CHAPTER 2

Vectors

The name may sound intimidating, but a vector is simply a list of values. R relies on vectors for many of its operations. This includes basic plots - we'll have you drawing graphs by the end of this chapter (and it's a lot easier than you might think)!

Course Tip: If you haven't already, try clicking on the expand icon ( ) in the upper-left corner of the sidebar. The expanded sidebar offers a more in-depth look at chapter sections and progress.

Vectors 2.1

A vector's values can be numbers, strings, logical values, or any other type, as long as they're all the same type. Try creating a vector of numbers, like this:

```
> c(4, 7, 9)
[1] 4 7 9
```

The `c` function (`c` is short for Combine) creates a new vector by combining a list of values.

Now try creating a vector with strings:

```
> c("a", "b", "c")
[1] "a" "b" "c"
```

Vectors cannot hold values with different modes (types). Try mixing modes and see what happens:

```
> c(1, TRUE, "three")
[1] "1"    "TRUE" "three"
```

All the values were converted to a single mode (characters) so that the vector can hold them all.

Sequence Vectors 2.2

If you need a vector with a sequence of numbers you can create it with `start:end` notation. Let's make a vector with values from 5 through 9:

```
> 5:9
[1] 5 6 7 8 9
```

A more versatile way to make sequences is to call the `seq` function. Let's do the same thing with `seq`:

```
> seq(5, 9)
[1] 5 6 7 8 9
```

`seq` also allows you to use increments other than 1. Try it with steps of 0.5:

```
> seq(5, 9, 0.5)
[1] 5.0 5.5 6.0 6.5 7.0 7.5 8.0 8.5 9.0
```

Now try making a vector with integers from 9 down to 5:

```
> 9:5
[1] 9 8 7 6 5
```

Vector Access 2.3

We're going to create a vector with some strings in it for you, and store it in the `sentence` variable.

You can retrieve an individual value within a vector by providing its numeric index in square brackets. Try getting the third value:

```
> sentence <- c("walk", "the", "plank")
> sentence[3]
[1] "plank"
```

Many languages start array indices at 0, but R's vector indices start at 1. Get the first value by typing:

```
> sentence[1]
[1] "walk"
```

You can assign new values within an existing vector. Try changing the third word to "dog":

```
> sentence[3] <- "dog"
```

If you add new values onto the end, the vector will grow to accommodate them. Let's add a fourth word:

```
> sentence[4] <- "to"
```

You can use a vector within the square brackets to access multiple values. Try getting the first and third words:

```
> sentence[c(1, 3)]
[1] "walk" "dog"
```

This means you can retrieve ranges of values. Get the second through fourth words:

```
> sentence[2:4]
[1] "the" "dog" "to"
```

You can also set ranges of values; just provide the values in a vector. Add words 5 through 7:

```
> sentence[5:7] <- c("the", "poop", "deck")
```

Now try accessing the sixth word of the `sentence` vector:

```
> sentence[6]
[1] "poop"
```

Vector Names 2.4

For this challenge, we'll make a 3-item vector for you, and store it in the `ranks` variable.

You can assign names to a vector's elements by passing a second vector filled with names to the `names` assignment function, like this:

```
> ranks <- 1:3
> names(ranks) <- c("first", "second", "third")
```

Assigning names for a vector can act as useful labels for the data. Below, you can see what our vector looks like now.

You can also use the names to access the vector's values. Try getting the value for the `"first"` rank:

```
> ranks
 first second  third
     1      2      3
> ranks["first"]
first
    1
```

Now set the current value for the `"third"` rank to a different value using the name rather than the position.

```
> ranks["third"] <- 4
```

Plotting One Vector 2.5

The `barplot` function draws a bar chart with a vector's values. We'll make a new vector for you, and store it in the `vesselsSunk` variable.

Now try passing the vector to the `barplot` function:

```
> vesselsSunk <- c(4, 5, 1)
> barplot(vesselsSunk)
```

If you assign names to the vector's values, R will use those names as labels on the bar plot. Let's use the `names` assignment function again:

```
> names(vesselsSunk) <- c("England", "France", "Norway")
```

Now, if you call `barplot` with the vector again, you'll see the labels:

```
> barplot(vesselsSunk)
```

Now, try calling barplot on a vector of integers ranging from 1 through 100:

```
> barplot(1:100)
```

Vector Math 2.6

Most arithmetic operations work just as well on vectors as they do on single values. We'll make another sample vector for you to work with, and store it in the `a` variable.

