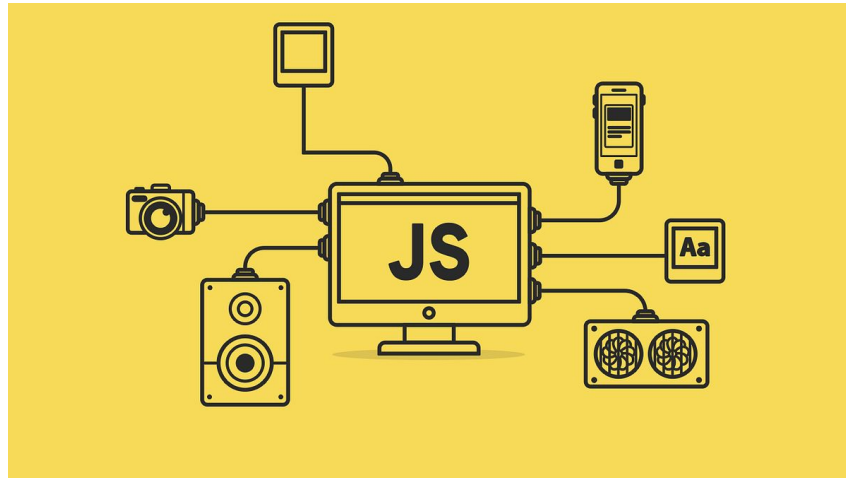


The slide features decorative geometric patterns in the top-left and bottom-right corners. These patterns consist of overlapping dark blue and light blue shapes, some with a dotted texture, and thin gold-colored lines forming geometric outlines.

JavaScript

Introduction

Знакомство с JavaScript




Изначально **JavaScript** был создан, чтобы «сделать веб-страницы живыми».

Программы на этом языке называются скриптами.

Они могут встраиваться в HTML и выполняться автоматически при загрузке веб-страницы.





Сегодня JavaScript может выполняться не только в браузере, но и на сервере или на любом другом устройстве, которое имеет специальную программу, называющуюся «движком» JavaScript.


У браузера есть собственный движок, который иногда называют «виртуальная машина JavaScript».

Разные движки имеют разные «кодовые имена». Например:

V8 – в Chrome, Opera и Edge.

SpiderMonkey – в Firefox.

...Ещё есть «Chakra» для IE, «JavaScriptCore», «Nitro» и «SquirrelFish» для Safari и т.д.



Движки сложны. Но основы понять легко.

1. Движок (встроенный, если это браузер) читает («парсит») текст скрипта.
2. Затем он преобразует («компилирует») скрипт в машинный язык.
3. После этого машинный код запускается и работает достаточно быстро.

Движок применяет оптимизации на каждом этапе. Он даже просматривает скомпилированный скрипт во время его работы, анализируя проходящие через него данные, и применяет оптимизации к машинному коду, полагаясь на полученные знания. В результате скрипты работают оче

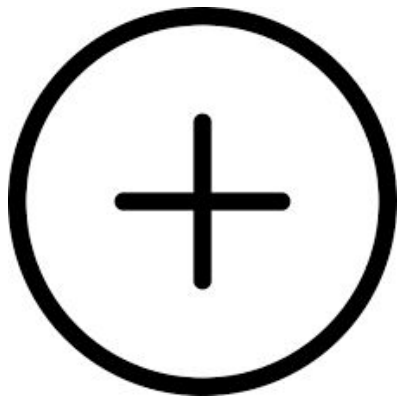


Что делает JavaScript особенным?

- Полная интеграция с HTML/CSS.
- Простые вещи делаются просто.
- Поддерживается всеми основными браузерами и включён по умолчанию.



Добавление JavaScript



Для добавления JavaScript в HTML, вы можно использовать тег `<script>`. Есть несколько способов включения JavaScript в HTML-документ.

