```
Selection: 1

| 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 Ve
ctors
 7: Matrices and Data Frames   8: Logic                      9: Functions
10: lapply and sapply         11: vapply and tapply         12: Looking at Da
ta
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 so
me 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 specifi
c variable of interest.
| By the end of this lesson, you'll know how to handle each of these scenario
s.

...

  |=====
|   5%
| I've created for you a vector called x that contains a random ordering of 2
0 numbers (from a
| standard normal distribution) and 20 NAs. Type x now to see what it looks l
ike.

> x
 [1]          NA          NA          NA          NA          NA -0.94887718
NA  0.13725073
 [9]          NA          NA -1.85099548          NA  0.94707769  0.47929006
-0.23134894          NA
[17]  0.03379646  1.41590243          NA -0.37050224  0.05623335  1.19070883
-1.19223571  1.56108480
[25]          NA -0.85689449  0.32979317  0.26235968 -1.38318148          NA
-0.43592424 -0.28931595
[33]          NA -0.26026676          NA          NA          NA          NA
NA          NA

| That's correct!

  |=======
|   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 th
e name of the vector.
```

```
...

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

> x[1:10]
 [1]          NA          NA          NA          NA          NA -0.9488772
NA  0.1372507          NA
[10]          NA

| You are amazing!

  |============
|  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 w
hich we'll cover in this
| lesson.

...

  |==============
|  15%
| Let's start by indexing with logical vectors. One common scenario when work
ing with real-world data
| is that we want to extract all elements of a vector that are not NA (i.e. m
issing 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 length 0
2: A vector of all NAs
3: A vector of TRUEs and FALSEs
4: A vector with no NAs

Selection: 3

| That's not exactly what I'm looking for. Try again.

| Remember that is.na(x) tells us where the NAs are in a vector. So if we sub
set x based on that, what
| do you expect to happen?

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: 2

| One more time. You can do it!

| Remember that is.na(x) tells us where the NAs are in a vector. So if we sub
set x based on that, what
| do you expect to happen?
```

```
1: A vector of TRUEs and FALSEs
2: A vector with no NAs
3: A vector of all NAs
4: A vector of length 0

Selection: 1

| Keep trying!

| Remember that is.na(x) tells us where the NAs are in a vector. So if we sub
set x based on that, what
| do you expect to happen?

1: A vector with no NAs
2: A vector of all NAs
3: A vector of TRUEs and FALSEs
4: A vector of length 0

Selection: 2

| Excellent job!

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

| You are doing so well!

  |====================
|  23%
| 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] -0.94887718  0.13725073 -1.85099548  0.94707769  0.47929006 -0.23134894
0.03379646  1.41590243
 [9] -0.37050224  0.05623335  1.19070883 -1.19223571  1.56108480 -0.85689449
0.32979317  0.26235968
[17] -1.38318148 -0.43592424 -0.28931595 -0.26026676

| That's correct!

  |========================
|  28%
| Now that we've isolated the non-missing values of x and put them in y, we c
an subset y as we please.

...
```

```
  |=============================
|  31%
| Recall that the expression y > 0 will give us a vector of logical values th
e same length as y, with
| TRUEs corresponding to values of y that are greater than zero and FALSEs co
rresponding 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 the negative elements of y
2: A vector of length 0
3: A vector of TRUEs and FALSEs
4: A vector of all NAs
5: A vector of all the positive elements of y

Selection: 5

| Nice work!

  |==============================
|  33%
| Type y[y > 0] to see that we get all of the positive elements of y, which a
re also the positive
| elements of our original vector x.

> y[y>0]
 [1] 0.13725073 0.94707769 0.47929006 0.03379646 1.41590243 0.05623335 1.1907
0883 1.56108480 0.32979317
[10] 0.26235968

| That's correct!

  |================================
|  36%
| You might wonder why we didn't just start with x[x > 0] to isolate the posi
tive elements of x. Try
| that now to see why.

> x[x>0]
 [1]          NA          NA          NA          NA          NA          NA 0.1372
5073          NA          NA
[10]          NA 0.94707769 0.47929006          NA 0.03379646 1.41590243
NA 0.05623335 1.19070883
[19] 1.56108480          NA 0.32979317 0.26235968          NA          NA
NA          NA          NA
[28]          NA          NA          NA

| You are doing so well!

  |==================================
|  38%
| 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 num
bers when we do this.

...

  |====================================
|  41%
| Combining our knowledge of logical operators with our new knowledge of subs
etting, we could do this
| -- x[!is.na(x) & x > 0]. Try it out.
```

```
> x[!is.na(x)$x>0]
Error in is.na(x)$x : $ operator is invalid for atomic vectors
> x[!is.na(x)&x>0]
 [1] 0.13725073 0.94707769 0.47929006 0.03379646 1.41590243 0.05623335 1.1907
0883 1.56108480 0.32979317
[10] 0.26235968

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

  |=========================================
|  44%
| In this case, we request only values of x that are both non-missing AND gre
ater than zero.

...

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

...

  |===========================================
|  49%
| Many programming languages use what's called 'zero-based indexing', which m
eans that the first
| element of a vector is considered element 0. R uses 'one-based indexing', w
hich (you guessed it!)
| means the first element of a vector is considered element 1.

...

  |============================================
|  51%
| Can you figure out how we'd subset the 3rd, 5th, and 7th elements of x? Hin
t -- Use the c() function
| to specify the element numbers as a numeric vector.

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

| Nice work!

  |=============================================
|  54%
| It's important that when using integer vectors to subset our vector x, we s
tick with the set of
| indexes {1, 2, ..., 40} since x only has 40 elements. What happens if we as
k for the zeroth element
| of x (i.e. x[0])? Give it a try.

> x[0]
numeric(0)

| You are doing so well!

  |==============================================
|  56%
| As you might expect, we get nothing useful. Unfortunately, R doesn't preven
t us from doing this.
```

