| Please choose a lesson, or type 0 to return to course menu.

1: Basic Building Blocks 2: Workspace and Files 3: Sequences of Numbers

4: Vectors 5: Missing Values 6: Subsetting Vectors

7: Matrices and Data Frames 8: Logic 9: Functions

10: lapply and sapply 11: vapply and tapply 12: Looking at Data

13: Simulation 14: Dates and Times 15: Base Graphics

Selection: 3

| | 0%

| In this lesson, you'll learn how to create sequences of numbers in R.

...

|==== | 5%

| 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

| You are quite good my friend!

|========= | 9%

| That gave us every integer between (and including) 1 and 20. We could also use it to create a

| sequence of real numbers. For example, try pi:10.

> pi:10

[1] 3.141593 4.141593 5.141593 6.141593 7.141593 8.141593 9.141593

| That's a job well done!

|============= | 14%

| 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.

...

|================= | 18%

| 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

| That's a job well done!

|===================== | 23%

| It counted backwards in increments of 1! It's unlikely we'd want this behavior, but nonetheless it's

| good to know how it could happen.

...

|========================== | 27%

| 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.)

...

|============================== | 32%

| Pull up the documentation for `:` now.

> ?':'

| That's correct!

|================================== | 36%

| Often, we'll desire more control over a sequence we're creating than what the `:` operator gives us.

| The seq() function serves this purpose.

...

|====================================== | 41%

| 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

| You got it right!

|=========================================== | 45%

| This gives us the same output as 1:20. However, let's say that instead 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 7.0 7.5 8.0 8.5 9.0 9.5

[21] 10.0

| You are really on a roll!

|=============================================== | 50%

| 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)

| You are really on a roll!

|=================================================== | 55%

| To confirm that my\_seq has length 30, we can use the length() function. Try it now.

> length(my\_seq)

[1] 30

| You are really on a roll!

|======================================================== | 59%

| 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.

...

|============================================================ | 64%

| 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 25 26 27 28 29 30

| You are quite good my friend!

|================================================================ | 68%

| 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 25 26 27 28 29 30

| You're the best!

|==================================================================== | 73%

| 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 25 26 27 28 29 30

| Keep up the great work!

|========================================================================= | 77%

| 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.

...

|============================================================================= | 82%

| 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. As you become a more advanced R programmer, you'll design your

| own functions to perform tasks when there are no better options. We'll explore writing your own

| functions in future lessons.

...

|================================================================================= | 86%

| One more function related to creating sequences of numbers is rep(), which stands for 'replicate'.

| Let's look at a few uses.

...

|===================================================================================== | 91%

| 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 0 0 0 0 0 0 0 0

| That's the answer I was looking for.

|========================================================================================== | 95%

| 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

| You nailed it! Good job!

|==============================================================================================| 100%

| 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 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2