| 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: 7

| | 0%

| In this lesson, we'll cover matrices and data frames. Both represent 'rectangular' data types,

| meaning that they are used to store tabular data, with rows and columns.

...

|=== | 3%

| The main difference, as you'll see, is that matrices can only contain a single class of data, while

| data frames can consist of many different classes of data.

...

|===== | 6%

| Let's create a vector containing the numbers 1 through 20 using the `:` operator. Store the result in

| a variable called my\_vector.

> my\_vector <- 1:20

| All that practice is paying off!

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

| View the contents of the vector you just created.

> my\_vector

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

| That's the answer I was looking for.

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

| The dim() function tells us the 'dimensions' of an object. What happens if we do dim(my\_vector)? Give

| it a try.

> dim(my\_vector)

NULL

| You are doing so well!

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

| Clearly, that's not very helpful! Since my\_vector is a vector, it doesn't have a `dim` attribute (so

| it's just NULL), but we can find its length using the length() function. Try that now.

> length(my\_vector)

[1] 20

| Excellent job!

|================ | 17%

| Ah! That's what we wanted. But, what happens if we give my\_vector a `dim` attribute? Let's give it a

| try. Type dim(my\_vector) <- c(4, 5).

> dim(my\_vector) <- c(4, 5)

| You nailed it! Good job!

|=================== | 20%

| It's okay if that last command seemed a little strange to you. It should! The dim() function allows

| you to get OR set the `dim` attribute for an R object. In this case, we assigned the value c(4, 5) to

| the `dim` attribute of my\_vector.

...

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

| Use dim(my\_vector) to confirm that we've set the `dim` attribute correctly.

> dim(my\_vector)

[1] 4 5

| You got it right!

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

| Another way to see this is by calling the attributes() function on my\_vector. Try it now.

> attributes(my\_vector)

$dim

[1] 4 5

| Nice work!

|=========================== | 29%

| Just like in math class, when dealing with a 2-dimensional object (think rectangular table), the

| first number is the number of rows and the second is the number of columns. Therefore, we just gave

| my\_vector 4 rows and 5 columns.

...

|============================== | 31%

| But, wait! That doesn't sound like a vector any more. Well, it's not. Now it's a matrix. View the

| contents of my\_vector now to see what it looks like.

> my\_vector

[,1] [,2] [,3] [,4] [,5]

[1,] 1 5 9 13 17

[2,] 2 6 10 14 18

[3,] 3 7 11 15 19

[4,] 4 8 12 16 20

| You're the best!

|================================ | 34%

| Now, let's confirm it's actually a matrix by using the class() function. Type class(my\_vector) to see

| what I mean.

> class(my\_vector)

[1] "matrix"

| Excellent job!

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

| Sure enough, my\_vector is now a matrix. We should store it in a new variable that helps us remember

| what it is. Store the value of my\_vector in a new variable called my\_matrix.

> my\_matrix <- my\_vector

| Keep up the great work!

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

| The example that we've used so far was meant to illustrate the point that a matrix is simply an

| atomic vector with a dimension attribute. A more direct method of creating the same matrix uses the

| matrix() function.

...

|======================================== | 43%

| Bring up the help file for the matrix() function now using the `?` function.

> ?matrix

| You are doing so well!

|=========================================== | 46%

| Now, look at the documentation for the matrix function and see if you can figure out how to create a

| matrix containing the same numbers (1-20) and dimensions (4 rows, 5 columns) by calling the matrix()

| function. Store the result in a variable called my\_matrix2.

> my\_matrix2 <- matrix(1:20, 4, 5)

| You're the best!

|============================================== | 49%

| Finally, let's confirm that my\_matrix and my\_matrix2 are actually identical. The identical() function

| will tell us if its first two arguments are the same. Try it out.

> identical(my\_matrix, my\_matrix2)

[1] TRUE

| You got it right!

|================================================ | 51%

| Now, imagine that the numbers in our table represent some measurements from a clinical experiment,

| where each row represents one patient and each column represents one variable for which measurements

| were taken.

