Javascript


HTML/CSS
NEW PROGAMMER
HTML/CSS
JAVASCRIPT
NEW PROGRAMMER
imgflip.com

Тег «script»

Программы на JavaScript могут быть вставлены в любое место HTML-документа с помощью тега **<script>**.

```
<script>
	alert( 'Привет, мир!' );
</script>

<script src="/path/to/script.js"></script>
```

Структура кода

Инструкции – это синтаксические конструкции и команды, которые выполняют действия. В нашем коде может быть столько инструкций, сколько мы захотим. Инструкции могут отделяться точкой с запятой.

```
alert('Привет');
alert('Мир');
```

В большинстве случаев точку с запятой можно не ставить, если есть переход на новую строку.

Комментарии

Со временем программы становятся всё сложнее и сложнее. Возникает необходимость добавлять комментарии, которые бы описывали, что делает код и почему. Комментарии могут находиться в любом месте скрипта. Они не влияют на его выполнение, поскольку движок просто игнорирует их.

```
// Этот комментарий занимает всю строку
/* Пример с двумя сообщениями. Это - многострочный комментарий. */
```

Строгий режим — "use strict"

Директива выглядит как строка: "use strict" или 'use strict'. Когда она находится в начале скрипта, весь сценарий работает в «современном» режиме.

```
"use strict";
// этот код работает в современном режиме
...
```

Переменные

Переменная – это «именованное хранилище» для данных. Мы можем использовать переменные для хранения товаров, посетителей и других данных.
Для создания переменной в JavaScript используйте ключевое слово let.
Приведённая ниже инструкция создаёт (другими словами, объявляет) переменную с именем «message»:

```
let message;
message = 'Hello'; // сохранить строку 'Hello' в переменной с именем message
```

Аналогия из жизни
Мы легко поймём концепцию «переменной», если представим её в виде «коробки» для данных с уникальным названием на ней.
Например, переменную message можно представить как коробку с названием "message" и значением "Hello!" внутри:

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.
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;
```

Семь из них называют «примитивными» типами данных:

- number для любых чисел: целочисленных или чисел с плавающей точкой; целочисленные значения ограничены диапазоном ±(253-1).
- 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.

Базовые операторы, математика

- Поддерживаются следующие математические операторы:
  - Сложение +,
  - Вычитание -,
  - Умножение *,
  - Деление /,
  - Взятие остатка от деления %,
  - Возведение в степень **.
- Взятие остатка %
- Возведение в степень **
- Сложение строк при помощи бинарного +
- Приведение к числу, унарный +
- Приоритет операторов
- Присваивание = возвращает значение

Операторы сравнения

Результат сравнения имеет логический тип
Все операторы сравнения возвращают значение логического типа:

- true – означает «да», «верно», «истина».
- false – означает «нет», «неверно», «ложь».

Сравнение строк

```
alert( 'Я' > 'А' ); // true
alert( 'Коты' > 'Кода' ); // true
alert( 'Сонный' > 'Сон' ); // true
```

Сравнение разных типов

```
alert( '2' > 1 ); // true, строка '2' становится числом 2
alert( '01' == 1 ); // true, строка '01' становится числом 1
```

Строгое сравнение
== и ===

Условное ветвление

Иногда нам нужно выполнить различные действия в зависимости от условий. Для этого мы можем использовать инструкцию if и условный оператор ?, который также называют оператором «вопросительный знак».

```
let accessAllowed;
if (age > 18) {
  accessAllowed = true;
} else {
  accessAllowed = false;
}

let accessAllowed = (age > 18) ? true : false;
```

Логические операторы

В JavaScript есть семь логических операторов:

- || (ИЛИ)
- && (И)
- ! (НЕ)
- ?? (Оператор нулевого слияния)

Несмотря на своё название, данные операторы могут применяться к значениям любых типов. Полученные результаты также могут иметь различный тип.

```
const result = a || b;
const result = a && b;
const result = !value;
const result = a ?? b;
```

Циклы while и for

```
let i = 0;
while (i < 3) { // выводит 0, затем 1, затем 2
  alert( i );
  i++;
}

for (let i = 0; i < 3; i++) { // выведет 0, затем 1, затем 2
  alert(i);
}
```

Функции

Зачастую нам надо повторять одно и то же действие во многих частях программы. Например, необходимо красиво вывести сообщение при приветствии посетителя, при выходе посетителя с сайта, ещё где-нибудь.
Чтобы не повторять один и тот же код во многих местах, придуманы функции. Функции являются основными «строительными блоками» программы.

