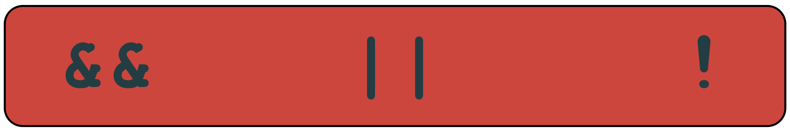
Logical Operators

In the following lesson, you will be introduced to logical operators.

We'll cover the following

- Types of Logical Operators
- Follow the Rules
- Short-Circuit Evaluation



Types of Logical Operators

Logical operators are operators that perform logic operations such as the Logical *AND* and Logical *OR*. They take `Boolean` type operands and yield `Boolean` type results. Below is a list of the logical operators supported by Scala.

Operator	Name	Use
`&&`	Logical AND	If both the operands are not false then the result is `true`
`||`	Logical OR	If any of the two operands is not false then the result is `true`
`!`	Logical NOT	Reverses the logical state of its operand. If a condition is `true` then the Logical *NOT*

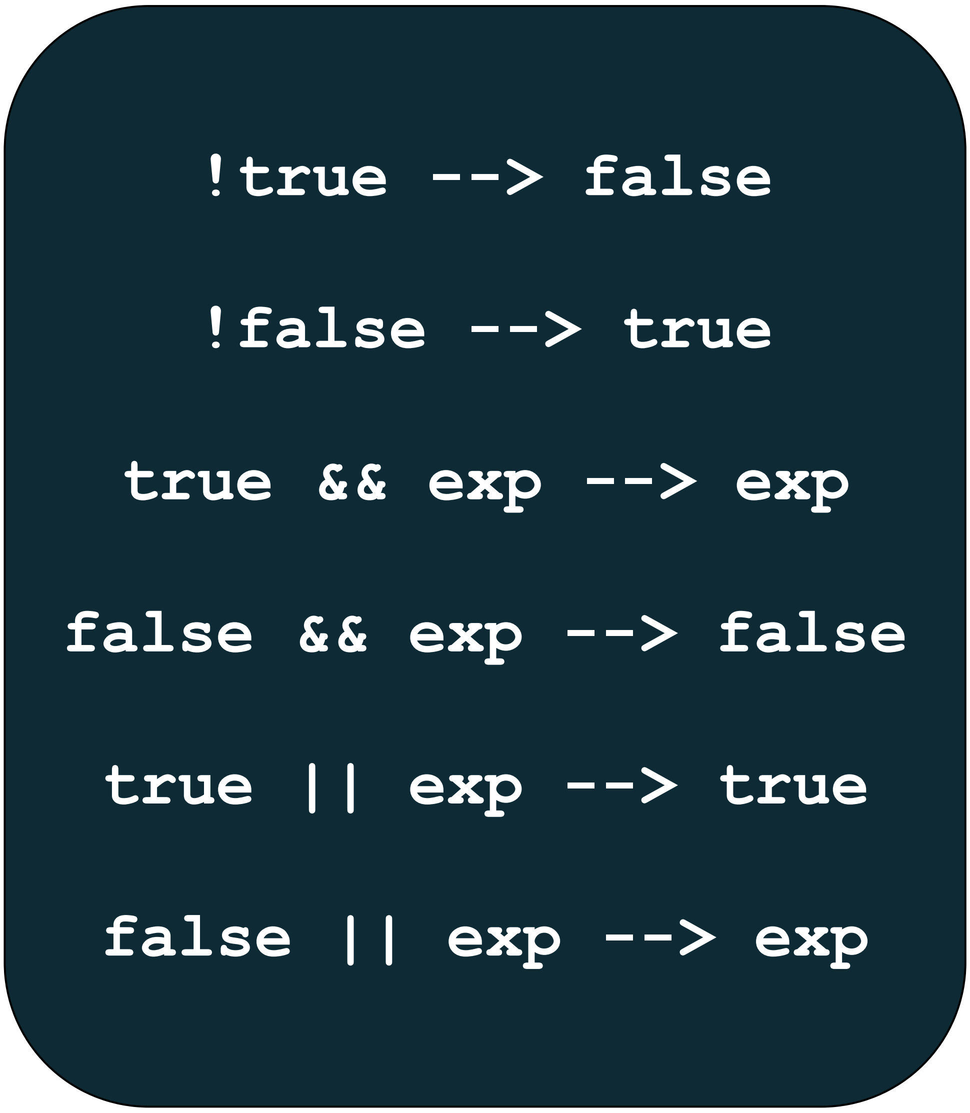
		operator will make it `false`

> `!` is a unary operator, i.e. it takes one operand.

Follow the Rules

Below, you'll find a list of the reduction rules for logical operators. The list is handy as it will summarize how each operator reduces expressions into their final form.

exp is an arbitrary expression that can be replaced with an operand of type `Boolean`. The operand can be `true` or `false` itself or can be an expression that reduces to `true` or `false.`



```
!true --> false

!false --> true

true && exp --> exp

false && exp --> false

true || exp --> true

false || exp --> exp
```

Let's now see these rules in action. For example, our arbitrary expression `exp` will be `A && B` where `A` is `true` and `B` is `false`.

> Try to figure out what the output would be before pressing **RUN**.

This code requires the following environment variables to execute:

LANG	C.UTF-8

```
val A = true
val B = false
val exp = A && B //false

println(!A)
println(!B)
println(true && exp)
println(false && exp)
println(true || exp)
println(false || exp)
```

`A && B` reduces to `false` as `B` is `false` and from our list of rules, we know that `false && exp --> false`

Short-Circuit Evaluation

Note that `&&` and `||` do not always need their right operand to be evaluated. For instance, when using `&&`, if our first operand is `false`, our final result will always be `false` irrespective of the second operand. Expressions which are built with `&&` and `||` are only evaluated as far as needed to determine the result. This is why we say these expressions use **short-circuit evaluation**.

Hence, when the compiler sees `false &&.....`, it will return `false` without evaluating what comes after `&&`. This reduces unnecessary compile time.

That sums up logical operators. Let’s move on to bitwise operators in the next lesson.