Introduction to R: Functions

Research Methods for Human Inquiry

Andrew Perfors

Using functions

Functions

There are not enough symbols on the keyboard to do everything you might want to do... so there are only a few operators

Most things are **functions**

Example: square root

```
> sqrt(25)
[1] 5
```

- The function is called `sqrt()`
- The 25 is the "**argument**" to the function.

Arguments

Every function has arguments. You can think of functions like recipes, and arguments like the ingredients. You can't make the recipe without the right ingredients, and the recipe combines the ingredients in a specific way.

Ingredients

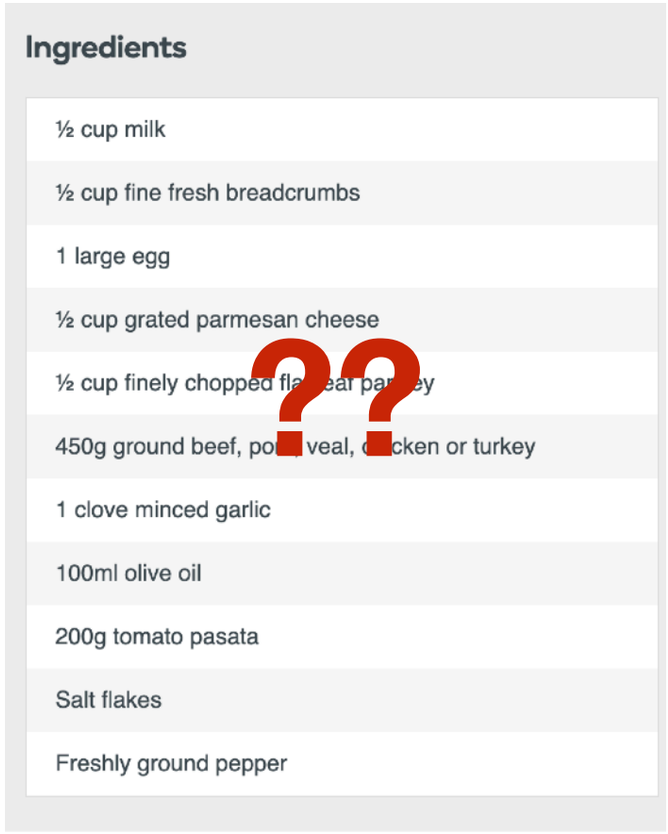
- ½ cup milk
- ½ cup fine fresh breadcrumbs
- 1 large egg
- ½ cup grated parmesan cheese
- ½ cup finely chopped flat leaf parsley
- 450g ground beef, pork, veal, chicken or turkey
- 1 clove minced garlic
- 100ml olive oil
- 200g tomato pasata
- Salt flakes
- Freshly ground pepper

`meatball_recipe(milk,breadcrumbs,egg,etc..)` → `meatballs!`

`sqrt(25)` → `5`

Arguments

Just like a recipe, it won't work if you don't give it the right ingredients (arguments)



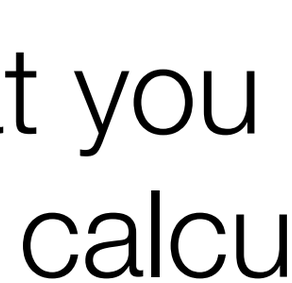
```
> sqrt()
Error in sqrt() : 0 arguments passed to 'sqrt' which requires 1
```

Is this a cooking show?
Are you going to teach us how to cook?

Because I really like meatballs.
And food in general.

Some functions that you might see on a scientific calculator

`sqrt()` - Square root
`round()` - Round a number
`log()` - Logarithm
`exp()` - Exponentiation
`abs()` - Absolute value



```
> sqrt(4)
[1] 2
> round(3.24)
[1] 3
> log(32)
[1] 3.465736
> exp(2)
[1] 7.389056
> abs(-3)
[1] 3
```

Some arguments have defaults

- A lot of arguments have “**default values**”.
- If you don’t tell R what value to use, it uses the default

```
> round( x = 3.1415 )
[1] 3
```

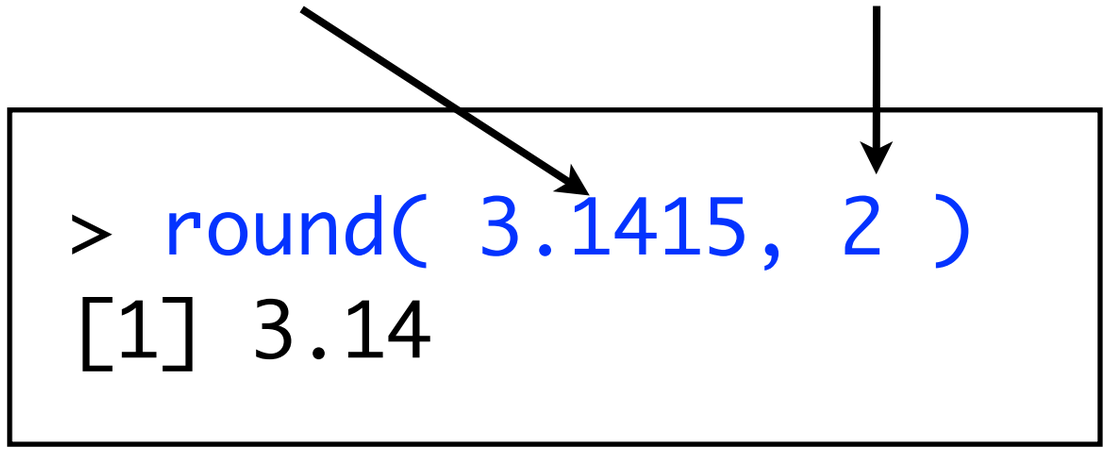
The default number of digits to round to is zero, so that’s what R uses here

Functions with multiple arguments

- Many functions can “take” more than one argument;
- Separate the arguments with commas.

Argument #1: The number that needs to be rounded

Argument #2: How many digits to round it to?



