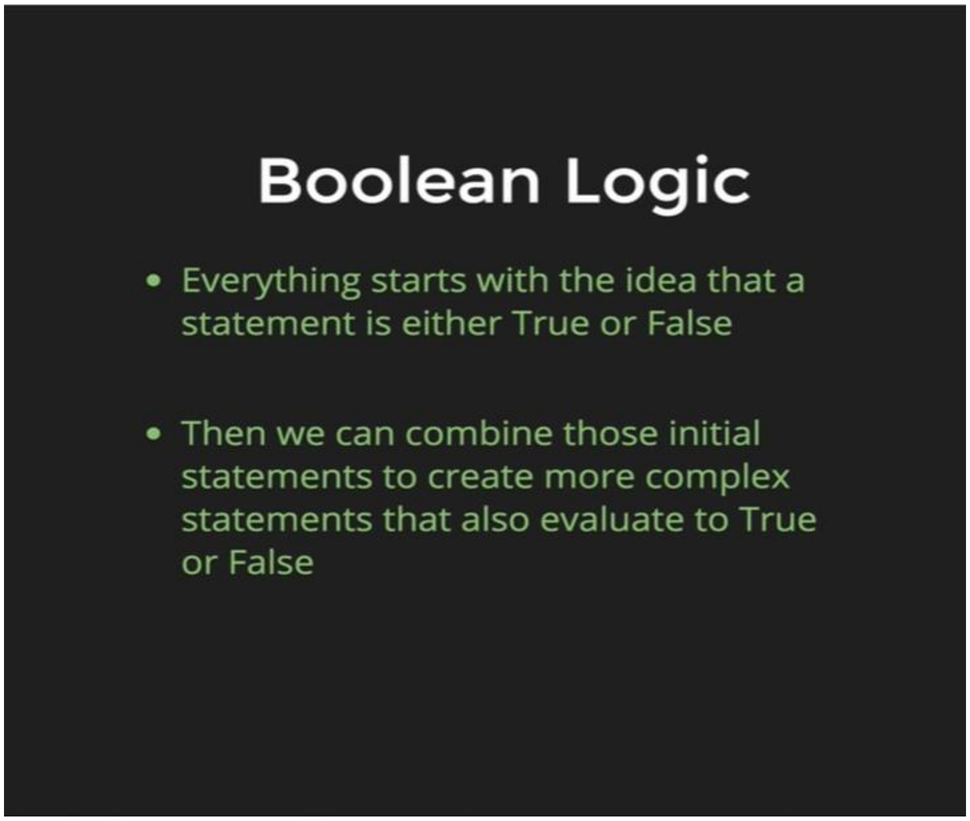
Booleans are very simple, there are two possible options for a Boolean value, true or false.

