# Тип данных Symbol

«Символ» представляет собой уникальный идентификатор.

```
// Создаём новый символ - id
let id = Symbol();
```

При создании символу можно дать описание (также называемое имя), в основном использующееся для отладки кода:

```
// Создаём символ id с описанием (именем) "id"
let id = Symbol("id");
```

Символы гарантированно уникальны. Даже если мы создадим множество символов с одинаковым описанием, это всё равно будут разные символы.

Символы не преобразуются автоматически в строки
К примеру, alert ниже выдаст ошибку:

```
let id = Symbol("id");
alert(id); // TypeError: Cannot convert a Symbol value to a string
```

Если же мы действительно хотим вывести символ с помощью alert, то необходимо явно преобразовать его с помощью метода .toString(), вот так:

```
let id = Symbol("id");
alert(id.toString()); // Symbol(id), теперь работает
```

Или мы можем обратиться к свойству symbol.description, чтобы вывести только описание:

```
let id = Symbol("id");
alert(id.description); // id
```

Символы позволяют создавать «скрытые» свойства объектов, к которым нельзя нечаянно обратиться и перезаписать их из других частей программы.

Если мы хотим использовать символ при литеральном объявлении объекта {...}, его необходимо заключить в квадратные скобки:

```
let id = Symbol("id");

let user = {
  name: "Вася",
  [id]: 123 // просто "id: 123" не сработает
};
```

Свойства, чьи ключи – символы, не перебираются циклом for..in.

Если мы хотим, чтобы одноимённые символы были равны, то следует использовать глобальный реестр: вызов Symbol.for(key) возвращает (или создаёт) глобальный символ с key в качестве имени. Многократные вызовы команды Symbol.for с одним и тем же аргументом возвращают один и тот же символ.

```
// читаем символ из глобального реестра и записываем его в переменную
let id = Symbol.for("id"); // если символа не существует, он будет создан

// читаем его снова в другую переменную (возможно, из другого места кода)
let idAgain = Symbol.for("id");

// проверяем -- это один и тот же символ
alert( id === idAgain ); // true
```

Символы, содержащиеся в реестре, называются **глобальными символами.**

Для глобальных символов, кроме Symbol.for(key), который ищет символ по имени, существует обратный метод: Symbol.keyFor(sym), который, наоборот, принимает глобальный символ и возвращает его имя. К примеру:

```
// получаем символ по имени
let sym = Symbol.for("name");
let sym2 = Symbol.for("id");

// получаем имя по символу
alert( Symbol.keyFor(sym) ); // name
alert( Symbol.keyFor(sym2) ); // id
```

Этот метод не будет работать для неглобальных символов. Если символ неглобальный, метод не сможет его найти и вернёт undefined. Впрочем, для любых символов доступно свойство description.

Существует множество «системных» символов, использующихся внутри самого JavaScript, и мы можем использовать их, чтобы настраивать различные аспекты поведения объектов.

Эти символы перечислены в спецификации в таблице Well-known symbols (https://tc39.es/ecma262/#sec-well-known-symbols):

- Symbol.hasInstance
- Symbol.isConcatSpreadable
- Symbol.iterator
- Symbol.toPrimitive
- …и так далее.