If you add a scalar (a single value) to a vector, the scalar will be added to each value in the vector, returning a new vector with the results. Try adding 1 to each element in our vector:

```
> a <- c(1, 2, 3)
> a + 1
[1] 2 3 4
```

The same is true of division, multiplication, or any other basic arithmetic. Try dividing our vector by 2:

```
> a / 2
[1] 0.5 1.0 1.5
```

Now try multiplying our vector by 2:

```
> a * 2
[1] 2 4 6
```

If you add two vectors, R will take each value from each vector and add them. We'll make a second vector for you to experiment with, and store it in the `b` variable.

Try adding it to the `a` vector.

```
> b <- c(4, 5, 6)
> a + b
[1] 5 7 9
```

Now try subtracting `b` from `a`:

```
> a - b
[1] -3 -3 -3
```

You can also take two vectors and compare each item. See which values in the `a` vector are equal to those in a second vector:

```
> a == c(1, 99, 3)
[1] TRUE FALSE TRUE
```

Notice that R didn't test whether the whole vectors were equal; it checked each value in the a vector against the value at the same index in our new vector.

Check if each value in the `a` vector is less than the corresponding value in another vector:

```
> a < c(1, 99, 3)
[1] FALSE TRUE FALSE
```

Functions that normally work with scalars can operate on each element of a vector, too. Try getting the sine of each value in our vector:

```
> sin(a)
[1] 0.8414710 0.9092974 0.1411200
```

Now try getting the square roots with `sqrt`:

```
> sqrt(a)
[1] 1.000000 1.414214 1.732051
```

Scatter Plots 2.7

The `plot` function takes two vectors, one for X values and one for Y values, and draws a graph of them.

Let's draw a graph showing the relationship of numbers and their sines.

First, we'll need some sample data. We'll create a vector for you with some fractional values between 0 and 20, and store it in the `x` variable.

Now, try creating a second vector with the sines of those values:

```
> x <- seq(1, 20, 0.1)
> y <- sin(x)
```

Then simply call `plot` with your two vectors:

```
> plot(x, y)
```

Great job! Notice on the graph that values from the first argument (`x`) are used for the horizontal axis, and values from the second (`y`) for the vertical.

Your turn. We'll create a vector with some negative and positive values for you, and store it in the `values` variable.

We'll also create a second vector with the absolute values of the first, and store it in the `absolutes` variable.

Try plotting the vectors, with `values` on the horizontal axis, and `absolutes` on the vertical axis.

```
> values <- -10:10
> absolutes <- abs(values)
> plot(values, absolutes)
```

NA Values 2.8

Sometimes, when working with sample data, a given value isn't available. But it's not a good idea to just throw those values out. R has a value that explicitly indicates a sample was not available: `NA`. Many functions that work with vectors treat this value specially.

We'll create a vector for you with a missing sample, and store it in the `q` variable.

Try to get the sum of its values, and see what the result is:

```
> q <- c(1, 3, NA, 7, 9)
> sum(q)
[1] NA
```

The sum is considered 'not available' by default because one of the vector's values was `NA`. This is the responsible thing to do. If R won't just blithely add up the numbers without warning you about the incomplete data, We can explicitly tell `sum` (and many other functions) to remove `NA` values before they do their calculations, however.

Remember that command to bring up help for a function? Bring up documentation for the `sum` function:

```
> help(sum)
sum                     package:base                     R Documentation

Sum of Vector Elements

Description:

     'sum' returns the sum of all the values present in its arguments.

Usage:

     sum(..., na.rm = FALSE)
```

As you see in the documentation, `sum` can take an optional named argument, `na.rm`. It's set to `FALSE` by default, but if you set it to `TRUE`, all `NA` arguments will be removed from the vector before the calculation is performed.

Try calling `sum` again, with `na.rm` set to `TRUE`:

```
> sum(q, na.rm = TRUE)
[1] 20
```

Chapter 2 Completed

You've traversed Chapter 2... and discovered another badge!

In this chapter, we've shown you all the basics of manipulating vectors - creating and accessing them, doing math with them, and making sequences. We've shown you how to make bar plots and scatter plots with vectors. And we've shown you how R treats vectors where one or more values are not available.

The vector is just the first of several data structures that R offers. See you in the next chapter, where we'll talk about... the matrix.

Continue