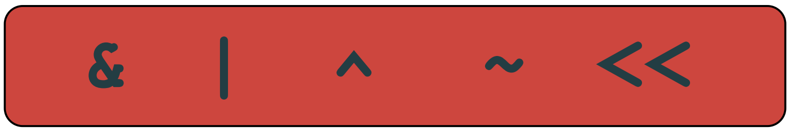
Bitwise Operators

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

We'll cover the following



Types of Bitwise Operators

Bitwise operators are operators that perform operations on individual bits of integer types. Below is a list of the bitwise operators supported by Scala.

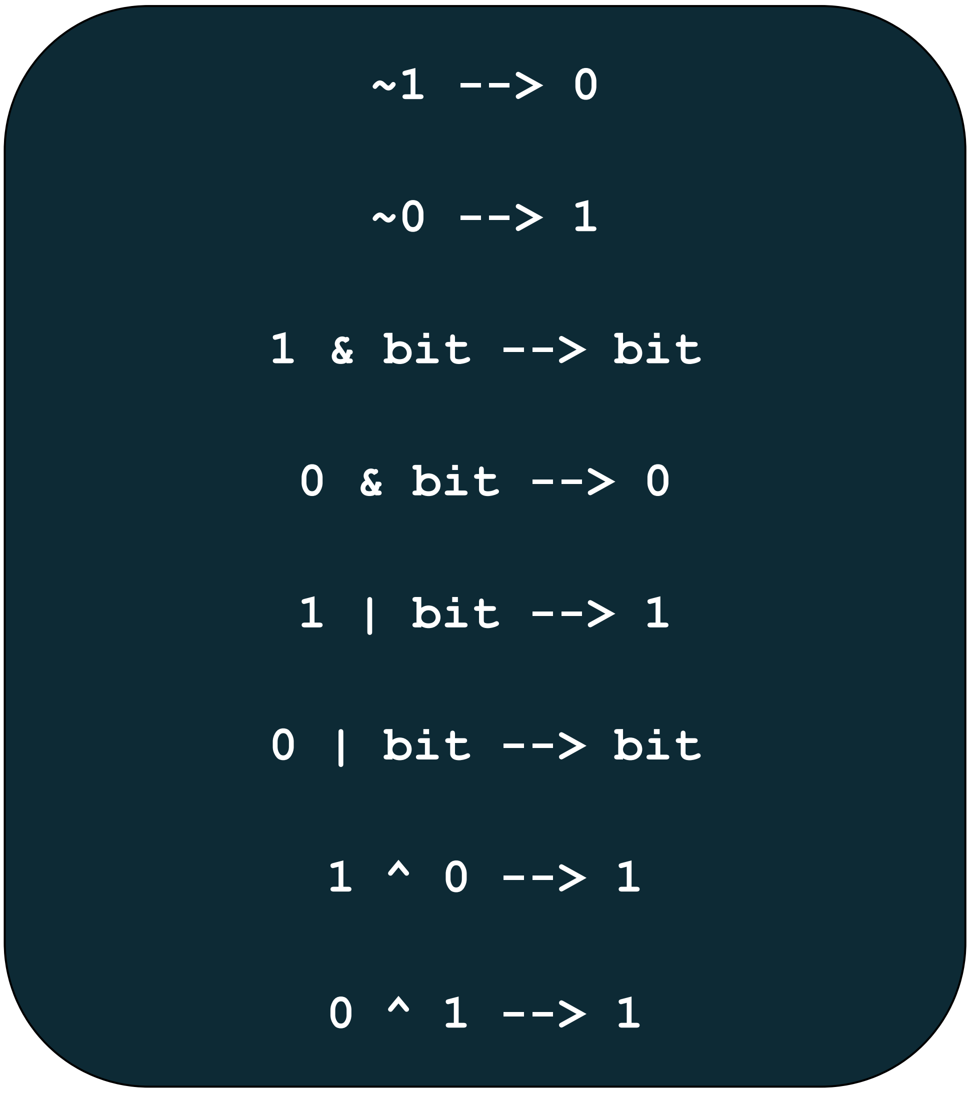
Operator	Name	Use
`&`	Bitwise AND	If the corresponding bit in both operands is 1 it will give a 1, else 0
`|`	Bitwise OR	If the corresponding bit in at least one operand is 1 it will give a 1 else 0
`^`	Bitwise XOR	If the corresponding bit in only one operand is 1 it will give a 1 else 0

`~`	Bitwise Ones Complement	Copies a bit to the result after reversing it

> The above operators work on binary numbers. They automatically convert each decimal number into its binary form, perform the specific operation on that number, convert the new binary number into decimal form, and return the result.