Arguments have names

- Most of the time, the arguments have "**names**",
- You can use the names when typing commands

The number that needs to be rounded is called `x`

The number of digits to round to is called `digits`

```
> round( x = 3.1415, digits = 2 )
[1] 3.14
```

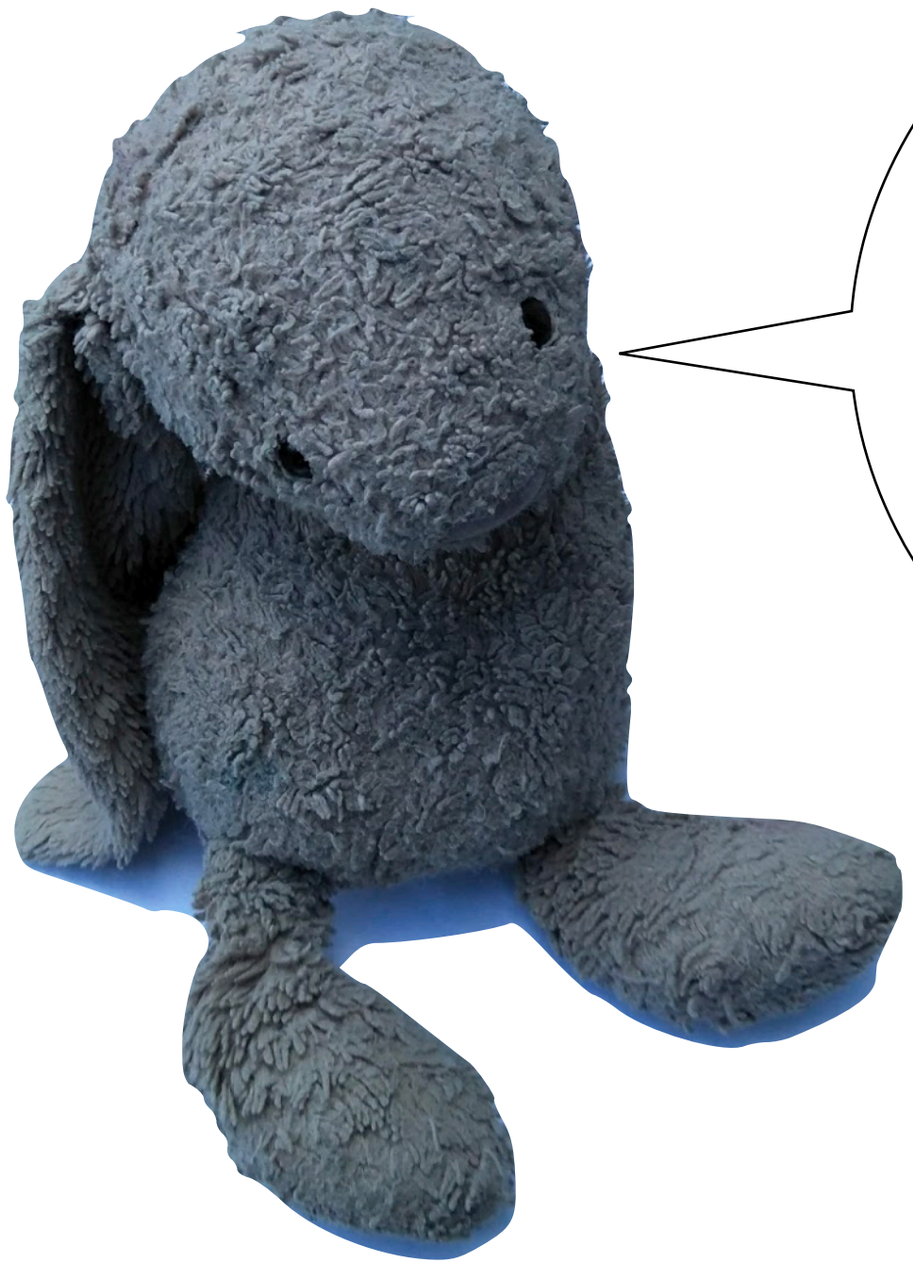
Arguments have names

If you specify the names, the order doesn't matter. But if you don't, it does!

```
> round (x=3.14159, digits=2)
[1] 3.14
> round (digits=2, x=3.14159)
[1] 3.14
```

```
> round(3.14159,2)
[1] 3.14
> round(2,3.14159)
[1] 2
```

This is known as a "silent fail." What we input didn't make sense, so it just went with the default, with no warning. Be careful of these!



I get it! You could make a function for awesomeness which requires both Bunny and Gladly!

But Bunny has to come first of course.

```
awesomeness(Bunny,Gladly)
```

“Nesting” functions

Just as a recipe can use the output of other recipes as ingredients, so too functions can use the output of other functions as arguments

Spaghetti

INGREDIENTS

1 lb. spaghetti

DIRECTIONS

1 In a large pot of boiling salted water, cook spaghetti according to package instructions. Drain.

Meatballs

1 lb. ground beef

1/3 c. bread crumbs

1/4 c. finely chopped parsley

1/4 c. freshly grated Parmesan, plus more for serving

1 egg

2 garlic cloves, minced

Kosher salt

1/2 tsp. red pepper flakes

2 tbsp. extra-virgin olive oil

1/2 c. onion, finely chopped

1 (28-oz.) can crushed tomatoes

1 bay leaf

Freshly ground black pepper

GET INGREDIENTS Powered by Chicory

2 In a large bowl, combine beef with bread crumbs, parsley, Parmesan, egg, garlic, 1 teaspoon salt, and red pepper flakes. Mix until just combined then form into 16 balls.

3 In a large pot over medium heat, heat oil. Add meatballs and cook, turning occasionally, until browned on all sides, about 10 minutes. Transfer meatballs to a plate.

4 Add onion to pot and cook until soft, 5 minutes. Add crushed tomatoes and bay leaf. Season with salt and pepper and bring to a simmer. Return meatballs to pot and cover. Simmer until sauce has thickened, 8 to 10 minutes.

