| 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: 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 TRUE 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 expressions. 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 familiar with arithmetic

| operators like `+`, `-`, `\*`, and `/`. The first logical operator we are going to discuss is the

| equality operator, represented by two equals signs `==`. Use the equality operator below to find out

| if TRUE is equal to TRUE.

> TRUE == TRUE

[1] TRUE

| Excellent work!

|======= | 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

| All that practice is paying off!

|=========== | 12%

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

> 6 == 7

[1] FALSE

| That's correct!

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

| The previous expression evaluates to FALSE because 6 is less than 7. Thankfully, there are inequality

| operators that allow us to test if a value is less than or greater than another value.

...

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

| 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 (called the right operand). Write

| an expression to test whether 6 is less than 7.

> 6 < 7

[1] TRUE

| Excellent job!

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

| 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

| Nice work!

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

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

| operators.

...

|==================== | 22%

| Which of the following evaluates to FALSE?

1: 6 < 8

2: 0 > -36

3: 7 == 7

4: 9 >= 10

Selection: 4

| You nailed it! Good job!

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

| Which of the following evaluates to TRUE?

1: 9 >= 10

2: 57 < 8

3: 7 == 9

4: -6 > -7

Selection: 4

| That's a job well done!

|======================== | 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 the 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

| Nice work!

|========================== | 27%

| In order to negate boolean expressions you can use the NOT operator. An exclamation 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)

[1] TRUE

| Perseverance, that's the answer.

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

| Let's take a moment to review. The equals operator `==` tests whether two boolean 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 become 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: 2

| Excellent work!

|=============================== | 33%

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

1: %>%

2: FALSE

3: Can there be objective truth when programming?

4: TRUE

Selection: 4

| You're the best!

|================================= | 35%

| At some point you may need to examine relationships between multiple logical 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 TRUE 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

| That's correct!

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

| You can use the `&` operator to evaluate AND across a vector. The `&&` version of AND only evaluates

| the first member of a vector. Let's test both for practice. Type the expression TRUE & c(TRUE, FALSE,

| FALSE).

> TRUE & c(TRUE, FALSE, FALSE)

[1] TRUE FALSE FALSE

| All that practice is paying off!

|======================================= | 41%

| What happens in this case is that the left operand `TRUE` is recycled across every element in the

| vector of the right operand. This is the equivalent statement as c(TRUE, TRUE, TRUE) & c(TRUE, FALSE,

| FALSE).

...

|========================================= | 43%

| 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 are really on a roll!

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

| In this case, the left operand is only evaluated with the first member of the right operand (the

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

...

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

| The OR operator follows a similar set of rules. The `|` version of OR evaluates OR across an entire

| vector, while the `||` version of OR only evaluates the first member of a vector.

...

|============================================== | 49%

| An expression using the OR operator will evaluate to TRUE if the left operand 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.

...

|================================================ | 51%

| Let's test out the vectorized version of the OR operator. Type the expression TRUE | c(TRUE, FALSE,

| FALSE).

> TRUE | c(TRUE, FALSE, FALSE)

[1] TRUE TRUE TRUE

| You're the best!

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

| Now let's try out the non-vectorized version of the OR operator. Type the expression TRUE || c(TRUE,

| FALSE, FALSE).

> TRUE || c(TRUE, FALSE, FALSE)

[1] TRUE

| Excellent job!

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

| Logical operators can be chained together just like arithmetic operators. The expressions: `6 != 10

| && FALSE && 1 >= 2` or `TRUE || 5 < 9.3 || FALSE` are perfectly normal to see.

...

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

| 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 are quite good my friend!

|======================================================= | 59%

| Let's walk through the order of operations in the above case. First the left and right operands of

| the AND operator are evaluated. 6 is not equal 8, 4 is greater than 3.9, therefore 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 evaluates to TRUE.

...

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

| Which one of the following expressions evaluates to TRUE?

1: 99.99 > 100 || 45 < 7.3 || 4 != 4.0

2: FALSE || TRUE && FALSE

3: TRUE && 62 < 62 && 44 >= 44

4: TRUE && FALSE || 9 >= 4 && 3 < 6

Selection: 4

| All that practice is paying off!

