Типы данных

Значение в JavaScript всегда относится к данным определённого типа. Например, это может быть строка или число.

Есть восемь основных типов данных в JavaScript. В этой главе мы рассмотрим их в общем, а в следующих главах поговорим подробнее о каждом.

Переменная в JavaScript может содержать любые данные. В один момент там может быть строка, а в другой – число:

```
// Не будет ошибкой
let message = "hello";


message = 123456;
```

Языки программирования, в которых такое возможно, называются «динамически типизированными». Это значит, что типы данных есть, но переменные не привязаны ни к одному из них.

Число

```
let n = 123;


n = 12.345;
```

Числовой тип данных (`number`) представляет как целочисленные значения, так и числа с плавающей точкой.

Существует множество операций для чисел, например, умножение `*`, деление `/`, сложение `+`, вычитание `-` и так далее.

Кроме обычных чисел, существуют так называемые «специальные числовые значения», которые относятся к этому типу данных: `Infinity`, `-Infinity` и `NaN`.

`Infinity` представляет собой математическую бесконечность ∞. Это особое значение, которое больше любого числа.
Мы можем получить его в результате деления на ноль:

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

Или задать его явно:

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

`NaN` означает вычислительную ошибку. Это результат неправильной или неопределённой математической операции, например:

```
alert( "не число" / 2 ); // NaN, такое деление является ошибкой
```

Значение `NaN` «прилипчиво». Любая математическая операция с `NaN` возвращает `NaN`:

```
alert( NaN + 1 ); // NaN
alert( 3 * NaN ); // NaN
```

```
alert( "не число" / 2 - 1 ); // NaN
```

Если где-то в математическом выражении есть `NaN`, то оно распространяется на весь результат (есть только одно исключение: `NaN ** 0` равно `1`).

Математические операции – безопасны

Математические операции в JavaScript «безопасны». Мы можем делать что угодно: делить на ноль, обращаться с нечисловыми строками как с числами и т.д.

Скрипт никогда не остановится с фатальной ошибкой (не «умрёт»). В худшем случае мы получим `NaN` как результат выполнения.

Специальные числовые значения относятся к типу «число». Конечно, это не числа в привычном значении этого слова.

Подробнее о работе с числами мы поговорим в главе Числа.

BigInt

В JavaScript тип `number` не может безопасно работать с числами, большими, чем $(2^{53}-1)$ (т. е. `9007199254740991`) или меньшими, чем $-(2^{53}-1)$ для отрицательных чисел.

Если говорить совсем точно, то, технически, тип `number` *может* хранить большие целые числа (до $1.7976931348623157 * 10^{308}$), но за пределами безопасного диапазона целых чисел $\pm(2^{53}-1)$ будет ошибка точности, так как не все цифры помещаются в фиксированную 64-битную память. Поэтому можно хранить «приблизительное» значение.

Например, эти два числа (прямо за пределами безопасного диапазона) совпадают:

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


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

То есть все нечетные целые числа, большие чем $(2^{53}-1)$, вообще не могут храниться в типе `number`.

В большинстве случаев безопасного диапазона чисел от $-(2^{53}-1)$ до $(2^{53}-1)$ вполне достаточно, но иногда нам требуется весь диапазон действительно гигантских целых чисел без каких-либо ограничений или пропущенных значений внутри него. Например, в криптографии или при использовании метки времени («timestamp») с микросекундами.

Тип `BigInt` был добавлен в JavaScript, чтобы дать возможность работать с целыми числами произвольной длины.

Чтобы создать значение типа `BigInt`, необходимо добавить `n` в конец числового литерала:

```
// символ "n" в конце означает, что это BigInt


const bigInt = 1234567890123456789012345678901234567890n;
```

Так как необходимость в использовании `BigInt`–чисел появляется достаточно редко, мы рассмотрим их в отдельной главе BigInt. Ознакомьтесь с ней, когда вам понадобятся настолько большие числа.

Поддержка

> В данный момент `BigInt` поддерживается только в браузерах Firefox, Chrome, Edge и Safari, но не поддерживается в IE.

Строка

Строка (`string`) в JavaScript должна быть заключена в кавычки.

```
let str = "Привет";
let str2 = 'Одинарные кавычки тоже подойдут';


let phrase = `Обратные кавычки позволяют встраивать переменные ${str}`;
```

В JavaScript существует три типа кавычек.

1. Двойные кавычки: `"Привет"`.
2. Одинарные кавычки: `'Привет'`.
3. Обратные кавычки: `` `Привет` ``.

Двойные или одинарные кавычки являются «простыми», между ними нет разницы в JavaScript.

Обратные же кавычки имеют расширенную функциональность. Они позволяют нам встраивать выражения в строку, заключая их в `${…}`. Например:

```
let name = "Иван";

// Вставим переменную
alert( `Привет, ${name}!` ); // Привет, Иван!

// Вставим выражение


alert( `результат: ${1 + 2}` ); // результат: 3
```

Выражение внутри `${…}` вычисляется, и его результат становится частью строки. Мы можем положить туда всё, что угодно: переменную `name`, или выражение `1 + 2`, или что-то более сложное.

Обратите внимание, что это можно делать только в обратных кавычках. Другие кавычки не имеют такой функциональности встраивания!

```
alert( "результат: ${1 + 2}" ); // результат: ${1 + 2} (двойные кавычки ничего не делают)
```

Мы рассмотрим строки более подробно в главе Строки.

Нет отдельного типа данных для одного символа.

> В некоторых языках, например C и Java, для хранения одного символа, например `"a"` или `"%"`, существует отдельный тип. В языках C и Java это `char`.

