Booleans

Everything is True or False

We'll cover the following

Booleans

At its most basic level, a computer is a series of on and off switches, a set of 0s and 1s that flip back and forth to, well, compute things.

The concept of using *binary values* to represent information (0 and 1, true and false) is so fundamental to computation that Javascript, along with most other programming languages, has a dedicated type to these values, referred to as a **boolean**.

Creating a boolean value

Boolean values can be created by assigning a value of `true` or `false` to a variable.

```
var theTruth = true;
var aLie = false;
```

Booleans from comparison operators

Boolean values can also be created by using **comparison operators**, such as:

- Greater Than - `>`

- Less Than - `<`
- Greater Than *or* Equal To - `>=`
- Less Than *or* Equal To - `<=`

```
console.log(101 > 100);
console.log(101 < 100);
console.log(10 >= 10);
console.log(20 <= 10);
```

Equality operators

To check whether or not one value is equal to another value, Javascript provides four different operators:

- Equal To:
  1. `==`
  2. `===`
- **Not** Equal To: 3. `!=` 4. `!==`

Comparing two values with an equality operator will return a **Boolean value**.

You may be wondering: Why there are two *different* operators for each comparison type? Let's explore the differences between these operators.

`===` and `!==`

The `===` and `!==` operators check that both values being compared are equal to (or not equal to) *the same type* and *the same value*.

```
var skyColor = "blue";
var carColor = "blue";
var hairColor = "black";

console.log(skyColor === carColor);
console.log(skyColor === hairColor);
console.log(skyColor !== hairColor);
```

Equality operators being used to compare strings.

With the `===` and `!==` operators, if the two values are two *different types*, the `===` operation will always return a value of `false` and the `!==` operation will always return a value of `true`.

```
var ageOfBill = 10;
var ageOfSally = "10";

console.log(ageOfBill === ageOfSally);
console.log(ageOfBill !== ageOfSally);
```

=== and !== being used to compare values of different types

`==` and `!=`

The `==` and `!=` operators are less strict about values being of *the same type*. Let's take a look at the age comparison again, this time using the `==` and `!=` instead:

```
var ageOfBill = 10;
var ageOfSally = "10";

console.log(ageOfBill == ageOfSally);
console.log(ageOfBill != ageOfSally);
```

== and != being used to compare values of different types. Why do you think the values returned are different?

What changed here? When we compare `ageOfBill` (a **number**) to `ageOfSally` (a **string**), Javascript tells us that the values are indeed equal to one another.

When you compare two values of different types using the `==` or `!=` operator, Javascript will attempt to **convert the type** of one of the values to make a valid comparison.

In this case, the *string* `ageOfSally` is **converted** to a *number*, and then the comparison is made, which is why the `==` comparison returns `true` and the `!=` comparison returns `false`.

If this seems confusing, don't worry too much about it. In the Javascript community, type conversion is a hotly debated topic and many feel it is a flaw in

the language’s design.

Which equality operators should be used?

For this course, we will focus on using `===` and `!==` operators, since the **intent** of these operations is more clear. This practice will guard against hidden errors that may pop up from Javascript automatically converting values from one type to another.

Check your Understanding

1 Given the following expression:

```
40 > 30
```

What value will be returned?

2 Given the following expression:

```
300 === "300"
```

What value will be returned?

3

Given the following expression:

```
31 <= 30
```

What value will be returned?

4

Given the following expression:

```
300 == "300"
```

What value will be returned?

Retake Quiz

Now that you have learned about booleans in javascript, let’s learn about null and undefined values in the next lesson.