Preliminaries in R

How to “talk” to R

How to “talk” to R

1. Open an **R session**
2. At the **prompt** in the **console**, enter an **R expression**
3. Read R’s “response” (the **output**)
4. Repeat 2 and 3
5. Close the R session

Opening an R session

An **R session** is an instance of you using R.

To open an R session, double-click on the icon for "RStudio" on you computer. When RStudio opens, you will be in a "fresh" R session, unless you restore a saved session (which I strongly recommend against).

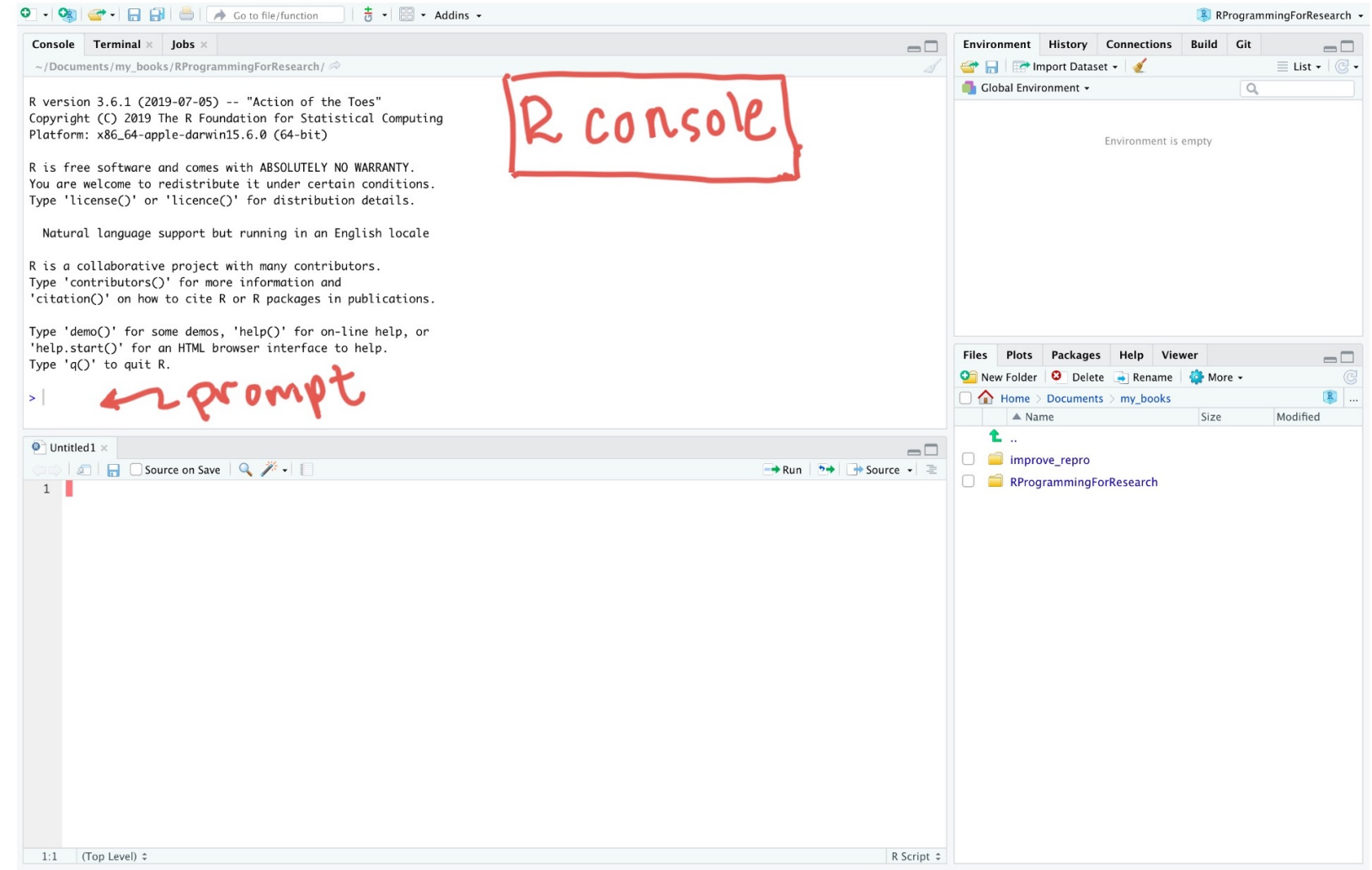
This means that, once you open RStudio, you will need to "set up" your session, including loading any packages you need (which we'll talk about later) and reading in any data (which we'll also talk about).

