

# Тема-3. JavaScript

Что тут есть?

JS: переменные и константы, типы данных, операторы, встроенные объекты, условные и циклические конструкции, функции и объекты, ООП, исключения, отладка, логирование, асинхронность, Web-API, документирование кода, версии JS

# JavaScript

Первая версия языка разработана в 1996 году для использования в браузере Netscape.

Сегодня это базовый язык для фронтенд-логики сайтов. Над ним есть много надстроек, например: TypeScript, CoffeeScript.

JavaScript (или JS) может использоваться не только в браузере, но и в других средах, например на сервере (см. NodeJS).

# JavaScript



Авторство: Darcy Padilla.  
<https://web.archive.org/web/20140209081556/http://blog.mozilla.org/press/bios/brendan-eich/>  
[https://web.archive.org/web/20131108073412/https://blog.mozilla.org/press/files/2012/04/Thumbnail-Full\\_Eich\\_04.jpg](https://web.archive.org/web/20131108073412/https://blog.mozilla.org/press/files/2012/04/Thumbnail-Full_Eich_04.jpg), CC BY-SA 3.0,  
<https://commons.wikimedia.org/w/index.php?curid=31783773>

**Брендан Эйх – создатель JavaScript**

# JavaScript

В 2011 году появился альтернативный язык для фронтенд-логики – Dart.

Вообще говоря, он может компилироваться и исполняться без JS, но поскольку браузеры поддерживают только JS, то этот язык также переводится в JS (как и TypeScript и другие надстройки).

# JavaScript

JavaScript – это динамический язык, то есть выполняется код на лету, без предварительной компиляции, переменные не имеют строго закрепленного типа данных.

JavaScript – расширяемый язык, то есть у него есть ядро к которому можно добавлять некоторые дополнительные возможности.

# JavaScript

Из-за того, что каждый браузер может в какой-то мере самостоятельно реализовывать движок JS, было принято решение стандартизировать язык.

JavaScript стандартизирован организацией ECMA (European Computer Manufacturers Association). Стандартизация называется ECMAScript.

Во всех средах, которые поддерживают ECMAScript JS работает одинаково.

# JavaScript

Первая версия ECMAScript вышла в 1997 году.  
Сокращенно эту версию называют ES1.

В 2009 году вышла версия ES5 (ещё её называют ES2009).  
Это базовая версия, которую мы будем рассматривать.  
Более старые версии мало где встречаются, а ES5  
поддерживается всеми современными браузерами.

# JavaScript

Все версии JS имеют обратную совместимость с предыдущими, то есть старый код всегда поддерживается в новых версиях.

Взамен, есть некоторые неудобные вещи, которые нельзя убрать из языка.

Перейдем к рассмотрению деталей языка.



# JavaScript: переменные

Есть три способа объявить переменную в JS: let, const и var.

Ключевое слово var не рекомендуется использовать, оно осталось из прошлого. Минус var в том, что:

- область видимости ограничена только функцией и скриптом (но не любыми блоками кода);
- var допускает повторное объявление (ошибки не будет);
- переменная автоматически всплывает в начало функции или скрипта при объявлении (фактически, если она объявлена не в начале, то она доступна до объявления). Это так называемый hoisting (но он бывает и полезен, например, при использовании функций).

# JavaScript: переменные

Объявления с помощью let и const не обладают такими недостатками.

При этом `let` и `const` отличаются друг от друга только тем, что `const` предназначено для константы (такую переменную невозможно перезаписать), а `let` для обычной переменной.

# JavaScript: переменные

Примеры объявления переменных в JS:

- let data = {};
- const MAX\_VALUE = 8;
- let minVal = 0, maxVal = 22, stepSize = 2;
- let queryAnswer;

// как и во многих других языках в конце оператора нужно  
ставить ;

// регистр в названиях важен (data и Data – разные переменные)

# JavaScript: окончание оператора

Точка с запятой не всегда обязательна в JS (она как бы автоматически вставляется при интерпретации кода).  
Тут можно про это (да и не только про это) почитать подробно:  
<https://262.ecma-international.org/6.0/#sec-automatic-semicolon-insertion>

Люди делятся на два типа:

- надо ставить везде точки с запятой;
- где можно надо не ставить их.

# JavaScript: типы данных

В JS есть несколько типов данных:

- Number – число (любое, хоть целое, хоть дробное – это один тип).
- String – строка (указывается в двойных или одинарных кавычках, какие брать не важно, но лучше использовать везде одинаковые. И конечно вот так „str’ нельзя).
- Boolean – булево значение (значения true или false).
- Array – массив (динамический, резервировать память не надо, внутри это Object)
- Object – объект (Объект это по сути словарь).
- Undefined – неопределённое значение (переменные, которым не присвоено значение имеют этот тип).
- Null – пустое значение (не то же самое, что undefined, но смысл похожий).

// Примечание. Некоторые другие типы данных опущены здесь

# JavaScript: типы данных

В JS есть ссылочные типы данных и примитивные.

К ссылочным относится объект и все типы, которые на нем основаны (например, Array).

Все остальные типы – примитивные.

Рассмотрим подробнее разные типы данных.

# JavaScript: типы данных

Тип данных Number. Интересные особенности:

- тип для любых чисел не больше, чем 2 в 53 степени (Для чисел больше есть тип BigInt);
- значение NaN – если в результате какой-то операции получилось что-то не определённое;
- значение Infinity – если число бесконечно;
- методы Number.parseInt() и Number.parseFloat() для получения числа из строки;
- метод Number.toFixed() для установки конкретного количества цифр в дробной части;
- 0b, 0o и 0x для 2-, 8- и 16 систем счисления;
- приведение к number: +variable.

