

## 3.2: Typing Commands at the R Console

One of the easiest things you can do with R is use it as a simple calculator, so it's a good place to start. For instance, try typing `10 + 20`, and hitting enter.<sup>18</sup> When you do this, you've entered a **command**, and R will "execute" that command. What you see on screen now will be this:

```
> 10 + 20
[1] 30
```

Not a lot of surprises in this extract. But there's a few things worth talking about, even with such a simple example. Firstly, it's important that you understand how to read the extract. In this example, what I typed was the `10 + 20` part. I didn't type the `>` symbol: that's just the R command prompt and isn't part of the actual command. And neither did I type the `[1] 30` part. That's what R printed out in response to my command.

Secondly, it's important to understand how the output is formatted. Obviously, the correct answer to the sum `10 + 20` is `30`, and not surprisingly R has printed that out as part of its response. But it's also printed out this `[1]` part, which probably doesn't make a lot of sense to you right now. You're going to see that a lot. I'll talk about what this means in a bit more detail later on, but for now you can think of `[1] 30` as if R were saying "the answer to the 1st question you asked is 30". That's not quite the truth, but it's close enough for now. And in any case it's not really very interesting at the moment: we only asked R to calculate one thing, so obviously there's only one answer printed on the screen. Later on this will change, and the `[1]` part will start to make a bit more sense. For now, I just don't want you to get confused or concerned by it.

### 3.2.1 important digression about formatting

Now that I've taught you these rules I'm going to change them pretty much immediately. That is because I want you to be able to copy code from the book directly into R if you want to test things or conduct your own analyses. However, if you copy this kind of code (that shows the command prompt and the results) directly into R you will get an error

```
> 10 + 20
[1] 30
```

```
## Error: <text>:1:1: unexpected '>'
## 1: >
##      ^
```

So instead, I'm going to provide code in a slightly different format so that it looks like this...

```
10 + 20
```

```
## [1] 30
```

There are two main differences.

- In your console, you type after the `>`, but from now I won't show the command prompt in the book.
- In the book, output is commented out with `##`, in your console it appears directly after your code.

These two differences mean that if you're working with an electronic version of the book, you can easily copy code out of the book and into the console.

So for example if you copied the two lines of code from the book you'd get this

```
10 + 20
```

```
## [1] 30
```

```
## [1] 30
```

### 3.2.2 very careful to avoid typos

Before we go on to talk about other types of calculations that we can do with R, there's a few other things I want to point out. The first thing is that, while R is good software, it's still software. It's pretty stupid, and because it's stupid it can't handle typos. It takes it on faith that you meant to type *exactly* what you did type. For example, suppose that you forgot to hit the shift key when trying to type `+`, and as a result your command ended up being `10 = 20` rather than `10 + 20`. Here's what happens:

```
10 = 20
```

```
## Error in 10 = 20: invalid (do_set) left-hand side to assignment
```

What's happened here is that R has attempted to interpret `10 = 20` as a command, and spits out an error message because the command doesn't make any sense to it. When a *human* looks at this, and then looks down at his or her keyboard and sees that `+` and `=` are on the same key, it's pretty obvious that the command was a typo. But R doesn't know this, so it gets upset. And, if you look at it from its perspective, this makes sense. All that R "knows" is that `10` is a legitimate number, `20` is a legitimate number, and `=` is a legitimate part of the language too. In other words, from its perspective this really does look like the user meant to type `10 = 20`, since all the individual parts of that statement are legitimate and it's too stupid to realise that this is probably a typo. Therefore, R takes it on faith that this is exactly what you meant... it only "discovers" that the command is nonsense when it tries to follow your instructions, typo and all. And then it whinges, and spits out an error.

Even more subtle is the fact that some typos won't produce errors at all, because they happen to correspond to "well-formed" R commands. For instance, suppose that not only did I forget to hit the shift key when trying to type `10 + 20`, I also managed to press the key next to one I meant to. The resulting typo would produce the command `10 - 20`. Clearly, R has no way of knowing that you meant to *add* 20 to 10, not *subtract* 20 from 10, so what happens this time is this:

```
10 - 20
```

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

In this case, R produces the right answer, but to the wrong question.

To some extent, I'm stating the obvious here, but it's important. The people who wrote R are smart. You, the user, are smart. But R itself is dumb. And because it's dumb, it has to be mindlessly obedient. It does *exactly* what you ask it to do. There is no equivalent to "autocorrect" in R, and for good reason. When doing advanced stuff – and even the simplest of statistics is pretty advanced in a lot of ways – it's dangerous to let a mindless automaton like R try to overrule the human user. But because of this, it's your responsibility to be careful. Always make sure you type *exactly what you mean*. When dealing with computers, it's not enough to type "approximately" the right thing. In general, you absolutely *must* be precise in what you say to R ... like all machines it is too stupid to be anything other than absurdly literal in its interpretation.

