```
Selection: 8

  |
|   0%

| This lesson is meant to be a short introduction to logical operations in R.

...

  |==
|   2%
| There are two logical values in R, also called boolean values. They are TRU
E and FALSE. In R you can
| construct logical expressions which will evaluate to either TRUE or FALSE.

...

  |====
|   4%
| Many of the questions in this lesson will involve evaluating logical expres
sions. It may be useful
| to open up a second R terminal where you can experiment with some of these
expressions.

...

  |=====
|   6%
| Creating logical expressions requires logical operators. You're probably fa
miliar with arithmetic
| operators like `+`, `-`, `*`, and `/`. The first logical operator we are go
ing to discuss is the
| equality operator, represented by two equals signs `==`. Use the equality o
perator below to find out
| if TRUE is equal to TRUE.

> TRUE==TRUE
[1] TRUE

| You are amazing!

  |=======
|   8%
| Just like arithmetic, logical expressions can be grouped by parenthesis so
that the entire
| expression (TRUE == TRUE) == TRUE evaluates to TRUE.

...

  |=========
|  10%
| To test out this property, try evaluating (FALSE == TRUE) == FALSE .

> (FALSE == TRUE) == FALSE
[1] TRUE

| That's a job well done!

  |===========
|  12%
| The equality operator can also be used to compare numbers. Use `==` to see
if 6 is equal to 7.

> 6==7
```

```
[1] FALSE

| Keep up the great work!

  |=============
|  13%
| The previous expression evaluates to FALSE because 6 is less than 7. Thankf
ully, there are
| inequality operators that allow us to test if a value is less than or great
er than another value.

...

  |==============
|  15%
| The less than operator `<` tests whether the number on the left side of the
operator (called the
| left operand) is less than the number on the right side of the operator (ca
lled the right operand).
| Write an expression to test whether 6 is less than 7.

> 6<7
[1] TRUE

| You're the best!

  |================
|  17%
| There is also a less-than-or-equal-to operator `<=` which tests whether the
left operand is less
| than or equal to the right operand. Write an expression to test whether 10
is less than or equal to
| 10.

> 10<=10
[1] TRUE

| You nailed it! Good job!

  |==================
|  19%
| Keep in mind that there are the corresponding greater than `>` and greater-
than-or-equal-to `>=`
| operators.

...

  |====================
|  21%
| Which of the following evaluates to FALSE?

1: 9 >= 10
2: 0 > -36
3: 7 == 7
4: 6 < 8

Selection: 1

| You are doing so well!

  |====================
|  23%
| Which of the following evaluates to TRUE?
```

```
1: 7 == 9
2: -6 > -7
3: 9 >= 10
4: 57 < 8

Selection: 2

| Perseverance, that's the answer.

  |=======================
| 25%
| The next operator we will discuss is the 'not equals' operator represented
by `!=`. Not equals tests
| whether two values are unequal, so TRUE != FALSE evaluates to TRUE. Like th
e equality operator, `!=`
| can also be used with numbers. Try writing an expression to see if 5 is not
equal to 7.

> 5!=7
[1] TRUE

| You are quite good my friend!

  |========================
| 27%
| In order to negate boolean expressions you can use the NOT operator. An exc
lamation point `!` will
| cause !TRUE (say: not true) to evaluate to FALSE and !FALSE (say: not false
) to evaluate to TRUE.
| Try using the NOT operator and the equals operator to find the opposite of
whether 5 is equal to 7.

> 5==7==!TRUE
Error: unexpected '==' in "5==7=="
> (5==7)==!TRUE
[1] TRUE

| Keep trying! Or, type info() for more options.

| This expression may be a little tricky, so think about negating the express
ion 5 == 7 (all you need
| is an exclamation point in front).

> 5!==7
Error: unexpected '=' in "5!=="
> 5!=7
[1] TRUE

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

| This expression may be a little tricky, so think about negating the express
ion 5 == 7 (all you need
| is an exclamation point in front).