# JavaScript: типы данных

Тип данных String. Интересные особенности:

- Можно использовать и двойные, и одинарные кавычки;
- Можно использовать обратные кавычки ``` для создания шаблонной строки: ``data: ${data}``;
- `"some string".length` – длина строки;
- `"some string"[i]` – для обращения по индексу;
- строки можно сравнивать;
- строки можно складывать с помощью `+`;
- `\` для продолжения строки с новой строки;



# JavaScript: типы данных

- методы indexOf(), includes(), endsWith(), startsWith() для поиска подстрок;
- методы replace() и replaceAll() для замены подстрок;
- метод slice() для извлечения подстроки;
- метод split() для разбиения строки по определённому разделителю в массив;
- методы toLowerCase() и toUpperCase() для изменения регистра;
- метод trim() для обрезания пробельных символов по краям;
- приведение к string: `\${variable}` или variable + ""

# JavaScript: типы данных

Тип данных Boolean. Интересные особенности:

- только 2 значения: true и false;
- приведение к boolean: !!variable;
- к false приводятся: 0, NaN, null, undefined, "", false;
- все остальные значения приводятся к true.

# JavaScript: типы данных

Тип данных Object. Интересные особенности:

- фактически это единственный ссылочный тип, все остальные типы (Array, Date и т.д.) основаны на нем;
- формат объекта: {key1: value1, key2: value2, ...};
- может содержать элементы разного типа;
- Object.keys(obj) – получение списка ключей объекта;
- Object.values(obj) – получение списка значений объекта;
- hasOwnProperty() – true, если объект имеет собственное свойство (не наследуемое), иначе false;
- иногда объект нужно обернуть в (), чтобы JS не посчитал его блоком кода.

# JavaScript: типы данных

Тип данных Array. Интересные особенности:

- формат массива: [val1, val2, ...];
- может содержать элементы разного типа;
- нумерация элементов с нуля;
- [...].length – длина массива;
- concat() – конкатенация массивов;
- filter() => {} – фильтрация массива;
- find() и findIndex() – поиск элементов массива;
- flat() – разворачивание массива в одномерный;
- forEach() => {} – выполнение некоторой функции для каждого элемента массива;

# JavaScript: типы данных

- map() => {} – то же самое, что и `forEach`, но возвращает массив из результатов выполнения;
- includes(), indexOf() – поиск элементов;
- join() – преобразование в строку;
- push() и unshift() – добавить элемент (в начало или конец);
- pop() и shift() – убрать элемент из массива (последний или первый);
- reverse() – инвертировать массив на месте (`inplace`);
- slice() – взять срез массива;
- sort() – сортировать массив на месте;
- splice() – добавить или удалить определенные элементы.

# JavaScript: типы данных

Тип данных Undefined. Имеет единственное значение undefined. Это специальное значение, которое присваивается неинициализированным переменным и свойствам объектов.

Функции возвращают `undefined`, если они ничего не возвращают.

# JavaScript: типы данных

Тип данных Null. Имеет единственное значение null.  
Отсутствующее значение. null !== undefined. Как правило  
указывается, если ожидается объект, но его нет.

То есть разница с `undefined` в том, что `null` – определённо  
отсутствующее значение, а `undefined` – неопределённое значение.

# JavaScript: объект Math

В JS есть встроенный объект Math, который предоставляет разные математические функции (например, тригонометрические, логарифмические) и константы ( $\pi$ ,  $e$ ).

**Документация по Math:**

[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Math](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math)



# JavaScript: объект Date

Также в JS есть объект Date. С помощью него можно представить определенный момент времени в различных форматах.

Внутри Date содержит просто число (Number), выражающее количество миллисекунд с момента 1 января 1970 года.

Если задать отрицательное число, то будет момент до 1 января 1970.

Объект позволяет также получить текущие дату и время, конвертировать дату и время в разные форматы.

# JavaScript: комментарии

**В JS есть два вида комментариев:**

- `/*многострочный*/`**
- `//однострочный`**

# JavaScript: условия

Есть две условные конструкции в JS:

```
if (condition) {  
    //code  
} else if (condition) {  
    //code  
} else {  
    //code  
}
```

Если в условии или case только один оператор, {} можно не ставить

```
switch (variable) {  
    case value1: {  
        //code  
        break;  
    }  
    case value2: {  
        //code  
        break;  
    }  
    default: {  
        //code  
    }  
}
```

# JavaScript: циклы

Как и в других языках, в JS есть циклы нескольких видов:

- for (let i = 0; i < someArray.length; i++) { ... }
- while (condition) { ... }
- do { ... } while (condition);

Если оператор только один, то {} также можно не ставить.

Пропуск итерации: continue;

Досрочное завершение цикла: break;

# JavaScript: циклы

Есть ещё пара циклов, специфичных для JS:

- for (let propName in someObject) { ... } -- для прохода по свойствам объекта.
- for (let val of someIterable) { ... } -- для прохода по значениям итерируемого объекта.

Итерируемые объекты: массив, строка, объект и т.д.

# JavaScript: операторы

Далее рассмотрим некоторые часто используемые операторы.

# JavaScript: операторы

## Операторы сравнения:

- ===, !== - без приведения типов (5 !== "5").
- ==, != - с приведением типов (5 == "5").
- in - true, если свойство есть в объекте, иначе false.
- <, <=, >, >=

# JavaScript: операторы

## Логические операторы:

- !
- ||
- &&



# JavaScript: операторы

Тернарный оператор:

- ?:

Пример: let color = isColorWhite(data) ? "white" : "black"

# JavaScript: операторы

## Арифметика:

- +
- =
- \*
- /
- % – остаток от деления
- \*\* – возведение в степень

# JavaScript: операторы

## Побитовые операторы:

- & – И
- | – ИЛИ
- ^ – исключающее ИЛИ (истина, если операнды разного значения)
- >> – сдвиг вправо
- << – сдвиг влево
- >>> – беззнаковый сдвиг вправо (бит знака двигается со всеми)

# JavaScript: операторы

## Операторы присваивание:

- =
- +=
- -=
- \*=
- /=
- %=
- \*\*=

Например, let data += 2; то же самое, что let data = data + 2;

# JavaScript: операторы

Деструктурирующее присваивание:

- let [a, b] = [1, 2]; // a === 1, b === 2
- let {a, b: anotherVar} = {a: 12, b: 3}; // a === 12, anotherVar === 3

То есть, можно объявить переменные сразу с присваиванием значений из некоторого объекта.