Follow the Rules

Below we have a list of rules that each operator follows. For bitwise operators, we work with binary numbers. Hence, instead of `false` and `true`, we will be using `1` and `0` where `1` acts as true and `0` acts as false. `bit` can be either `1` or `0`.



```
~1 --> 0

~0 --> 1

1 & bit --> bit

0 & bit --> 0

1 | bit --> 1

0 | bit --> bit

1 ^ 0 --> 1

0 ^ 1 --> 1
```

Let’s now see these rules in action. We will take the first operand `A` to be **12** and second operand `B` to be **5**.

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 = 12
val B = 5

println(~A)
println(~B)
println(A & B)
println(A | B)
println(A ^ B)
```

Understanding the Code

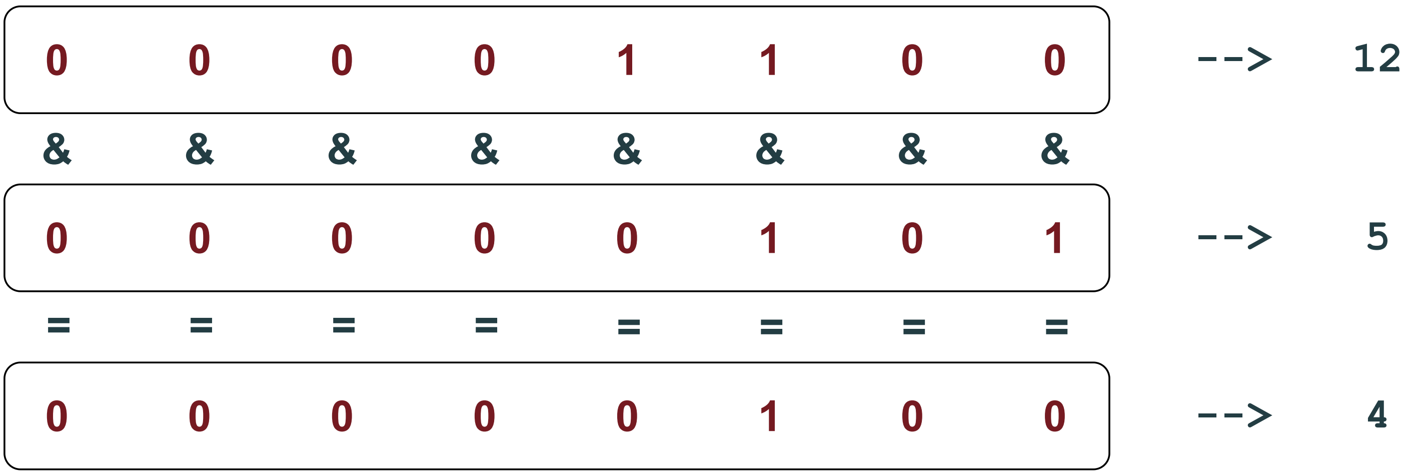
The output of the code might not be as clear as the outputs we've been looking at throughout this chapter. Let's break down what the operators are doing one step at a time.

`A & B`

The operator will first convert each decimal operand into its binary form.

- **12** in binary form is 0000 1100
- **5** in binary form is 0000 0101

From there, it will apply the above rules for the binary AND operator (`&`) on each pair of bits, i.e. the first bit of **12** with the first bit of **5** and so on.



And that is how `A & B` will yield a result of `4`.

Try to map the rules onto the other operations and see if you get the same result as the output above.

That sums up bitwise operators. Let’s move on to our final type of operator, assignment operator, in the next lesson.