> info()

| When you are at the R prompt (>):
| -- Typing skip() allows you to skip the current question.
| -- Typing play() lets you experiment with R on your own; swirl will ignore
what you do...
| -- UNTIL you type nxt() which will regain swirl's attention.
| -- Typing bye() causes swirl to exit. Your progress will be saved.
| -- Typing main() returns you to swirl's main menu.
```

```
| -- Typing info() displays these options again.

> !5==7
[1] TRUE

| Perseverance, that's the answer.

  |=========================
|  29%
| Let's take a moment to review. The equals operator `==` tests whether two b
oolean values or numbers
| are equal, the not equals operator `!=` tests whether two boolean values or
numbers are unequal, and
| the NOT operator `!` negates logical expressions so that TRUE expressions b
ecome FALSE and FALSE
| expressions become TRUE.

...

  |============================
|  31%
| Which of the following evaluates to FALSE?

1: 7 != 8
2: !(0 >= -1)
3: 9 < 10
4: !FALSE

Selection:
Enter an item from the menu, or 0 to exit
Selection: 2

| Keep up the great work!

  |============================
|  33%
| What do you think the following expression will evaluate to?: (TRUE != FALS
E) == !(6 == 7)

1: %>%
2: FALSE
3: TRUE
4: Can there be objective truth when programming?

Selection: 2

| Not exactly. Give it another go.

| Try to evaluate each expression in isolation and build up an answer.

1: TRUE
2: %>%
3: FALSE
4: Can there be objective truth when programming?

Selection: 1

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

  |===============================
|  35%
| At some point you may need to examine relationships between multiple logica
l expressions. This is
| where the AND operator and the OR operator come in.
```

```
...

  |==================================
|  37%
| Let's look at how the AND operator works. There are two AND operators in R,
`&` and `&&`. Both
| operators work similarly, if the right and left operands of AND are both TR
UE the entire expression
| is TRUE, otherwise it is FALSE. For example, TRUE & TRUE evaluates to TRUE.
Try typing FALSE & FALSE
| to how it is evaluated.

> FALSE&FALSE
[1] FALSE

| You are really on a roll!

  |===================================
|  38%
| You can use the `&` operator to evaluate AND across a vector. The `&&` vers
ion of AND only evaluates
| the first member of a vector. Let's test both for practice. Type the expres
sion TRUE & c(TRUE,
| FALSE, FALSE).

> TRUE&c(TRUE,TRUE,FALSE)
[1]  TRUE  TRUE FALSE

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

| Now to see how the AND operator works with a vector, type: TRUE & c(TRUE, F
ALSE, FALSE)

> TRUE & c(TRUE, FALSE, FALSE)
[1]  TRUE FALSE FALSE

| Excellent job!

  |====================================
|  40%
| What happens in this case is that the left operand `TRUE` is recycled acros
s every element in the
| vector of the right operand. This is the equivalent statement as c(TRUE, TR
UE, TRUE) & c(TRUE,
| FALSE, FALSE).

...

  |====================================
|  42%
| Now we'll type the same expression except we'll use the `&&` operator. Type
the expression TRUE &&
| c(TRUE, FALSE, FALSE).

> TRUE && c(TRUE, FALSE, FALSE)
[1] TRUE

| You got it right!

  |====================================
|  44%
| In this case, the left operand is only evaluated with the first member of t
he right operand (the
```

```
| vector). The rest of the elements in the vector aren't evaluated at all in
this expression.

...

  |=========================================
|  46%
| The OR operator follows a similar set of rules. The `|` version of OR evalu
ates OR across an entire
| vector, while the `||` version of OR only evaluates the first member of a v
ector.

...

  |==========================================
|  48%
| An expression using the OR operator will evaluate to TRUE if the left opera
nd or the right operand
| is TRUE. If both are TRUE, the expression will evaluate to TRUE, however if
neither are TRUE, then
| the expression will be FALSE.

...