# JavaScript: операторы

Оператор typeof позволяет проверить тип переменной.

Например: typeof 5 === "number" истина.

# JavaScript: операторы

Оператор delete позволяет удалить из объекта свойство.

Например:

```
let someObject = {a: 12, b: 3};  
delete someObject.b;  
console.log(someObject); // {a: 12}
```

# JavaScript: операторы

Операторы инкремента и декремента:

- i++ - постфиксный инкремент.
- i-- - постфиксный декремент.
- ++i - префиксный инкремент.
- --i - префиксный декремент.



# JavaScript: операторы

Оператор распространения (spreading):

- ...variable – позволяет разобрать массив или объект на отдельные аргументы.

Например:

```
let arr = [12, 13, 14];  
let arr2 = [10, 11, ...arr];  
console.log(arr2); // [10, 11, 12, 13, 14]
```

```
let obj = {a: 33, b: 13};  
let obj2 = {...obj, c: 12};  
console.log(obj2); // {a: 33, b: 13, c: 12}
```

# JavaScript: операторы

Операторы доступа `.`, `?.` и `[]` используются для получения доступа к элементам объектов.

Примеры:

- `obj.a` – доступ к свойству `a` объекта `obj`.
- `obj?.a` – то же самое, что выше, но если `obj === undefined`, то здесь возвратится `undefined`, а выше произойдет ошибка `AttributeError`.
- `obj["a"]` – то же самое, что и `obj.a`; этот способ полезен, если Вы не знаете заранее название свойства (можно вставить переменную) или если свойство не удовлетворяет правилам именования идентификатора (например, число)

# JavaScript: операторы

Оператор регулярных выражений /.../ используется для записи регулярного выражения.

Например: let regex = /\d{4}\.\d\d.\d\d \d+:\d+/;

После регулярного выражения можно добавлять флаги, например: /.../gi – глобальный поиск (найдет все совпадения) с игнорированием регистра.

# JavaScript: функции

В JS есть два способа определить функцию:

- function funcName() { ... } – обычная функция, „всплывающая“.
- const funcName = () => { ... } – „стрелочная“ функция, „не всплывает“.

В JS функция может быть присвоена переменной. Соответственно, функцию можно вернуть из функции, передать как аргумент и т.д.

# JavaScript: функции

Если некоторые параметры не были переданы в функцию, они будут иметь значение `undefined`.

Можно передать и больше аргументов, чем функция принимает.

Все переданные аргументы находятся во встроенном в функцию объекте `arguments`. Через него можно получить доступ даже к тем аргументам, которые функция не принимает.

Объект `arguments` подобен массиву, конкретный аргумент можно получить по индексу.

# JavaScript: функции

Как и в других языках, в JS есть так называемые генераторы – функции, которые помнят который раз их вызывают.

Синтаксис:

```
function* someGenerator (someParams...) {  
  //code  
  yield someResult;  
}
```

# JavaScript: функции

Как уже известно, переменные и функции могут быть не видны глобально. Их видимость можно ограничить внешней функцией. Например:

```
function externalFunc() {  
    function internalFunc() { //такие функции называют замыканиями  
        //code  
    }  
    //internalFunc существует  
}  
//internalFunc не существует
```

# JavaScript: функции

Функция может устанавливать значения параметров по умолчанию:

```
function someFunc(param1, param2=4) {
```

```
...
```

```
}
```

Также можно использовать „остаточный параметр“ (туда запишутся все оставшиеся переданные значения):

```
function someFunc(param1, ...restParams) {
```

```
...
```

```
}
```



# JavaScript: strict mode

JS иногда позволяет писать код, приводящий к сложно обнаружимым ошибкам. Поэтому в JS появился так называемый строгий режим (strict mode), который запрещает многие плохие практики.

Чтобы активировать строгий режим в блоке кода (например, в функции), нужно использовать конструкцию:  
"use strict";

# JavaScript: ООП

Для создания множества объектов определенной структуры можно использовать функцию-конструктор.

Можно сказать, что это аналог класса в других языках.

Функция-конструктор позволяет задать структуру объекта, а при вызове создавать конкретные объекты.

# JavaScript: ООП

Пример функции-конструктора:

```
function Car(brand, model, year) {  
  this.brand = brand;  
  this.model = model;  
  this.year = year;  
}
```

...

```
let car = new Car("Volvo", "S40", 2012);
```

# JavaScript: ООП

В JS прототипная модель наследования. То есть, есть конкретный объект, а есть его прототип – объект, от которого первый наследуется.

В JS нет строгого разделения на классы и экземпляры.

Множественного наследования нет, но можно использовать миксины (о них далее).

# JavaScript: ООП

Доступ к свойствам прототипа происходит через специальное свойство prototype.

Если в свойство прототипа что-то записать, то это изменение применится для всех объектах, которые основаны на этом прототипе.

Пример: someObj.prototype.someProp = () => {...}

# JavaScript: ООП

Свойства объекта, если говорить в контексте ООП, могут быть полями или методами. Поля – это свойства, значения которых являются данными, методы – это свойства, значениями которых являются функции.

# JavaScript: ООП

Как и в других языках, в JS также можно определять геттеры и сеттеры для полей:

```
var someObj = {  
  a: 7,  
  get getA() {  
    return this.a;  
  },  
  set updateA(x) {  
    this.a = x;  
  }  
};
```

# JavaScript: ООП

В JS есть синтаксический сахар для создания объектов – классы. Фактически, внутри классы это функции.

Пример класса:

```
class Rectangle {  
  constructor(height, width) {  
    this.height = height;  
    this.width = width;  
  }  
}
```



# JavaScript: ООП

Классы в JS не „всплывают“, их нужно определить обязательно перед использованием.

