**JAVASCRIPT DATA TYPES**

**JavaScript String**

String is used to store text. In JavaScript, strings are surrounded by quotes:

* Single quotes: 'Hello'
* Double quotes: "Hello"
* Backticks: `Hello`

For example,

//strings example

const name = 'ram';

const name1 = "hari";

const result = `The names are ${name} and ${name1}`;

Single quotes and double quotes are practically the same and you can use either of them.

Backticks are generally used when you need to include variables or expressions into a string. This is done by wrapping variables or expressions with ${variable or expression} as shown above.

**JavaScript Number**

Number represents integer and floating numbers (decimals and exponentials). For example,

const number1 = 3;

const number2 = 3.433;

const number3 = 3e5 // 3 \* 10^5

In JavaScript, we can shorten a number by appending the letter "e" to it and specifying the zeroes count:

*let billion = 1e9; // 1 billion, literally: 1 and 9 zeroes*

*alert( 7.3e9 ); // 7.3 billions (same as 7300000000 or 7\_300\_000\_000)*

**JavaScript BigInt**

In JavaScript, Number type can only represent numbers less than **(253** **- 1)** and more than **-(253** **- 1)**. However, if you need to use a larger number than that, you can use the BigInt data type.

A BigInt number is created by appending **n** to the end of an integer. For example,

// BigInt value

const value1 = 900719925124740998n;

// Adding two big integers

const result1 = value1 + 1n;

console.log(result1); // "900719925124740999n"

const value2 = 900719925124740998n;

// Error! BitInt and number cannot be added

const result2 = value2 + 1;

console.log(result2);

[Run Code](https://www.programiz.com/javascript/online-compiler)

**Output**

900719925124740999n

Uncaught TypeError: Cannot mix BigInt and other types

**Note:** BigInt was introduced in the newer version of JavaScript and is not supported by many browsers including Safari. Visit [JavaScript BigInt support](https://caniuse.com/#feat=bigint) to learn more.

**JavaScript Boolean**

This data type represents logical entities. Boolean represents one of two values: true or false. It is easier to think of it as a yes/no switch. For example,

const dataChecked = true;

const valueCounted = false;

**JavaScript undefined**

The undefined data type represents **value that is not assigned**. If a variable is declared but the value is not assigned, then the value of that variable will be undefined. For example,

let name;

console.log(name); // undefined

[Run Code](https://www.programiz.com/javascript/online-compiler)

It is also possible to explicitly assign a variable value undefined. For example,

let name = undefined;

console.log(name); // undefined

[Run Code](https://www.programiz.com/javascript/online-compiler)

**Note:** It is recommended not to explicitly assign undefined to a variable. Usually, null is used to assign 'unknown' or 'empty' value to a variable.

**JavaScript null**

In JavaScript, null is a special value that represents **empty** or **unknown value**. For example,

const number = null;

The code above suggests that the number variable is empty.

**Note**: null is not the same as NULL or Null.

**JavaScript Symbol**

This data type was introduced in a newer version of JavaScript (from ES2015).

A “symbol” represents a unique identifier.

A value of this type can be created using Symbol():

let id = Symbol();

Upon creation, we can give symbols a description (also called a symbol name), mostly useful for debugging purposes:

// id is a symbol with the description "id"

let id = Symbol("id");

Symbols are guaranteed to be unique. Even if we create many symbols with exactly the same description, they are different values. The description is just a label that doesn’t affect anything.

let id1 = Symbol("id");

let id2 = Symbol("id");

alert(id1 == id2); // false

When to use symbols

**Symbols don’t auto-convert to a string**

Most values in JavaScript support implicit conversion to a string. For instance, we can alert almost any value, and it will work. Symbols are special. They don’t auto-convert.

For instance, this alert will show an error:

let id = Symbol("id");

alert(id); // TypeError: Cannot convert a Symbol value to a string

That’s a “language guard” against messing up, because strings and symbols are fundamentally different and should not accidentally convert one into another.

If we really want to show a symbol, we need to explicitly call .toString() on it, like here:

let id = Symbol("id");

alert(id.toString()); // Symbol(id), now it works

Or get symbol.description property to show the description only:

let id = Symbol("id");

alert(id.description); // id

## [“Hidden” properties](https://javascript.info/symbol" \l "hidden-properties)

Symbols allow us to create “hidden” properties of an object, that no other part of code can accidentally access or overwrite.

For instance, if we’re working with user objects, that belong to a third-party code. We’d like to add identifiers to them.

Let’s use a symbol key for it:

let user = { // belongs to another code

name: "John"

};

let id = Symbol("id");

user[id] = 1;

alert( user[id] ); // we can access the data using the symbol as the key

What’s the benefit of using Symbol("id") over a string "id"?

As user objects belong to another codebase, it’s unsafe to add fields to them, since we might affect pre-defined behavior in that other codebase. However, symbols cannot be accessed accidentally. The third-party code won’t be aware of newly defined symbols, so it’s safe to add symbols to the user objects.

**JavaScript Object**

An object is a complex data type that allows us to store collections of data. For example,

const student = {

firstName: 'ram',

lastName: null,

class: 10

};

**JavaScript Type**

JavaScript is a dynamically typed (loosely typed) language. JavaScript automatically determines the variables' data type for you.

It also means that a variable can be of one data type and later it can be changed to another data type. For example,

// data is of undefined type

let data;

// data is of integer type

data = 5;

// data is of string type

data = "JavaScript Programming";

**JavaScript typeof**

To find the type of a variable, you can use the typeof operator. For example,

 const name = 'ram';

console.log(typeof(name)); // returns "string"

const number = 4;

console.log(typeof(number)); //returns "number"

const valueChecked = true;

console.log(typeof(valueChecked)); //returns "boolean"

const a = null;

console.log(typeof(a)); // returns "object"

Notice that typeof returned "object" for the null type. This is a known issue in JavaScript since its first release.