The prompt in the console

In RStudio, there screen is divided into several "panes". We'll start with the pane called "Console".

The **console** lets you "talk" to R. This is where you can "talk" to R by typing an **expression** at the **prompt** (the caret symbol, ">"). You press the "Return" key to send this message to R.

The prompt in the console



How R might respond

Once you press "Return", R will respond in one of three ways:

1. R does whatever you asked it to do with the expression and prints the output (if any) of doing that, as well as a new prompt so you can ask it something new
2. R doesn't think you've finished asking you something, and instead of giving you a new prompt ("$>$") it gives you a "$+$". This means that R is still listening, waiting for you to finish asking it something.
3. R tries to do what you asked it to, but it can't. It gives you an **error message**, as well as a new prompt so you can try again or ask it something new.

R expressions, function calls, and objects

To "talk" with R, you need to know how to give it a complete **expression**.

Most expressions you'll want to give R will be some combination of two elements:

1. **Function calls**
2. **Object assignments**

We'll go through both these pieces and also look at how you can combine them together for some expressions.

R expressions, function calls, and objects

According to John Chambers, one of the creators of R's precursor S:

1. Everything that exists in R is an **object**
2. Everything that happens in R is a **call to a function**

Function calls

In general, function calls in R take the following structure:

```
## Generic code (this won't run)
function_name(formal_argument_1 = named_argument_1,
              formal_argument_2 = named_argument_2,
              [etc.])
```

A function call forms a complete R expression, and the output will be the result of running `print` or `show` on the object that is output by the function call.

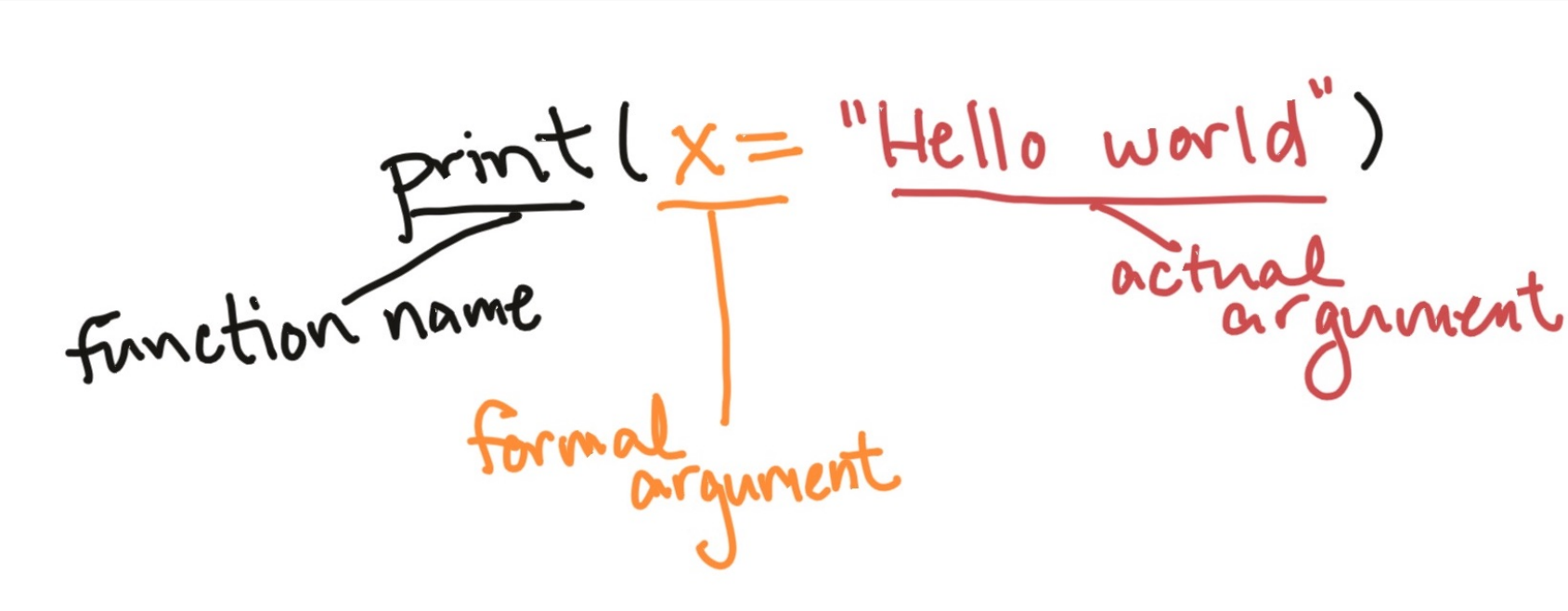
Function calls

Here is an example of this structure:

```
print(x = "Hello world")
```

```
## [1] "Hello world"
```

Function calls



In this example, we're **calling** a function with the **name** `print`. It has one **argument**, with a **formal argument** of `x`, which in this call we've provided the **named argument** "Hello world".

