| 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: 6

| | 0%

| In this lesson, we'll see how to extract elements from a vector based on some conditions that we

| specify.

...

|== | 3%

| For example, we may only be interested in the first 20 elements of a vector, or only the elements

| that are not NA, or only those that are positive or correspond to a specific variable of interest. By

| the end of this lesson, you'll know how to handle each of these scenarios.

...

|===== | 5%

| I've created for you a vector called x that contains a random ordering of 20 numbers (from a standard

| normal distribution) and 20 NAs. Type x now to see what it looks like.

> x

[1] NA NA NA -1.5855552 1.1221620 -1.6624327 NA NA NA

[10] -0.5647859 NA NA NA 1.2659898 NA -0.6587773 -1.8504481 NA

[19] NA NA NA 0.1768346 NA -0.1265546 -0.4426295 0.4020785 NA

[28] NA 1.2674332 NA NA 1.0175948 0.5133586 1.6058826 -0.9052851 0.5638049

[37] 1.1070491 NA 0.1115255 0.5625621

| All that hard work is paying off!

|======= | 8%

| The way you tell R that you want to select some particular elements (i.e. a 'subset') from a vector

| is by placing an 'index vector' in square brackets immediately following the name of the vector.

...

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

| For a simple example, try x[1:10] to view the first ten elements of x.

> x[1:10]

[1] NA NA NA -1.5855552 1.1221620 -1.6624327 NA NA NA

[10] -0.5647859

| You're the best!

|============ | 13%

| Index vectors come in four different flavors -- logical vectors, vectors of positive integers,

| vectors of negative integers, and vectors of character strings -- each of which we'll cover in this

| lesson.

...

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

| Let's start by indexing with logical vectors. One common scenario when working with real-world data

| is that we want to extract all elements of a vector that are not NA (i.e. missing data). Recall that

| is.na(x) yields a vector of logical values the same length as x, with TRUEs corresponding to NA

| values in x and FALSEs corresponding to non-NA values in x.

...

|================= | 18%

| What do you think x[is.na(x)] will give you?

1: A vector of all NAs

2: A vector of length 0

3: A vector with no NAs

4: A vector of TRUEs and FALSEs

Selection: 1

| Keep up the great work!

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

| Prove it to yourself by typing x[is.na(x)].

> x[is.na(x)]

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

| Your dedication is inspiring!

|====================== | 24%

| Recall that `!` gives us the negation of a logical expression, so !is.na(x) can be read as 'is not

| NA'. Therefore, if we want to create a vector called y that contains all of the non-NA values from x,

| we can use y <- x[!is.na(x)]. Give it a try.

> y <- x[!is.na(x)]

| Perseverance, that's the answer.

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

| Print y to the console.

> y

[1] -1.5855552 1.1221620 -1.6624327 -0.5647859 1.2659898 -0.6587773 -1.8504481 0.1768346 -0.1265546

[10] -0.4426295 0.4020785 1.2674332 1.0175948 0.5133586 1.6058826 -0.9052851 0.5638049 1.1070491

[19] 0.1115255 0.5625621

| You're the best!

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

| Now that we've isolated the non-missing values of x and put them in y, we can subset y as we please.

...

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

| Recall that the expression y > 0 will give us a vector of logical values the same length as y, with

| TRUEs corresponding to values of y that are greater than zero and FALSEs corresponding to values of y

| that are less than or equal to zero. What do you think y[y > 0] will give you?

1: A vector of all NAs

2: A vector of TRUEs and FALSEs

3: A vector of all the negative elements of y

4: A vector of length 0

5: A vector of all the positive elements of y

Selection: 5

| All that practice is paying off!

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

| Type y[y > 0] to see that we get all of the positive elements of y, which are also the positive

| elements of our original vector x.

> y[y > 0]

[1] 1.1221620 1.2659898 0.1768346 0.4020785 1.2674332 1.0175948 0.5133586 1.6058826 0.5638049 1.1070491

[11] 0.1115255 0.5625621

| Great job!

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

| You might wonder why we didn't just start with x[x > 0] to isolate the positive elements of x. Try

| that now to see why.

> x[x > 0]

[1] NA NA NA 1.1221620 NA NA NA NA NA NA

[11] 1.2659898 NA NA NA NA NA 0.1768346 NA 0.4020785 NA

[21] NA 1.2674332 NA NA 1.0175948 0.5133586 1.6058826 0.5638049 1.1070491 NA

[31] 0.1115255 0.5625621

| Nice work!

|===================================== | 39%

| Since NA is not a value, but rather a placeholder for an unknown quantity, the expression NA > 0

| evaluates to NA. Hence we get a bunch of NAs mixed in with our positive numbers when we do this.

...

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

| Combining our knowledge of logical operators with our new knowledge of subsetting, we could do this

| -- x[!is.na(x) & x > 0]. Try it out.

> x[!is.na(x) & x > 0]

[1] 1.1221620 1.2659898 0.1768346 0.4020785 1.2674332 1.0175948 0.5133586 1.6058826 0.5638049 1.1070491

[11] 0.1115255 0.5625621

| Excellent work!

|========================================== | 45%

| In this case, we request only values of x that are both non-missing AND greater than zero.