Родительский конструктор вызывается с помощью специальной функции super().

Можно объявлять статические методы и поля, используя ключевое слово static.

# JavaScript: ООП

**Впоследствии планируется добавить в JS поддержку приватных и публичных полей, но пока этого нет.**

# JavaScript: ООП

**Наследование классов делается с помощью ключевого слова `extends`:**

```
class Dog extends Animal {  
  constructor(name) {  
    super(name);  
  }  
  
  speak() {  
    console.log(`${this.name} лает.`);  
  }  
}
```

# JavaScript: ООП

В JS можно обогащать классы с помощью миксинов (mix-ins, примеси). Пример определения миксина:

```
let calculatorMixin = Base => class extends Base {  
  calc() {...}  
};
```

# JavaScript: ООП

Пример использования миксинов:

class SomeClass {...}

class NewClass extends calculatorMixin(SomeClass) {...}

# JavaScript: исключения

Обработка исключений в JS происходит с использованием следующей конструкции:

```
try {  
  //code  
  if (comeCondition) {  
    throw someError;  
  }  
  //code  
} catch (e) {  
  //on-exception code  
} finally {  
  //exception-independent code  
}
```

# JavaScript: исключения

Ключевое слово `throw` позволяет бросить в качестве исключения любую информацию. Например, строку:

```
throw "Ошибка";
```

Но хорошей практикой считается выбрасывание специальных объектов ошибок. Базовый объект для этого – Error. Пример:

```
throw Error("Ошибка");
```

# JavaScript: исключения

**Есть также другие встроенные объекты ошибок в JS, например:**

- `InternalError`**
- `RangeError`**
- `ReferenceError`**
- `SyntaxError`**
- `TypeError`**
- `URIError`**



# JavaScript: исключения

В блоке catch какая конкретно ошибка произошла можно с помощью ключевого слова `instanceof`. Пример:

```
try {  
    //code  
} catch (e) {  
    if (e instanceof EvalError) {  
        //code  
    } else if (e instanceof RangeError) {  
        //code  
    }  
}
```

# JavaScript: исключения

У выброшенной ошибки есть несколько атрибутов:

- name – название ошибки;
- message – сообщение, которое возвращает ошибка.

# JavaScript: отладка

Все современные браузеры имеют, так называемые инструменты разработчика.

Их можно вызвать с помощью разных комбинаций клавиш (от них зависит какая вкладка будет открыта по умолчанию), например: Ctrl+Shift+C.

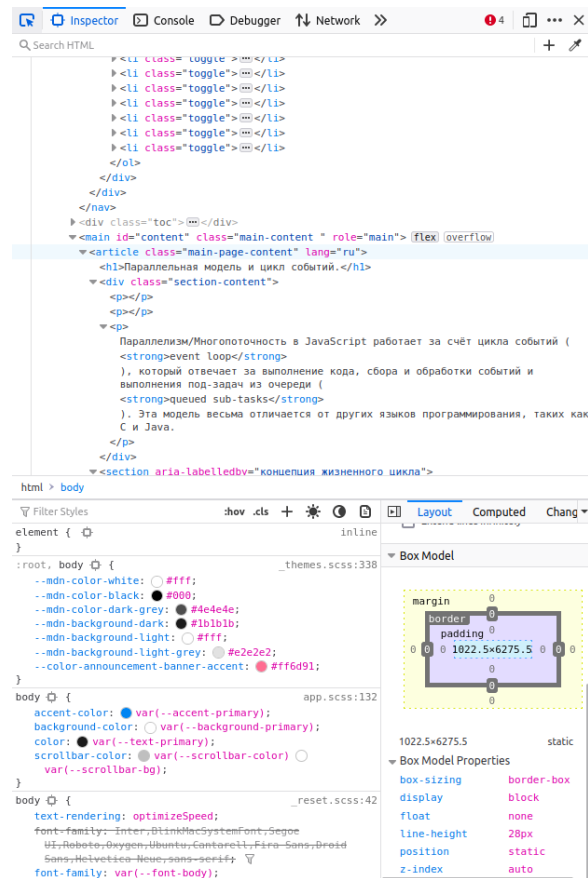
Нужно просто зайти на любую страницу и можно **вызвать**.

# JavaScript: отладка

Вкладка inspector / elements (в Firefox и Chrome называется по-разному).

Позволяет просматривать структуру документа, стили, параметры элементов.

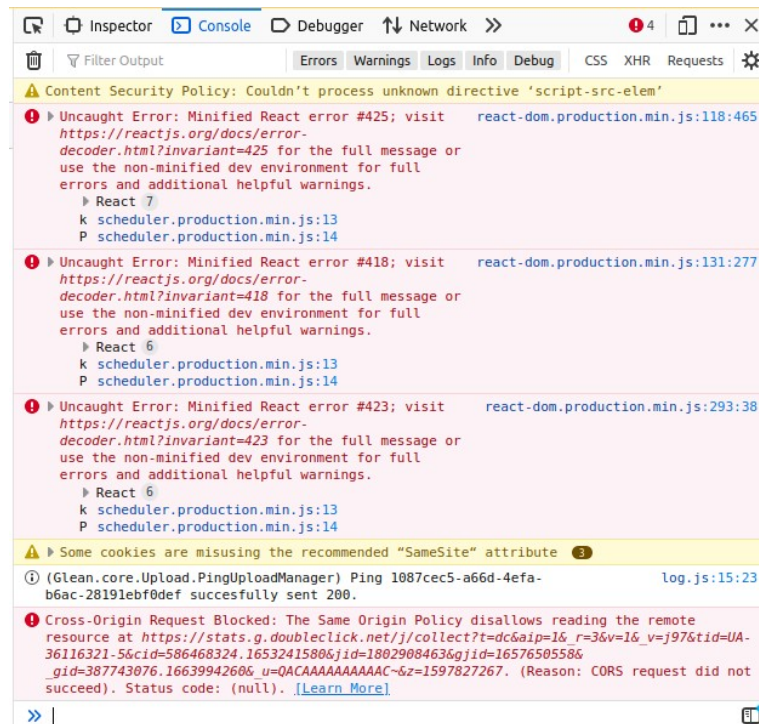
Позволяет менять HTML- и CSS-код на лету. Можно прямо в этом окошке что-то писать и результат будет виден на экране. Но до перезагрузки страницы, изменения не записываются в исходные файлы.