Spaghetti & meatballs

INGREDIENTS

1 lb. spaghetti

1 lb. ground beef

1/3 c. bread crumbs

1/4 c. finely chopped parsley

1/4 c. freshly grated Parmesan, plus more for serving

1 egg

2 garlic cloves, minced

Kosher salt

1/2 tsp. red pepper flakes

2 tbsp. extra-virgin olive oil

1/2 c. onion, finely chopped

1 (28-oz.) can crushed tomatoes

1 bay leaf

Freshly ground black pepper

GET INGREDIENTS Powered by Chicory

DIRECTIONS

1 In a large pot of boiling salted water, cook spaghetti according to package instructions. Drain.

2 In a large bowl, combine beef with bread crumbs, parsley, Parmesan, egg, garlic, 1 teaspoon salt, and red pepper flakes. Mix until just combined then form into 16 balls.

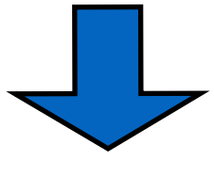
3 In a large pot over medium heat, heat oil. Add meatballs and cook, turning occasionally, until browned on all sides, about 10 minutes. Transfer meatballs to a plate.

4 Add onion to pot and cook until soft, 5 minutes. Add crushed tomatoes and bay leaf. Season with salt and pepper and bring to a simmer. Return meatballs to pot and cover. Simmer until sauce has thickened, 8 to 10 minutes.

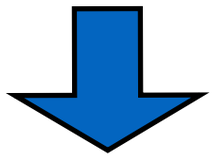
“Nesting” functions

Functions can take other functions as arguments!

```
sqrt ( round(4.45) )
```



```
sqrt ( 4 )
```



```
2
```

The inside functions “convert” to their answers.
So to evaluate it, you work from the inside out.

“Nesting” functions

Functions can take other functions as arguments!

```
sqrt ( round(4.45) )
```

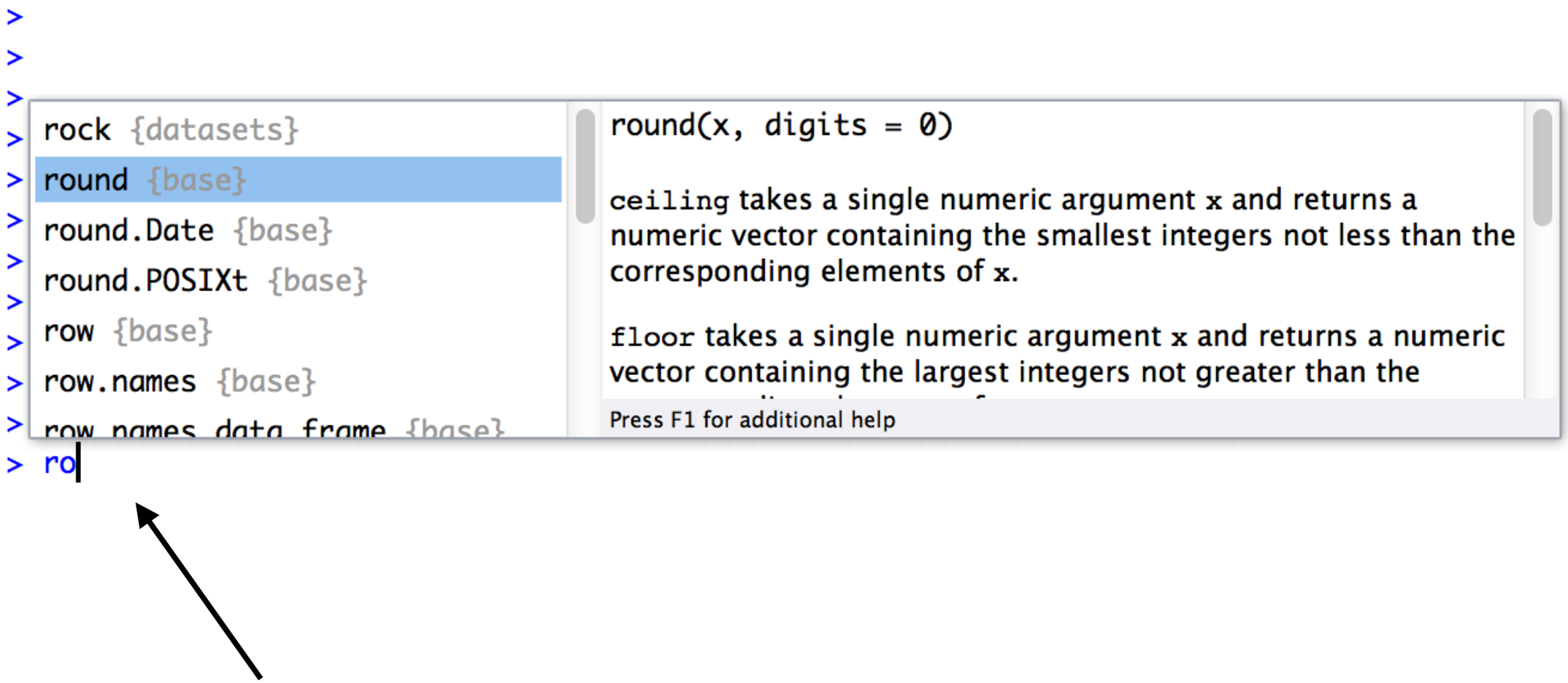
Note that the parentheses are balanced. If they aren’t, this can cause a problem because R won’t know what goes inside what.

```
> sqrt(round4.33))
Error: unexpected ')' in "sqrt(round4.33))"
```

```
> sqrt(round(4.33)
+
```

Navigation hints

“Tab” autocomplete



Type `ro` and then hit tab.

