Boolean Variables

The next data type that I'd like to introduce you to is called **booleans**, named after the philosopher, logician, and mathematician, George Boole. Boolean values can hold one of two values, `true` or `false`.

```
var one = 1;
var two = 2;
```

These two variables are not equal to each other. How could I determine that programmatically? In JavaScript you can use comparison operators to see the relationship between two variables, and the result of that expression is a boolean value:

```
var same = (value1 == value2);
console.log(same);
```

The value of `same` is `false`.

That double equals sign is one of the comparison operators that you can use in JavaScript to compare two values and turn out a boolean value.

Coercion and Boolean Comparison

It's important to know the differences among the various comparison operators though, because just saying that two things are the same doesn't necessarily mean that they're actually the same in JavaScript.

For example, comparing with the two equal signs forces the two data types of two variables to match. So if one is a string, and one is a number, they are going to be converted to the same data type before the comparison happens. If you want to preserve the data types between two variables before doing the comparison, use three equal signs. And what you're going to find is frequently when you're programming, you're going to use the three equal signs comparison instead of the two equal sign comparison, in order to be sure that what you're comparing actually is the same.

```
var value1 = 1;
var value2 = "1";
var same = (value1 == value2);
var exact = (value1 === value2);

console.log(same);
console.log(exact);
```

Boolean Comparison Operators

You can do more with JavaScript in terms of comparing variables then just seeing whether they're equal to the same value.

You can also use a greater than to see whether the first variable is actually larger than the second variable.

You can use greater than or equal to, which sees whether or not the first variable is larger than or the same as the second variable.

You can use less than, which shows whetherthe first variable was less than or a match for the second variable.

And then you can also use the exclamation mark equal sign, which shows whether or not the two variables do not equal each other.

Examples of Boolean Comparison Operators

So here is an example:

```
var number1 = 1;
var number2 = 2;
var same = (number1 == number2);
var exact = (number1 === number2);
var greater = (number1 > number2);
var less = (number1 < number2);
var unequal = (number1 != number2);

console.log(same);
console.log(exact);
console.log(greater);
console.log(less);
console.log(unequal);
```

You can use these operators when you're testing values in order to determine what you want your program to do next. It's important to know which values come back as `true` according to JavaScript when you test them.

True Values

JavaScript is going to treat any real value that is not empty or equal to zero as if it were `true`. So the boolean value of a string that has content in it, or any number other than zero, is going to return `true`, and it's important to know this.

JavaScript includes a method called `Boolean` that let's you test the boolean value of any variable.

```
console.log(Boolean(1));
console.log(Boolean("Word"));
console.log(Boolean(-1000));
console.log(Boolean(2 + 2));
```

`true` will be returned for all cases.

Knowing what JavaScript considers `true` can prevent you from making mistakes when you're evaluating situations, and trying to determine what you want to do next.

False Values

JavaScript will return `false` for zero, for empty strings, for undefined variables, and expressions that return `NaN` values.

```
 var valueless;
console.log(valueless);
console.log(Boolean(valueless));

var notNumber = 2.0 * "Hello";
console.log(notNumber);
console.log(Boolean(notNumber));
console.log(Boolean(0));
console.log(Boolean(""));
```

You'll see that all of these values are considered `false` by JavaScript.

Not all programming languages would consider zero, or an empty string, as `false` values. So, if you're coming from a different programming language to JavaScript, it's important to know that that's the situation here.