Primitive

In JavaScript, a **primitive** (primitive value, primitive data type) is data that is not an object and has no methods. There are 6 primitive data types: string, number, bigint, boolean, undefined, and symbol. There also is null, which is seemingly primitive, but indeed is a special case for every `Object`: and any structured type is derived from `null` by the Prototype Chain.

Most of the time, a primitive value is represented directly at the lowest level of the language implementation.

All primitives are **immutable**, i.e., they cannot be altered. It is important not to confuse a primitive itself with a variable assigned a primitive value. The variable may be reassigned a new value, but the existing value can not be changed in the ways that objects, arrays, and functions can be altered.

Example

This example will help you understand that primitive values are **immutable.**

JavaScript

```
// Using a string method doesn't mutate the string
var bar = "baz";
console.log(bar);               // baz
bar.toUpperCase();
console.log(bar);               // baz

// Using an array method mutates the array
var foo = [];
console.log(foo);               // []
foo.push("plugh");
console.log(foo);               // ["plugh"]
```

```
12
13 // Assignment gives the primitive a new (not a mutated) value
14 bar = bar.toUpperCase();        // BAZ
```

A primitive can be replaced, but it can't be directly altered.

Another Example [Step-by-step]

The following example will help you go through how JavaScript deals with primitives.

JavaScript

```
1  // The primitive
2  let foo = 5;
3
4  // Defining a function that should change the primitive value
5  function addTwo(num) {
6     num += 2;
7  }
8  // Another function trying to do the same thing
9  function addTwo_v2(foo) {
10    foo += 2;
11 }
12
13 // Calling our first function while passing our primitive as an a
14 addTwo(foo);
15 // Getting the current primitive value
16 console.log(foo);   // 5
17
18 // Trying again with our second function...
19 addTwo_v2(foo);
20 console.log(foo);   // 5
```

Did you expect it to be `7` instead of `5`? If so, read how this code runs:

- For both the `addTwo` and `addTwo_v2` functions calls, JavaScript looks up the value for the identifier `foo`. It correctly finds our variable instantiated with our first statement

- After finding it, the expression is evaluated, `foo` is replaced by 5 and the JavaScript engine passes that value to the functions as an argument
- Before executing the statements inside the functions' bodies, **JavaScript takes a copy of the originally passed argument** (which is a primitive) and creates a local copy. These copies, existing only inside the functions' scopes, are accessible via the identifiers we specified in the functions' definitions (`num` for `addTwo`, `foo` for `addTwo_v2`)
- Then, the functions' statements are executed:
  - In the first function, a local `num` variable had been created. We are increasing its value by 2, not the original `foo`'s value!
  - In the second function, a local `foo` variable had been created. We are increasing its value by 2, not the original (external) `foo`'s value! Also, in this situation, the external `foo` variable cannot be accessed directly. This is because of JavaScript's lexical scoping and the resulting variable shadowing. The local `foo` hides the external `foo`. For more information, see Closures. (Note that `window.foo` could still be used to access the external `foo` variable.)
- In conclusion, any changes inside our functions **won't** affect the original `foo` at all, as we are modifying **copies** of it

That's why primitives are immutable - instead of changing them directly, we're modifying a *copy, without affecting the original*.

Primitive wrapper objects in JavaScript

Except for `null` and `undefined`, all primitive values have object equivalents that wrap around the primitive values:

- `String` for the string primitive.
- `Number` for the number primitive.
- `BigInt` for the bigint primitive.
- `Boolean` for the boolean primitive.
- `Symbol` for the symbol primitive.

The wrapper's `valueOf()` method returns the primitive value.

Learn more

General knowledge

- Introduction to JavaScript data types
- Primitive data type on Wikipedia

Last modified: Aug 13, 2020, by MDN contributors

Related Topics

Glossary

- JavaScript
- string
- number
- bigint
- boolean
- null
- undefined
- symbol

JavaScript data types