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

> x[3000]
[1] NA

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

  |=============================================================
|  59%
| Again, nothing useful, but R doesn't prevent us from asking for it. This sh
ould be a cautionary
| tale. You should always make sure that what you are asking for is within th
e bounds of the vector
| you're working with.

...

  |=============================================================
|  62%
| What if we're interested in all elements of x EXCEPT the 2nd and 10th? It w
ould be pretty tedious to
| construct a vector containing all numbers 1 through 40 EXCEPT 2 and 10.

...

  |=============================================================
|  64%
| Luckily, R accepts negative integer indexes. Whereas x[c(2, 10)] gives us O
NLY 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          NA          NA -0.94887718          NA
0.13725073          NA
 [9] -1.85099548          NA  0.94707769  0.47929006 -0.23134894          NA
0.03379646  1.41590243
[17]          NA -0.37050224  0.05623335  1.19070883 -1.19223571  1.56108480
NA -0.85689449
[25]  0.32979317  0.26235968 -1.38318148          NA -0.43592424 -0.28931595
NA -0.26026676
[33]          NA          NA          NA          NA          NA          NA

| Excellent job!

  |=============================================================
|  67%
| A shorthand way of specifying multiple negative numbers is to put the negat
ive sign out in front of
| the vector of positive numbers. Type x[-c(2, 10)] to get the exact same res
ult.

> x[-c(2, 10)]
 [1]          NA          NA          NA          NA -0.94887718          NA
0.13725073          NA
 [9] -1.85099548          NA  0.94707769  0.47929006 -0.23134894          NA
0.03379646  1.41590243
[17]          NA -0.37050224  0.05623335  1.19070883 -1.19223571  1.56108480
NA -0.85689449
[25]  0.32979317  0.26235968 -1.38318148          NA -0.43592424 -0.28931595
NA -0.26026676
[33]          NA          NA          NA          NA          NA          NA
```

```
| Great job!

  |=======================================================================
|  69%
| So far, we've covered three types of index vectors -- logical, positive int
eger, and negative
| integer. The only remaining type requires us to introduce the concept of 'n
amed' elements.

...

  |=======================================================================
|  72%
| 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)

| Great job!

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

| Excellent job!

  |=======================================================================
|  77%
| We can also get the names of vect by passing vect as an argument to the nam
es() function. Give that
| a try.

> names(vect)
[1] "foo"  "bar"  "norf"

| You nailed it! Good job!

  |=======================================================================
|  79%
| Alternatively, we can create an unnamed vector vect2 with c(11, 2, NA). Do
that now.

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

| Try again. Getting it right on the first try is boring anyway! Or, type inf
o() for more options.

| Create an ordinary (unnamed) vector called vect2 that contains c(11, 2, NA)
.

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

| Your dedication is inspiring!

  |=======================================================================
==                  |  82%
| Then, we can add the `names` attribute to vect2 after the fact with names(v
ect2) <- c("foo", "bar",
| "norf"). Go ahead.
```

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

| Nice work!

  |=========================================================================
=====                |  85%
| Now, let's check that vect and vect2 are the same by passing them as argume
nts to the identical()
| function.

> identical(vect,vect2)
[1] TRUE

| Great job!

  |=========================================================================
=======              |  87%
| Indeed, vect and vect2 are identical named vectors.

...

  |=========================================================================
=========            |  90%
| 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["2"]
3: vect[bar]

Selection: 1

| You are really on a roll!

  |=========================================================================
============         |  92%
| Now, try it out.

> vec[t"bar"]
Error: unexpected string constant in "vec[t"bar""
> vec["bar"]
Error: object 'vec' not found
> vect["bar"]
bar
  2

| Excellent work!

  |=========================================================================
==============       |  95%
| Likewise, we can specify a vector of names with vect[c("foo", "bar")]. Try
it out.

> vect[c("foo", "bar")]
foo bar
 11   2

| Perseverance, that's the answer.

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

```
| Now you know all four methods of subsetting data from vectors. Different ap
proaches are best in
| different scenarios and when in doubt, try it out!

...

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

1: No
2: Yes
```