```
| The simplest and most common data structure in R is the vector.

...

  |==
|   3%
| Vectors come in two different flavors: atomic vectors and lists. An atomic
vector contains exactly
| one data type, whereas a list may contain multiple data types. We'll explor
e atomic vectors further
| before we get to lists.

...

  |=====
|   5%
| In previous lessons, we dealt entirely with numeric vectors, which are one
type of atomic vector.
| Other types of atomic vectors include logical, character, integer, and comp
lex. In this lesson,
| we'll take a closer look at logical and character vectors.

...

  |=======
|   8%
| Logical vectors can contain the values TRUE, FALSE, and NA (for 'not availa
ble'). These values are
| generated as the result of logical 'conditions'. Let's experiment with some
simple conditions.

...

  |==========
|  11%
| First, create a numeric vector num_vect that contains the values 0.5, 55, -
10, and 6.

> num_vect<-c(0.5,55,-10,6)

| Great job!

  |============
|  13%
| Now, create a variable called tf that gets the result of num_vect < 1, whic
h is read as 'num_vect
| is less than 1'.

> tf<-num_vect<1

| That's correct!

  |===============
|  16%
| What do you think tf will look like?

1: a single logical value
2: a vector of 4 logical values

Selection: 1

| Keep trying!
```

```
| Remember our lesson on vector arithmetic? The theme was that R performs man
y operations on an
| element-by-element basis. We called these 'vectorized' operations.

1: a vector of 4 logical values
2: a single logical value

Selection: 1

| That's correct!

  |=================
| 18%
| Print the contents of tf now.

> if
+
+
+
+ tf
Error: unexpected symbol in:
"
tf"
> tf
[1]  TRUE FALSE  TRUE FALSE

| You're the best!

  |===================
| 21%
| The statement num_vect < 1 is a condition and tf tells us whether each corr
esponding element of our
| numeric vector num_vect satisfies this condition.

...

  |======================
| 24%
| The first element of num_vect is 0.5, which is less than 1 and therefore th
e statement 0.5 < 1 is
| TRUE. The second element of num_vect is 55, which is greater than 1, so the
statement 55 < 1 is
| FALSE. The same logic applies for the third and fourth elements.

...

  |========================
| 26%
| Let's try another. Type num_vect >= 6 without assigning the result to a new
variable.

> num_vect>=6
[1] FALSE  TRUE FALSE  TRUE

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

  |===========================
| 29%
| This time, we are asking whether each individual element of num_vect is gre
ater than OR equal to 6.
| Since only 55 and 6 are greater than or equal to 6, the second and fourth e
lements of the result
| are TRUE and the first and third elements are FALSE.
```

...

  |============================
| 32%
| The `<` and `>=` symbols in these examples are called 'logical operators'. Other logical operators
| include `>`, `<=`, `==` for exact equality, and `!=` for inequality.

...

  |==============================
| 34%
| If we have two logical expressions, A and B, we can ask whether at least one is TRUE with A | B
| (logical 'or' a.k.a. 'union') or whether they are both TRUE with A & B (logical 'and' a.k.a.
| 'intersection'). Lastly, !A is the negation of A and is TRUE when A is FALSE and vice versa.

...

  |================================
| 37%
| It's a good idea to spend some time playing around with various combinations of these logical
| operators until you get comfortable with their use. We'll do a few examples here to get you
| started.

...

  |==================================
| 39%
| Try your best to predict the result of each of the following statements. You can use pencil and
| paper to work them out if it's helpful. If you get stuck, just guess and you've got a 50% chance of
| getting the right answer!

...

  |==
| 42%
| (3 > 5) & (4 == 4)

1: TRUE
2: FALSE

Selection: 2

| You are amazing!

  |==
| 45%
| (TRUE == TRUE) | (TRUE == FALSE)

1: TRUE
2: FALSE

Selection: 1

| Excellent job!

```
  |=============================================
|  47%
| ((111 >= 111) | !(TRUE)) & ((4 + 1) == 5)

1: TRUE
2: FALSE

Selection: 1

| All that practice is paying off!

  |==============================================
|  50%
| Don't worry if you found these to be tricky. They're supposed to be. Workin
g with logical
| statements in R takes practice, but your efforts will be rewarded in future
lessons (e.g.
| subsetting and control structures).

...

  |===============================================
|  53%
| Character vectors are also very common in R. Double quotes are used to dist
inguish character
| objects, as in the following example.

...

  |================================================
|  55%
| Create a character vector that contains the following words: "My", "name",
"is". Remember to
| enclose each word in its own set of double quotes, so that R knows they are
character strings.
| Store the vector in a variable called my_char.

> my_char<-("My","name","name")
Error: unexpected ',' in "my_char<-("My","
> my<-c("My","name","name")

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

| Type my_char <- c("My", "name", "is") to create a new variable called my_ch
ar that contains a
| character vector of length 3. Make sure that the commas separating the word
s are OUTSIDE of the
| double quotes, or else R thinks the commas are part of the words.

> my_char<-c("My","name","name")

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

| Type my_char <- c("My", "name", "is") to create a new variable called my_ch
ar that contains a
| character vector of length 3. Make sure that the commas separating the word
s are OUTSIDE of the
| double quotes, or else R thinks the commas are part of the words.

> my_char<-c("My","name","is")

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

```
  |=======================================================
|  58%
| Print the contents of my_char to see what it looks like.