> В JavaScript подобного типа нет, есть только тип `string`. Строка может содержать ноль символов (быть пустой), один символ или множество.

Булевый (логический) тип

Булевый тип (`boolean`) может принимать только два значения: `true` (истина) и `false` (ложь).

Такой тип, как правило, используется для хранения значений да/нет: `true` значит «да, правильно», а `false` значит «нет, не правильно».

Например:

```
let nameFieldChecked = true; // да, поле отмечено

let ageFieldChecked = false; // нет, поле не отмечено
```

Булевые значения также могут быть результатом сравнений:

```
let isGreater = 4 > 1;
```

```
alert( isGreater ); // true (результатом сравнения будет "да")
```

Мы рассмотрим булевые значения более подробно в главе Логические операторы.

Значение «null»

Специальное значение `null` не относится ни к одному из типов, описанных выше.

Оно формирует отдельный тип, который содержит только значение `null`:

```
let age = null;
```

В JavaScript `null` не является «ссылкой на несуществующий объект» или «нулевым указателем», как в некоторых других языках.

Это просто специальное значение, которое представляет собой «ничего», «пусто» или «значение неизвестно».

В приведённом выше коде указано, что значение переменной `age` неизвестно.

Значение «undefined»

Специальное значение `undefined` также стоит особняком. Оно формирует тип из самого себя так же, как и `null`.

Оно означает, что «значение не было присвоено».

Если переменная объявлена, но ей не присвоено никакого значения, то её значением будет `undefined`:

```
let age;
```

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

Технически мы можем присвоить значение `undefined` любой переменной:

```
let age = 123;

// изменяем значение на undefined
age = undefined;



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

…Но так делать не рекомендуется. Обычно `null` используется для присвоения переменной «пустого» или «неизвестного» значения, а `undefined` – для проверок, была ли переменная назначена.

Объекты и символы

Тип `object` (объект) – особенный.

Все остальные типы называются «примитивными», потому что их значениями могут быть только простые значения (будь то строка, или число, или что-то ещё). В объектах же хранят коллекции данных или более сложные структуры.

Объекты занимают важное место в языке и требуют особого внимания. Мы разберёмся с ними в главе Объекты после того, как узнаем больше о примитивах.

Тип `symbol` (символ) используется для создания уникальных идентификаторов в объектах. Мы упоминаем здесь о нём для полноты картины, изучим этот тип после объектов.

Оператор typeof

Оператор `typeof` возвращает тип аргумента. Это полезно, когда мы хотим обрабатывать значения различных типов по-разному или просто хотим сделать проверку.

У него есть две синтаксические формы:

```
// Обычный синтаксис
typeof 5 // Выведет "number"
// Синтаксис, напоминающий вызов функции (встречается реже)



typeof(5) // Также выведет "number"
```

Если передается выражение, то нужно заключать его в скобки, т.к. typeof имеет более высокий приоритет, чем бинарные операторы:

```
typeof 50 + " Квартир"; // Выведет "number Квартир"


typeof (50 + " Квартир"); // Выведет "string"
```

Другими словами, скобки необходимы для определения типа значения, которое получилось в результате выполнения выражения в них.

Вызов `typeof x` возвращает строку с именем типа:

```
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)
```

Последние три строки нуждаются в пояснении:

1. `Math` — это встроенный объект, который предоставляет математические операции и константы. Мы рассмотрим его подробнее в главе Числа. Здесь он служит лишь примером объекта.
2. Результатом вызова `typeof null` является `"object"`. Это официально признанная ошибка в `typeof`, ведущая начало с времён создания JavaScript и сохранённая для совместимости. Конечно, `null` не является объектом. Это специальное значение с отдельным типом.
3. Вызов `typeof alert` возвращает `"function"`, потому что `alert` является функцией. Мы изучим функции в следующих главах, где

заодно увидим, что в JavaScript нет специального типа «функция». Функции относятся к объектному типу. Но `typeof` обрабатывает их особым образом, возвращая `"function"`. Так тоже повелось от создания JavaScript. Формально это неверно, но может быть удобным на практике.

Итого

В JavaScript есть 8 основных типов данных.

Семь из них называют «примитивными» типами данных:

`number` для любых чисел: целочисленных или чисел с плавающей точкой; целочисленные значения ограничены диапазоном $\pm(2^{53}-1)$.

`bigint` для целых чисел произвольной длины.

`string` для строк. Строка может содержать ноль или больше символов, нет отдельного символьного типа.

`boolean` для `true`/`false`.

`null` для неизвестных значений – отдельный тип, имеющий одно значение `null`.

`undefined` для неприсвоенных значений – отдельный тип, имеющий одно значение `undefined`.

`symbol` для уникальных идентификаторов.

И один не является «примитивным» и стоит особняком:

`object` для более сложных структур данных.

Оператор `typeof` позволяет нам увидеть, какой тип данных сохранён в переменной.

Имеет две формы: `typeof x` или `typeof(x)`.

Возвращает строку с именем типа. Например, `"string"`.

Для `null` возвращается `"object"` – это ошибка в языке, на самом деле это не объект.

В следующих главах мы сконцентрируемся на примитивных значениях, а когда познакомимся с ними, перейдём к объектам.

Задачи

Шаблонные строки

важность: 5

Что выведет этот скрипт?

```
let name = "Ilya";

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

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


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

решение

Обратные кавычки позволяют вставить выражение внутри `${...}` в строку.

```
let name = "Ilya";

// выражение - число 1
alert( `hello ${1}` ); // hello 1

// выражение - строка "name"
alert( `hello ${"name"}` ); // hello name

// выражение - переменная, вставим её в строку
```

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