```
library(swirl)

| Hi! I see that you have some variables saved in your workspace. To keep thi
ngs running smoothly, I
| recommend you clean up before starting swirl.

| Type ls() to see a list of the variables in your workspace. Then, type rm(l
ist=ls()) to clear your
| workspace.

| Type swirl() when you are ready to begin.

Warning message:
package ‘swirl’ was built under R version 3.6.3
> swirl()

| Welcome to swirl! Please sign in. If you've been here before, use the same
name as you did then. If
| you are new, call yourself something unique.

What shall I call you? RobinTeotia

| Please choose a course, or type 0 to exit swirl.

1: R Programming
2: Take me to the swirl course repository!

Selection: 1

| 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 Ve
ctors
 7: Matrices and Data Frames   8: Logic                      9: Functions
10: lapply and sapply         11: vapply and tapply         12: Looking at Da
ta
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.

...

  |====
|   4%
| The simplest way to create a sequence of numbers in R is by using the `:` o
perator. 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

| Nice work!

  |========
|   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

| Keep up the great work!

  |============
|  13%
| The result is a vector of real numbers starting with pi (3.142...) and incr
easing in increments of
| 1. The upper limit of 10 is never reached, since the next number in our seq
uence would be greater
| than 10.

...

  |================
|  17%
| 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 the answer I was looking for.

  |====================
|  22%
| It counted backwards in increments of 1! It's unlikely we'd want this behav
ior, but nonetheless
| it's good to know how it could happen.

...

  |========================
|  26%
| 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. Ho
wever, in the case of an
| operator like the colon used above, you must enclose the symbol in backtick
s 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.)

...?`:`

  |============================
|  30%
| Pull up the documentation for `:` now.

> `:`
.Primitive(":")

| You're close...I can feel it! Try it again. Or, type info() for more option
s.

| In order to view the documentation for a symbol like the colon operator, yo
u have to use backticks
```

```
| (or quotes). This is so R knows you are not attempting to use the symbol in
the command. Here's
| what it looks like: ?`:`. Don't forget the question mark out front.

> ?``
Error: attempt to use zero-length variable name
> ?`:`

| You are amazing!

  |================================
|  35%
| Often, we'll desire more control over a sequence we're creating than what t
he `:` operator gives
| us. The seq() function serves this purpose.

...

  |====================================
|  39%
| 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

| Keep working like that and you'll get there!

  |========================================
|  43%
| This gives us the same output as 1:20. However, let's say that instead we w
ant 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
[20]  9.5 10.0

| That's a job well done!

  |============================================
|  48%
| 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.

> seq(5,10, length=30)
 [1]  5.000000  5.172414  5.344828  5.517241  5.689655  5.862069  6.034483  6
.206897  6.379310
[10]  6.551724  6.724138  6.896552  7.068966  7.241379  7.413793  7.586207  7
.758621  7.931034
[19]  8.103448  8.275862  8.448276  8.620690  8.793103  8.965517  9.137931  9
.310345  9.482759
[28]  9.655172  9.827586 10.000000

| That's not exactly what I'm looking for. Try again. Or, type info() for mor
e options.
```

```
| You're using the same function here, but changing its arguments for differe
nt results. Be sure to
| store the result in a new variable called my_seq, like this: my_seq <- seq(
5, 10, length=30).

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

| All that hard work is paying off!

  |================================================                      
|  52%
| To confirm that my_seq has length 30, we can use the length() function. Try
it now.

> length(my_seq)
[1] 30

| All that hard work is paying off!

  |====================================================                  
|  57%
| 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 w
ords, we want a new
| vector (1, 2, 3, ...) that is the same length as my_seq.

...

  |========================================================              
|  61%
| 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're the best!

  |============================================================          
|  65%
| 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

| Nice work!

  |================================================================      
|  70%
| However, as is the case with many common tasks, R has a separate built-in f
unction 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

| Perseverance, that's the answer.
```

```
  |==================================================================
|  74%
| There are often several approaches to solving the same problem, particularl
y 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.

...

  |======================================================================
|  78%
| If R has a built-in function for a particular task, it's likely that functi
on is highly optimized
| for that purpose and is your best option. As you become a more advanced R p
rogrammer, 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.

...

  |==========================================================================
==                |  83%
| One more function related to creating sequences of numbers is rep(), which
stands for 'replicate'.
| Let's look at a few uses.

...

  |==========================================================================
======            |  87%
| 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

| Your dedication is inspiring!

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

| That's the answer I was looking for.

  |==========================================================================
==============    |  96%
| 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
```

```
| That's a job well done!

  |=============================================================================
===================| 100%
| Would you like to receive credit for completing this course on Coursera.org
?

1: No
2: Yes

Selection: 2
What is your email address? robteotia@gmail.com
What is your assignment token? S9tSMpVeDfzTWXVF
Grade submission succeeded!
```