Logic

Raphael Carvalho

27/05/2019

Logic

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
```

Just like arithmetic, logical expressions can be grouped by parenthesis so that the entire expression (TRUE == TRUE) == TRUE evaluates to TRUE. To test out this property, try evaluating (FALSE == TRUE) == FALSE.

```
(FALSE == TRUE) == FALSE
```

```
## [1] TRUE
```

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

```
6 == 7
```

```
## [1] FALSE
```

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
```

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
```

Which of the following evaluates to FALSE?

```
print("3: 9 >= 10")
```

```
## [1] "3: 9 >= 10"
```

Which of the following evaluates to TRUE?

```
print("1: -6 > -7")
```

```
## [1] "1: -6 > -7"
```

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
```

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.

```
6 < 7
```

```
## [1] TRUE
```

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
```

Which of the following evaluates to FALSE?

```
print("3: !(0 >= -1)")
```

```
## [1] "3: !(0 >= -1)"
```

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

```
print("1: TRUE")
```

```
## [1] "1: TRUE"
```

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
```

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
```

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). 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
```

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.

```
FALSE & FALSE
```

```
## [1] FALSE
```

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. 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.

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
```

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
```

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. 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
```

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.

Which one of the following expressions evaluates to TRUE?

```
print("3: TRUE && FALSE || 9 >= 4 && 3 < 6")
```

```
## [1] "3: TRUE && FALSE || 9 >= 4 && 3 < 6"
```

Which one of the following expressions evaluates to FALSE?

```
print("2: FALSE && 6 >= 6 || 7 >= 8 || 50 <= 49.5")
```

```
## [1] "2: FALSE && 6 >= 6 || 7 >= 8 || 50 <= 49.5"
```

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
```

Which of the following evaluates to TRUE?

```
print("1: !isTRUE(4 < 3)")
```

```
## [1] "1: !isTRUE(4 < 3)"
```

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
```

Which of the following evaluates to TRUE?

```
print("2: identical(5 > 4, 3 < 3.1)")
```

```
## [1] "2: identical(5 > 4, 3 < 3.1)"
```

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)

```
print("xor(5 == 6, !FALSE)")
```

```
## [1] "xor(5 == 6, !FALSE)"
```

Which of the following evaluates to FALSE?

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

```
## [1] "4: xor(4 >= 9, 8 != 8.0)"
```

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)
```

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]  TRUE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE FALSE FALSE
```

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).

```
which(ints > 7)
```

```
## [1] 1 5 7
```

Which of the following commands would produce the indices of the elements in ints that are less than or equal to 2?

```
4: which(ints <= 2)
```

```
## Warning in 4:which(ints <= 2): numerical expression has 2 elements: only
## the first used
```

```
## [1] 4 3 2
```

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.

```
any(ints < 0)
```

```
## [1] FALSE
```

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

```
all(ints > 0)
```

```
## [1] TRUE
```

Which of the following evaluates to TRUE?

```r
print("4: any(ints == 10)")
```

```
## [1] "4: any(ints == 10)"
```