning of the vector
weights
```

What happens here is that we take the original vector `weights`, and we are adding another item first to the end of the other ones, and then another item at the beginning. We can do this over and over again to grow a vector, or assemble a dataset. As we program, this may be useful to add results that we are collecting or calculating.

We just saw 2 of the 6 **atomic vector** types that R uses: `"character"` and `"numeric"`. These are the basic building blocks that all R objects are built from. The other 4 are:

- `"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`) and that's all we're going to say about them
- `"raw"` that we won't discuss further

Vectors are one of the many **data structures** that R uses. Other important ones are lists (`list`), matrices (`matrix`), data frames (`data.frame`) and factors (`factor`).

**Challenge**

- **Question**: We've seen that atomic vectors can be of type character, numeric, integer, and logical. But what happens if we try to mix these types in a single vector?
- *Answer*: R implicitly converts them to all be the same type
- **Question**: Why do you think it happens?
- *Answer*: Vectors can be of only one data type. R tries to convert (=coerce) the content of this vector to find a "common denominator".
- **Question**: Can you draw a diagram that represents the hierarchy of the data types?
- *Answer*: `logical --> numeric --> character <-- logical`

```
## We've seen that atomic vectors can be of type character, numeric, integer, and
## logical. But what happens if we try to mix these types in a single
## vector?

## What will happen in each of these examples? (hint: use `class()` to
## check the data type of your object)
num_char <- c(1, 2, 3, 'a')

num_logical <- c(1, 2, 3, TRUE)

char_logical <- c('a', 'b', 'c', TRUE)

tricky <- c(1, 2, 3, '4')

## Why do you think it happens?

## Can you draw a diagram that represents the hierarchy of the data
## types?
```

## Missing data

As R was designed to work with data, it includes the concept of missing data (which is uncommon in other programming languages). Missing data are represented as `NA`.

```
planets <- c("Mercury", "Venus", "Earth", "Mars", "Jupiter", "Saturn", "Uranus",
             "Neptune", NA)
```

When doing operations on numbers, most functions will return `NA` if the data you are working with include missing values. It is a safer behavior as otherwise you may overlook that you are dealing with missing data. You can add the argument `na.rm=TRUE` to calculate the result while ignoring the missing values.

```
heights <- c(2, 4, 4, NA, 6)
mean(heights)
max(heights)
mean(heights, na.rm = TRUE)
max(heights, na.rm = TRUE)
```

If your data include missing values, you may want to become familiar with the functions `is.na()`, `na.omit()`, and `complete.cases()`.

Next, we will use the “surveys” dataset to explore the `data.frame` data structure, which is one of the most common types of R objects.