Brings up a window showing possible commands you might like to use

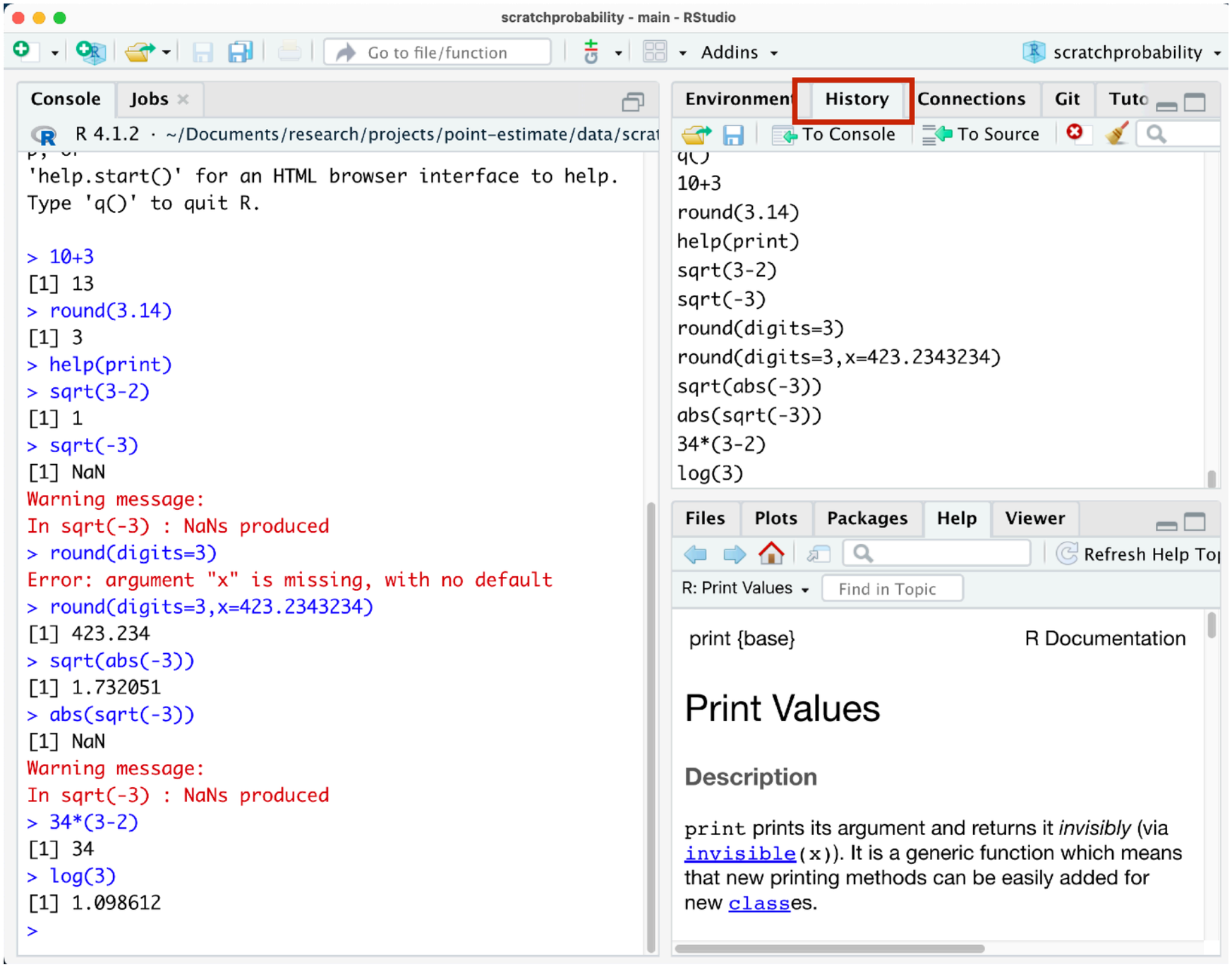
The up arrow

```
> 3+4+5
[1] 12
> 3*4*5
[1] 60
> sqrt(4)
[1] 2
```

If you type the up arrow it will let you go through all your previous commands in reverse order.

(This would first show **`sqrt(4)`**, then **`3*4*5`**, then **`3+4+5`**)