1. Внутренний скрипт (в теле HTML):

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>JavaScript в HTML</title>
</head>
<body>

  <!-- Внутренний скрипт -->
  <script>
    // Ваш JavaScript код здесь
    console.log("Привет, это JavaScript внутри HTML!");
  </script>

</body>
</html>
```


2. Внешний файл скрипта (в голове HTML):

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>JavaScript в HTML</title>

  <!-- Внешний файл скрипта -->
  <script src="script.js"></script>
</head>
<body>
  <!-- Содержимое тела документа -->
</body>
</html>
```

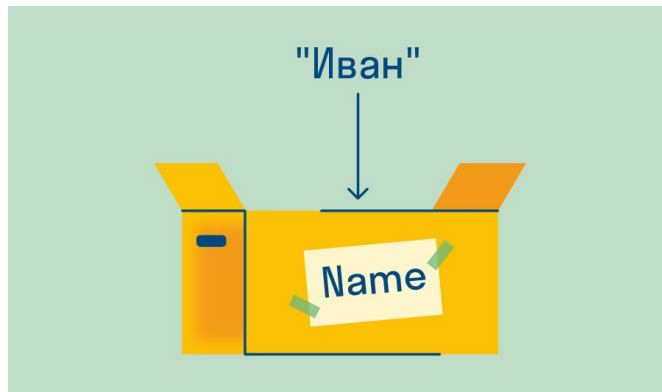
3. Внешний файл скрипта (в конце тела HTML):

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>JavaScript в HTML</title>
</head>
<body>

  <!-- Содержимое тела документа -->

  <!-- Внешний файл скрипта в конце тела документа -->
  <script src="script.js"></script>
</body>
</html>
```

Переменные



Переменные

JavaScript-приложению обычно нужно работать с информацией. Например:

1. **Интернет-магазин** – информация может включать продаваемые товары и корзину покупок.
2. **Чат** – информация может включать пользователей, сообщения и многое другое.

Переменные используются для хранения этой информации.

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»:

```
1 let message;
```

Теперь можно поместить в неё данные (другими словами, определить переменную), используя оператор присваивания =:

```
1 let message;  
2  
3 message = 'Hello'; // сохранить строку 'Hello' в переменной с именем message
```

Строка сохраняется в области памяти, связанной с переменной. Мы можем получить к ней доступ, используя имя переменной:

```
1 let message;  
2 message = 'Hello!';  
3  
4 alert(message); // показывает содержимое переменной
```

Аналогия из жизни

Например, переменную `message` можно представить как коробку с названием `"message"` и значением `"Hello!"` внутри:

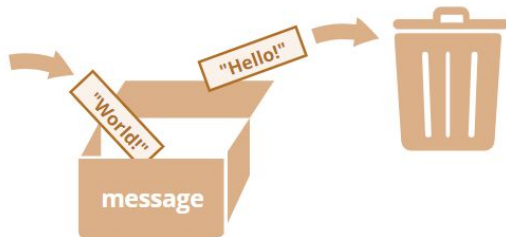


Мы можем положить любое значение в коробку.

Мы также можем изменить его столько раз, сколько захотим:

```
1 let message;
2
3 message = 'Hello!';
4
5 message = 'World!'; // значение изменено
6
7 alert(message);
```


При изменении значения старые данные удаляются из переменной:



Мы также можем объявить две переменные и скопировать данные из одной в другую.

```
1 let hello = 'Hello world!';  
2  
3 let message;  
4  
5 // копируем значение 'Hello world' из переменной hello в переменную message  
6 message = hello;  
7  
8 // теперь две переменные содержат одинаковые данные  
9 alert(hello); // Hello world!  
10 alert(message); // Hello world!
```

Имена переменных

В JavaScript есть два ограничения, касающиеся имён переменных:

1. Имя переменной должно содержать только буквы, цифры или символы \$ и _.
2. Первый символ не должен быть цифрой.

