Introduction to Operators

Welcome to lesson three! In this lesson we will discuss **operators**.

Operators Make Variables Do Things

Operators are used to make variables do things in JavaScript. Operators are the symbols that show what's going to happen to each variable or value in the statement.

```
var smallNumber = 2;
var bigNumber = 2000;

console.log(smallNumber + bigNumber);
// 2002

console.log(smallNumber < bigNumber);
// true
```

`=` is the assignment operator. It is telling the variable to set itself to the value onthe other side of the operator.

`=` is another operator telling JavaScript to add `smallNumber` and `bigNumber` together.

`>` creates a boolean out of the comparison between `smallNumber` and `bigNumber`.

Working with operators may seem so intuitive that you might have just skipped over thinking about them. But it's important to be conscious of what an operator is and how it works in order to be effective in using them in JavaScript.

Operators Respond to Types

One of the reasons you need to know about operators, is because some of them behave differently depending on the type of the variable that you're using them with. The same operator can do different things to variables and values of different types, and if you're not expecting it, it can be a bit of a surprise.

```
var firstWord = "hello";
var secondWord = "world";

console.log(firstWord + " " + secondWord);
// "hello world"
```

You'll get a new string as the result of that operation. In this case, the `+` operator that we used for mathematical addition with numbers, is being used for concatenation with strings.

Knowing which operators you can use with which variable types, and how they're going to affect those variable types is going to be helpful in avoiding problems when you're working with JavaScript.

JavaScript is Loosely Typed

The reason that it's important to keep track of this is that JavaScript is **loosely typed**. What that means is that variables as we've seen are declared without a type. Variables can hold data of any type, and a single variable can actually hold different data types at different times. That's why we have to pay attention to what our variable types are as we perform operations on them.

The operation we perform may do something different, depending on the type of variable. Type coercion can actually change the type of a variable. So the results can be surprising if you don't know what the rules are and what to expect.

```
var letter = "A";
var number = 1;

console.log(number + number);
// 2

console.log(letter + letter);
// "AA"

console.log(number + letter);
// "1A"

console.log(letter + number);
// "A1"

console.log(letter + letter + number);
// "AA1"

console.log(number + number + letter);
// "2A"
```

Note what happens if we try to use `+` operator to combine a string and a number: we get the letter and the number represented together as a string.

Remember, when we performed mathematical operations on strings before, we got JavaScript returning `NaN` as a result. In this case, JavaScript is coercing our number into a string and then performing the appropriate string concatenation method.

Also note the last and the most trickiest example. Surprisingly, we get `2A`. This is because of the order in which JavaScript forms the concatenation and addition operations. The first thing JavaScript does is add together the two numbers and then it concatenates the result of adding those two numbers together, as a string with letter.