The history panel



Shows you all your previous commands in reverse order

The `help()` function

Suppose you want to know more about a function…

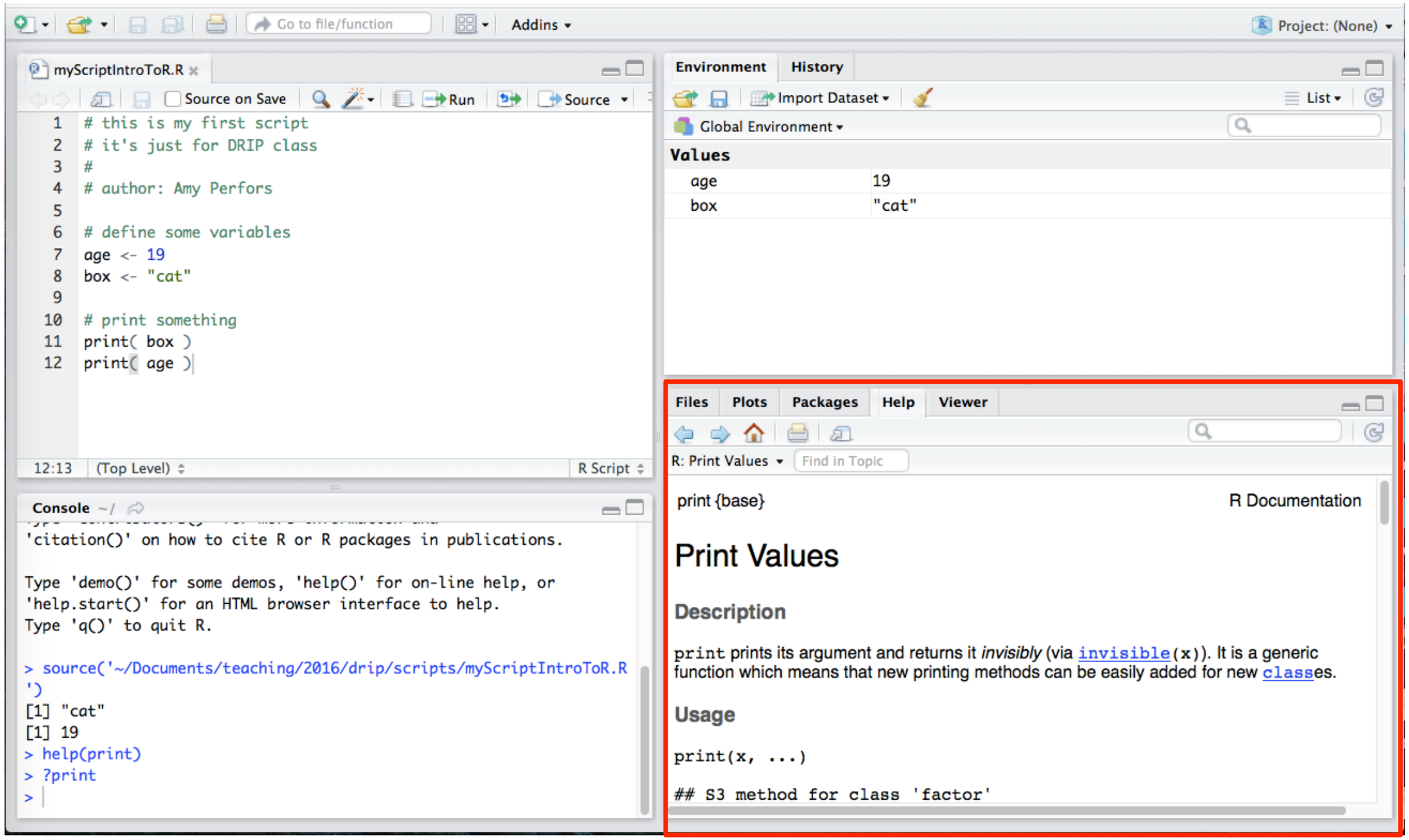
```
print( “hello” )
```

Every function comes with documentation

`help(print)` or `?print`

R documentation

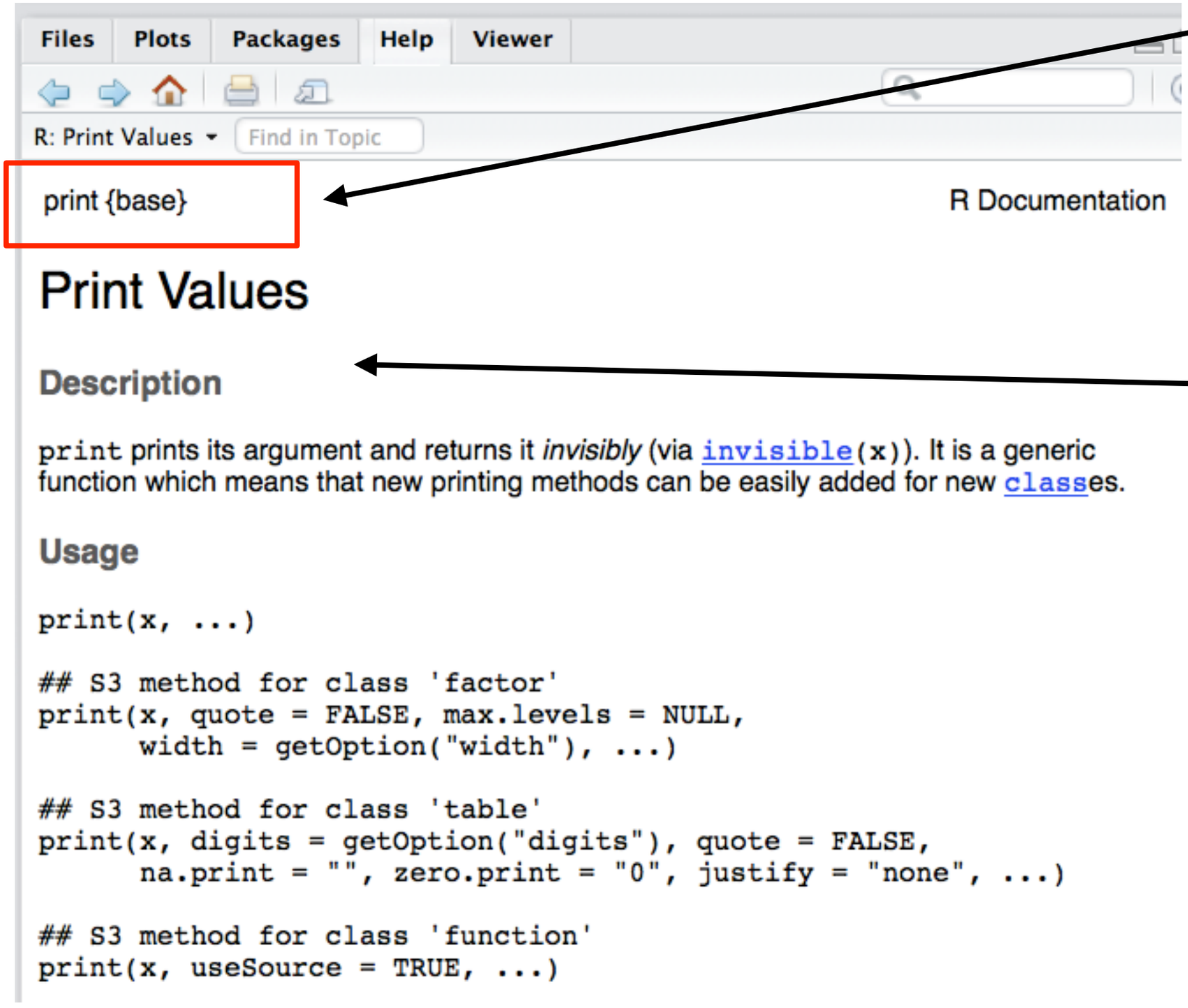
When you type `help()`, it shows up in the lower right panel



R documentation

tells you what function the documentation is for

describes what the function does



R documentation

Files | Plots | Packages | Help | Viewer

R: Print Values | Find in Topic

print {base} R Documentation

Print Values

Description

`print` prints its argument and returns it *invisibly* (via `invisible(x)`). It is a generic function which means that new printing methods can be easily added for new `class`es.