```
1 let userName;  
2 let test123;
```

```
1 let $ = 1; // объявили переменную с именем "$"  
2 let _ = 2; // а теперь переменную с именем "_"  
3  
4 alert($ + _); // 3
```

Константы

Чтобы объявить константную, то есть, неизменяемую переменную, используйте `const` вместо `let`:

```
1  const myBirthday = '18.04.1982';
```

Переменные, объявленные с помощью `const`, называются «константами». Их нельзя изменить. Попытка сделать это приведёт к ошибке:

```
1  const myBirthday = '18.04.1982';  
2  
3  myBirthday = '01.01.2001'; // ошибка, константу нельзя перезаписать!
```

Константы в верхнем регистре

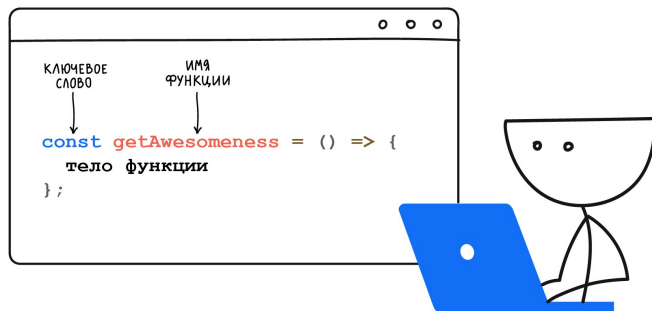
Широко распространена практика использования констант в качестве псевдонимов для трудно запоминаемых значений, которые известны до начала исполнения скрипта.


Названия таких констант пишутся с использованием заглавных букв и подчёркивания.

Например, сделаем константы для различных цветов в «шестнадцатеричном формате»:

```
1  const COLOR_RED = "#F00";
2  const COLOR_GREEN = "#0F0";
3  const COLOR_BLUE = "#00F";
4  const COLOR_ORANGE = "#FF7F00";
5
6  // ...когда нам нужно выбрать цвет
7  let color = COLOR_ORANGE;
8  alert(color); // #FF7F00
```

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.



Объявление функции

Для создания функций мы можем использовать объявление функции.

```
1 function showMessage() {  
2     alert( 'Всем привет!' );  
3 }
```

Вначале идёт ключевое слово `function`, после него имя функции, затем список параметров в круглых скобках через запятую

```
1 function имя(параметры) {  
2     ...тело...  
3 }
```

Вызов функции

Наша новая функция может быть вызвана по своему имени: **showMessage()**.

```
1 function showMessage() {  
2     alert( 'Всем привет!' );  
3 }  
4  
5 showMessage();  
6 showMessage();
```


Локальные переменные

Переменные, объявленные внутри функции, видны только внутри этой функции.

```
1 function showMessage() {  
2   let message = "Привет, я JavaScript!"; // локальная переменная  
3  
4   alert( message );  
5 }  
6  
7 showMessage(); // Привет, я JavaScript!  
8  
9 alert( message ); // <-- будет ошибка, т.к. переменная видна только внутри функции
```

Внешние переменные

У функции есть доступ к внешним переменным, например:

```
1 let userName = 'Вася';  
2  
3 function showMessage() {  
4   let message = 'Привет, ' + userName;  
5   alert(message);  
6 }  
7  
8 showMessage(); // Привет, Вася
```

Если одноимённая переменная объявляется внутри функции, тогда она перекрывает внешнюю.

Параметры

Мы можем передать внутрь функции любую информацию, используя параметры.

В нижеприведённом примере функции передаются два параметра: **from** и **text**.

```
1 function showMessage(from, text) { // параметры: from, text
2   alert(from + ': ' + text);
3 }
4
5 showMessage('Аня', 'Привет!'); // Аня: Привет! (*)
6 showMessage('Аня', "Как дела?"); // Аня: Как дела? (**)
```

Возврат значения

Функция может вернуть результат, который будет передан в вызвавший её код.

Простейшим примером может служить функция сложения двух чисел:

```
1 function sum(a, b) {  
2   return a + b;  
3 }  
4  
5 let result = sum(1, 2);  
6 alert( result ); // 3
```

Выбор имени функции

Функция – это действие. Поэтому имя функции обычно является глаголом. Оно должно быть кратким, точным и описывать действие функции, чтобы программист, который будет читать код, получил верное представление о том, что делает функция.

Например, функции, начинающиеся с "show" обычно что-то показывают.

Функции, начинающиеся с...

- "get..." – возвращают значение,

- "calc..." – что-то вычисляют,

- "create..." – что-то создают,

- "check..." – что-то проверяют и возвращают логическое значение, и т.д.

Выбор имени функции

Примеры имен

```
1 showMessage(..)    // показывает сообщение
2 getAge(..)         // возвращает возраст (получая его каким-то образом)
3 calcSum(..)        // вычисляет сумму и возвращает результат
4 createForm(..)     // создаёт форму (и обычно возвращает её)
5 checkPermission(..) // проверяет доступ, возвращая true/false
```

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:

```
1 // Не будет ошибкой
2 let message = "hello";
3 message = 123456;
```


Число

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`.