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

| Which one of the following expressions evaluates to FALSE?

1: 6 >= -9 && !(6 > 7) && !(!TRUE)

2: FALSE || TRUE && 6 != 4 || 9 > 4

3: !(8 > 4) || 5 == 5.0 && 7.8 >= 7.79

4: FALSE && 6 >= 6 || 7 >= 8 || 50 <= 49.5

Selection: 4

| That's the answer I was looking for.

|============================================================= | 65%

| 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.

...

|=============================================================== | 67%

| The function isTRUE() takes one argument. If that argument evaluates to TRUE, the function will

| return TRUE. Otherwise, the function will return FALSE. Try using this function by typing: isTRUE(6 >

| 4)

> isTRUE(6 > 4)

[1] TRUE

| Nice work!

|================================================================= | 69%

| Which of the following evaluates to TRUE?

1: isTRUE(3)

2: !isTRUE(8 != 5)

3: isTRUE(NA)

4: !isTRUE(4 < 3)

5: isTRUE(!TRUE)

Selection: 4

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

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

| 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

| You are really on a roll!

|==================================================================== | 73%

| Which of the following evaluates to TRUE?

1: !identical(7, 7)

2: identical('hello', 'Hello')

3: identical(5 > 4, 3 < 3.1)

4: identical(4, 3.1)

Selection: 3

| You are really on a roll!

|====================================================================== | 75%

| 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 evaluates 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

| All that hard work is paying off!

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

| 5 == 6 evaluates to FALSE, !FALSE evaluates to TRUE, so xor(FALSE, TRUE) evaluates to TRUE. On the

| other hand if the first argument was changed to 5 == 5 and the second argument was unchanged then

| both arguments would have been TRUE, so xor(TRUE, TRUE) would have evaluated to FALSE.

...

|========================================================================== | 78%

| Which of the following evaluates to FALSE?

1: xor(identical(xor, 'xor'), 7 == 7.0)

2: xor(!isTRUE(TRUE), 6 > -1)

3: xor(!!TRUE, !!FALSE)

4: xor(4 >= 9, 8 != 8.0)

Selection: 4

| That's a job well done!

|============================================================================ | 80%

| For the next few questions, we're going to need to create a vector of integers called ints. Create

| this vector by typing: ints <- sample(10)

> ints <- sample(10)

| Nice work!

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

| Now simply display the contents of ints.

> ints

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

| Keep up the great work!

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

| The vector `ints` is a random sampling of integers from 1 to 10 without replacement. Let's say we

| wanted to ask some logical questions about contents of ints. If we type ints > 5, we will get a

| logical vector corresponding to whether each element of ints is greater than 5. Try typing: ints > 5

> ints > 5

[1] FALSE FALSE TRUE TRUE TRUE FALSE TRUE TRUE FALSE FALSE

| That's the answer I was looking for.

|================================================================================= | 86%

| 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).

...

|=================================================================================== | 88%

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

> which(ints > 7)

[1] 3 4 8

| That's a job well done!

|===================================================================================== | 90%

| Which of the following commands would produce the indices of the elements in ints that are less than

| or equal to 2?

1: ints < 2

2: ints <= 2

3: which(ints <= 2)

4: which(ints < 2)

Selection: 3

| Your dedication is inspiring!

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

| Like the which() function, the functions any() and all() take logical vectors as their argument. The

| any() function will return TRUE if one or more of the elements in the logical vector is TRUE. The

| all() function will return TRUE if every element in the logical vector is TRUE.

...

|======================================================================================== | 94%

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

> any(ints < 0)

[1] FALSE

| Great job!

|========================================================================================== | 96%

| Use the all() function to see if all of the elements of ints are greater than zero.

> all(ints > 0)

[1] TRUE

| Perseverance, that's the answer.

|============================================================================================ | 98%

| Which of the following evaluates to TRUE?

1: all(c(TRUE, FALSE, TRUE))

2: any(ints == 2.5)

3: all(ints == 10)

4: any(ints == 10)

Selection: 4

| Nice work!

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

| That's all for this introduction to logic in R. If you really want to see what you can do with logic,

| check out the control flow lesson!

...