# JavaScript: отладка

Вкладка console.

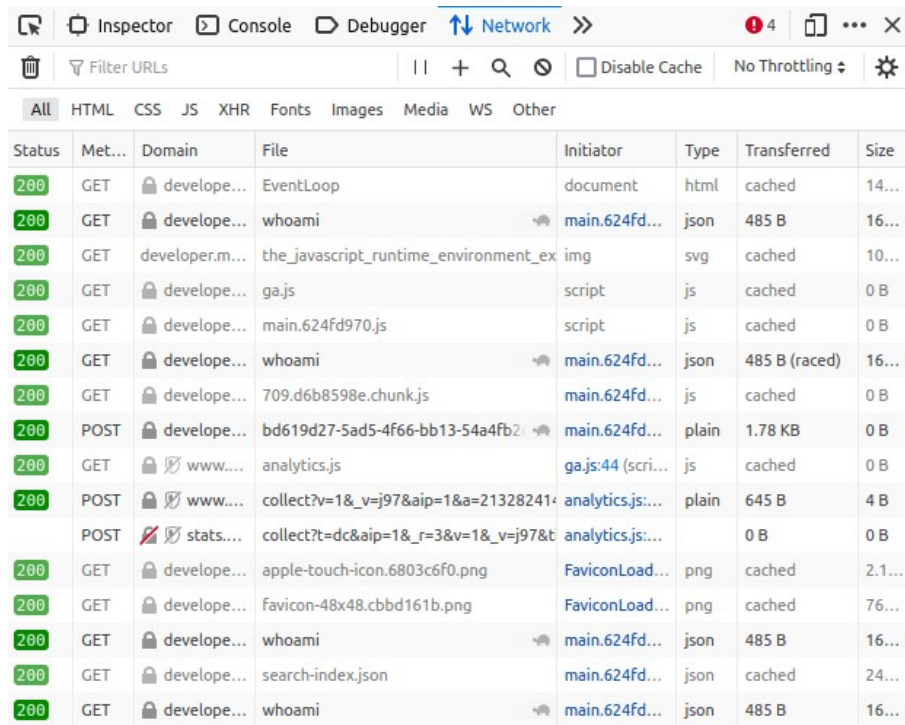
Выводит всю отладочную информацию приложения и позволяет писать команды на JS. Обычно используется, чтобы проверить что-то.



# JavaScript: отладка

## Вкладка network.

Позволяет посмотреть всю информацию о выполнении запросов (заголовки, данные, статусы и т.д.).



The screenshot shows the Chrome DevTools Network tab. The top bar includes icons for Inspector, Console, Debugger, and Network (selected). Below the bar is a filter input and a 'Filter URLs' button. The main area displays a table of network requests. The table has columns for Status, Method, Domain, File, Initiator, Type, Transferred, and Size. The requests are listed in chronological order, showing various file types like HTML, JSON, SVG, JS, and PNG. The status for all requests is 200, indicating successful completion.

Status	Met...	Domain	File	Initiator	Type	Transferred	Size
200	GET	develope...	EventLoop	document	html	cached	14...
200	GET	develope...	whoami	main.624fd...	json	485 B	16...
200	GET	developer.m...	the_javascript_runtime_environment_ex	img	svg	cached	10...
200	GET	develope...	ga.js	script	js	cached	0 B
200	GET	develope...	main.624fd970.js	script	js	cached	0 B
200	GET	develope...	whoami	main.624fd...	json	485 B (raced)	16...
200	GET	develope...	709.d6b8598e.chunk.js	main.624fd...	js	cached	0 B
200	POST	develope...	bd619d27-5ad5-4f66-bb13-54a4fb2...	main.624fd...	plain	1.78 KB	0 B
200	GET	www....	analytics.js	ga.js:44 (scri...	js	cached	0 B
200	POST	www....	collect?v=1&_v=j97&aip=1&a=213282414	analytics.js:...	plain	645 B	4 B
200	POST	stats....	collect?t=dc&aip=1&_r=3&v=1&_v=j97&t	analytics.js:...		0 B	0 B
200	GET	develope...	apple-touch-icon.6803c6f0.png	FaviconLoad...	png	cached	2.1...
200	GET	develope...	favicon-48x48.cbdd161b.png	FaviconLoad...	png	cached	76...
200	GET	develope...	whoami	main.624fd...	json	485 B	16...
200	GET	develope...	search-index.json	main.624fd...	json	cached	24...
200	GET	develope...	whoami	main.624fd...	json	485 B	16...

# JavaScript: отладка

Есть и другие вкладки, но здесь мы их не будем рассматривать.

# JavaScript: логирование

Чтобы вывести на консоль какую-либо информацию из приложения, можно использовать специальный объект – console.

Есть разные потоки вывода, основные:

- log – для логов;
- warn – для предупреждений;
- error – для вывода ошибок.

Пример: console.log(`current state: \${currentState}, mode: \${mode}`);



# JavaScript: логирование

**Логи можно стилизовать с помощью CSS:**

`console.log("%cSome log", "color: green; font-size: 2rem;");`

```
>> console.log("Some log");
Some log                                     debugger eval code:1:9
← undefined
>> console.log("%cSome log", "color: green; font-size: 2rem;");
Some log                                     debugger eval code:1:9
← undefined
>> |
```

A screenshot of a JavaScript console interface. It shows two log entries. The first entry is the result of 'console.log("Some log");', displaying 'Some log' in a standard font. The second entry is the result of 'console.log("%cSome log", "color: green; font-size: 2rem;");', displaying 'Some log' in a larger, green font. Each log entry is followed by '← undefined' and a source location 'debugger eval code:1:9'. At the bottom, there is a prompt '>> |' and a small icon of a document with a blue dot.

