More on Operators

In the previous lecture, we explored key operators that you'll use all the time when doing math with JavaScript (which, as it turns out, is also something you'll do all the time ;-)).

- `+`: Add two numbers OR concatenate two strings
- `-`: Subtract two numbers
- `*`: Multiple two numbers
- `/`: Divide two numbers
- `%`: Derive the remainder of a division

There also are **useful shorthand operators** that we haven't explored in the video but which will also be shown later in the course:

- `++` (e.g. `age++`): Shorthand notation for `age = age + 1` (increment a value stored in a variable by 1 and store it back into that variable)
- `--` (e.g. `age--`): Shorthand notation for `age = age - 1` (decrement a value stored in a variable by 1 and store it back into that variable)
- `**` (e.g. `age = 4 ** 3`): Exponentiation operator (i.e. replacement for `age = 4 * 4 * 4`)
- `+=` (e.g. `age += 2`): Shorthand notation for `age = age + 2` (increase a value stored in a variable and store it back into that variable)
- `-=` (e.g. `age -= 2`): Shorthand notation for `age = age - 2` (decrease a value stored in a variable and store it back into that variable)
- `*=` (e.g. `age *= 2`): Shorthand notation for `age = age * 2` (multiply a value stored in a variable and store it back into that variable)
- `/=` (e.g. `age /= 2`): Shorthand notation for `age = age / 2` (divide a value stored in a variable and store it back into that variable)

More on Value Types

It is also important to understand and keep in mind, that all values have certain value types. E.g. `2` is a *number* whereas `'hi'` or `'2'` are *strings* (note the `'`!).

Especially the `'2'` case is important: It looks like a number to us humans - and of course it technically is a number. But it's stored as text, not as a "raw number".

Why does that matter?

It matters once you start performing operations with the operators mentioned above. Here are some example code snippets and the results that would be produced by them *(you can test them all in the JavaScript console in the developer tools of your browser)*:

```
let a = 'hi' + ' there'; // 'hi there' => a string
let b = 'the number' + ' 2'; // 'the number 2' => a string
let c = 'the number' + 2; // 'the number2' => a string
let d = 2 + 2; // 4 => a number
let e = 2 + '2'; // '22' => a string! ('2' and '2' concatenated)
let f = '2' + '2'; // '22' => a string! ('2' and '2' concatenated)
let g = '2' * 3; // 6 => a number
```

If you go through the above examples, there might be some code lines that you maybe didn't expect to produce the results they do produce.

Especially `e`, `f` and `g` can be confusing.

But it's less confusing if you keep in mind, that values do have certain value types. If you wrap something with quotes (double or single quotes, doesn't matter), it's a string. Even if it looks like a number to us.

And if you use the `+` operator on a string (even if the other value is a number), JavaScript will do what it always does when working with `+` on strings: It creates a new string by concatenating the values. That's why `e` and `f` yield `'22'` as results.

`g` might then again be confusing, as the result now suddenly is a number and not a string but the reason for that is simple: JavaScript doesn't know what to do with `*`, `/` or `-` on strings. But instead of failing immediately, it tries to convert the string to a number behind the scenes and use that converted number in the operation instead. That's why we get the number `6` as a result.

If it would fail to convert the string to a number (e.g. because you tried `'hi' * 3`), you would get a special result: `NaN` which stands for "*Not a Number*". A special value that exists in JavaScript for cases like this.

There are other value types as well (objects, arrays, functions and more) but numbers and strings are the most important ones right now and we'll explore more values and value types throughout the course!