  |==========================================
|  50%
| Let's test out the vectorized version of the OR operator. Type the expressi
on TRUE | c(TRUE, FALSE,
| FALSE).

> TRUE|c(TRUE, FALSE, FALSE)
[1] TRUE TRUE TRUE

| Excellent job!

  |==========================================
|  52%
| Now let's try out the non-vectorized version of the OR operator. Type the e
xpression TRUE || c(TRUE,
| FALSE, FALSE).

> TRUE||c(TRUE, FALSE, FALSE)
[1] TRUE

| You are doing so well!

  |==========================================
|  54%
| Logical operators can be chained together just like arithmetic operators. T
he expressions: `6 != 10
| && FALSE && 1 >= 2` or `TRUE || 5 < 9.3 || FALSE` are perfectly normal to s
ee.

...

  |==========================================
|  56%
| As you may recall, arithmetic has an order of operations and so do logical
expressions. All AND
| operators are evaluated before OR operators. Let's look at an example of an
ambiguous case. Type: 5
| > 8 || 6 != 8 && 4 > 3.9

> 5> 8 || 6 != 8 && 4 > 3.9
```

```
[1] TRUE

| You got it!

  |=======================================================
|  58%
| Let's walk through the order of operations in the above case. First the lef
t and right operands of
| the AND operator are evaluated. 6 is not equal 8, 4 is greater than 3.9, th
erefore both operands are
| TRUE so the resulting expression `TRUE && TRUE` evaluates to TRUE. Then the
left operand of the OR
| operator is evaluated: 5 is not greater than 8 so the entire expression is
reduced to FALSE || TRUE.
| Since the right operand of this expression is TRUE the entire expression ev
aluates to TRUE.

...

  |=======================================================
|  60%
| Which one of the following expressions evaluates to TRUE?

1: TRUE && 62 < 62 && 44 >= 44
2: TRUE && FALSE || 9 >= 4 && 3 < 6
3: 99.99 > 100 || 45 < 7.3 || 4 != 4.0
4: FALSE || TRUE && FALSE

Selection: 2

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

  |=========================================================
|  62%
| Which one of the following expressions evaluates to FALSE?

1: 6 >= -9 && !(6 > 7) && !(!TRUE)
2: FALSE && 6 >= 6 || 7 >= 8 || 50 <= 49.5
3: !(8 > 4) ||  5 == 5.0 && 7.8 >= 7.79
4: FALSE || TRUE && 6 != 4 || 9 > 4

Selection:
Enter an item from the menu, or 0 to exit
Selection:
Enter an item from the menu, or 0 to exit
Selection: 2

| You are amazing!

  |=========================================================
|  63%
| Now that you're familiar with R's logical operators you can take advantage
of a few functions that R
| provides for dealing with logical expressions.

...

  |===========================================================
|  65%
| The function isTRUE() takes one argument. If that argument evaluates to TRU
E, the function will
| return TRUE. Otherwise, the function will return FALSE. Try using this func
tion by typing: isTRUE(6
| > 4)
```

```
> isTRUE(6>4)
[1] TRUE

| All that hard work is paying off!

  |======================================================================
|  67%
| Which of the following evaluates to TRUE?

1: !isTRUE(4 < 3)
2: isTRUE(!TRUE)
3: isTRUE(3)
4: isTRUE(NA)
5: !isTRUE(8 != 5)

Selection: 1

| You are quite good my friend!

  |======================================================================
|  69%
| The function identical() will return TRUE if the two R objects passed to it
as arguments are
| identical. Try out the identical() function by typing: identical('twins', '
twins')

> identical('twins', 'twins')
[1] TRUE

| Nice work!

  |======================================================================
|  71%
| Which of the following evaluates to TRUE?

1: identical(4, 3.1)
2: identical('hello', 'Hello')
3: !identical(7, 7)
4: identical(5 > 4, 3 < 3.1)

Selection: 4

| That's correct!

  |======================================================================
|  73%
| You should also be aware of the xor() function, which takes two arguments.
