**Data types**

A variable in JavaScript can contain any data. A variable can at one moment be a string and later receive a numeric value:

// no error

let message = "hello";

message = 123456;

There are 7 basic data types in JavaScript.

[**A number**](https://javascript.info/types#a-number)

let n = 123;

n = 12.345;

The *number* type serves both for integer(whole numbers) and floating point numbers (decimal numbers).

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

Besides regular numbers, there are so-called “special numeric values” which also belong to that type: Infinity, -Infinity and NaN.

* Infinity represents the mathematical [Infinity](https://en.wikipedia.org/wiki/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 mention it in the code 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 operation on NaN would give NaN:

alert( "not a number" / 2 + 5 ); // NaN

So, if there’s NaN somewhere in a mathematical expression, it propagates to the whole result.

**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 a common sense of this word.

**[A string](https://javascript.info/types" \l "a-string)**

A string in JavaScript must be quoted.

let str = "Hello";

let str2 = 'Single quotes are ok too';

let phrase = `can embed ${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 no difference between them in JavaScript.

Backticks are “extended functionality” quotes. They allow 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 there: a variable like name or an arithmetical expression like 1 + 2 or something more complex.

Please note that this only can be done in backticks, other quotes do not allow such embedding!

alert( "the result is ${1 + 2}" ); // the result is ${1 + 2} (double quotes do nothing)

**[A boolean (logical type)](https://javascript.info/types" \l "a-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 the “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")

[**The “null” value**](https://javascript.info/types#the-null-value)

The special null value does not belong to any type of those described above.

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

let age = null;

The code above states that the age is unknown or empty for some reason.

**[The “undefined” value](https://javascript.info/types" \l "the-undefined-value)**

The special value undefined 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 exactly undefined:

let x;

alert(x); // shows "undefined"

Technically, it is possible to assign any variable to undefined:

let x = 123;

x = undefined;

alert(x); // "undefined"

…But it’s not recommended to do that. Normally, we use null to write an “empty” or an “unknown” value into the variable, and undefined is only used for checks, to see if the variable is assigned or similar.

**[Objects and Symbols](https://javascript.info/types" \l "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 data and more complex entities.

The symbol type is used to create unique identifiers for objects.

[**The typeof operator**](https://javascript.info/types#type-typeof)

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

It supports two forms of syntax:

1. As an operator: typeof x.
2. Function style: typeof(x).

In other words, it works both with the brackets or without them. The result is the same.

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

typeof undefined // "undefined"

typeof 0 // "number"

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 explanations:

1. Math is a built-in object that provides mathematical operations.
2. The result of typeof null is "object". That’s wrong. It is an officially recognized error in typeof, kept for compatibility. Of course, null is not an object. It is a special value with a separate type of its own
3. The result of typeof alert is "function", because alert is a function of the language
4. Functions belong to the object type. But typeof treats them differently. Formally, it’s incorrect, but very convenient in practice.

**[Summary](https://javascript.info/types" \l "summary)**

There are 7 basic types in JavaScript.

* number for numbers of any kind: integer or floating-point.
* string for strings. A string may have one 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.
* object for more complex data structures.
* symbol for unique identifiers.

The typeof operator allows to see which type is stored in the variable.

* Two forms: typeof x or typeof(x).
* Returns a string with the name of the type, like "string".
* For null returns "object" – that’s the error in the language, it’s not an object in fact.

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

**Questions and Exercises**

* 1. What is the output of the script?

let name = "Ilya";

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

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

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

hello 1, hello name, hello Ilya.

* 1. Test the script above by creating a web page and linked JS source file.
  2. List the different data types in JS.

Number, string, Boolean, null, undefined, object, symbol

* 1. What does **typeof** do?

Tells what type of data is

* 1. What is a floating point number?

Number with decimal

* 1. Write a JS program that displays a Roses are red, violets are blue …. rhyme but uses variables to store and represent all nouns.
  2. Do some research. Find out how to create a random number in JS. Create a program that displays random numbers in the following ranges:

1 to 10

1 to 100

10 to 20

0 to 0.999