Usage

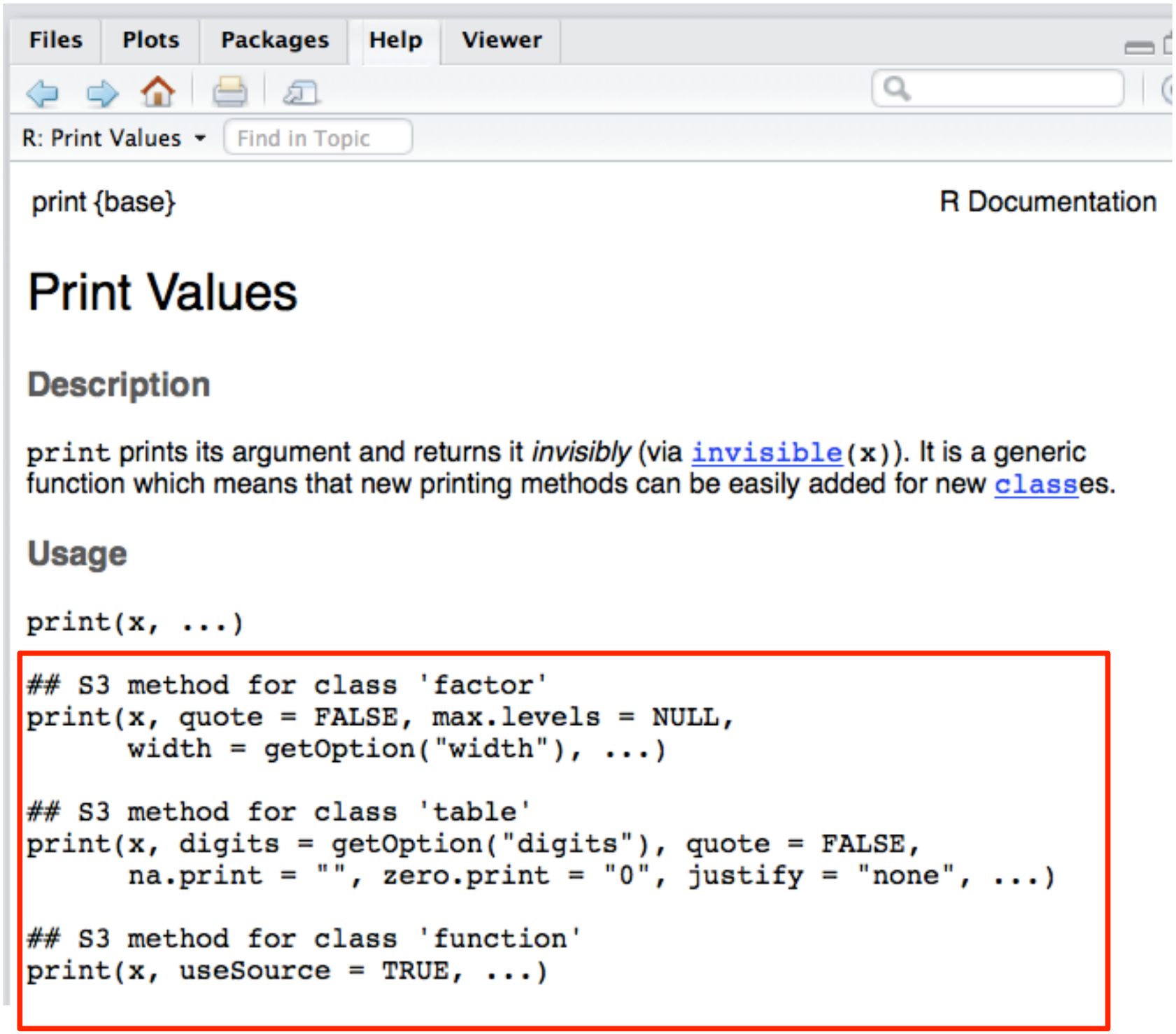
```
print(x, ...)

## S3 method for class 'factor'
print(x, quote = FALSE, max.levels = NULL,
      width = getOption("width"), ...)

## S3 method for class 'table'
print(x, digits = getOption("digits"), quote = FALSE,
      na.print = "", zero.print = "0", justify = "none", ...)

## S3 method for class 'function'
print(x, useSource = TRUE, ...)
```

