Theory: Type conversion

9 minutes 0 / 5 problems solved

JavaScript has automatic data type conversion. This means that the interpreter independently converts values to the type it expects to see in a particular part of your program. Such conversion is called **implicit**. At first glance, this looks useful and seems to eliminate unnecessary actions. Sometimes it really is, but at other times the features of JS may be confusing, leaving you to wonder why your code yields error after error or behaves strangely, not the way you wanted.

Throughout your programming practice, you will often find yourself in situations where you need to **explicitly** (that is, by ourselves) indicate that you need to change the value of a certain data type. That's why today we're going to learn how *strings, boolean*, and *numbers* can be converted to each other.

§1. String conversion

String conversion occurs when you want to represent something as a string. To *explicitly* cast a value to a string, the `String()` function must be applied to it. For example:

```
String(123);   // "123"
String(false); // "false"
String(-12.3); // "-12.3"
String(true);  // "true"
```

As you can see, the conversion is simple and intuitive.

In JS, an *implicit conversion* will be called by the binary `+` operator when one of the operands is a string:

```
"3" + 4                      // "34"
4 + ""                       // "4"
true + "detective"           // "truedetective"
"You are " + 25 + " years old" // "You are 25 years old"
```

The automatic conversion will take place regardless of the location of the operand string on the right or left side of the expression.

Remember the order of arithmetic operations. If there are several numbers before the string, these numbers will be added before the conversion:

```
3 + 10 + "1" // "131", not "3101"
```

Implicit conversion may be confusing, so be attentive when writing programs.

§2. Numeric conversion

Numeric conversion occurs when you want to represent something as a number. To perform *explicit* conversion, apply the `Number()` function:

```
Number("1");    // 1
Number(" 37 "); // 37
Number("");     // 0
Number("\n3");  // 3
Number("\n");   // 0
Number("\t");   // 0
Number(true);   // 1
Number(false);  // 0
```

When converting a string to a number, spaces and characters `\n`, `\t` at the beginning and the end of the string are cut off. If the string turns out to be empty, the result will be `0`. The boolean type behaves as expected: `false`

Current topic:

Type conversion

Topic depends on:

- ✕ Arithmetic operators
- ✕ Boolean and logical operators
- ✕ Null and Undefined

Topic is required for:

- Function constructor
- Comparison operators
- Object methods and keyword "this"

Table of contents:

turns into `0`, `true` turns into `1`.

> Not all values can be converted into numbers. The result of such attempts to convert is `NaN`. For example, `Number("apple")` will return a `NaN` value, which means **Not-a-Number**. Usually, this value is returned when an operation with numbers is performed incorrectly.

The *implicit* conversion is a little more confusing. It occurs in almost all mathematical functions and expressions:

```
true + 43 // 44
3 - false // 3
10 / "5"  // 2
-true     // -1
+"85"     // 85
```

For explicit conversion to a number, the string must be a correct number as well.

> Also, remember that binary plus *cannot* turn a string into a number, while the `Number()` function can. So the result of the expression `"5" + 7` will be a string, and the result `Number("5" + 7)` will be a number.

§3. Boolean conversion

Boolean conversion occurs when you want to represent something as a boolean. To *explicitly* convert to a boolean value, the `Boolean()` function must be applied:

```
Boolean(1);            // true
Boolean(0);            // false
Boolean("Am I nice?"); // true
Boolean("");           // false
```

The rules for using this function are simple. The following values as a result of conversion give the value `false`: `false`, `undefined`, `null`, `0`, `NaN`, `""`. All other values in the result of conversion give the value `true`.

An *implicit* conversion occurs when using logical operators (`||` `&&` `!`):

```
!!3                    // true
0 || "go"              // "go"
"Master" && "Margarita"  // "Margarita"
```

An implicit conversion with operators `||` and `&&` occurs under the hood and eventually returns the original value of one of the operators.

Report a typo

122 users liked this theory. **8** didn't like it. **What about you?**

Start practicing

Comments (15) Hints (0) Useful links (0) Show discussion