July 9, 2024

Data types

A value in JavaScript is always of a certain type. For example, a string or a number.

There are eight basic data types in JavaScript. Here, we'll cover them in general and in the next chapters we'll talk about each of them in detail.

We can put any type in a variable. For example, a variable can at one moment be a string and then store a number:

```
// no error
let message = "hello";
message = 123456;
```

Programming languages that allow such things, such as JavaScript, are called "dynamically typed", meaning that there exist data types, but variables are not bound to any of them.

Number

```
let n = 123;
n = 12.345;
```

The *number* type represents both integer and floating point numbers.

There are many operations for numbers, e.g. multiplication `*`, division `/`, addition `+`, subtraction `-`, and so on.

Besides regular numbers, there are so-called "special numeric values" which also belong to this data type: `Infinity`, `-Infinity` and `NaN`.

- `Infinity` represents the mathematical Infinity ∞. It is a special value that's greater than any number.

  We can get it as a result of division by zero:

  ```
  alert( 1 / 0 ); // Infinity
  ```

  Or just reference it directly:

```
alert( Infinity ); // Infinity
```

- `NaN` represents a computational error. It is a result of an incorrect or an undefined mathematical operation, for instance:

```
alert( "not a number" / 2 ); // NaN, such division is erroneous
```

`NaN` is sticky. Any further mathematical operation on `NaN` returns `NaN`:

```
alert( NaN + 1 ); // NaN
alert( 3 * NaN ); // NaN
alert( "not a number" / 2 - 1 ); // NaN
```

So, if there's a `NaN` somewhere in a mathematical expression, it propagates to the whole result (there's only one exception to that: `NaN ** 0` is `1`).

Mathematical operations are safe

Doing maths is "safe" in JavaScript. We can do anything: divide by zero, treat non-numeric strings as numbers, etc.

The script will never stop with a fatal error ("die"). At worst, we'll get `NaN` as the result.

Special numeric values formally belong to the "number" type. Of course they are not numbers in the common sense of this word.

We'll see more about working with numbers in the chapter Numbers.

BigInt

In JavaScript, the "number" type cannot safely represent integer values larger than $(2^{53}-1)$ (that's `9007199254740991`), or less than $-(2^{53}-1)$ for negatives.

To be really precise, the "number" type can store larger integers (up to $1.7976931348623157 * 10^{308}$), but outside of the safe integer range $\pm(2^{53}-1)$ there'll be a precision error, because not all digits fit into the fixed 64-bit storage. So an "approximate" value may be stored.

For example, these two numbers (right above the safe range) are the same:

```
console.log(9007199254740991 + 1); // 9007199254740992
console.log(9007199254740991 + 2); // 9007199254740992
```

So to say, all odd integers greater than $(2^{53}-1)$ can't be stored at all in the "number" type.

For most purposes `±(2^53-1)` range is quite enough, but sometimes we need the entire range of really big integers, e.g. for cryptography or microsecond-precision timestamps.

`BigInt` type was recently added to the language to represent integers of arbitrary length.

A `BigInt` value is created by appending `n` to the end of an integer:

```
// the "n" at the end means it's a BigInt
const bigInt = 1234567890123456789012345678901234567890n;
```

As `BigInt` numbers are rarely needed, we don't cover them here, but devoted them a separate chapter BigInt. Read it when you need such big numbers.

String

A string in JavaScript must be surrounded by quotes.

```
let str = "Hello";
let str2 = 'Single quotes are ok too';
let phrase = `can embed another ${str}`;
```

In JavaScript, there are 3 types of quotes.

1. Double quotes: `"Hello"` .
2. Single quotes: `'Hello'` .
3. Backticks: `` `Hello` `` .

Double and single quotes are "simple" quotes. There's practically no difference between them in JavaScript.

Backticks are "extended functionality" quotes. They allow us to embed variables and expressions into a string by wrapping them in `${…}` , for example:

```
let name = "John";

// embed a variable
alert( `Hello, ${name}!` ); // Hello, John!

// embed an expression
alert( `the result is ${1 + 2}` ); // the result is 3
```

The expression inside `${…}` is evaluated and the result becomes a part of the string. We can put anything in there: a variable like `name` or an arithmetical expression like `1 + 2` or something more complex.

Please note that this can only be done in backticks. Other quotes don't have this embedding functionality!

```
alert( "the result is ${1 + 2}" ); // the result is ${1 + 2} (double quotes do
```

We'll cover strings more thoroughly in the chapter Strings.

> **There is no *character* type.**
>
> In some languages, there is a special "character" type for a single character. For example, in the C language and in Java it is called "char".
>
> In JavaScript, there is no such type. There's only one type: `string`. A string may consist of zero characters (be empty), one character or many of them.

Boolean (logical type)

The boolean type has only two values: `true` and `false`.

This type is commonly used to store yes/no values: `true` means "yes, correct", and `false` means "no, incorrect".