what you have to type in order to get the function to run

which arguments are obligatory

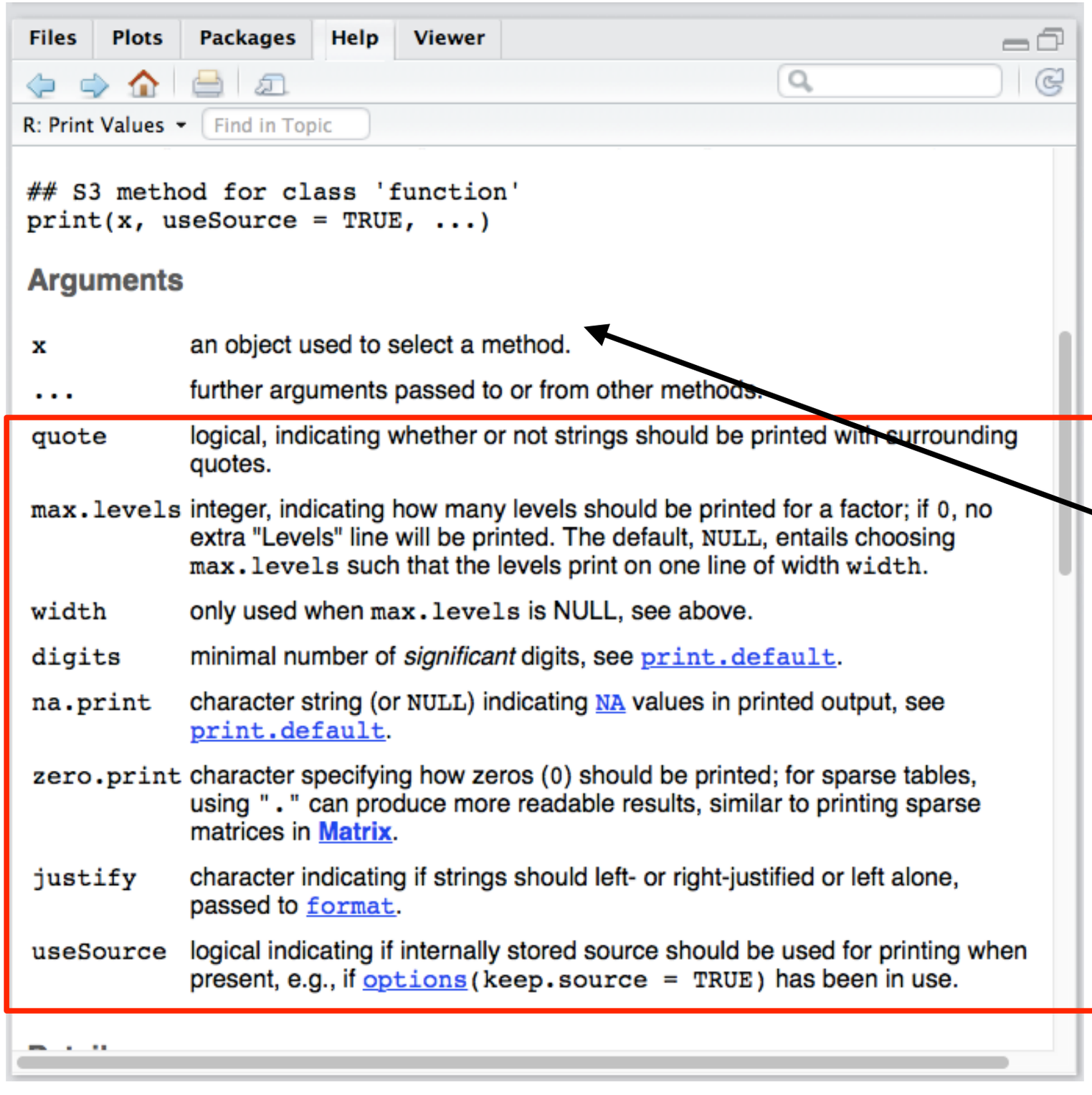
indicates there are optional arguments

R documentation



don’t worry about this!