```
function showMessage() {
  alert( 'Всем привет!' );
}
```

- Локальные переменные
- Внешние переменные
- Параметры
- Function Expression

Массивы

Объекты позволяют хранить данные со строковыми ключами. Это замечательно.
Но довольно часто мы понимаем, что нам необходима упорядоченная коллекция данных, в которой присутствуют 1-й, 2-й, 3-й элементы и т.д. Например, она понадобится нам для хранения списка чего-либо: пользователей, товаров, элементов HTML и т.д.
В этом случае использовать объект неудобно, так как он не предоставляет методов управления порядком элементов. Мы не можем вставить новое свойство «между» уже существующими. Объекты просто не предназначены для этих целей.
Для хранения упорядоченных коллекций существует особая структура данных, которая называется массив, Array.

```
let arr = new Array();
let arr = [];

let fruits = ["Яблоко", "Апельсин", "Слива"];
alert( fruits[0] ); // Яблоко
alert( fruits[1] ); // Апельсин
alert( fruits[2] ); // Слива
```

Объекты

Объекты же используются для хранения коллекций различных значений и более сложных сущностей. В JavaScript объекты используются очень часто, это одна из основ языка.

Объект может быть создан с помощью фигурных скобок {…} с необязательным списком свойств. Свойство – это пара «ключ: значение», где ключ – это строка (также называемая «именем свойства»), а значение может быть чем угодно.

Мы можем представить объект в виде ящика с подписанными папками. Каждый элемент данных хранится в своей папке, на которой написан ключ. По ключу папку легко найти, удалить или добавить в неё что-либо.

```
let user = {     // объект
  name: "John",  // под ключом "name" хранится значение "John"
  age: 30        // под ключом "age" хранится значение 30
};
```

DOM-дерево

Основой HTML-документа являются теги.

В соответствии с объектной моделью документа («Document Object Model», коротко DOM), каждый HTML-тег является объектом. Вложенные теги являются «детьми» родительского элемента. Текст, который находится внутри тега, также является объектом.

Все эти объекты доступны при помощи JavaScript, мы можем использовать их для изменения страницы.

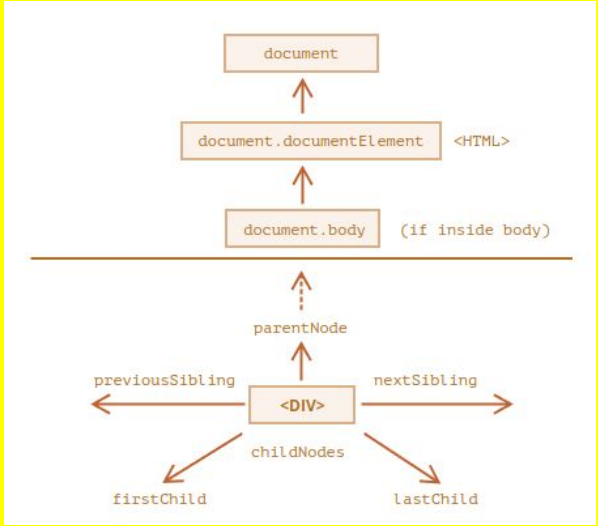
Например, document.body – объект для тега <body>.

Навигация по DOM-элементам

DOM позволяет нам делать что угодно с элементами и их содержимым, но для начала нужно получить соответствующий DOM-объект.

Все операции с DOM начинаются с объекта document. Это главная «точка входа» в DOM. Из него мы можем получить доступ к любому узлу.

Так выглядят основные ссылки, по которым можно переходить между узлами DOM:



Поиск: getElement*, querySelector*

```
// получить элемент
 let elem = document.getElementById('elem');

let elements = document.querySelectorAll('ul > li:last-child');
```

Браузерные события

Событие – это сигнал от браузера о том, что что-то произошло. Все DOM-узлы подают такие сигналы (хотя события бывают и не только в DOM).

События мыши:

- click – происходит, когда кликнули на элемент левой кнопкой мыши (на устройствах с сенсорными экранами оно происходит при касании).
- contextmenu – происходит, когда кликнули на элемент правой кнопкой мыши.
- mouseover / mouseout – когда мышь наводится на / покидает элемент.
- mousedown / mouseup – когда нажали / отжали кнопку мыши на элементе.
- mousemove – при движении мыши.

События на элементах управления:

- submit – пользователь отправил форму <form>.
- focus – пользователь фокусируется на элементе, например нажимает на <input>.

Клавиатурные события:

- keydown и keyup – когда пользователь нажимает / отпускает клавишу.

События документа:

- DOMContentLoaded – когда HTML загружен и обработан, DOM документа полностью построен и доступен.

CSS events:

- transitionend – когда CSS-анимация завершена.

Обработчики событий

addEventListener

Фундаментальный недостаток описанных выше способов назначения обработчика – невозможность повесить несколько обработчиков на одно событие.

Например, одна часть кода хочет при клике на кнопку делать её подсвеченной, а другая – выдавать сообщение.

Мы хотим назначить два обработчика для этого. Но новое DOM-свойство перезапишет предыдущее:

element.addEventListener(event, handler, [options]);