Function calls

The **arguments** are how you customize the call to an R function.

For example, you can use change the named argument value to print different messages with the `print` function:

```
print(x = "Hello world")
```

```
## [1] "Hello world"
```

```
print(x = "Hi Fort Collins")
```

```
## [1] "Hi Fort Collins"
```

Function calls

Some functions do not require any arguments. For example, the `getRversion` function will print out the version of R you are using.

```
getRversion()
```

```
## [1] '4.0.2'
```

Function calls

Some functions will accept multiple arguments. For example, the `print` function allows you to specify whether the output should include quotation marks, using the `quote` formal argument:

```
print(x = "Hello world", quote = TRUE)
```

```
## [1] "Hello world"
```

```
print(x = "Hello world", quote = FALSE)
```

```
## [1] Hello world
```

Function calls

Arguments can be **required** or **optional**.

For a required argument, if you don't provide a value for the argument when you call the function, R will respond with an error. For example, `x` is a **required argument** for the `print` function, so if you try to call the function without it, you'll get an error:

```
print()
```

```
Error in print.default() : argument "x" is
  missing, with no default
```

Function calls

For an **optional argument** on the other hand, R knows a **default value** for that argument, so if you don't give it a value for that argument, it will just use the default value for that argument.

For example, for the `print` function, the `quote` argument has the default value `TRUE`. So if you don't specify a value for that argument, R will assume it should use `quote = TRUE`. That's why the following two calls give the same result:

```
print(x = "Hello world", quote = TRUE)
```

```
## [1] "Hello world"
```

```
print(x = "Hello world")
```

```
## [1] "Hello world"
```

Function helpfiles

Often, you'll want to find out more about a function, including:

- Examples of how to use the function
- Which arguments you can include for the function
- Which arguments are required versus optional
- What the default values are for optional arguments.

You can find out all this information in the function's **helpfile**, which you can access using the function `?`.