…scrolling down…

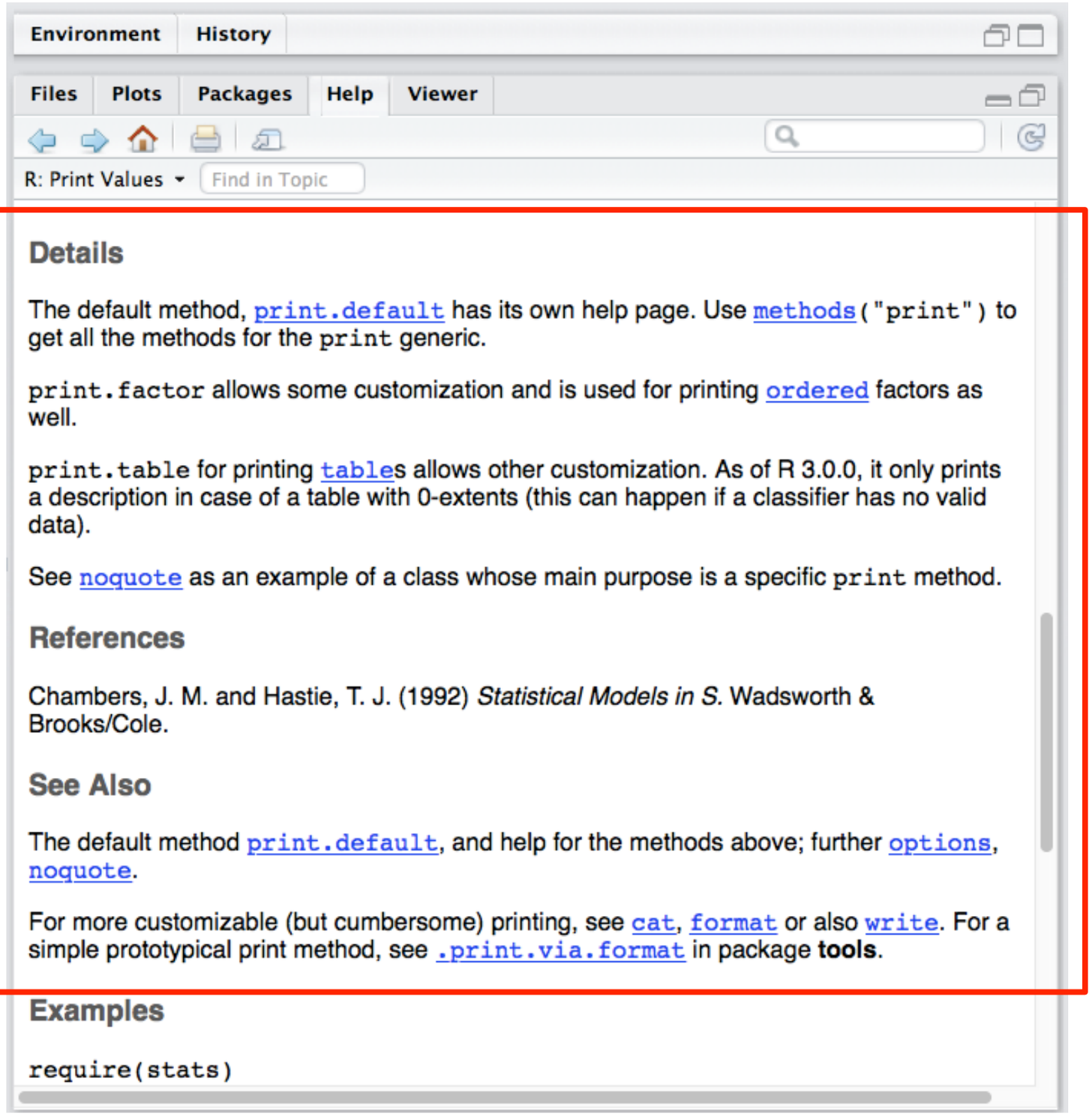


here it tells you what it needs to take as an argument

remember this was required

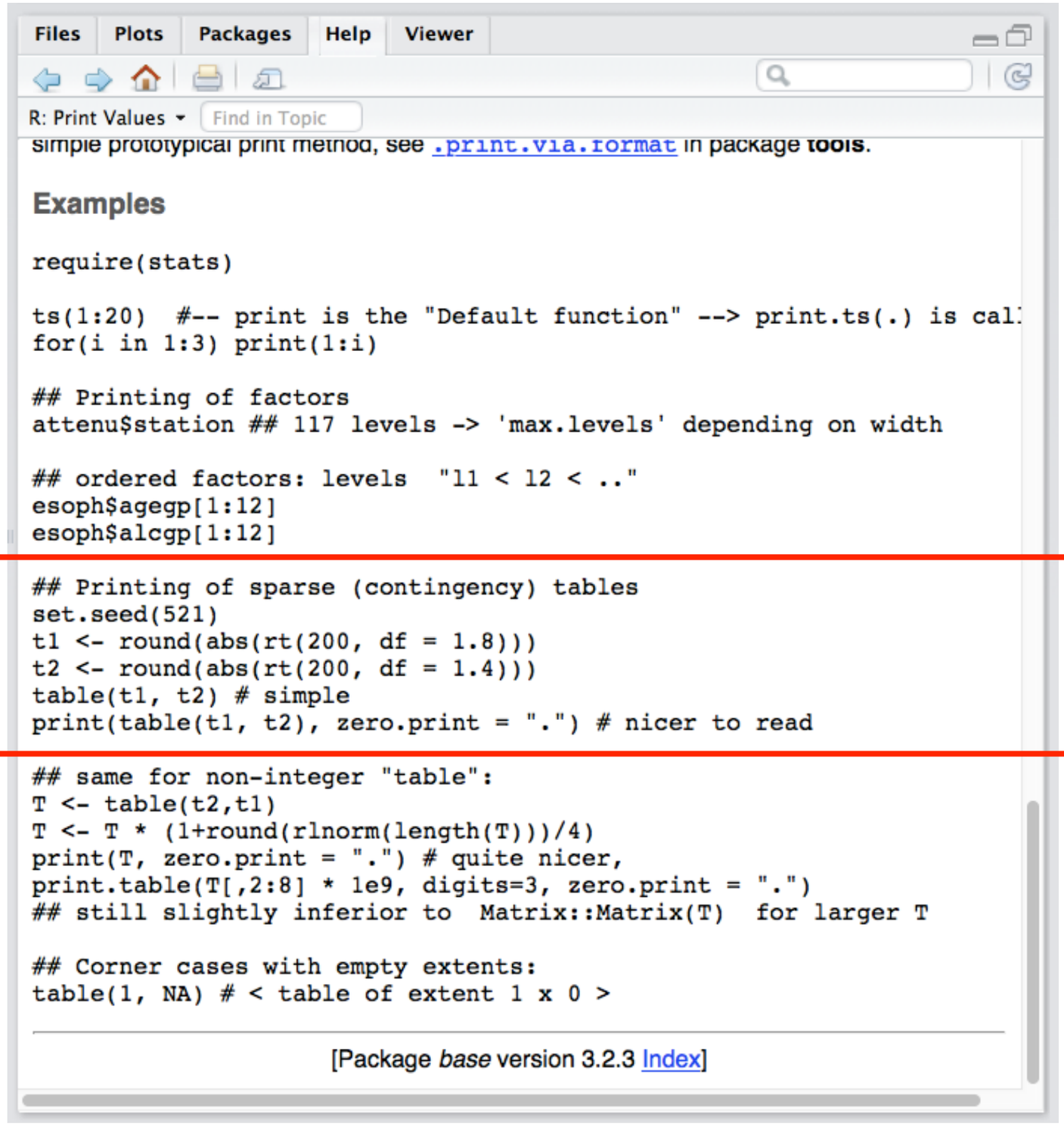
and these are optional

… scrolling even more…



you can pretty much ignore all of this (it's far advanced of what you'll need usually)

the end of the scrolling…



These can be useful to make sense of how to use some of the optional arguments.

But if they are confusing it's because it's almost certainly not something you need to understand!

Other debugging hints

** Remember that 80% of programming is debugging. Everyone has to debug *all of the time*. Having bugs means nothing about your skill (or lack thereof). Nothing at all. Good programmers = good debuggers.

1. Double-check that you have spelled everything right and have all of the right punctuation! Especially brackets. R is *super* picky.

2. READ AND THINK ABOUT THE ERROR MESSAGE. It is often really helpful. If you don't know what it means, google it exactly.

3. If you have a long command or multiple lines, separate it into parts and make sure each part is doing what you expect, one by one

4. Experiment around and try things! It's really really hard to break R. Do strange stuff, that's how you learn.

Exercises

1. Calculate the square root of 81.

2. What happens if you try to take the square root of -81? Use the `abs()` function to make it non-negative first.

3. What is the exponent of the log of 2?

4. Round 4328.21874 to two digits after the decimal place. How can you make it round it to 4000?

5. See if you can figure out how the `floor()` function is different from `round()`. Hint: just experiment around by giving it different values, or use the `help()` function.