```
Selection: 5

  |
|   0%

| Missing values play an important role in statistics and data analysis. Ofte
n, 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.

...

  |=========
|  10%
| 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 job!

  |==============
|  15%
| Now, let's multiply x by 3.

> x*3
[1] 132  NA  15  NA

| All that hard work is paying off!

  |===================
|  20%
| Notice that the elements of the resulting vector that correspond with the N
A values in x are also
| NA.

...

  |=======================
|  25%
| To make things a little more interesting, lets create a vector containing 1
000 draws from a standard
| normal distribution with y <- rnorm(1000).

> y<-rnorm(1000)

| You got it!

  |============================
|  30%
| Next, let's create a vector containing 1000 NAs with z <- rep(NA, 1000).

> z<-rep(NA,1000)
```

```
| All that hard work is paying off!

  |=================================
|  35%
| Finally, let's select 100 elements at random from these 2000 values (combin
ing 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(x,y),100)

| Not quite right, but keep trying. Or, type info() for more options.

| The sample() function draws a random sample from the data provided as its f
irst argument (in this
| case c(y, z)) of the size specified by the second argument (100). The comma
nd my_data <- sample(c(y,
| z), 100) will give us what we want.

> my_data<-sample(c(y,z),100)

| You're the best!

  |====================================
|  40%
| 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)

| Keep up the great work!

  |=========================================
|  45%
| Now, print my_na to see what you came up with.

> print(my_na)
  [1] FALSE  TRUE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE
TRUE  TRUE FALSE  TRUE
 [17]  TRUE FALSE  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE FALSE  TRUE FALSE
TRUE FALSE FALSE FALSE
 [33] FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
FALSE  TRUE  TRUE FALSE
 [49]  TRUE FALSE  TRUE FALSE FALSE  TRUE  TRUE FALSE  TRUE FALSE  TRUE  TRUE
FALSE  TRUE FALSE  TRUE
 [65] FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
FALSE FALSE  TRUE FALSE
 [81]  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE  TRUE  TRUE FALSE FALSE
FALSE  TRUE  TRUE FALSE
 [97]  TRUE  TRUE FALSE FALSE

| One more time. You can do it! Or, type info() for more options.

| Type my_na to view its contents.

> my_na
  [1] FALSE  TRUE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE
TRUE  TRUE FALSE  TRUE
 [17]  TRUE FALSE  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE FALSE  TRUE FALSE
TRUE FALSE FALSE FALSE
```

```
 [33] FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
FALSE  TRUE  TRUE FALSE
 [49]  TRUE FALSE  TRUE FALSE FALSE  TRUE  TRUE FALSE  TRUE FALSE  TRUE  TRUE
FALSE  TRUE FALSE  TRUE
 [65] FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
FALSE FALSE  TRUE FALSE
 [81]  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE  TRUE  TRUE FALSE FALSE
FALSE  TRUE  TRUE FALSE
 [97]  TRUE  TRUE FALSE FALSE

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

  |=============================================
|  50%
| 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 ou
r random draws from the
| standard normal distribution.

...

  |=============================================
|  55%
| In our previous discussion of logical operators, we introduced the `==` ope
rator as a method of
| testing for equality between two objects. So, you might think the expressio
n 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
 [33] 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
 [65] 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
 [97] NA NA NA NA

| You are doing so well!

  |=============================================
|  60%
| The reason you got a vector of all NAs is that NA is not really a value, bu
t just a placeholder for
| a quantity that is not available. Therefore the logical expression is incom
plete and R has no choice
| but to return a vector of the same length as my_data that contains all NAs.

...

  |=============================================
|  65%
| Don't worry if that's a little confusing. The key takeaway is to be cautiou
s when using logical
| expressions anytime NAs might creep in, since a single NA value can derail
the entire thing.

...

  |=============================================
|  70%
| 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 ou
r data.

...

  |=======================================================================
|  75%
| 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 FALS
Es, we get the total
| number of TRUEs.

...

  |=======================================================================
|  80%
| Let's give that a try here. Call the sum() function on my_na to count the t
otal 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] 50

| That's a job well done!

  |=======================================================================
=====                |  85%
| Pretty cool, huh? Finally, let's take a look at the data to convince oursel
ves that everything 'adds
| up'. Print my_data to the console.

> my_data
  [1] -0.17733867          NA -1.06050021 -1.14537219 -2.30695343  1.83236081
0.62491533          NA
  [9] -0.94503607          NA -0.51633811  0.48545499          NA          NA
0.16130984          NA
 [17]          NA  0.23385634          NA          NA          NA -0.45027005
-1.20058051          NA
 [25]          NA  0.67473158          NA -0.09956944          NA -1.41903250
-0.57385386 -3.31309333
 [33]  0.86943634  2.10342162  0.76827620 -2.03965107  2.56117510  0.52945433
1.13700857          NA
 [41]          NA          NA          NA          NA -1.23250776          NA
NA -0.17453034
 [49]          NA -0.71810295          NA  1.19079820  0.10015901          NA
NA -0.28725762
 [57]          NA -0.99850121          NA          NA  0.49677143          NA
-1.33882834          NA
 [65]  0.88721069 -0.97666644          NA          NA          NA          NA
NA          NA
 [73]          NA          NA          NA          NA  0.84399563 -0.27468744
NA  2.22872850
 [81]          NA -0.18796114  0.86885334  0.56219574          NA  1.95116457
-0.28008940          NA
 [89]          NA          NA  1.03878081  0.22828963 -0.63867417          NA
NA  0.85403906
 [97]          NA          NA  1.87882052  1.73679521

| Great job!

  |=======================================================================
==========          |  90%
```

```
| Now that we've got NAs down pat, let's look at a second type of missing val
ue -- NaN, which stands
| for 'not a number'. To generate NaN, try dividing (using a forward slash) 0
by 0 now.

> 0/0
[1] NaN

| That's correct!

  |=======================================================================
==============     |  95%
| Let's do one more, just for fun. In R, Inf stands for infinity. What happen
s if you subtract Inf
| from Inf?

> Inf-Inf
[1] NaN

| Nice work!

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