# JavaScript: цикл событий

В JS есть так называемый цикл событий – это бесконечный цикл, который выполняет задачи, создаваемые пользователем (клики мышкой, отправка запросов и т.д.).

Параллельного выполнения задач в JS нет. Все задачи выполняются друг за другом и не прерываются.

# JavaScript: асинхронные операции

Но код в JS выполняется не всегда последовательно.

Например, запросы к серверу выполняются отдельно от остального кода. Пока запрос обрабатывается, скрипт может выполняться дальше. Т.е. запрос выполняется асинхронно.

В этом случае код может быть разбит на несколько конкретных задач, каждая из которых выполнится при вызове нужного события.

# JavaScript: асинхронные операции

Бывает так, что нужно получить результат асинхронной операции, например:

```
let data = asyncFunc();  
if (data) {  
  //code  
} else {  
  //another code  
}
```

В примере выше это не сработает. Условие будет выполнено раньше, чем данные придут.

# JavaScript: асинхронные операции

В JS есть два способа выполнить асинхронную операцию и получить результат:

- использовать объект Promise;
- использовать ключевые слова async и await.

# JavaScript: асинхронные операции

## Пример использования Promise:

```
function asyncFunc(param) {  
  return new Promise((resolve, reject) => {  
    function getData() {  
      resolve(34);  
    }  
  
    setTimeout(getData, param);  
  });  
}
```

# JavaScript: асинхронные операции

## Обычная работа с Promise:

```
const externalFunc = () => {  
  ...  
  asyncFunc("api/v1/data/").then(  
    data => {  
      if (data) {  
        //code  
      } else {  
        //another code  
      }  
    }  
  );  
}
```

# JavaScript: асинхронные операции

`async` и `await` позволяют избежать вложенности, перенеся асинхронность на уровень выше:

```
const externalFunc = async () => {  
  ...  
  let data = await asyncFunc("api/v1/data/");  
  if (data) {  
    //code  
  } else {  
    //another code  
  }  
}
```



# Работа из JavaScript с HTML и CSS

JS, используемый в браузере имеет специальные языковые расширения для взаимодействия со средой (HTML-документом, куками и т.д.). Эти расширения предустановлены, их не нужно специально устанавливать или подключать.

Основной объект, который появляется в веб-версии JS – window. Это глобальный объект. Все идентификаторы, которые есть в скрипте, находятся в нем.

# Работа из JavaScript с HTML и CSS

То есть, фактически к любому глобальному идентификатору можно обратиться двумя эквивалентными способами:

- window.someVar;
- someVar;

Первый вариант длинный, но позволяет обращаться к глобальному идентификатору тогда, когда уже есть локальный идентификатор с таким же именем.

# Работа из JavaScript с HTML и CSS

Некоторые полезные поля объекта window (они же глобальные переменные):

- console; – объект для вывода на консоль;
- devicePixelRatio; – сколько px в аппаратном пикселе;
- document; – объект документа;
- history; – объект с историей перехода по ссылкам;
- innerWidth; и innerHeight; – размер окна без всех меню браузера;

# Работа из JavaScript с HTML и CSS

Некоторые полезные поля объекта window:

- localStorage; – объект локального хранилища;
- location; – объект текущего URL;
- navigator; – объект с информацией о браузере;
- outerWidth; и outerHeight; – размер окна с учетом всех меню браузера;
- screen; – объект, содержащий параметры экрана;
- scrollX; и scrollY; – прокрутка документа.

# Работа из JavaScript с HTML и CSS

Некоторые полезные методы объекта window (они же глобальные функции):

- blur(); – убрать фокус с окна;
- close(); – закрыть текущее окно;
- scroll(); – прокрутить окно в определенное место;
- setTimeout(); и clearTimeout(); – установить таймаут для выполнения;
- setInterval(); и clearInterval(); – установить интервал для выполнения;
- open(); – открыть новое окно.

# Работа из JavaScript с HTML и CSS

Объект document содержит информацию о текущем документе. С помощью этого объекта можно получить информацию обо всех элементах в документе (а также динамически изменять элементы). Он также позволяет создавать новые элементы в документе.

Этот объект используется для того, чтобы динамически манипулировать DOM-деревом.

# Работа из JavaScript с HTML и CSS

Некоторые полезные поля document:

- activeElement; – возвращает элемент, который сейчас в фокусе;
- body; – возвращает элемент, содержащий тело документа.

# Работа из JavaScript с HTML и CSS

Некоторые полезные методы document:

- createElement(); – создает новый элемент;
- getElementById(); – возвращает элемент по заданному id;
- getElementsByClassName(); – возвращает элементы содержащие указанный класс;
- getElementsByTagName(); – возвращает элементы, в основе которых указанный тег;
- getSelection(); – возвращает текст, который выделил пользователь.



# Работа из JavaScript с HTML и CSS

Каждый элемент, который можно получить с помощью объекта `document`, имеет специальный тип Element.

Взаимодействие с элементами в документе происходит именно через такой объект. Он может быть получен, например, с помощью метода `document.getElementById()`;

# Работа из JavaScript с HTML и CSS

Некоторые полезные поля объектов типа Element:

- attributes; – все атрибуты;
- children; – список всех дочерних элементов;
- className; – значение атрибута class элемента;
- clientHeight; и clientWidth; – размер элемента;
- id; – id элемента;
- innerHTML; – HTML-код внутри элемента;
- nextElementSibling; (previousElementSibling;) – сосед элемента справа (слева);
- parentNode; – родительский элемент;
- scrollLeft; и scrollTop; – прокрутка внутри элемента;
- tagName; – имя тега элемента;

# Работа из JavaScript с HTML и CSS

Некоторые полезные поля объектов типа Element:

- append(); – добавить дочерний элемент;
- getAttribute(); – получить значение атрибута;
- getBoundingClientRect(); – получить позицию и размер элемента;
- remove(); – удалить элемента;
- setAttribute(); – установить значение атрибуту;

# Работа из JavaScript с HTML и CSS

Элементы имеют также специальные методы, которые обрабатывают события, которые происходят с элементом.

