

Sequences of Numbers

In this lesson, you'll learn how to create sequences of numbers in R. Sequences of numbers are used in many different tasks, from plotting the axes of graphs to generating simulated data.

The simplest way to create a sequence of numbers in R is by using the `:` operator. Type `1:20` to see how it works.

```
1:20
## [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

That gave us every integer between (and including) 1 and 20 (an integer is a positive or negative counting number, including 0).

We could also use it to create a sequence of real numbers (a real number is a positive, negative, or 0 with an infinite or finite sequence of digits after the decimal place). For example, try typing `pi:10`.

```
pi:10
## [1] 3.141593 4.141593 5.141593 6.141593 7.141593 8.141593 9.141593
```

The result is a vector of real numbers starting with pi (3.142...) and increasing in increments of 1. The upper limit of 10 is never reached, since the next number in our sequence would be greater than 10.

Note also that pi is one of the few constants built in to R. Type `?pi` to check the others.

```
?pi
```

What happens if we do `15:1`? Give it a try to find out.

```
15:1
## [1] 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
```

It counted backwards in increments of 1! This is sometimes useful for plotting coefficients from models in reverse order.

Remember that if you have questions about a particular R function, you can access its documentation with a question mark followed by the function name: `?function_name_here`. However, in the case of an operator like the colon used above, you must enclose the symbol in backticks like this: `?:`:``. (NOTE: The backtick (```) key is generally located in the top left corner of a keyboard, above the Tab key. If you don't have a backtick key, you can use regular quotes.)

Pull up the documentation for `:` now.

```
?:`:`
```

Often, we'll desire more control over a sequence we're creating than what the `:` operator gives us. The `seq()` function serves this purpose.

The most basic use of `seq()` does exactly the same thing as the `:` operator. Try `seq(1, 20)` to see this.

```
seq(1, 20)
## [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

This gives us the same output as `1:20`. Check the help file for `seq()`.

The help files show the arguments listed for the `seq()` function. The first two arguments are “from =” and “to =”. In R, you do not have to specify the arguments by name if you write out their values in the same order as written in the function. However, for complex functions it is often best practice to do so and makes your code much clearer.

For example, `seq(from = 1, to = 20)` will give the same output as `seq(1, 20)`. Try it!

```
seq(from = 1, to = 20)
## [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

OK, let's say that instead of 1 to 20, we want a vector of numbers ranging from 0 to 10, incremented by 0.5. `seq(0, 10, by = 0.5)` does just that. Try it out.

```
seq(0, 10, by = 0.5)
## [1] 0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5
## [15] 7.0 7.5 8.0 8.5 9.0 9.5 10.0
```

Or maybe we don't care what the increment is and we just want a sequence of 30 numbers between 5 and 10. `seq(5, 10, length = 30)` does the trick. Give it a shot now and store the result in a new variable called `my_seq`.

```
my_seq <- seq(5, 10, length = 30)
```

If you look closely again at the help file for `?seq`, you will not see an argument “length =”, but only “length.out =”. You can actually use any abbreviation of the argument name, as long as it is different from any other argument. You could even use just “l =”!

To confirm that `my_seq` has length 30, we can use the `length()` function. Try it now. To do this, you need to include the object ‘`my_seq`’ as the value of argument ‘`x`’ of `length()`.

```
length(my_seq)
## [1] 30
```

Let's pretend we don't know the length of `my_seq`, but we want to generate a sequence of integers from 1 to N, where N represents the length of the `my_seq` vector. In other words, we want a new vector (1, 2, 3, ...) that is the same length as `my_seq`.

There are several ways we could do this. One possibility is to combine the `:` operator and the `length()` function like this: `1:length(my_seq)`. Give that a try.

```
1:length(my_seq)
## [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
## [24] 24 25 26 27 28 29 30
```

Another option is to use `seq(along.with = my_seq)`. Give that a try.

```
seq(along.with = my_seq)
## [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
## [24] 24 25 26 27 28 29 30
```

However, as is the case with many common tasks, R has a separate built-in function for this purpose called `seq_along()`. Type `seq_along(my_seq)` to see it in action.

```
seq_along(my_seq)
```

```
## [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
## [24] 24 25 26 27 28 29 30
```

There are often several approaches to solving the same problem, particularly in R. Simple approaches that involve less typing are generally best. It's also important for your code to be readable, so that you and others can figure out what's going on without too much hassle.

If R has a built-in function for a particular task, it's likely that function is highly optimized for that purpose and is your best option. One of the philosophies of R (and Unix more generally) is to have tools (or functions) that do specific things very well and then link these together, rather than a single multi-purpose tool that does many things poorly.

This approach is like having a separate knife, fork, and spoon, rather than a Spork ... In most situations, cutlery ("silverware") is superior to the Spork.

As you become a more advanced R programmer, you will learn how to link and nest these apparently simple functions to do incredibly powerful tasks. You will also design your own functions to perform tasks when there are no better options. We'll explore writing your own functions in future lessons.

OK, back to the show. One more function related to creating sequences of numbers is `rep()`, which stands for 'replicate'. Let's look at a few uses.

If we're interested in creating a vector that contains 40 zeros, we can use `rep(0, times = 40)`. Try it out.

```
rep(0, times = 40)
## [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [36] 0 0 0 0 0
```

If instead we want our vector to contain 10 repetitions of the vector (0, 1, 2), we can do `rep(c(0, 1, 2), times = 10)`. Go ahead.

```
rep(c(0, 1, 2), times = 10)
## [1] 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2
```

Finally, let's say that rather than repeating the vector (0, 1, 2) over and over again, we want our vector to contain 10 zeros, then 10 ones, then 10 twos. We can do this with the `each` argument. Try `rep(c(0, 1, 2), each = 10)`.

```
rep(c(0, 1, 2), each = 10)
## [1] 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
```

Congratulations! Now you have several powerful tools that you can use to generate sequences of numbers. You also learnt to use the function `length()` and the `:` operator. Your R skills are building!