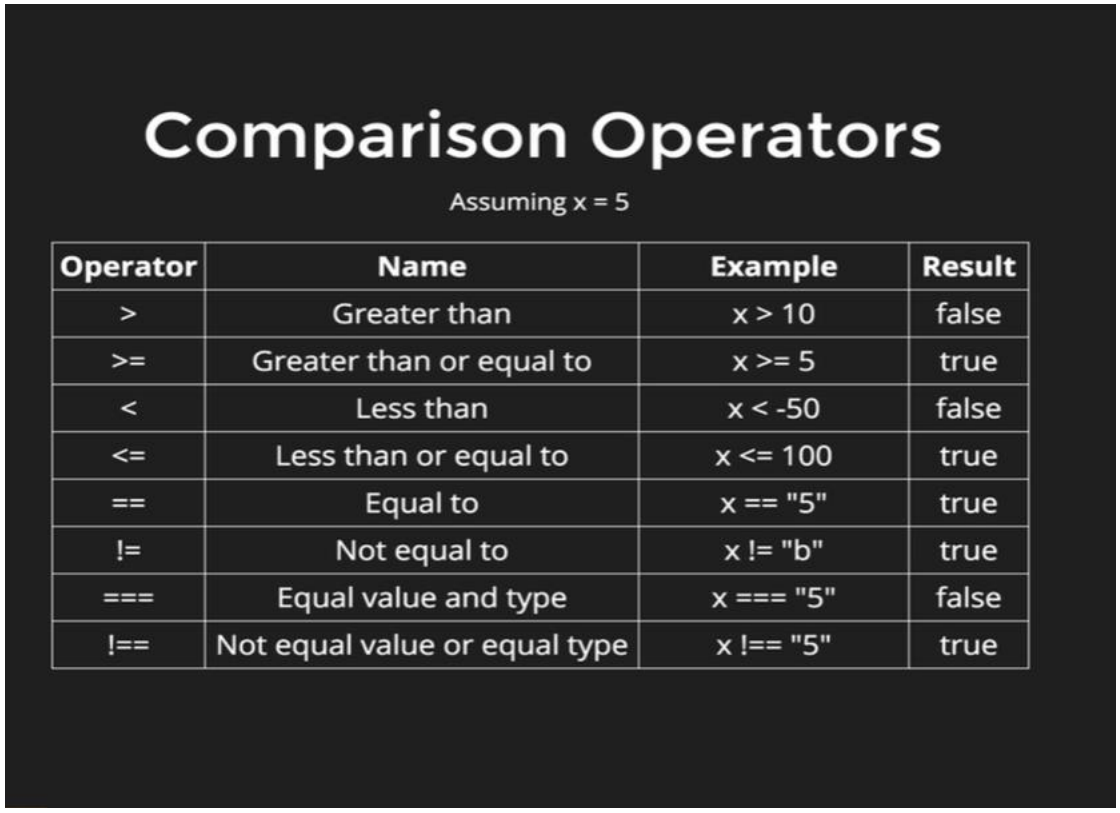


Boolean Logic

- Everything starts with the idea that a statement is either True or False
- Then we can combine those initial statements to create more complex statements that also evaluate to True or False

Boolean logic is simply writing statements that evaluate to be true or false. Eventually we will be writing these statements if something is true and some other parts of our statements if something is false. So, these statements can be things like, a user is logged in, that is true or false. But it can also be something as simple as if subtotal is greater than $10000 and those evaluate to be true or false and we do something about them in our code.

The first thing we must start is with comparison operators. Comparison operators are things like most of us use in our day to day lives. Its how we compare two values to one another, its how JavaScript does the same thing. So, the chart below goes through all the comparison operators.



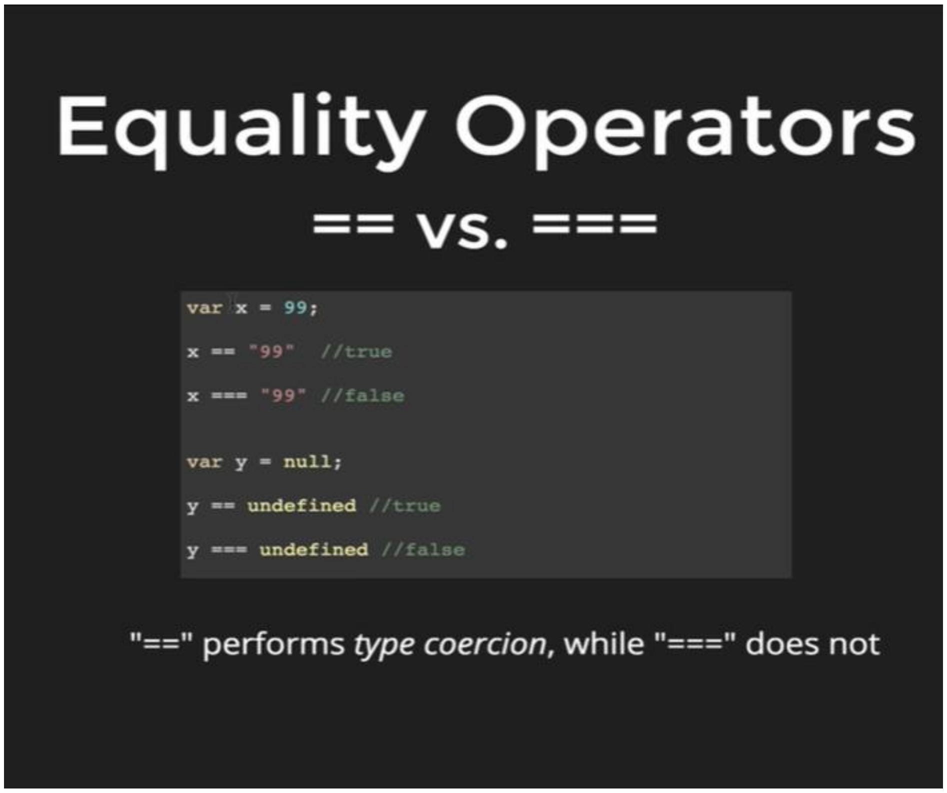
Comparison Operators

Assuming x = 5

Operator	Name	Example	Result
>	Greater than	x > 10	false
>=	Greater than or equal to	x >= 5	true
<	Less than	x < -50	false
<=	Less than or equal to	x <= 100	true
==	Equal to	x == "5"	true
!=	Not equal to	x != "b"	true
===	Equal value and type	x === "5"	false
!==	Not equal value or equal type	x !== "5"	true

It assumes that x = 5. The last four entries in the chart is a little bit different than the above four. They cover equality and non-equality. So, its how we check if x is exactly equal to the number 5, or x is equal to the string "5" or x is anything but the number 20, its how we basically check for equality. One thing we can notice that there are two ways to check for equality, so there's one with two equal signs (==), and one with three equal signs (===). Likewise, there are two ways to check for non-equality one with one equal sign and an exclamation sign (!=) and another with two equal signs and one exclamation sign (!==).