```
1 let n = 123;  
2 n = 12.345;
```

Число

- `Infinity` представляет собой математическую бесконечность ∞ . Это особое значение, которое больше любого числа.

Мы можем получить его в результате деления на ноль:

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

Или задать его явно:

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

Число

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

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

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

```
1 alert( NaN + 1 ); // NaN
2 alert( 3 * NaN ); // NaN
3 alert( "не число" / 2 - 1 ); // NaN
```

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 $1.7976931348623157 \cdot 10^{308}$), но за пределами безопасного диапазона целых чисел $\pm (2^{53}-1)$ будет ошибка точности

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

```
1 // символ "n" в конце означает, что это BigInt
2 const bigInt = 1234567890123456789012345678901234567890n;
```

Строка

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

```
1 let str = "Привет";  
2 let str2 = 'Одинарные кавычки тоже подойдут';  
3 let phrase = `Обратные кавычки позволяют встраивать переменные ${str}`;
```

Строка

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:

```
1 let name = "Иван";
2
3 // Вставим переменную
4 alert( `Привет, ${name}!` ); // Привет, Иван!
5
6 // Вставим выражение
7 alert( `результат: ${1 + 2}` ); // результат: 3
```

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».

```
1 let nameFieldChecked = true; // да, поле отмечено
2 let ageFieldChecked = false; // нет, поле не отмечено
```

Булевы значения также могут быть результатом сравнений:

```
1 let isGreater = 4 > 1;
2
3 alert( isGreater ); // true (результатом сравнения будет "да")
```

Значение «null»

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

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

```
1 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».

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`:

```
1 let age;  
2  
3 alert(age); // выведет "undefined"
```

Обычно `null` используется для присвоения переменной «пустого» или «неизвестного» значения, а `undefined` – для проверок, была ли переменная назначена.

Объекты

Следующий тип данных - object, он используется для более сложных структур данных.

В JavaScript объекты представляют собой коллекции ключ-значение, где ключи являются строками или символами, а значениями могут быть любые типы данных, они относятся к типу данных - **Object**.

```
const person = {  
  name: 'John',  
  age: 30,  
  gender: 'male'  
};
```

Функции и массивы также относят к сложному типу данных - **Object**

Символы

Символ (Symbol) в JavaScript является уникальным и неизменным примитивным типом данных. Каждый созданный символ уникален, и он может быть использован в качестве уникального идентификатора для свойств объекта. Создание символа происходит с использованием функции `Symbol()`.