...

|=================================================== | 54%

| We may want to label the rows, so that we know which numbers belong to each patient in the

| experiment. One way to do this is to add a column to the matrix, which contains the names of all four

| people.

...

|====================================================== | 57%

| Let's start by creating a character vector containing the names of our patients -- Bill, Gina, Kelly,

| and Sean. Remember that double quotes tell R that something is a character string. Store the result

| in a variable called patients.

> patients <- c("Bill", "Gina", "Kelly", "Sean")

| You are doing so well!

|======================================================== | 60%

| Now we'll use the cbind() function to 'combine columns'. Don't worry about storing the result in a

| new variable. Just call cbind() with two arguments -- the patients vector and my\_matrix.

> cbind(patients, my\_matrix)

patients

[1,] "Bill" "1" "5" "9" "13" "17"

[2,] "Gina" "2" "6" "10" "14" "18"

[3,] "Kelly" "3" "7" "11" "15" "19"

[4,] "Sean" "4" "8" "12" "16" "20"

| Your dedication is inspiring!

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

| Something is fishy about our result! It appears that combining the character vector with our matrix

| of numbers caused everything to be enclosed in double quotes. This means we're left with a matrix of

| character strings, which is no good.

...

|============================================================== | 66%

| If you remember back to the beginning of this lesson, I told you that matrices can only contain ONE

| class of data. Therefore, when we tried to combine a character vector with a numeric matrix, R was

| forced to 'coerce' the numbers to characters, hence the double quotes.

...

|================================================================ | 69%

| This is called 'implicit coercion', because we didn't ask for it. It just happened. But why didn't R

| just convert the names of our patients to numbers? I'll let you ponder that question on your own.

...

|=================================================================== | 71%

| So, we're still left with the question of how to include the names of our patients in the table

| without destroying the integrity of our numeric data. Try the following -- my\_data <-

| data.frame(patients, my\_matrix)

> my\_data <- data.frame(patients, my\_matrix)

| You are amazing!

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

| Now view the contents of my\_data to see what we've come up with.

> my\_data

patients X1 X2 X3 X4 X5

1 Bill 1 5 9 13 17

2 Gina 2 6 10 14 18

3 Kelly 3 7 11 15 19

4 Sean 4 8 12 16 20

| Excellent job!

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

| It looks like the data.frame() function allowed us to store our character vector of names right

| alongside our matrix of numbers. That's exactly what we were hoping for!

...

|=========================================================================== | 80%

| Behind the scenes, the data.frame() function takes any number of arguments and returns a single

| object of class `data.frame` that is composed of the original objects.

...

|============================================================================== | 83%

| Let's confirm this by calling the class() function on our newly created data frame.

> class(my\_data)

[1] "data.frame"

| That's a job well done!

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

| It's also possible to assign names to the individual rows and columns of a data frame, which presents

| another possible way of determining which row of values in our table belongs to each patient.

...

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

| However, since we've already solved that problem, let's solve a different problem by assigning names

| to the columns of our data frame so that we know what type of measurement each column represents.

...

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

| Since we have six columns (including patient names), we'll need to first create a vector containing

| one element for each column. Create a character vector called cnames that contains the following

| values (in order) -- "patient", "age", "weight", "bp", "rating", "test".

> cnames <- c("patient", "age", "weight", "bp", "rating", "test")

| You are really on a roll!

|========================================================================================= | 94%

| Now, use the colnames() function to set the `colnames` attribute for our data frame. This is similar

| to the way we used the dim() function earlier in this lesson.

> colnames(my\_data) <- cnames

| All that hard work is paying off!

|=========================================================================================== | 97%

| Let's see if that got the job done. Print the contents of my\_data.

> my\_data

patient age weight bp rating test

1 Bill 1 5 9 13 17

2 Gina 2 6 10 14 18

3 Kelly 3 7 11 15 19

4 Sean 4 8 12 16 20

| Excellent work!

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

| In this lesson, you learned the basics of working with two very important and common data structures

| -- matrices and data frames. There's much more to learn and we'll be covering more advanced topics,

| particularly with respect to data frames, in future lessons.

...