So, they both go together in pairs, x == "5" is true but x === "5" is false. The reason for that is *type coercion* so when we use double equals (==) it performs type coercion and what that means is it basically takes two numbers or two strings or two variables whatever they are and it tries to turn them into similar type so that it can compare them.



Equality Operators

== vs. ===

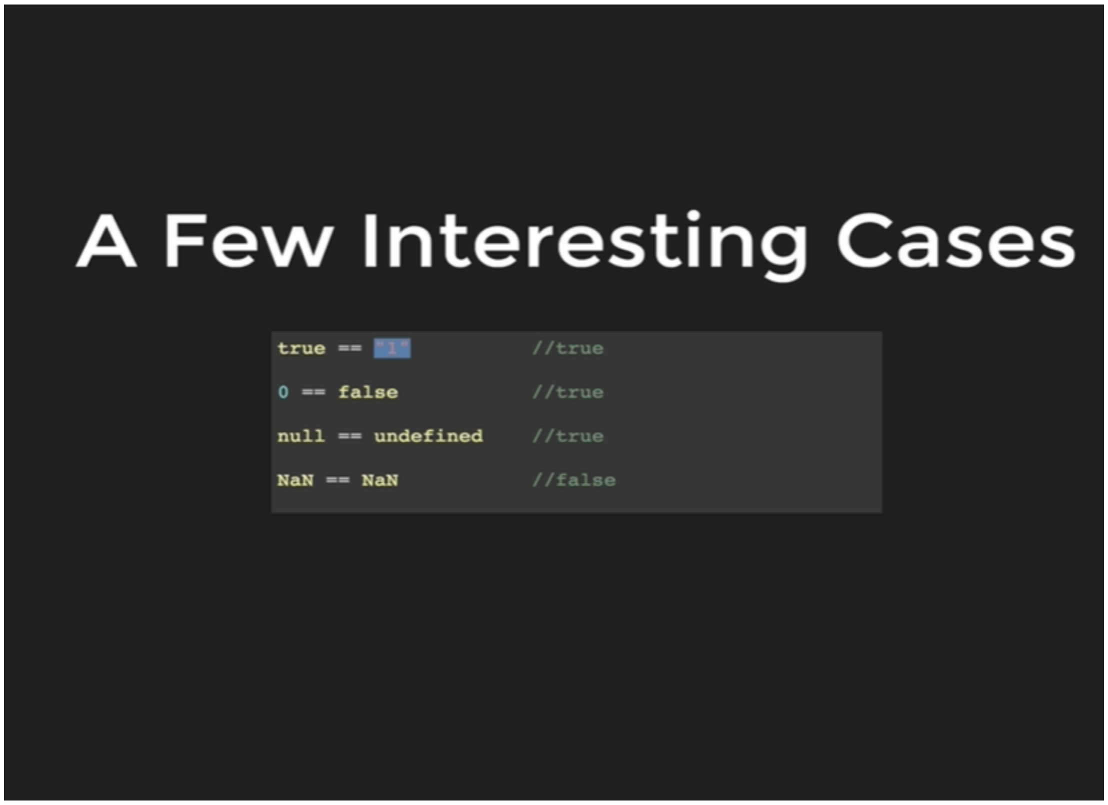
```
var x = 99;
x == "99"  //true
x === "99" //false

var y = null;
y == undefined //true
y === undefined //false
```

"==" performs *type coercion*, while "===" does not

In this example x is a number, 99, and when we double equal (==) with string "99" that is true because JavaScript performs type coercion, it tries to get x and "99" into same format, and then compare between them, triple equals (===) does not perform type coercion and it defines x as a number and "99" as a string which is not equal as triple equal (===), and that is why it ends up as false.

That is why as rule of thumb we should always use triple equals (===), it is much safer; it is much more specific.

And as another example we can see y = null. Null double equals undefined is true, even though they are very different values they are not the same thing, double equals (==) considers them to be true and triple equals (===) consider them to be false.



A Few Interesting Cases

```
true == "1"          //true
0 == false           //true
null == undefined    //true
NaN == NaN           //false
```