Working with function arguments

Learning Objectives

After completing this tutorial, you will be able to:

- Define the purpose of a function argument.
- Use default vs. required function arguments in a function.

What you need

You will need a computer with internet access to complete this lesson.

In the previous lessons, we have used many different functions and function arguments to customize our code.

For example, we used numerous arguments to plot our data including:

1. `main` to add a title
2. `axes = FALSE` to remove the axes of our plot
3. `box = FALSE` to remove the box surrounding the plot.

In the example below, we call each argument by name and then assign it a value based on the type of argument it is. For example the value for the `main =` argument is a text string which is the title that we want `R` to add to our plot.

```
# import and plot landsat
landsat_ndvi <- raster("data/week6/outputs/landsat_ndvi.tif")
plot(landsat_ndvi,
     main = "ndvi title - rendered using a function argument",
     axes = FALSE,
     box = FALSE)
```

Function arguments allow us to customize how a function runs. For example, we can use the `plot()` function to plot many different types of data! And we can use the `main` argument to customize the title. We use axes and box to customize how the plot looks. This is a powerful function as it can be used to do many different things and is customizable in many ways that we may need / want!

Argument order matters

Let's next talk about the order or arguments in a function. `R` has three ways that arguments supplied by you are matched to the *formal arguments* of the function definition:

1. **By complete name:** i.e. you type `main = ""` and `R` matches main to the argument called `main`.
2. **By order or position when you call an argument:** i.e. you call `plot(raster, "title here")`, `R` will read these two variables in the order that you provide them. This can cause the function to fail if they are not in the right order!
3. **By partial name:** (matching on initial n characters of the argument name) - we are not going to review this in class. Beware using this "feature".

Arguments are matched in the manner outlined above.

- R first tries to find arguments according to the complete name,
- then by partial matching of names,
- and finally by position.

With that in mind, let's look at the help for `read.csv()`:

```
# view help for the csv function
?read.csv
```

There's a lot of information available in the help. The the most important part is the first couple of lines:

```
read.csv(file, header = TRUE, sep = ",", quote = "\"",
         dec = ".", fill = TRUE, comment.char = "", ...)
```

This tells us that `read.csv()` has one argument, `file`, that doesn't have a default value, and six other arguments that do have a default value.

Now we understand why the following code returns an error:

```
precip_data <- read.csv(FALSE, "data/week2/precipitation/precip-boulder-aug-oct-2013.csv")
```

The code above fails because `FALSE` is assigned to `file` and the filename is assigned to the argument `header`.

Default function arguments

We have passed arguments to functions in two ways:

1. Directly: `plot(landsat_ndvi)`,
2. and by name: `read.csv(file = "data/week2/precipitation/precip-boulder-aug-oct-2013.csv", header = FALSE)`.

We can pass the arguments to `read.csv` without naming them if they are in the order that R expects.

```
precip_data <- read.csv("data/week2/precipitation/precip-boulder-aug-oct-2013.csv",
                FALSE)
```

However, the position of the arguments matter if they are not named. Does the code below return an error?

```
# import csv
precip_data <- read.csv(header = FALSE,
                file = "data/week2/precipitation/precip-boulder-aug-oct-2013.csv")
```

But this code below doesn't work. Make sense?

```
dat <- read.csv(FALSE,
                "data/week2/precipitation/precip-boulder-aug-oct-2013.csv")
```