### 3.2.3 bit) flexible with spacing

Of course, now that I've been so uptight about the importance of always being precise, I should point out that there are some exceptions. Or, more accurately, there are some situations in which R does show a bit more flexibility than my previous description suggests. The first thing R is smart enough to do is ignore redundant spacing. What I mean by this is that, when I typed `10 + 20` before, I could equally have done this

```
10  + 20
```

```
## [1] 30
```

or this

```
10+20
```

```
## [1] 30
```

and I would get exactly the same answer. However, that doesn't mean that you can insert spaces in any old place. When we looked at the startup documentation in Section 3.1.5 it suggested that you could type `citation()` to get some information about how to cite R. If I do so...

```
citation()
```

```
##
## To cite R in publications use:
##
##   R Core Team (2018). R: A language and environment for
##   statistical computing. R Foundation for Statistical Computing,
##   Vienna, Austria. URL https://www.R-project.org/.
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {R: A Language and Environment for Statistical Computing},
##     author = {{R Core Team}},
##     organization = {R Foundation for Statistical Computing},
##     address = {Vienna, Austria},
##     year = {2018},
##     url = {https://www.R-project.org/},
##   }
##
## We have invested a lot of time and effort in creating R, please
## cite it when using it for data analysis. See also
## 'citation("pkgname")' for citing R packages.
```

... it tells me to cite the R manual (R Core Team 2013). Let's see what happens when I try changing the spacing. If I insert spaces in between the word and the parentheses, or inside the parentheses themselves, then all is well. That is, either of these two commands

```
citation ()
```

```
citation( )
```

will produce exactly the same response. However, what I can't do is insert spaces in the middle of the word. If I try to do this, R gets upset:

```
citat ion()
```

```
## Error: <text>:1:7: unexpected symbol
## 1: citat ion
##           ^
```

Throughout this book I'll vary the way I use spacing a little bit, just to give you a feel for the different ways in which spacing can be used. I'll try not to do it too much though, since it's generally considered to be good practice to be consistent in how you format your commands.

### 3.2.4 sometimes tell that you're not finished yet (but not often)

One more thing I should point out. If you hit enter in a situation where it's "obvious" to R that you haven't actually finished typing the command, R is just smart enough to keep waiting. For example, if you type `10 +` and then press enter, even R is smart enough to realise that you probably wanted to type in another number. So here's what happens (for illustrative purposes I'm breaking my own code formatting rules in this section):

```
> 10+
+
```

and there's a blinking cursor next to the plus sign. What this means is that R is still waiting for you to finish. It "thinks" you're still typing your command, so it hasn't tried to execute it yet. In other words, this plus sign is actually another command prompt. It's different from the usual one (i.e., the `>` symbol) to remind you that R is going to "add" whatever you type now to what you typed last time. For example, if I then go on to type `3` and hit enter, what I get is this:

```
> 10 +
+ 20
[1] 30
```

And as far as R is concerned, this is *exactly* the same as if you had typed `10 + 20`. Similarly, consider the `citation()` command that we talked about in the previous section. Suppose you hit enter after typing `citation(`. Once again, R is smart enough to realise that there must be more coming – since you need to add the `)` character – so it waits. I can even hit enter several times and it will keep waiting:

```
> citation(
+
+
+ )
```

I'll make use of this a lot in this book. A lot of the commands that we'll have to type are pretty long, and they're visually a bit easier to read if I break it up over several lines. If you start doing this yourself, you'll eventually get yourself in trouble (it happens to us all). Maybe you start typing a command, and then you realise you've screwed up. For example,

```
> citblation(
+
+
```

You'd probably prefer R not to try running this command, right? If you want to get out of this situation, just hit the 'escape' key.<sup>19</sup> R will return you to the normal command prompt (i.e. `>`) *without* attempting to execute the botched command.

That being said, it's not often the case that R is smart enough to tell that there's more coming. For instance, in the same way that I can't add a space in the middle of a word, I can't hit enter in the middle of a word either. If I hit enter after typing `citat` I get an error, because R thinks I'm interested in an "object" called `citat` and can't find it:

```
> citat
Error: object 'citat' not found
```

What about if I typed `citation` and hit enter? In this case we get something very odd, something that we definitely *don't* want, at least at this stage. Here's what happens:

```
> [citation]
function (package = "base", lib.loc = NULL, auto = NULL)
{
  dir <- system.file(package = package, lib.loc = lib.loc)
  if (dir == "")
    stop(gettextf("package '%s' not found", package), domain = NA)

  BLAH BLAH BLAH
```

where the `BLAH BLAH BLAH` goes on for rather a long time, and you don't know enough R yet to understand what all this gibberish actually means (of course, it doesn't actually say `BLAH BLAH BLAH` - it says some other things we don't understand or need to know that I've edited for length) This incomprehensible output can be quite intimidating to novice users, and unfortunately it's very easy to forget to type the parentheses; so almost certainly you'll do this by accident. Do not panic when this happens. Simply ignore the gibberish. As you become more experienced this gibberish will start to make sense, and you'll find it quite handy to print this stuff out.<sup>20</sup> But for now just try to remember to add the parentheses when typing your commands.

---

This page titled [3.2: Typing Commands at the R Console](#) is shared under a [CC BY-SA 4.0](#) license and was authored, remixed, and/or curated by [Danielle Navarro](#) via [source content](#) that was edited to the style and standards of the LibreTexts platform.