

3.7: Storing Many Numbers As a Vector

At this point we've covered functions in enough detail to get us safely through the next couple of chapters (with one small exception: see Section 4.11, so let's return to our discussion of variables. When I introduced variables in Section 3.4 I showed you how we can use variables to store a single number. In this section, we'll extend this idea and look at how to store multiple numbers within the one variable. In R, the name for a variable that can store multiple values is a **vector**. So let's create one.

3.7.1 Creating a vector

Let's stick to my silly "get rich quick by textbook writing" example. Suppose the textbook company (if I actually had one, that is) sends me sales data on a monthly basis. Since my class start in late February, we might expect most of the sales to occur towards the start of the year. Let's suppose that I have 100 sales in February, 200 sales in March and 50 sales in April, and no other sales for the rest of the year. What I would like to do is have a variable – let's call it `sales.by.month` – that stores all this sales data. The first number stored should be `0` since I had no sales in January, the second should be `100`, and so on. The simplest way to do this in R is to use the **combine** function, `c()`. To do so, all we have to do is type all the numbers you want to store in a comma separated list, like this:³⁵

```
sales.by.month <- c(0, 100, 200, 50, 0, 0, 0, 0, 0, 0, 0, 0)
sales.by.month
```

```
## [1] 0 100 200 50 0 0 0 0 0 0 0 0
```

To use the correct terminology here, we have a single variable here called `sales.by.month`: this variable is a vector that consists of 12 **elements**.

3.7.2 handy digression

Now that we've learned how to put information into a vector, the next thing to understand is how to pull that information back out again. However, before I do so it's worth taking a slight detour. If you've been following along, typing all the commands into R yourself, it's possible that the output that you saw when we printed out the `sales.by.month` vector was slightly different to what I showed above. This would have happened if the window (or the RStudio panel) that contains the R console is really, really narrow. If that were the case, you might have seen output that looks something like this:

```
sales.by.month
```

```
## [1] 0 100 200 50 0 0 0 0 0 0 0 0
```

Because there wasn't much room on the screen, R has printed out the results over two lines. But that's not the important thing to notice. The important point is that the first line has a `[1]` in front of it, whereas the second line starts with `[9]`. It's pretty clear what's happening here. For the first row, R has printed out the 1st element through to the 8th element, so it starts that row with a `[1]`. For the second row, R has printed out the 9th element of the vector through to the 12th one, and so it begins that row with a `[9]` so that you can tell where it's up to at a glance. It might seem a bit odd to you that R does this, but in some ways it's a kindness, especially when dealing with larger data sets!

3.7.3 Getting information out of vectors

To get back to the main story, let's consider the problem of how to get information out of a vector. At this point, you might have a sneaking suspicion that the answer has something to do with the `[1]` and `[9]` things that R has been printing out. And of course you are correct. Suppose I want to pull out the February sales data only. February is the second month of the year, so let's try this:

```
sales.by.month[2]
```

```
## [1] 100
```

Yep, that's the February sales all right. But there's a subtle detail to be aware of here: notice that R outputs `[1] 100`, not `[2] 100`. This is because R is being extremely literal. When we typed in `sales.by.month[2]`, we asked R to find exactly *one* thing, and that one thing happens to be the second element of our `sales.by.month` vector. So, when it outputs `[1] 100` what R is saying is that the first number *that we just asked for* is `100`. This behaviour makes more sense when you realise that we can use this trick to create new variables. For example, I could create a `february.sales` variable like this:

```
february.sales <- sales.by.month[2]  
february.sales
```

```
## [1] 100
```

Obviously, the new variable `february.sales` should only have one element and so when I print it out this new variable, the R output begins with a `[1]` because `100` is the value of the first (and only) element of `february.sales`. The fact that this also happens to be the value of the second element of `sales.by.month` is irrelevant. We'll pick this topic up again shortly (Section 3.10).

3.7.4 Altering the elements of a vector

Sometimes you'll want to change the values stored in a vector. Imagine my surprise when the publisher rings me up to tell me that the sales data for May are wrong. There were actually an additional 25 books sold in May, but there was an error or something so they hadn't told me about it. How can I fix my `sales.by.month` variable? One possibility would be to assign the whole vector again from the beginning, using `c()`. But that's a lot of typing. Also, it's a little wasteful: why should R have to redefine the sales figures for all 12 months, when only the 5th one is wrong? Fortunately, we can tell R to change only the 5th element, using this trick:

```
sales.by.month[5] <- 25  
sales.by.month
```

```
## [1] 0 100 200 50 25 0 0 0 0 0 0 0
```

Another way to edit variables is to use the `edit()` or `fix()` functions. I won't discuss them in detail right now, but you can check them out on your own.

3.7.5 Useful things to know about vectors

Before moving on, I want to mention a couple of other things about vectors. Firstly, you often find yourself wanting to know how many elements there are in a vector (usually because you've forgotten). You can use the `length()` function to do this. It's quite straightforward:

```
length( x = sales.by.month )
```

```
## [1] 12
```

Secondly, you often want to alter all of the elements of a vector at once. For instance, suppose I wanted to figure out how much money I made in each month. Since I'm earning an exciting \$7 per book (no seriously, that's actually pretty close to what authors get on the very expensive textbooks that you're expected to purchase), what I want to do is multiply each element in the `sales.by.month` vector by `7`. R makes this pretty easy, as the following example shows:

```
sales.by.month * 7
```

```
## [1] 0 700 1400 350 175 0 0 0 0 0 0 0
```

In other words, when you multiply a vector by a single number, all elements in the vector get multiplied. The same is true for addition, subtraction, division and taking powers. So that's neat. On the other hand, suppose I wanted to know how much money I was making per day, rather than per month. Since not every month has the same number of days, I need to do something slightly different. Firstly, I'll create two new vectors:

```
days.per.month <- c(31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31)
profit <- sales.by.month * 7
```

Obviously, the `profit` variable is the same one we created earlier, and the `days.per.month` variable is pretty straightforward. What I want to do is divide every element of `profit` by the *corresponding* element of `days.per.month`. Again, R makes this pretty easy:

```
profit / days.per.month
```

```
## [1] 0.000000 25.000000 45.161290 11.666667 5.645161 0.000000 0.000000
## [8] 0.000000 0.000000 0.000000 0.000000 0.000000
```

I still don't like all those zeros, but that's not what matters here. Notice that the second element of the output is 25, because R has divided the second element of `profit` (i.e. 700) by the second element of `days.per.month` (i.e. 28). Similarly, the third element of the output is equal to 1400 divided by 31, and so on. We'll talk more about calculations involving vectors later on (and in particular a thing called the "recycling rule"; Section 7.12.2, but that's enough detail for now).

This page titled [3.7: Storing Many Numbers As a Vector](#) 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.