...

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

| I've already shown you how to subset just the first ten values of x using x[1:10]. In this case,

| we're providing a vector of positive integers inside of the square brackets, which tells R to return

| only the elements of x numbered 1 through 10.

...

|=============================================== | 50%

| Many programming languages use what's called 'zero-based indexing', which means that the first

| element of a vector is considered element 0. R uses 'one-based indexing', which (you guessed it!)

| means the first element of a vector is considered element 1.

...

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

| Can you figure out how we'd subset the 3rd, 5th, and 7th elements of x? Hint -- Use the c() function

| to specify the element numbers as a numeric vector.

> x[c(3, 5, 7)]

[1] NA 1.122162 NA

| You got it!

|==================================================== | 55%

| It's important that when using integer vectors to subset our vector x, we stick with the set of

| indexes {1, 2, ..., 40} since x only has 40 elements. What happens if we ask for the zeroth element

| of x (i.e. x[0])? Give it a try.

> x[0]

numeric(0)

| Perseverance, that's the answer.

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

| As you might expect, we get nothing useful. Unfortunately, R doesn't prevent us from doing this. What

| if we ask for the 3000th element of x? Try it out.

> x[3000]

[1] NA

| That's the answer I was looking for.

|========================================================= | 61%

| Again, nothing useful, but R doesn't prevent us from asking for it. This should be a cautionary tale.

| You should always make sure that what you are asking for is within the bounds of the vector you're

| working with.

...

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

| What if we're interested in all elements of x EXCEPT the 2nd and 10th? It would be pretty tedious to

| construct a vector containing all numbers 1 through 40 EXCEPT 2 and 10.

...

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

| Luckily, R accepts negative integer indexes. Whereas x[c(2, 10)] gives us ONLY the 2nd and 10th

| elements of x, x[c(-2, -10)] gives us all elements of x EXCEPT for the 2nd and 10 elements. Try

| x[c(-2, -10)] now to see this.

> x[c(-2, -10)]

[1] NA NA -1.5855552 1.1221620 -1.6624327 NA NA NA NA

[10] NA NA 1.2659898 NA -0.6587773 -1.8504481 NA NA NA

[19] NA 0.1768346 NA -0.1265546 -0.4426295 0.4020785 NA NA 1.2674332

[28] NA NA 1.0175948 0.5133586 1.6058826 -0.9052851 0.5638049 1.1070491 NA

[37] 0.1115255 0.5625621

| You are doing so well!

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

| A shorthand way of specifying multiple negative numbers is to put the negative sign out in front of

| the vector of positive numbers. Type x[-c(2, 10)] to get the exact same result.

> x[-c(2,10)]

[1] NA NA -1.5855552 1.1221620 -1.6624327 NA NA NA NA

[10] NA NA 1.2659898 NA -0.6587773 -1.8504481 NA NA NA

[19] NA 0.1768346 NA -0.1265546 -0.4426295 0.4020785 NA NA 1.2674332

[28] NA NA 1.0175948 0.5133586 1.6058826 -0.9052851 0.5638049 1.1070491 NA

[37] 0.1115255 0.5625621

| Keep up the great work!

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

| So far, we've covered three types of index vectors -- logical, positive integer, and negative

| integer. The only remaining type requires us to introduce the concept of 'named' elements.

...

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

| Create a numeric vector with three named elements using vect <- c(foo = 11, bar = 2, norf = NA).

> vect <- c(foo = 11, bar = 2, norf = NA)

| All that hard work is paying off!

|======================================================================== | 76%

| When we print vect to the console, you'll see that each element has a name. Try it out.

> vect

foo bar norf

11 2 NA

| That's correct!

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

| We can also get the names of vect by passing vect as an argument to the names() function. Give that a

| try.

> names(vect)

[1] "foo" "bar" "norf"

| Excellent job!

|============================================================================= | 82%

| Alternatively, we can create an unnamed vector vect2 with c(11, 2, NA). Do that now.

> vect2 <- c(11, 2, NA)

| That's correct!

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

| Then, we can add the `names` attribute to vect2 after the fact with names(vect2) <- c("foo", "bar",

| "norf"). Go ahead.

> names(vect2) <- c("foo", "bar", "norf")

| You nailed it! Good job!

|================================================================================== | 87%

| Now, let's check that vect and vect2 are the same by passing them as arguments to the identical()

| function.

> identical(vect, vect2)

[1] TRUE

| Great job!

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

| Indeed, vect and vect2 are identical named vectors.

...

|======================================================================================= | 92%

| Now, back to the matter of subsetting a vector by named elements. Which of the following commands do

| you think would give us the second element of vect?

1: vect[bar]

2: vect["bar"]

3: vect["2"]

Selection: 2

| You are amazing!

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

| Now, try it out.

> vect["bar"]

bar

2

| Excellent work!

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

| Likewise, we can specify a vector of names with vect[c("foo", "bar")]. Try it out.

> vect[c("foo", "bar")]

foo bar

11 2

| You are really on a roll!

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

| Now you know all four methods of subsetting data from vectors. Different approaches are best in

| different scenarios and when in doubt, try it out!

...