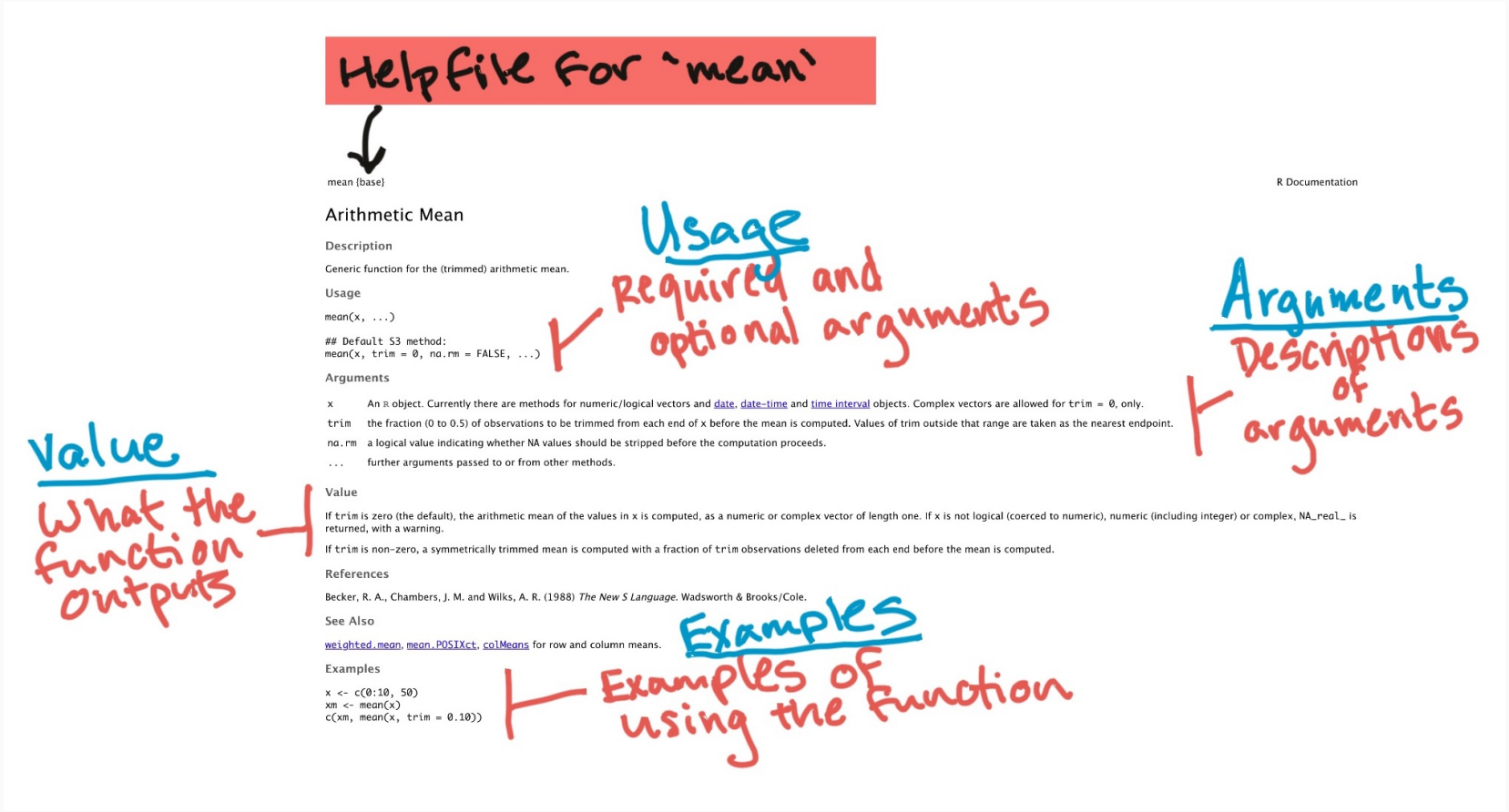
Function helpfiles

For example, the `mean` function will let you calculate the mean (average) of a group of numbers. To find out more about this function, at the console type:

```
?mean
```

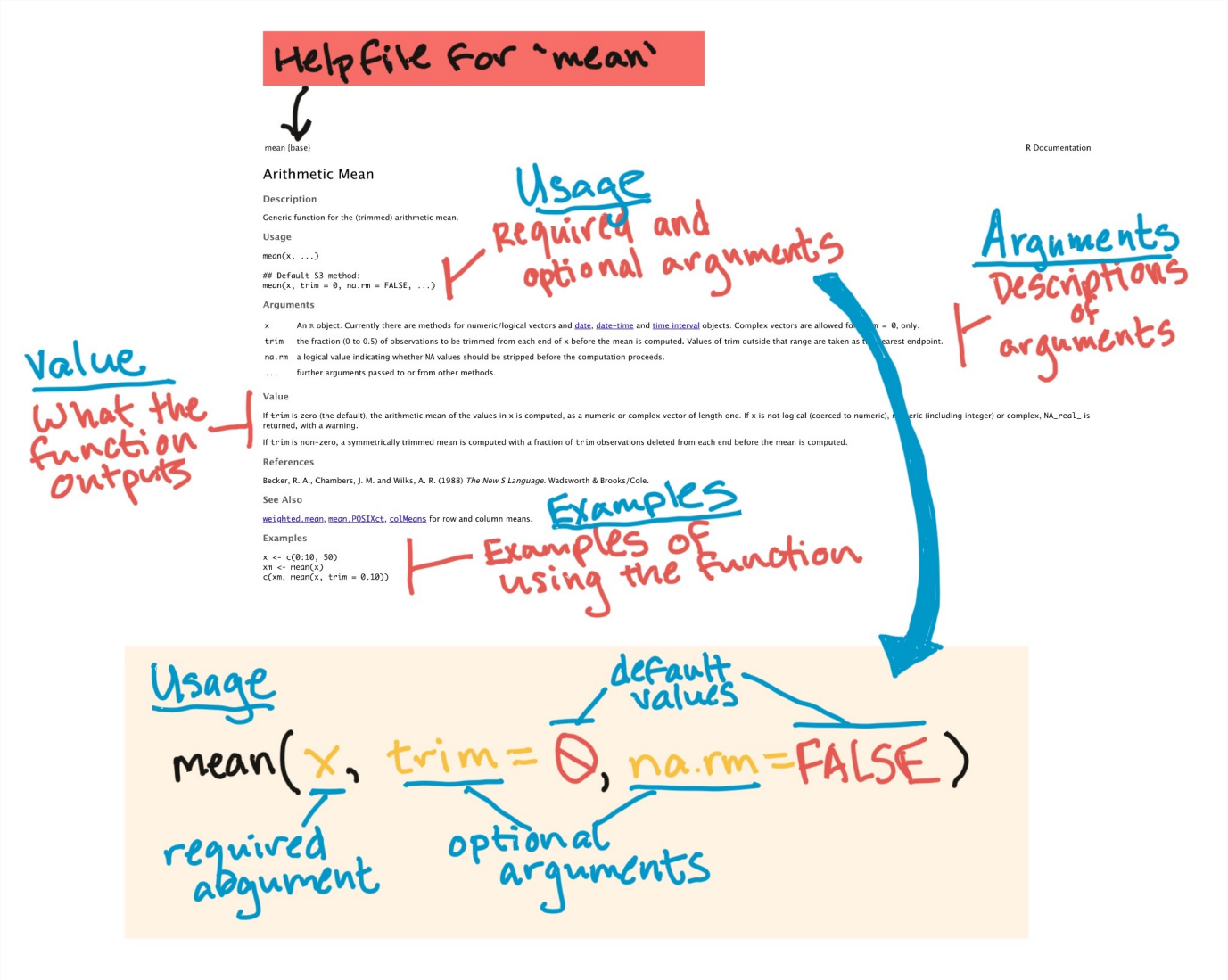
This will open a helpfile in the "Help" pane in RStudio.

