JS | Data Types in JavaScript-boolean, undefined & null and Immutability

Learning Goals

After this lesson you will be able to:

- Use boolean as data type
- Use undefined and null as a data types
- Understand primitives and immutability
- Understand `value` vs. `reference` in JavaScript

A boolean as data type

Some questions can only be answered with two possibilities: `yes` or `no`. For example:

- Are you going out tonight? `No`
- Will you learn a lot of new stuff at Ironhack? `Yes`
- Do you like your TAs? Hell, `yes`!

These answers are the equivalent of the boolean values in programming.

A **Boolean** or **bool** expression can result in the value of either **TRUE** or **FALSE**.

Booleans are often used in conditional statements, but we will come to that in a bit. Let's first get familiar with **logical operators**.

Boolean logic operators

We use logical operators to combine two (or more) conditions and depending on the conditions and the logical operator(s), we will get as a result a **true** or a **false**.

We have three different logical operators:

- **or**,
- **and** and
- **not**.

OR Operator (`||`)

The *or* operator, represented by `||`, returns `true` if **one of the evaluated expressions is `true`**.

```
expr1 || expr2
```

If `expr1` or `expr2` is `true`, the result will be `true`. If they both are `false`, the result of the expression will be `false`.

```
true  || true       // => true
true  || false      // => true
false || true       // => true
false || false      // => false
false || (4 > 2)    // true
```

AND Operator (`&&`)

The **and** operator, represented by `&&`, returns true just if **all the evaluated expressions are true**.

```
expr1 && expr2
```

If `expr1` and `expr2` are `true`, the result will be `true`. If one of the expressions is `false`, the result will be `false`. If both expressions are `false`, of course, the result will be `false`.

```
true  && true       // => true
true  && false      // => false
false && true       // => false
false && false      // => false
true  && (4 > 2)    // => true
```

A short-circuit evaluation 🤓

As logical expressions in JavaScript are evaluated left to right, they are tested for possible "short-circuit" evaluation using the following rules:

```
false && (anything) is a short-circuit evaluated to false.
true || (anything) is a short-circuit evaluated to true.
```

The rules of logic guarantee that these evaluations are **always correct**.

NOT Operator (`!`)

Here we have to mention nothing less important - the **not** operator. It's used to negate the value of an expression.

```
!expr1
```

If the expression is **true**, the result will be **false**, and vice versa.

```
!true        // => false
!false             // => true
!(4 > 2)     // => false
```

Keep these rules in mind, you will be using them quite a lot ✅

An undefined as data type

`undefined` is primitive value automatically assigned to variables when they are declared.

```
let name;
console.log(name); // <== undefined
```

A null as data type

In computer science, a **null** value represents a reference that points, generally intentionally, to a nonexistent address, meaning the variable that hasn't been even declared yet.

However, in JavaScript, `null` is often used to represent `value unknown` variables:

```
let name = null;
console.log(name); // <== null
```

You will often use this value when checking if a variable has even been declared or when you intentionally want to reassign the value of some variable to *null* because of some changes in its status in your application. (this is just hypothetically speaking, it will be more clear later through the course)

We also mentioned `symbol` as a primitive value. However, we will not dig into symbols since we won't use it during this course. If you would like to know more about it here is MDN - Symbol page and there are many other online resources.

Immutability

As we said at the very beginning, all primitive data types are **immutable**.

Immutability means that once one of the primitive values is created, it can't be modified.

Let's explain this:

```
let city = "miami";
console.log(city[0]); // <== m
city[0] = "M";
console.log(city); // <== miami
```

Don't get confused here because **values are immutable** but variables are mutable which means you can reassign them:

```
let city = "miami";
console.log(city); // <== miami

city = "berlin";
console.log(city); // <== berlin
```

```
city[0] = "B";
console.log(city); // <== berlin
```

✔ You can see from the previous example that you can reassign the variable with a new value but you can't alter the existing value.

As you can see, when we practiced the string methods, each of them returned a new string and the original string stayed untouched.

```
const message = "Don't be sad, be happy!";
console.log(message.slice(0,3)); // <== Don
console.log(message); // <== Don't be sad, be happy!
```

Numbers are immutable as well - but that is more a common sense, right? Obviously, if number "5" is of value "5", it will stay forever the same value - you can't change it.

Immutability is a very important topic in JavaScript and we will come back to it a bit later in this module when we talk about Objects. Objects (and arrays) are mutable data types by default and we will see what that means in a couple of learning units.

Summary

Extra Resources

- MDN - Logical Operators
- MDN - undefined as data type
- MDN - null as data type
- MDN - Difference between null and undefined
- MDN - Mutable