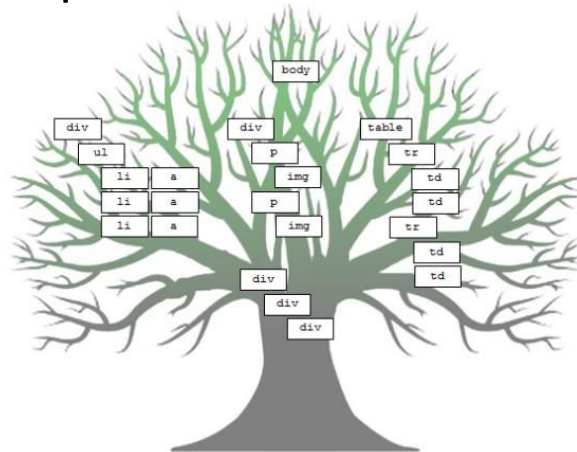
```
// Создание символа
const mySymbol = Symbol();

// Использование символа в качестве ключа для свойства объекта
const obj = {
  [mySymbol]: 'Hello, Symbol!'
};

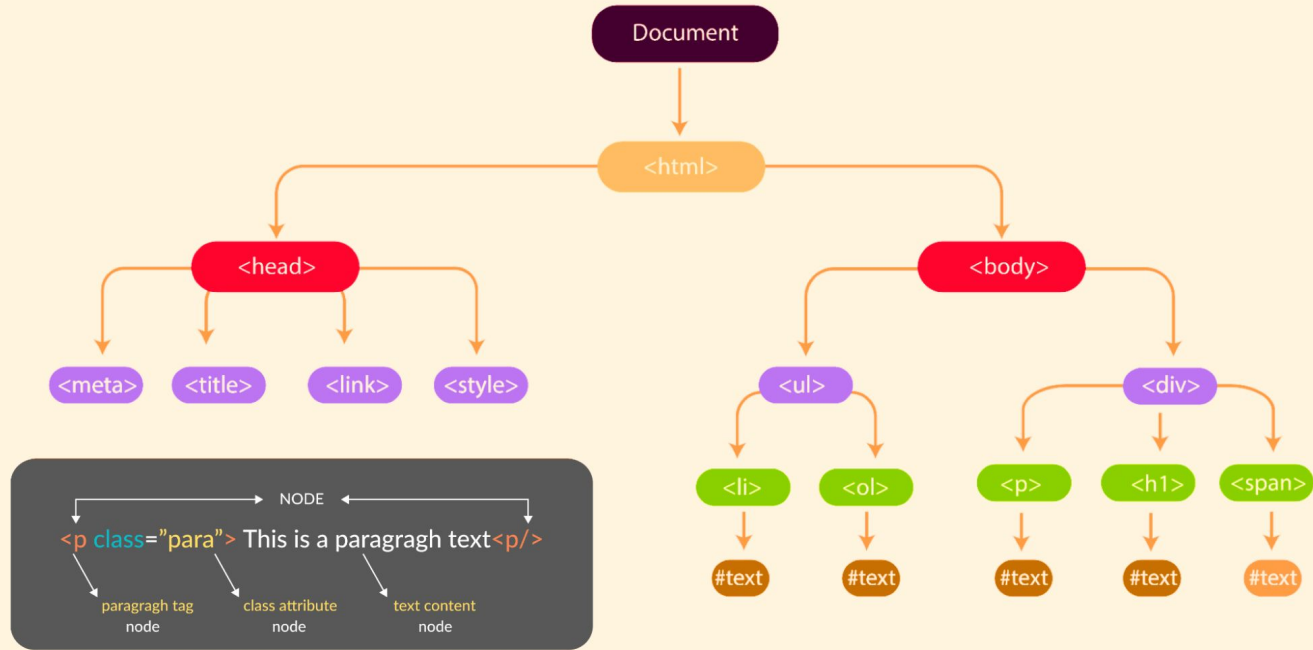
console.log(obj[mySymbol]); // Hello, Symbol!
```


DOM (Document Object Model)– это объектная модель документа, которую браузер создает в памяти компьютера на основании HTML-кода.

Иными словами, это представление HTML-документа в виде дерева тегов. Такое дерево нужно для правильного отображения сайта и внесения изменений на страницах с помощью JavaScript



The DOM Structure/ DOM TREE



Возьмем простой документ:
узлов:

```
<!doctype html>
<html lang="en">
  <head>
    <title>My first web page</title>
  </head>
  <body>
    <h1>Hello, world!</h1>
    <p>How are you?</p>
  </body>
</html>
```

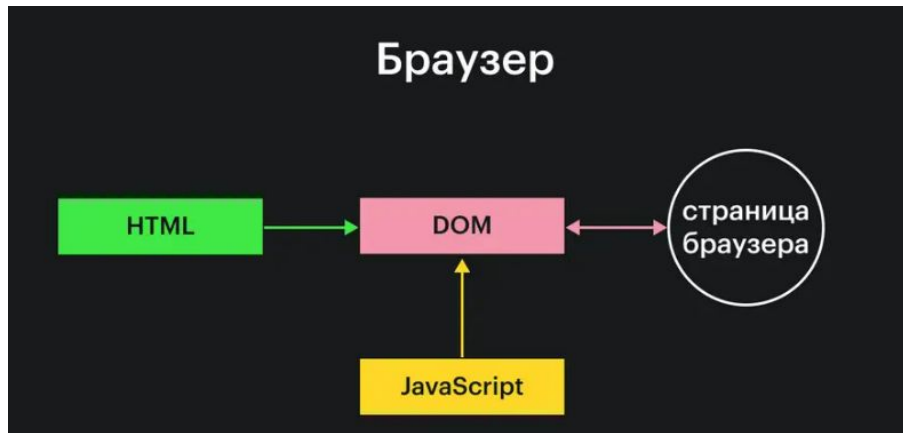
Он может быть представлен в виде дерева



DOM позволяет управлять HTML-разметкой из JavaScript-кода.

