| 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 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: 5

| | 0%

| Missing values play an important role in statistics and data analysis. Often, missing values must not

| be ignored, but rather they should be carefully studied to see if there's an underlying pattern or

| cause for their missingness.

...

|===== | 5%

| In R, NA is used to represent any value that is 'not available' or 'missing' (in the statistical

| sense). In this lesson, we'll explore missing values further.

...

|========== | 11%

| Any operation involving NA generally yields NA as the result. To illustrate, let's create a vector

| c(44, NA, 5, NA) and assign it to a variable x.

> x <- c(44, NA, 5, NA)

| Excellent work!

|=============== | 16%

| Now, let's multiply x by 3.

> x \* 3

[1] 132 NA 15 NA

| All that practice is paying off!

|==================== | 21%

| Notice that the elements of the resulting vector that correspond with the NA values in x are also NA.

...

|========================= | 26%

| To make things a little more interesting, lets create a vector containing 1000 draws from a standard

| normal distribution with y <- rnorm(1000).

> y <- rnorm(1000)

| You nailed it! Good job!

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

| Next, let's create a vector containing 1000 NAs with z <- rep(NA, 1000).

> z <- rep(NA, 1000)

| Excellent job!

|=================================== | 37%

| Finally, let's select 100 elements at random from these 2000 values (combining y and z) such that we

| don't know how many NAs we'll wind up with or what positions they'll occupy in our final vector --

| my\_data <- sample(c(y, z), 100).

> my\_data <- sample(c(y, z), 100)

| You nailed it! Good job!

|======================================== | 42%

| Let's first ask the question of where our NAs are located in our data. The is.na() function tells us

| whether each element of a vector is NA. Call is.na() on my\_data and assign the result to my\_na.

> my\_na <- is.na(my\_data)

| You are quite good my friend!

|============================================= | 47%

| Now, print my\_na to see what you came up with.

> my\_na

[1] TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE TRUE TRUE FALSE FALSE TRUE FALSE FALSE

[17] TRUE FALSE TRUE TRUE FALSE FALSE FALSE TRUE FALSE TRUE TRUE FALSE FALSE TRUE TRUE FALSE

[33] FALSE FALSE FALSE TRUE TRUE FALSE FALSE TRUE FALSE FALSE FALSE FALSE TRUE FALSE TRUE FALSE

[49] TRUE FALSE TRUE TRUE FALSE FALSE FALSE TRUE FALSE FALSE FALSE TRUE FALSE TRUE TRUE FALSE

[65] FALSE FALSE FALSE FALSE TRUE TRUE TRUE TRUE TRUE TRUE FALSE TRUE FALSE FALSE TRUE FALSE

[81] FALSE TRUE FALSE TRUE TRUE FALSE TRUE FALSE TRUE FALSE TRUE TRUE FALSE TRUE FALSE FALSE

[97] FALSE FALSE TRUE FALSE

| Nice work!

|================================================= | 53%

| Everywhere you see a TRUE, you know the corresponding element of my\_data is NA. Likewise, everywhere

| you see a FALSE, you know the corresponding element of my\_data is one of our random draws from the

| standard normal distribution.

...

|====================================================== | 58%

| In our previous discussion of logical operators, we introduced the `==` operator as a method of

| testing for equality between two objects. So, you might think the expression my\_data == NA yields the

| same results as is.na(). Give it a try.

> my\_data == NA

[1] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA

[34] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA

[67] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA

[100] NA

| You got it!

|=========================================================== | 63%

| The reason you got a vector of all NAs is that NA is not really a value, but just a placeholder for a

| quantity that is not available. Therefore the logical expression is incomplete and R has no choice

| but to return a vector of the same length as my\_data that contains all NAs.

...

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

| Don't worry if that's a little confusing. The key takeaway is to be cautious when using logical

| expressions anytime NAs might creep in, since a single NA value can derail the entire thing.

...

|===================================================================== | 74%

| So, back to the task at hand. Now that we have a vector, my\_na, that has a TRUE for every NA and

| FALSE for every numeric value, we can compute the total number of NAs in our data.

...

|========================================================================== | 79%

| The trick is to recognize that underneath the surface, R represents TRUE as the number 1 and FALSE as

| the number 0. Therefore, if we take the sum of a bunch of TRUEs and FALSEs, we get the total number

| of TRUEs.

...

|=============================================================================== | 84%

| Let's give that a try here. Call the sum() function on my\_na to count the total number of TRUEs in

| my\_na, and thus the total number of NAs in my\_data. Don't assign the result to a new variable.

> sum(my\_na)

[1] 42

| You're the best!

|==================================================================================== | 89%

| Pretty cool, huh? Finally, let's take a look at the data to convince ourselves that everything 'adds

| up'. Print my\_data to the console.

> my\_data

[1] NA -1.38463507 1.56444659 1.78345168 -0.83856391 1.26743319 1.91287948 1.92442516

[9] NA NA NA 0.60280550 0.81097166 NA 0.63202138 -0.59402569

[17] NA 0.13081729 NA NA -0.46770756 1.38885839 0.45682028 NA

[25] -0.96341258 NA NA -0.50285214 1.41462742 NA NA 0.72066250

[33] 0.24787138 -0.69308179 1.21720901 NA NA -2.84001919 1.18853983 NA

[41] -0.54839667 -0.48909168 0.23850471 1.27453838 NA -0.38670974 NA -0.36536528

[49] NA 0.12852504 NA NA -0.60173648 -0.56478588 0.31604006 NA

[57] -0.61552648 0.07807398 -0.20939570 NA -0.36173850 NA NA 0.81153502

[65] 0.22038136 -1.60572113 1.11315299 -1.55126530 NA NA NA NA

[73] NA NA -1.02660159 NA 1.45911617 -0.69332870 NA -0.77492469

[81] 1.02890786 NA 0.98965859 NA NA 0.03423575 NA -0.21076533

[89] NA 0.21367001 NA NA 2.74593561 NA -0.01502913 -1.94611035

[97] 0.79819155 -0.67819316 NA -0.37150577

| That's correct!

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

| Now that we've got NAs down pat, let's look at a second type of missing value -- NaN, which stands

| for 'not a number'. To generate NaN, try dividing (using a forward slash) 0 by 0 now.

> 0 / 0

[1] NaN

| Excellent work!

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

| Let's do one more, just for fun. In R, Inf stands for infinity. What happens if you subtract Inf from

| Inf?

> Inf - Inf

[1] NaN

| Perseverance, that's the answer.