> my_char
[1] "My"   "name" "is"

| You are really on a roll!

  |=======================================================
|  61%
| Right now, my_char is a character vector of length 3. Let's say we want to
join the elements of
| my_char together into one continuous character string (i.e. a character vec
tor of length 1). We can
| do this using the paste() function.

...

  |=======================================================
|  63%
| Type paste(my_char, collapse = " ") now. Make sure there's a space between
the double quotes in the
| `collapse` argument. You'll see why in a second.

> paste(my_char,collapse="")
[1] "Mynameis"

| Not exactly. Give it another go. Or, type info() for more options.

| Use paste(my_char, collapse = " ") to collapse the words in the vector so t
hey almost form a
| sentence. There should be a single space between the double quotes in the `
collapse` argument so
| that there are single spaces separating the words.

> paste(my_char, collapse = " ")
[1] "My name is"

| Your dedication is inspiring!

  |=======================================================
|  66%
| The `collapse` argument to the paste() function tells R that when we join t
ogether the elements of
| the my_char character vector, we'd like to separate them with single spaces
.

...

  |=======================================================
|  68%
| It seems that we're missing something.... Ah, yes! Your name!

...

  |=======================================================
|  71%
| To add (or 'concatenate') your name to the end of my_char, use the c() func
tion like this:
| c(my_char, "your_name_here"). Place your name in double quotes where I've p
ut "your_name_here". Try
| it now, storing the result in a new variable called my_name.
```

```
> my_name<-c(my_char,"Robin")

| Nice work!

  |======================================================================
|  74%
| Take a look at the contents of my_name.

> my_name
[1] "My"    "name"  "is"    "Robin"

| Excellent work!

  |======================================================================
|  76%
| Now, use the paste() function once more to join the words in my_name togeth
er into a single
| character string. Don't forget to say collapse = " "!

> paste(my_name, collapse = " ")
[1] "My name is Robin"

| You are amazing!

  |======================================================================
|  79%
| In this example, we used the paste() function to collapse the elements of a
single character
| vector. paste() can also be used to join the elements of multiple character
vectors.

...

  |======================================================================
=                     |  82%
| In the simplest case, we can join two character vectors that are each of le
ngth 1 (i.e. join two
| words). Try paste("Hello", "world!", sep = " "), where the `sep` argument t
ells R that we want to
| separate the joined elements with a single space.

> paste("hello","world!",sep=" ")
[1] "hello world!"

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

| Enter paste("Hello", "world!", sep = " ") to join the two words "Hello" and
"world", separated by a
| single space. There should be a single space between the double quotes in t
he `sep` argument to the
| paste() function.

> paste("Hello", "world!", sep=" ")
[1] "Hello world!"

| You nailed it! Good job!

  |======================================================================
===                   |  84%
| For a slightly more complicated example, we can join two vectors, each of l
ength 3. Use paste() to
| join the integer vector 1:3 with the character vector c("X", "Y", "Z"). Thi
s time, use sep = "" to
```

```
| leave no space between the joined elements.

> paste(1:3,c("X","y","Z"),sep=" ")
[1] "1 X" "2 y" "3 Z"

| Almost! Try again. Or, type info() for more options.

| Use paste(1:3, c("X", "Y", "Z"), sep = "") to see what happens when we join
two vectors of equal
| length using paste().

> paste(1:3, c("X", "y", "Z"), sep="")
[1] "1X" "2y" "3Z"

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

| Use paste(1:3, c("X", "Y", "Z"), sep = "") to see what happens when we join
two vectors of equal
| length using paste().

> paste(1:3,c("X","Y","Z"),sep="")
[1] "1X" "2Y" "3Z"

| Nice work!

  |=======================================================================
======              |  87%
| What do you think will happen if our vectors are of different length? (Hint
: we talked about this
| in a previous lesson.)

...

  |=======================================================================
========            |  89%
| Vector recycling! Try paste(LETTERS, 1:4, sep = "-"), where LETTERS is a pr
edefined variable in R
| containing a character vector of all 26 letters in the English alphabet.

> paste(LETTERS, 1:4, sep = "-")
 [1] "A-1" "B-2" "C-3" "D-4" "E-1" "F-2" "G-3" "H-4" "I-1" "J-2" "K-3" "L-4"
"M-1" "N-2" "O-3" "P-4"
[17] "Q-1" "R-2" "S-3" "T-4" "U-1" "V-2" "W-3" "X-4" "Y-1" "Z-2"

| You are amazing!

  |=======================================================================
===========         |  92%
| Since the character vector LETTERS is longer than the numeric vector 1:4, R
simply recycles, or
| repeats, 1:4 until it matches the length of LETTERS.

...

  |=======================================================================
=============       |  95%
| Also worth noting is that the numeric vector 1:4 gets 'coerced' into a char
acter vector by the
| paste() function.

...

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

```
| We'll discuss coercion in another lesson, but all it really means is that t
he numbers 1, 2, 3, and
| 4 in the output above are no longer numbers to R, but rather characters "1"
, "2", "3", and "4".

...

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

1: No
2: Yes
```