Управление обычно состоит из:

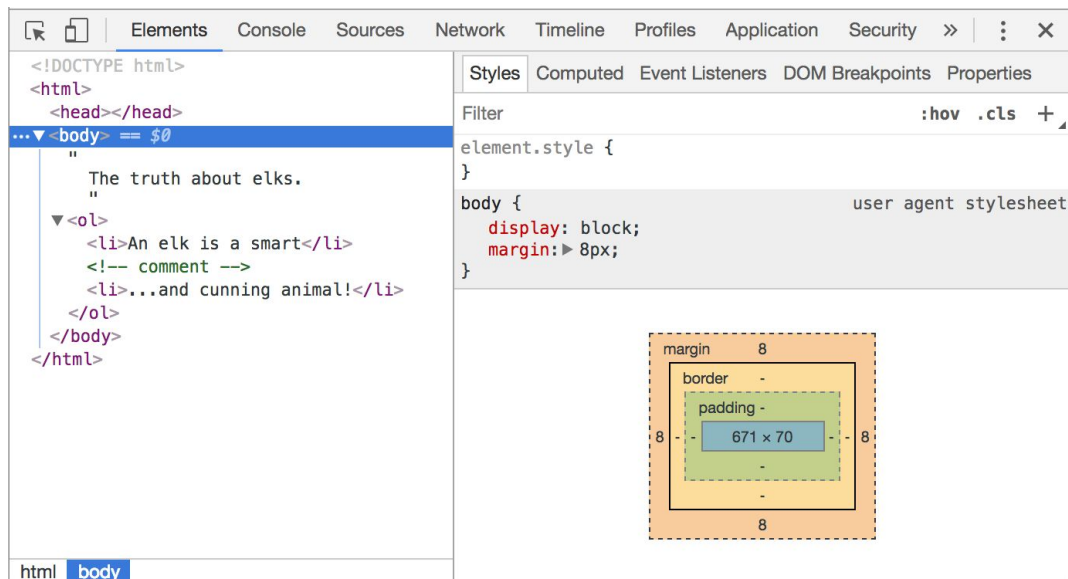
- добавления элементов
- удаления элементов
- изменения стилей и содержимого элементов



Способ исследовать DOM – это использовать инструменты разработчика браузера. Это то, что мы каждый день делаем при разработке.

Для этого откройте страницу, включите инструменты разработчика и перейдите на вкладку Elements.

Выглядит примерно так:



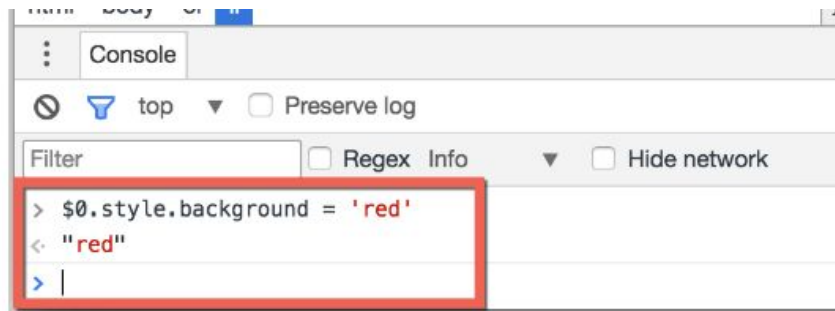
Взаимодействие с консолью

При работе с DOM нам часто требуется применить к нему JavaScript. Например: получить узел и запустить какой-нибудь код для его изменения, чтобы посмотреть результат.

- На вкладке Elements выберите текстовый элемент.
- Нажмите **Esc** – прямо под вкладкой Elements откроется Console.