Function helpfiles



The helpfile includes sections giving the function's **usage**, **arguments**, **value**, and **examples**.

Function helpfiles



You can figure out which arguments are **required** and which are **optional** in the Usage section of the helpfile.

Operators

There's one class of functions that looks a bit different from others. These are the infix **operator** functions.

Instead using parentheses after the function name, they usually go *between* two arguments.

One common example is the + operator:

```
2 + 3
```

```
## [1] 5
```

Operators

There are operators for several mathematical functions: `+`, `-`, `*`, `/`.

There are also other operators, including **logical operators** and **assignment operators**, which we'll cover later.

Objects, object names, and assignment expressions

Function calls will usually produce something called an **object**.

If you just call a function, as we've been doing, then R will respond by printing out that object.

However, we'll often want to use that object some more. For example, we might want to use it as an argument later in our "conversation" with R, when we call another function later.

If you want to re-use the results of a function call later, you can **assign** that **object** to an **object name**.

This kind of expression is called an **assignment expression**.

Assignment expressions

The **gets arrow**, <-, is R's assignment operator. It takes whatever you've created on the right hand side of the <- and saves it as an object with the name you put on the left hand side of the <- :

```
## Note: Generic code-- this will not work
[object name] <- [object]
```

Assignment expressions

For example, if I just type `"Hello world"`, R will print it back to me, but won't save it anywhere for me to use later:

```
"Hello world"
```

```
## [1] "Hello world"
```

Assignment expressions

However, if I assign it to an object, I can "refer" to that object in a later expression.

For example, the code below assigns the **object** `"Hello world"` the **object name** `message`. Later, I can just refer to this object using the name `message`, for example in a function call to the `print` function:

```
message <- "Hello world"
print(x = message)
```

```
## [1] "Hello world"
```

History of <-

Assignment expressions

When you enter an **assignment expression** like this at the R console, if everything goes right, then R will "respond" by giving you a new prompt, without any kind of message.

However, there are three ways you can check to make sure that the object was assigned to the object name:

1. Enter the object's name at the prompt and press return. The default if you do this is for R to "respond" by calling the `print` function with that object as the `x` argument.
2. Call the `ls` function (which doesn't require any arguments). This will list all the object names that have been assigned in the current R session.
3. Look in the "Environment" pane in RStudio. This also lists all the object names that have been assigned in the current R session.

