Type Conversion and Coercion

Introduction to Type Conversion and Coercion

In this video, we need to go back to value types. Types are one of the fundamental aspects in programming, and converting between types is something that we do in every programming language. For example, converting a string to a number or a number into a Boolean is something that we do all the time. It is important that we learn about this before being able to move on further in the course. This is especially true for a language like JavaScript, which sometimes behaves in a weird way, as we will see in this video.

Type Conversion vs. Type Coercion

In JavaScript, there is type conversion and type coercion. They sound very similar but are different. Type conversion is when we manually convert from one type to another. On the other hand, type coercion is when JavaScript automatically converts types behind the scenes for us. That is necessary in some situations but it happens implicitly, completely hidden from us.

Manual Type Conversion

Let us start with type conversion, which, remember, is when we explicitly want to convert from one type to another. Suppose we have an input field on a web page for the user to input their birth year. These inputs from input fields usually come as strings. Let us say that the **inputYear** that we get from the user interface is a string with the value 1991. If we want to do some calculations with this, this will not really work. For example, adding 18 to this string will result in string concatenation, not arithmetic addition.

javascript Code Sample

```
console.log(inputYear + 18)
```

So, we need a way of fixing this, which means that we need a way of converting this string to a number. The way we convert this string to a number is by using the built-in **Number** function.

javascript Code Sample

```
console.log(Number(inputYear))
```

Doing this operation will return the string as a number. The original value is not converted;
the **inputYear** variable itself is still a string. Using the **Number** function will simply give us a converted version. If you want to perform a calculation, you need to use **Number** here as well.

javascript Code Sample

```
console.log(Number(inputYear) + 18)
```

Now, what if we try to convert something to a number that is impossible to convert? For example, converting the string 'Jonas' to a number. JavaScript will try to convert it but it will not work. Instead, we get **NaN**, which stands for Not a Number. JavaScript gives us this value whenever an operation that involves numbers fails to produce a new number. Not a Number actually means invalid number.

javascript Code Sample

```
console.log(Number('Jonas'))
```

We can check the type of **NaN**.

javascript Code Sample

```
console.log(typeof NaN)
```

The type of Not a Number is actually **number**, so Not a Number means an invalid number. We get Not a Number whenever an operation involving numbers fails to give us a new number.

Converting Numbers to Strings

Of course, we can also do the opposite: convert numbers to strings. To do it the other way around, we use the **String** function. Remember to start it with a capital S, just like the **Number** function needs to start with a capital N.

javascript Code Sample

```
console.log(String(23))
```

JavaScript can only convert to three types: number, string, or Boolean. We cannot, for example, convert something to undefined or to null. Here, we only converted to numbers and to strings but not to Booleans. Booleans behave in a special way, and there is a separate lecture on truthy and falsy values.

Automatic Type Coercion

In practice, we rarely have to do type conversion manually because JavaScript does type coercion automatically for us in many situations. Type coercion happens whenever an operator is dealing with two values that have different types. JavaScript will then, behind the scenes, convert one of the values to match the other so that the operation can be executed.

javascript Code Sample

```
// Type conversion

// Type coercion
```

For example, when using the plus operator with a string and a number, the number will be converted to a string. This is why the following produces a string:

javascript Code Sample

```
console.log('I am ' + 23 + ' years old')
```

The plus operator triggers coercion to strings. Whenever there is an operation between a string and a number, the number will be converted to a string. This also happens in template literals. If JavaScript did not have automatic type coercion, we would have to manually convert the number to a string.

javascript Code Sample

```
console.log('I am ' + String(23) + ' years old')
```

Not all operators do type coercion to string. For example, the minus operator triggers the opposite conversion: strings are converted to numbers.

javascript Code Sample

```
console.log('23' - '10' - 3)
```

If we use the plus operator, the three is converted to a string and the strings are concatenated.

javascript Code Sample

```
console.log('23' + '10' + 3)
```

The same is true for multiplication and division. The values are converted to numbers before the operation.

javascript Code Sample

```
console.log('23' * '2')

console.log('23' / '2')
```

Guess the Output: Type Coercion in Action

Let us play a game called guess the output. Consider the following code:

javascript Code Sample

```
let n = '1' + 1

n = n - 1

console.log(n)
```

In the first line, '1' + 1 will result in '11' (a string), because the plus operator converts the number to a string. Then, the minus operator converts the string '11' to the number 11, and 11 - 1 is 10.

javascript Code Sample

Another example:

```
console.log(2 + 3 + 4 + '5')
```

Here, 2 + 3 is 5, 5 + 4 is 9, and then 9 + '5' results in '95' as a string.

javascript Code Sample

```
console.log('10' - '4' - '3' - 2 + '5')
```

In this example, '10' - '4' is 6, 6 - '3' is 3, 3 - 2 is 1, and then 1 + '5' results in '15' as a string.

Importance of Understanding Type Coercion

It is really important to know about type conversion and coercion right from the start so that you can write your code with all of this in mind. Many people do not like type coercion and think that it is a bad practice to rely on it, because it can introduce unexpected bugs. However, this only happens when we do not really know what we are doing. If you understand how type coercion works, it is easier to avoid these errors. Coercion is actually a great mechanism that allows us to write less code and more readable code. Take some time to understand how type coercion works and embrace it in your code.

Key Takeaways

- Type conversion is the manual process of changing a value from one type to another in JavaScript, using functions like **Number()** and **String()**.
- Type coercion is JavaScript's automatic conversion of types during operations, often triggered by operators like **+**, **-**, *****, and **/**.
- The **+** operator with a string and a number results in string concatenation, while **-**, *****, and **/** convert strings to numbers if possible.
- Understanding the difference between type conversion and coercion is essential to avoid unexpected bugs and write more readable code.