Рассмотрим некоторые из них:

- blur и focus – активируются когда элемент в фокусе и когда фокус уходит с элемента;
- change – активируется при вводе/удалении значений в поле;
- click – активируется при клике по элементу;
- dblclick – активируется при двойном клике по элементу;
- keydown и keyup – активируются при зажатии и отпускании клавиши на клавиатуре, когда элемент в фокусе;
- keypress – активируется при нажатии клавиши на клавиатуре, когда элемент в фокусе;

# Работа из JavaScript с HTML и CSS

Продолжение:

- mousedown и mouseup – активируются при зажатии и отпускании кнопки мыши на элементе;
- mouseenter и mouseleave – активируются, когда мышка наводится или покидает элемент или его потомка;
- mousemove – активируются, когда мышка движется, находясь над элементом;
- mouseout и mouseover – активируются, когда мышка наводится или покидает элемент;
- scroll – активируется при прокрутке.

# Работа из JavaScript с HTML и CSS

Каждый из обработчиков при вызове получает в качестве параметра объект типа, наследуемого от Event. Например, если событие вызвано мышью, то это будет MouseEvent, если это была клавиатура, то KeyboardEvent и т.д. (в конце есть ссылка, где почитать подробнее про свойства этих событий).

Объекты событий позволяют узнать, на каком элементе оно произошло, при каких условиях (например, какая именно кнопка была нажата, или где находилась мышка).

# Работа из JavaScript с HTML и CSS

У объекта Event есть одно важное свойство – target – возвращает элемент, на котором сработало событие.

# Работа из JavaScript с HTML и CSS

Рассмотрим некоторые из объектов событий подробнее.

KeyboardEvent – событие, вызванное с клавиатуры.

- code – код нажатой клавиши;
- repeat – клавиша нажата и повторяется;



# Работа из JavaScript с HTML и CSS

MouseEvent – событие, вызванное движением мыши или нажатием её кнопок. Некоторые полезные свойства:

- altKey;, ctrlKey и shiftKey; – зажат ли Alt (Ctrl, Shift) на момент вызова события;
- button; – какая кнопка мыши вызвала событие;
- buttons; – какие кнопки мыши зажаты во время события.
- clientX; и clientY; – координаты указателя мыши на момент выхода события.

# Работа из JavaScript с HTML и CSS

WheelEvent – событие, вызванное движением колесика мыши (или чем-то подобным).

Полезные свойства:

- deltaX; – горизонтальная прокрутка;
- deltaY; – вертикальная прокрутка.

Примечание: величина может разниться в разных браузерах.

# Работа из JavaScript с HTML и CSS

В браузерах есть так называемые модальные окна. Это такие окна, которые перехватывают фокус с основного при своем появлении.

В расширении JavaScript для браузеров есть специальные объекты, которые позволяют вызывать такие окна. Таких объектов три:

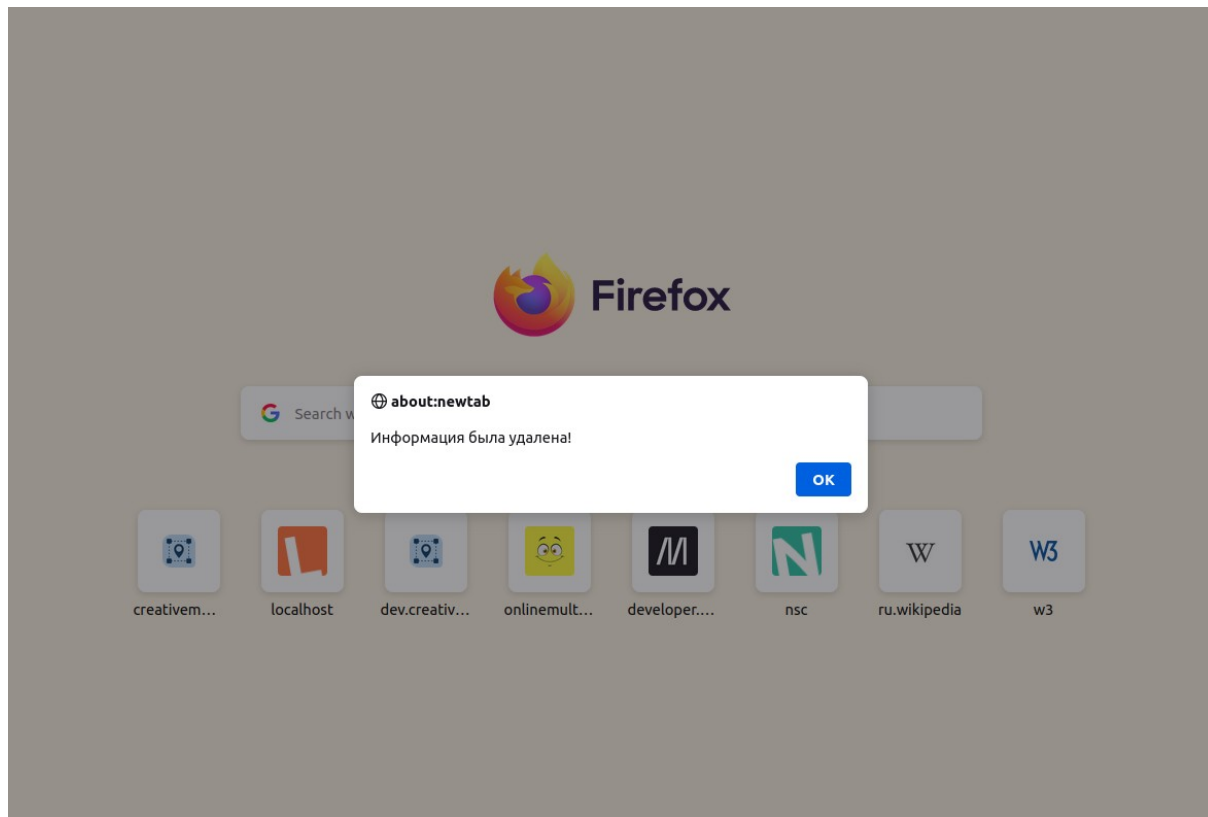
- alert;
- prompt;
- confirm.