Assignment expressions

Here's an example of the first two strategies:

1. Enter the object's name at the prompt and press return:

```
message
```

```
## [1] "Hello world"
```

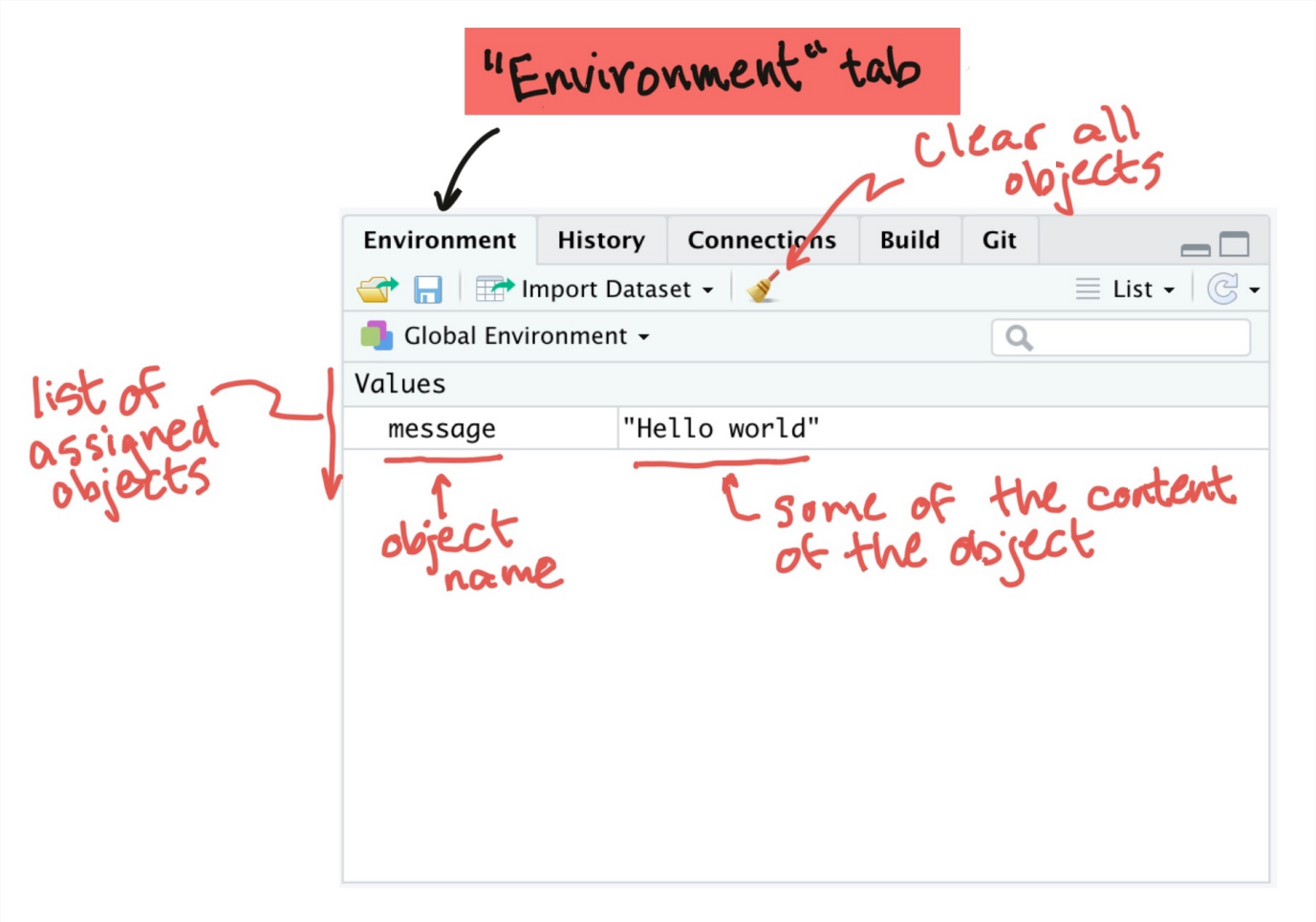
2. Call the `ls` function:

```
ls()
```

```
## [1] "message"
```

"Environment" pane

Here's an example of the third method:



Object names

There are some absolute **rules** for the names you can use for an object name:

- Use only letters, numbers, and underscores
- Don't start with anything but a letter

Assigning objects to object names

If you try to assign an object to a name that doesn't follow the "hard" rules, you'll get an error.

For example, all of these expressions will give you an error:

```
1message <- "Hello world"
_message <- "Hello world"
message! <- "Hello world"
```

Object names

There are also some **guidelines** for picking *good* object names:

From Hadley Wickham's R style guide

- Use lower case for variable names (`message`, not `Message`)
- Use an underscore as a separator (`message_one`, not `messageOne`)
- Avoid using names that are already defined in R (e.g., don't name an object `mean`, because a `mean` function exists)

“Composing” to combine function calls

What if you want to “compose” a call from more than one function call?

One way to do it is to assign the output from the first function call to a name and then use that name for the next call.