The xor() function stands
| for exclusive OR. If one argument evaluates to TRUE and one argument evalua
tes to FALSE, then this
| function will return TRUE, otherwise it will return FALSE. Try out the xor(
) function by typing:
| xor(5 == 6, !FALSE)

> xor(5 == 6, !FALSE)
[1] TRUE

| You are amazing!

  |======================================================================
|  75%
| 5 == 6 evaluates to FALSE, !FALSE evaluates to TRUE, so xor(FALSE, TRUE) ev
aluates to TRUE. On the
```

```
| other hand if the first argument was changed to 5 == 5 and the second argum
ent was unchanged then
| both arguments would have been TRUE, so xor(TRUE, TRUE) would have evaluate
d to FALSE.

...

  |=======================================================================
|  77%
| Which of the following evaluates to FALSE?

1: xor(!!TRUE, !!FALSE)
2: xor(4 >= 9, 8 != 8.0)
3: xor(identical(xor, 'xor'), 7 == 7.0)
4: xor(!isTRUE(TRUE), 6 > -1)

Selection: 2

| You nailed it! Good job!

  |=======================================================================
|  79%
| For the next few questions, we're going to need to create a vector of integ
ers called ints. Create
| this vector by typing: ints <- sample(10)

> ints <- sample(10)

| All that hard work is paying off!

  |=======================================================================
=                    |  81%
| Now simply display the contents of ints.

> ints
 [1]  2  8  6  1  7  3  5  4 10  9

| That's the answer I was looking for.

  |=======================================================================
===                  |  83%
| The vector `ints` is a random sampling of integers from 1 to 10 without rep
lacement. Let's say we
| wanted to ask some logical questions about contents of ints. If we type int
s > 5, we will get a
| logical vector corresponding to whether each element of ints is greater tha
n 5. Try typing: ints > 5

> ints > 5
 [1] FALSE  TRUE  TRUE FALSE  TRUE FALSE FALSE FALSE  TRUE  TRUE

| You got it right!

  |=======================================================================
=====                |  85%
| We can use the resulting logical vector to ask other questions about ints.
The which() function
| takes a logical vector as an argument and returns the indices of the vector
that are TRUE. For
| example which(c(TRUE, FALSE, TRUE)) would return the vector c(1, 3).

...
```

```
  |=========================================================================
======              |  87%
| Use the which() function to find the indices of ints that are greater than
7.

> which(TRUE)
[1] 1

| Keep trying! Or, type info() for more options.

| Use the which() function on the logical vector produced by: `ints > 7`

> which(ints>7)
[1]  2  9 10

| You're the best!

  |=========================================================================
========            |  88%
| Which of the following commands would produce the indices of the elements i
n ints that are less than
| or equal to 2?

1: which(ints <= 2)
2: ints <= 2
3: ints < 2
4: which(ints < 2)

Selection: 1

| Nice work!

  |=========================================================================
==========          |  90%
| Like the which() function, the functions any() and all() take logical vecto
rs as their argument. The
| any() function will return TRUE if one or more of the elements in the logic
al vector is TRUE. The
| all() function will return TRUE if every element in the logical vector is T
RUE.

...

  |=========================================================================
============        |  92%
| Use the any() function to see if any of the elements of ints are less than
zero.

> any(ints<0)
[1] FALSE

| Keep up the great work!

  |=========================================================================
==============      |  94%
| Use the all() function to see if all of the elements of ints are greater th
an zero.

> all(ints>0)
[1] TRUE

| You got it right!
```

```
  |=======================================================================
===============     |  96%
| Which of the following evaluates to TRUE?

1: any(ints == 10)
2: all(ints == 10)
3: any(ints == 2.5)
4: all(c(TRUE, FALSE, TRUE))

Selection: 1

| You are quite good my friend!

  |=======================================================================
================  |  98%
| That's all for this introduction to logic in R. If you really want to see w
hat you can do with
| logic, check out the control flow lesson!

...

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