```
1: R Programming Matrices and Data Frames
2: No. Let me start something new.

Selection: 1


| Let's create a vector containing the numbers 1 through 20 using the `:` ope
rator. Store the result in a variable called
| my_vector.

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

| Not quite, but you're learning! Try again. Or, type info() for more options
.

| You learned about the `:` operator in the lesson on sequences. If you wante
d to create a vector containing the numbers 1,
| 2, and 3 (in that order), you could use either c(1, 2, 3) or 1:3. In this c
ase, we want the numbers 1 through 20 stored in
| a variable called my_vector. Also, remember that you don't need the c() fun
ction when using `:`.

> my_vector <- 1:20

| Nice work!

  |==========
|   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 a job well done!

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

| The dim() function tells us the 'dimensions' of an object. What happens if
we do dim(my_vector)? Give it try.

> dim(my_vector)
NULL

| All that hard work is paying off!

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

| Clearly, that's not very helpful! Since my_vector is a vector, it doesn't h
ave 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

| That's the answer I was looking for.

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

```
| Ah! That's what we wanted. But, what happens if we give my_vector a `dim` a
ttribute? Let's give it a try. Type
| dim(my_vector) <- c(4, 5).

> dim(my_vector) <- c(4,5)

| Excellent job!

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

| It's okay if that last command seemed a little strange to you. It should! T
he 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

| Great job!

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

| Another way to see this is by calling the attributes() function on my_vecto
r. 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 re
ctangular 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 i
t'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

| Excellent job!

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

| Now, let's confirm it's actually a matrix by using the class() function. Ty
pe class(my_vector) to see what I mean.

> class(my_vector)
[1] "matrix"

| That's the answer I was looking for.

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

| Sure enough, my_vector is now a matrix. We should store it in a new variabl
e that helps us remember what it is. Store the
| value of my_vector in a new variable called my_matrix.

> my_matrix <- my
Error: object 'my' not found
> my_matrix <- my_vector

| You are amazing!

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

| The command ?matrix will do the trick.

> ?matrix

| You're the best!

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

| Now, look at the documentation for the matrix function and see if you can f
igure out how to create a matrix containing the
| same numbers (1-20) and dimensions (4 rows, 5 columns) by calling the matri
x() function. Store the result in a variable
| called my_matrix2.
```

```
> my_matrix2 <- matrix(1:20, nrow = 4, ncol = 5)

| You are doing so well!

  |=======================================================
|  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

| Keep up the great work!

  |=========================================================
|  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 me
asurements where 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 peop
le.

...

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

| Let's start by creating a character vector containing the names of our pati
ents -- Bill, Gina, Kelly, and Sean. Remember
| that double quotes tell R that something is a character string. Store the r
esult in a variable called patients.

>
> c("Bill","Gina","Kelly","Sean")
[1] "Bill"  "Gina"  "Kelly" "Sean"

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

| Make sure to capitalize the first letter of each name and to store the resu
lt in a variable called patients. Also, don't
| get the order of the patients mixed up! That would be a disaster!

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

| You got it!

  |===============================================================
|  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(my_matrix, patients)
                      patients
[1,] "1" "5" "9"  "13" "17" "Bill"
[2,] "2" "6" "10" "14" "18" "Gina"
[3,] "3" "7" "11" "15" "19" "Kelly"
[4,] "4" "8" "12" "16" "20" "Sean"

| Give it another try. Or, type info() for more options.

| Type cbind(patients, my_matrix) to add the names of our patients to the mat
rix of numbers.

> 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"

| Perseverance, that's the answer.

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

| Something is fishy about our result! It appears that combining the characte
r vector with our matrix of numbers caused
| everything to be enclosed in double quotes. This means we're left with a ma
trix of character strings, which is no good.

...

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

| If you remember back to the beginning of this lesson, I told you that matri
ces can only contain ONE class of data.
| Therefore, when we tried to combine a character vector with a numeric matri
x, 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 h
appened. 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 p
atients in the table without destroying the
| integrity of our numeric data. Try the following -- my_data <- data.frame(p
atients, my_matrix)
```

```
> my_data <- data.frame(patients,my_matrix)

| Excellent job!

  |=======================================================================
==========                                    |  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

| Nice work!

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

| It looks like the data.frame() function allowed us to store our character v
ector 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 dat
a frame.

>
> class(my_data)
[1] "data.frame"

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

  |=======================================================================
=========================                     |  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 p
roblem 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 cr
eate 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 da
ta frame. This is similar to the way we used
| the dim() function earlier in this lesson.

> colnames(my_data)
[1] "patients" "X1"       "X2"       "X3"       "X4"       "X5"

| Not quite! Try again. Or, type info() for more options.

| Try colnames(my_data) <- cnames.

> colnames(my_data) <- cnames

| Great job!

  |=======================================================================
====================================      |  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

| You're the best!

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

| In this lesson, you learned the basics of working with two very important a
nd common data structures -- matrices and data
| frames. There's much more to learn and we'll be covering more advanced topi
cs, particularly with respect to data frames,
| in future lessons.
```