For example:

```
message <- paste("Hello", "world")
print(x = message)
```

```
## [1] "Hello world"
```

“Composing” to combine function calls

You can also “nest” one function call inside another function call. For example:

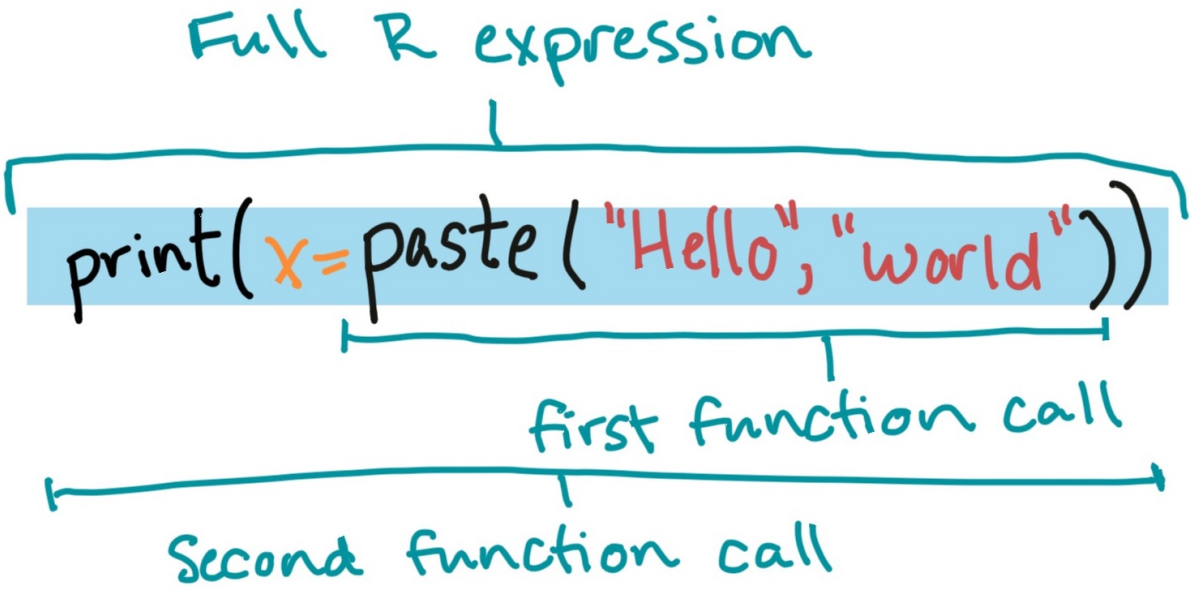
```
print(x = paste("Hello", "world"))
```

```
## [1] "Hello world"
```

Just like with math, the order that the functions are evaluated moves from the inner set of parentheses to the outer one.

There’s one more way we’ll look at later...

“Composing” to combine function calls



1. paste ("Hello", "world") ⇨ "Hello world"

2. print(x = "Hello world")

Using R scripts

The console can be great for quick functions to explore the data.

However, for most data analysis work you'll want to use a script, so you can save all the expressions you used for the analysis.

This improves the *reproducibility* of your analysis.

R scripts

An **R script** is a plain text file where you can write down and save R code.

When you write, run, and save your R code in a script rather than running it one line at a time in the console, you can easily go back and re-do exactly what you did again later.

You can also share the script for someone else to use, or run it on a different computer.

R scripts

RStudio has one pane that shows any R scripts you have open. If you'd like to create new R scripts, you can do that in RStudio with the following steps:

- Open a new script file in RStudio: `File -> New File -> R Script`.
- I recommend that you make an "R" folder in all of the R project directories that you create and save all your script files in that folder.
- Save scripts using the extension `.R`