Последний элемент, выбранный во вкладке Elements, доступен в консоли как `$0`; предыдущий, выбранный до него, как `$1` и т.д.

- Теперь мы можем запускать на них команды. Например `$0.style.background = 'red'` сделает выбранный элемент красным

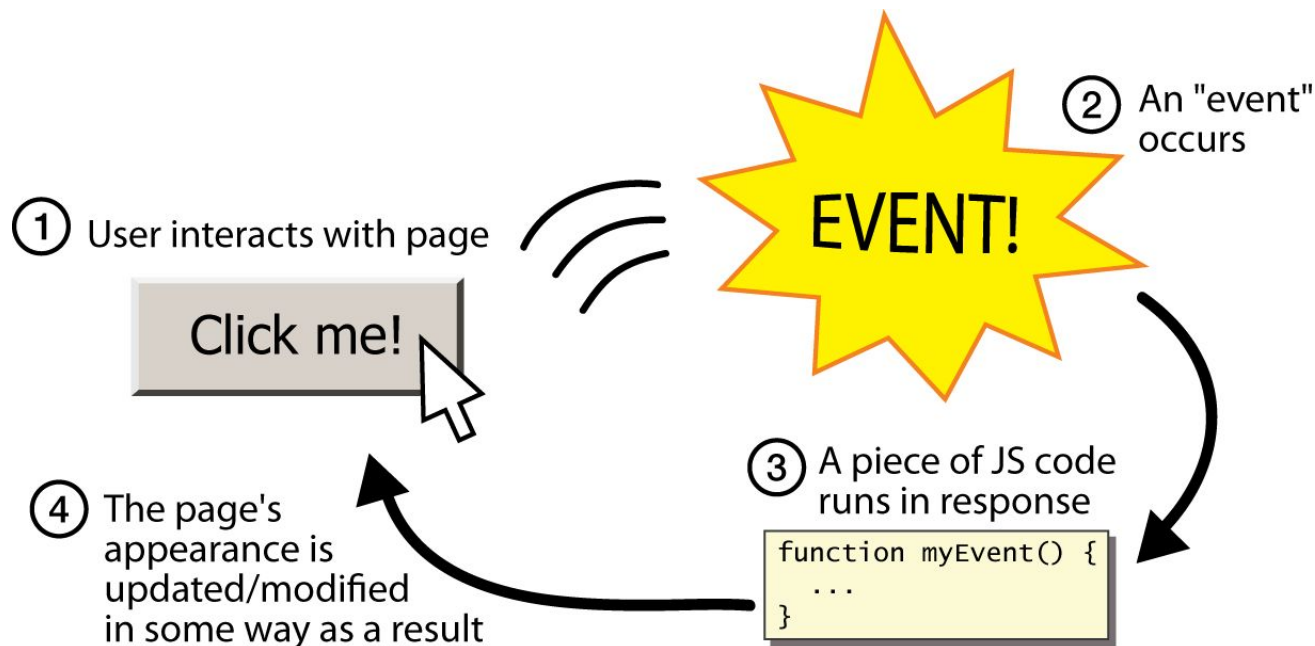


Поиск элемента по ID

Задача: получить элемент с id="elem"

```
let elem = document.getElementById('elem');
```

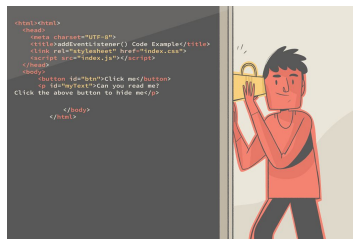
Любой DOM элемент запускает событие, когда мы с ним как-то взаимодействуем (кликаем, наводим мышь и др.). Обработчики событий в JS используются для того, чтобы реагировать на эти события.