For instance:

```
let nameFieldChecked = true; // yes, name field is checked
let ageFieldChecked = false; // no, age field is not checked
```

Boolean values also come as a result of comparisons:

```
let isGreater = 4 > 1;

alert( isGreater ); // true (the comparison result is "yes")
```

We'll cover booleans more deeply in the chapter Logical operators.

The "null" value

The special `null` value does not belong to any of the types described above.

It forms a separate type of its own which contains only the `null` value:

```
let age = null;
```

In JavaScript, `null` is not a "reference to a non-existing object" or a "null pointer" like in some other languages.

It's just a special value which represents "nothing", "empty" or "value unknown".

The code above states that `age` is unknown.

The “undefined” value

The special value `undefined` also stands apart. It makes a type of its own, just like `null`.

The meaning of `undefined` is “value is not assigned”.

If a variable is declared, but not assigned, then its value is `undefined`:

```
let age;

alert(age); // shows "undefined"
```

Technically, it is possible to explicitly assign `undefined` to a variable:

```
let age = 100;

// change the value to undefined
age = undefined;

alert(age); // "undefined"
```

...But we don’t recommend doing that. Normally, one uses `null` to assign an “empty” or “unknown” value to a variable, while `undefined` is reserved as a default initial value for unassigned things.

Objects and Symbols

The `object` type is special.

All other types are called “primitive” because their values can contain only a single thing (be it a string or a number or whatever). In contrast, objects are used to store collections of data and more complex entities.

Being that important, objects deserve a special treatment. We’ll deal with them later in the chapter Objects, after we learn more about primitives.

The `symbol` type is used to create unique identifiers for objects. We have to mention it here for the sake of completeness, but also postpone the details till we know objects.

The typeof operator

The `typeof` operator returns the type of the operand. It’s useful when we want to process values of different types differently or just want to do a quick check.

A call to `typeof x` returns a string with the type name:

```
typeof undefined // "undefined"

typeof 0 // "number"

typeof 10n // "bigint"
```

```
typeof true // "boolean"

typeof "foo" // "string"

typeof Symbol("id") // "symbol"

typeof Math // "object"  (1)

typeof null // "object"  (2)

typeof alert // "function"  (3)
```

The last three lines may need additional explanation:

1. `Math` is a built-in object that provides mathematical operations. We will learn it in the chapter Numbers. Here, it serves just as an example of an object.
2. The result of `typeof null` is `"object"`. That's an officially recognized error in `typeof`, coming from very early days of JavaScript and kept for compatibility. Definitely, `null` is not an object. It is a special value with a separate type of its own. The behavior of `typeof` is wrong here.
3. The result of `typeof alert` is `"function"`, because `alert` is a function. We'll study functions in the next chapters where we'll also see that there's no special "function" type in JavaScript. Functions belong to the object type. But `typeof` treats them differently, returning `"function"`. That also comes from the early days of JavaScript. Technically, such behavior isn't correct, but can be convenient in practice.

The `typeof(x)` syntax

You may also come across another syntax: `typeof(x)`. It's the same as `typeof x`.

To put it clear: `typeof` is an operator, not a function. The parentheses here aren't a part of `typeof`. It's the kind of parentheses used for mathematical grouping.

Usually, such parentheses contain a mathematical expression, such as `(2 + 2)`, but here they contain only one argument `(x)`. Syntactically, they allow to avoid a space between the `typeof` operator and its argument, and some people like it.

Some people prefer `typeof(x)`, although the `typeof x` syntax is much more common.

Summary

There are 8 basic data types in JavaScript.

- Seven primitive data types:
  - `number` for numbers of any kind: integer or floating-point, integers are limited by $\pm(2^{53}-1)$.
  - `bigint` for integer numbers of arbitrary length.
  - `string` for strings. A string may have zero or more characters, there's no separate single-character type.
  - `boolean` for `true` / `false`.
  - `null` for unknown values – a standalone type that has a single value `null`.
  - `undefined` for unassigned values – a standalone type that has a single value `undefined`.

  - `symbol` for unique identifiers.
- And one non-primitive data type:
  - `object` for more complex data structures.

The `typeof` operator allows us to see which type is stored in a variable.

- Usually used as `typeof x`, but `typeof(x)` is also possible.
- Returns a string with the name of the type, like `"string"`.
- For `null` returns `"object"` – this is an error in the language, it's not actually an object.

In the next chapters, we'll concentrate on primitive values and once we're familiar with them, we'll move on to objects.

Tasks

String quotes

importance: 5

What is the output of the script?

```
let name = "Ilya";

alert( `hello ${1}` ); // ?

alert( `hello ${"name"}` ); // ?

alert( `hello ${name}` ); // ?
```

solution

Previous lesson Next lesson

Share

Tutorial map

Comments

- If you have suggestions what to improve - please submit a GitHub issue or a pull request instead of commenting.
- If you can't understand something in the article – please elaborate.
- To insert few words of code, use the `<code>` tag, for several lines – wrap them in `<pre>` tag, for more than 10 lines – use a sandbox (plnkr, jsbin, codepen...)

Should governments regulate the use of AI in political campaigns?

Choose one

© 2007—2025 Ilya Kantorabout the projectcontact usterms of usage privacy policy