Basic Operators

Introduction to Basic Operators

Let's now learn some more JavaScript fundamentals. In this video, we will focus on basic operators. To begin, let's define what an operator actually is. An operator allows us to transform values or combine multiple values and perform various operations on them. There are many categories of operators, such as mathematical operators, comparison operators, logical operators, assignment operators, and more.

Mathematical or Arithmetic Operators

We have already used the plus and minus operators, but of course, we can perform all arithmetic operations. These include multiplication, division, and others. Let's use the minus operator to calculate ages based on a person's birth year.

javascript Code Sample

```
const ageJonas = 2037 - 1991;

console.log(ageJonas);
```

Here, we assume the current year is 2037, which is far in the future, but it makes this example fun. We subtract the birth year 1991 from 2037 to calculate Jonas's age. Logging this to the console shows that Jonas's age is 46.

javascript Code Sample

```
const ageSarah = 2037 - 2018;

console.log(ageSarah);
```

Similarly, we calculate Sarah's age by subtracting her birth year 2018 from 2037. Logging this shows her age as 19.

We can also log multiple values at the same time by separating them with commas in the console.log function.

javascript Code Sample

```
console.log(ageJonas, ageSarah);
```

This will output both ages together, 46 and 19 years, respectively.

Using Variables to Avoid Repetition

Currently, the year 2037 is repeated in the code, which is not ideal. If the year changes, we would need to update it in multiple places. To avoid this, we use variables to store such values.

javascript Code Sample

```
const now = 2037;

const ageJonas = now - 1991;

const ageSarah = now - 2018;

console.log(ageJonas, ageSarah);
```

By assigning 2037 to the variable **now**, we only need to update the year in one place if it changes. This is a good use case for variables. Note that **const** is used here because these values do not change.

More Arithmetic Operations

We can perform other mathematical operations such as multiplication, division, and exponentiation.

javascript Code Sample

```
console.log(ageJonas * 2, ageJonas / 2, 2 ** 3);
```

This logs Jonas's age times two, divided by two, and two to the power of three (which is 2 multiplied by itself three times, equal to 8).

String Concatenation with Plus Operator

The plus operator can also join strings, which is called concatenation.

javascript Code Sample

```
const firstName = 'Jonas';

const lastName = 'Schmedtmann';

console.log(firstName + lastName);
```

This concatenates the two strings without space. To add a space between them, concatenate a space string as well.

javascript Code Sample

```
console.log(firstName + ' ' + lastName);
```

This outputs "Jonas Schmedtmann" with a space between the first and last names.

The typeof Operator

The **typeof** operator returns the type of a value. We have used it before to check the type of variables.

Assignment Operators

The simplest assignment operator is the equal sign **=**. For example:

javascript Code Sample

```
let x = 10 + 5;

console.log(x);
```

Here, the plus operator is executed before the assignment operator, so **x** is assigned the value 15.

Compound Assignment Operators

There are shorthand assignment operators such as **+=** which add a value to a variable and assign the result back to it.

javascript Code Sample

```
x += 10; // equivalent to x = x + 10

console.log(x);
```

After this operation, **x** becomes 25 because it was 15 before and we added 10.

Similarly, there are other compound assignment operators like ***=** and **/=**.

javascript Code Sample

```
x *= 4; // equivalent to x = x * 4

console.log(x);
```

Now, **x** is 100 because 25 times 4 equals 100.

Increment and Decrement Operators

The increment operator **++** increases a variable by one, and the decrement operator **--** decreases it by one.

javascript Code Sample

```
x++;

console.log(x);

x--;

x--;

console.log(x);
```

After incrementing, **x** becomes 101. After decrementing twice, it becomes 99.

Comparison Operators

Comparison operators produce Boolean values (**true** or **false**). For example, to check if Jonas's age is greater than Sarah's:

javascript Code Sample

```
console.log(ageJonas > ageSarah);
```

This outputs **true** because 46 is greater than 19. Comparison operators include greater than (**>**), less than (**<**), greater than or equal (**>=**), and less than or equal (**<=**).

Checking Full Age

To check if Sarah is of full age (at least 18), we can use the greater than or equal operator:

javascript Code Sample

```
const isFullAge = ageSarah >= 18;

console.log(isFullAge);
```

This will output **true** if Sarah is 18 or older, and **false** otherwise. Storing the result in a variable like **isFullAge** is useful for later use in the code.

Combining Calculations and Comparisons

We can perform calculations and comparisons in one expression without intermediate variables. For example:

javascript Code Sample

```
const isFullAge = (now - 1991) >= (now - 2018);

console.log(isFullAge);
```

Here, the subtraction operations are performed first, then the comparison. JavaScript uses operator precedence rules to determine the order of operations, ensuring correct evaluation.

Key Takeaways

- Operators in JavaScript allow transformation and combination of values, including mathematical, comparison, logical, and assignment operators.
- Arithmetic operators include addition, subtraction, multiplication, division, and exponentiation.
- Variables help avoid repetition and make code more maintainable by storing values like the current year.
- The plus operator can concatenate strings, and spaces can be added by concatenating a space string.
- Assignment operators like **+=**, ***=**, and **-=** provide shorthand for updating variable values.
- Comparison operators produce Boolean values useful for decision-making, such as greater than, less than, and equality checks.
- Operator precedence determines the order in which operations are performed, ensuring correct evaluation of expressions.