→ The JavaScript language → JavaScript Fundamentals

1st December 2019

Data types

A variable in JavaScript can contain any data. A variable can at one moment be a string and at another be a number:

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

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

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.

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 operation on `NaN` returns `NaN`:

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

  So, if there’s a `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 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 represent integer values larger than 2^{53} (or less than -2^{53} for negatives), that’s a technical limitation caused by their internal representation. That’s about 16 decimal digits, so for most purposes the limitation isn’t a problem, but sometimes we need really big numbers, e.g. for cryptography or microsecond-precision timestamps.

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

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

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

As `BigInt` numbers are rarely needed, we devoted them a separate chapter BigInt.

Compatability issues

Right now `BigInt` is supported in Firefox and Chrome, but not in Safari/IE/Edge.

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 d
```

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 only 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 or empty for some reason.

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 x;

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

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

```
let x = 123;

x = undefined;

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

…But we don’t recommend doing that. Normally, we use `null` to assign an “empty” or “unknown” value to a variable, and we use `undefined` for checks like seeing if a variable has been assigned.

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. 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 mention it here for completeness, but we’ll study it after objects.

The typeof operator

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 do a quick check.

It supports two forms of syntax:

1. As an operator: `typeof x`.
2. As a function: `typeof(x)`.

In other words, it works with parentheses 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 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 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. So, again, this is an error in the language.
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’s not quite correct, but very convenient in practice.

Summary

There are 8 basic data types in JavaScript.

- `number` for numbers of any kind: integer or floating-point, integers are limited by $\pm 2^{53}$.
- `bigint` is for integer numbers of arbitrary length.
- `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 us to see which type is stored in a variable.

- Two forms: `typeof x` or `typeof(x)` .
- 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 a few words of code, use the `<code>` tag, for several lines – use `<pre>`, for more than 10 lines – use a sandbox (plnkr, JSBin, codepen…)