Чтобы "повесить" обработчик событий на наш элемент button, нужно использовать специальный метод - **addEventListener**. Этот метод принимает 2 аргумента:

1. **Тип события** (мы будем "слушать" событие "click").
2. Так называемую **колбэк** (callback) функцию, которая запускается после срабатывания нужного события.

```
element.addEventListener('click', handleClickFunction)
```



Пример

Найдём кнопку на странице и будем выводить сообщение в консоль, когда произошёл клик по этой кнопке.

```
const element = document.getElementById('button')
```

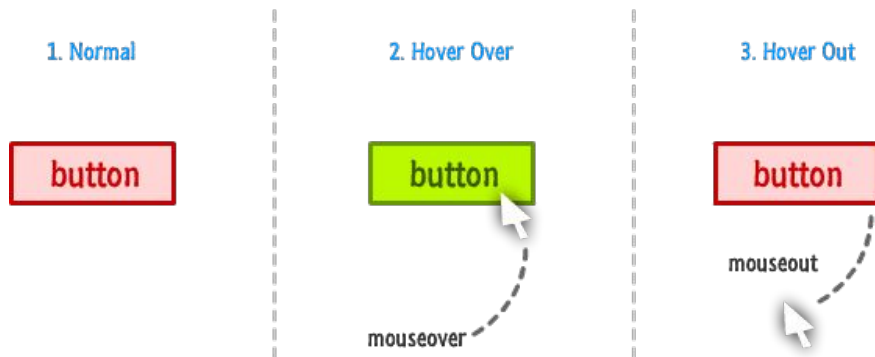
```
element.addEventListener('click', function () {  
  console.log('Произошло событие')  
})
```

Типы событий

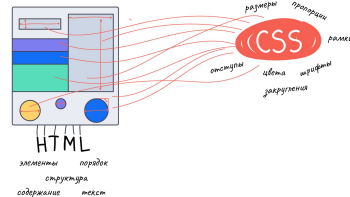
eventType (первый аргумент `addEventListener`) - строка, содержащая название события.

Наиболее популярные события:

- 'click'
- 'change'
- 'submit'
- 'keydown'
- 'keyup'
- 'mousemove'
- 'mouseenter'
- 'mouseleave'



Изменение стилей



HTML DOM позволяет JavaScript изменять стиль HTML элементов.

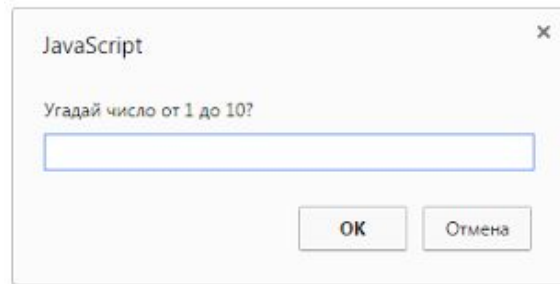
Синтаксис:

```
document.getElementById(id).style.свойство = новый стиль
```

Применение:

```
document.getElementById("p2").style.color = "blue";
```


Встроенные функции. Взаимодействие с пользователем



A screenshot of a JavaScript alert dialog box. The title bar reads "JavaScript" with a close button (X) on the right. The main text inside the box says "Угадай число от 1 до 10?". Below the text is a single-line text input field. At the bottom right of the dialog are two buttons: "ОК" and "Отмена".

console.log()

Чтобы вывести что-то на консоль из нашего кода, существует функция **console.log**.

Обычный пользователь сайта не увидит такой вывод, так как он в консоли. Чтобы увидеть его, либо откройте консольную панель инструментов разработчика, либо нажмите Esc, находясь в другой панели: это откроет консоль внизу.

```
let message = "Привет, мир!";  
console.log(message);
```

alert()

- alert используется для вывода всплывающего диалогового окна с сообщением.
- Принимает один параметр - текст сообщения.

Она показывает сообщение и ждёт, пока пользователь нажмёт кнопку «ОК».

```
1 alert("Hello");
```

prompt()

- **prompt** используется для вывода всплывающего диалогового окна с полем для ввода текста.
- Принимает два параметра: текст сообщения и необязательное значение по умолчанию для поля ввода.

```
1 let age = prompt('Сколько тебе лет?', 100);  
2  
3 alert(`Тебе ${age} лет!`); // Тебе 100 лет!
```

confirm()

Функция **confirm** отображает модальное окно с текстом вопроса `question` и двумя кнопками: ОК и Отмена.

```
1 let isBoss = confirm("Ты здесь главный?");  
2  
3 alert( isBoss ); // true, если нажата ОК
```