R scripts

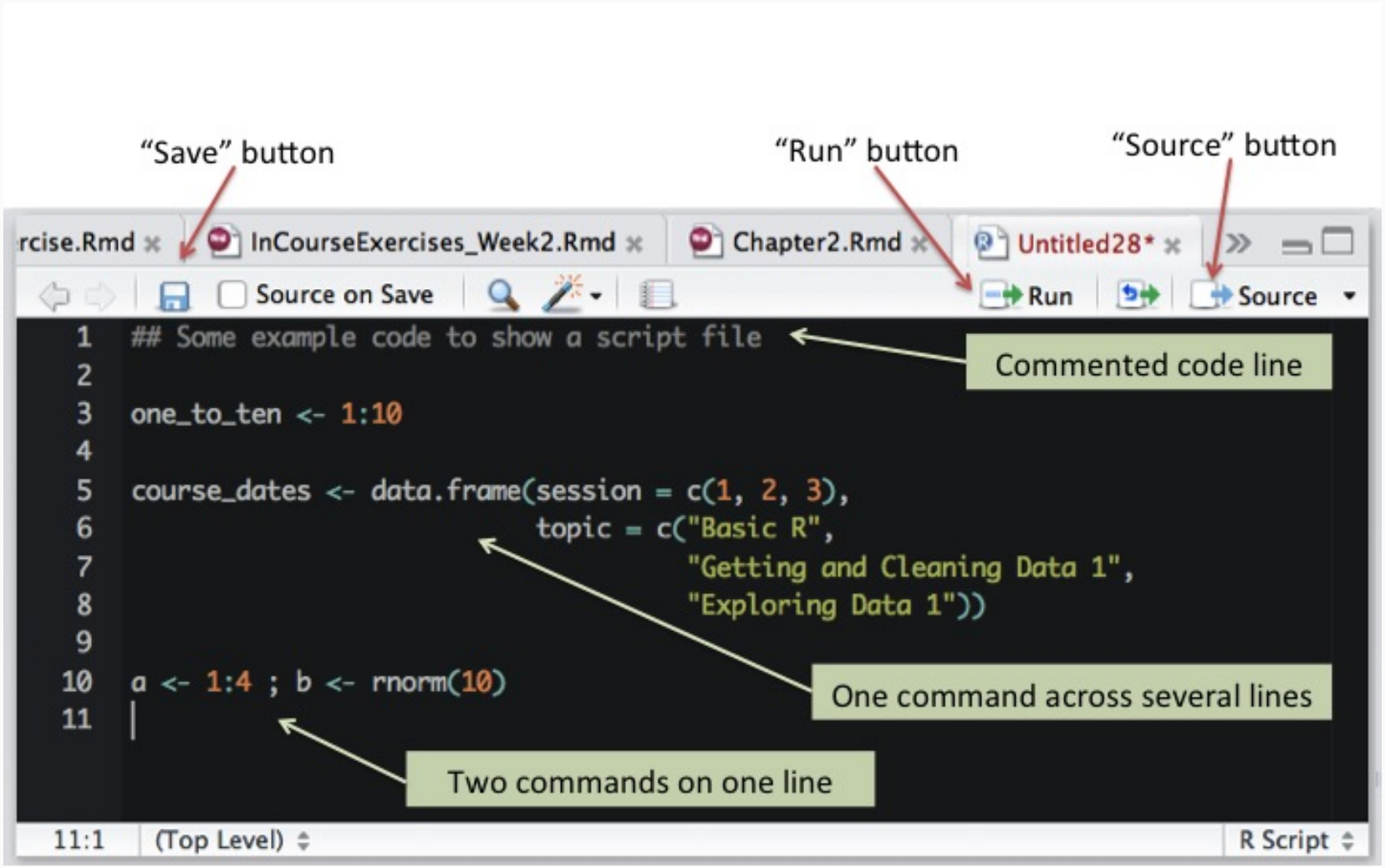
Running code in R scripts line-by-line:

- To run code from an R script file in RStudio, you can use the `Run` button (or `Command-R`).
- This will run whatever's on your cursor line or whatever's highlighted.

Sourcing an R script (i.e., running all the code saved in the script):

- To run the whole script, you can also use the `source` function with the filename.
- You can also use the "Source" button on the script pane.

R scripts


"Save" button
"Run" button
"Source" button
rcise.Rmd
InCourseExercises_Week2.Rmd
Chapter2.Rmd
Untitled28*
Source on Save
Run
Source
1 ## Some example code to show a script file
2
3 one_to_ten <- 1:10
4
5 course_dates <- data.frame(session = c(1, 2, 3),
6 topic = c("Basic R",
7 "Getting and Cleaning Data 1",
8 "Exploring Data 1"))
9
10 a <- 1:4 ; b <- rnorm(10)
11
Commented code line
One command across several lines
Two commands on one line
11:1
(Top Level)
R Script

Comment characters

Sometimes, you'll want to include notes in your code. You can do this in all programming languages by using a **comment character** to start the line with your comment.

In R, the comment character is the hash symbol, #. R will skip any line that starts with # in a script.

```
# Don't print this.
"But print this"
```

```
## [1] "But print this"
```

Closing an R session

Do **not** save the history of your R session when you close RStudio.
Instead, get in the habit of writing your R code in reproducible formats (R scripts, RMarkdown documents)