# Работа из JavaScript с HTML и CSS

## Окно alert

предназначено для вывода извещений / предупреждений, на которые пользователь не может повлиять.

Вызывается оно так:  
`alert("Информация  
была удалена!");`

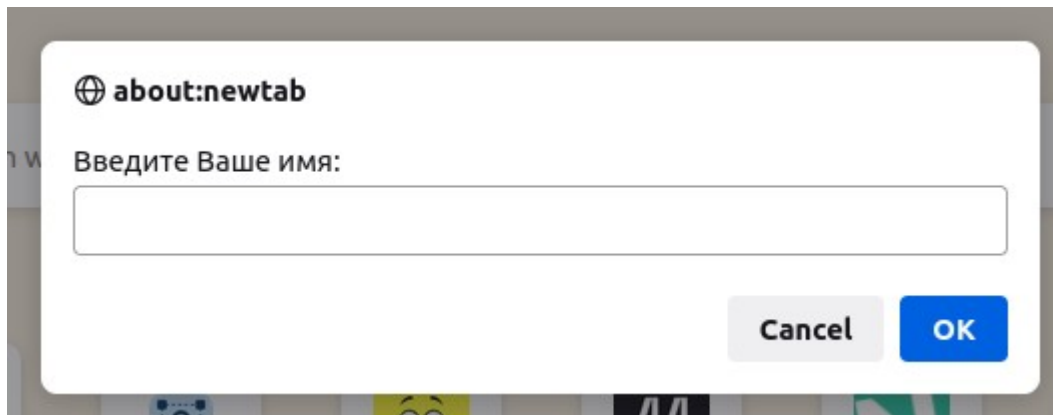


# Работа из JavaScript с HTML и CSS

Окно prompt предназначено для получения ответа от пользователя.

Вызывается оно так:

let answer = prompt("Введите Ваше имя:");

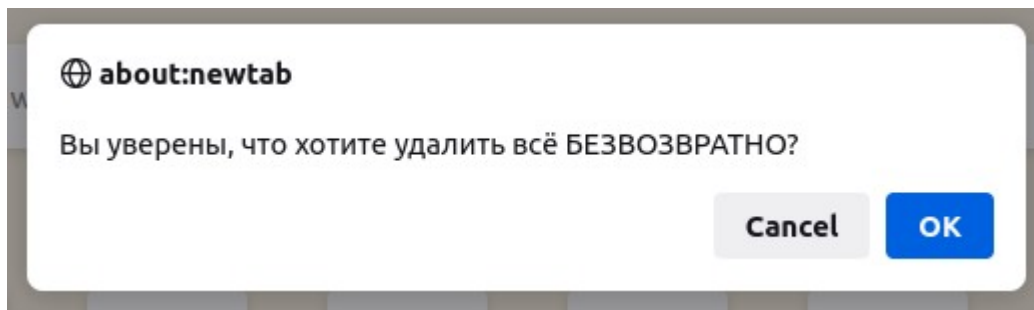


# Работа из JavaScript с HTML и CSS

Окно confirm предназначено для подтверждения пользователя.

Вызывается оно так:

let answer = confirm("Вы уверены, что хотите удалить всё БЕЗВОЗВРАТНО?");



# JavaScript: документирование кода

В JavaScript принято документировать код по правилам JSDoc.

## Пример документирования функции:

```
/**  
 * Create simple unclosed polyline path from point array  
 * @param {Array<string>} points The source array of points (like this: ["12,3", "10,5",  
"11,6"])  
 * @return {string}          The new unclosed polyline path.  
 * @example  
 * createFromPoints(["12,3", "10,5", "11,6"]);  
 * // returns: "M 12,3 L 10,5 L 11,6"  
 */
```

# JavaScript: документирование кода

## При документировании можно создавать свои типы данных:

```
/**  
 * Get base point of index-th segment in simpleIndex-th elementary path  
 * @typedef {{x: number, y: number}} Point  
 * @param {string} path The source path  
 * @param {number} simpleIndex The index of path in whole path (like "M ... " in "M ..  
M .. M ..")  
 * @param {number} index The index of segment path (like "L .." in "M .. L .. L ..")  
 * @return {Point} return base point (last in segment)  
 * @example  
 * getBasePoint("M 1,1 L 1,6 Q 34,2 1,1", 0, 0);  
 * // returns: {x: 1, y: 6}  
 */
```



# Версии JavaScript

После ES5 в 2015 году вышло большое обновление ECMAScript – ES6 (или ES2015).

После ES6 было решено выпускать новую версию стандарта ежегодно, чтобы не было слишком больших обновлений.

То есть, ES7 – это ES2016, ES12 – это ES2021 и т.д.

Последние версии браузеров обычно поддерживают новейшую версию JS.

# JavaScript: ES6

## Что нового в ES6:

- добавлено ключевые слова `let` и `const`;
- стрелочные функции `() => {}` и значения по умолчанию для аргументов;
- добавлены `import` и `export`;
- `spreading (...)`;
- двоичные и восьмеричные литералы `0o10` и `0b1010`;
- деструктуризация `let [a, b, c] = getData()`;
- добавлены „классы“ и ключевые слова `static`, `extend` и `super()`;
- добавлен цикл `for ... of`;
- строковые шаблоны ``string ${variable}``;
- добавлены генераторы;
- добавлены `Promise()`
- есть ещё много что.

# JavaScript: ES7 и ES8

## Что нового в ES7:

- добавлена функция `Array.includes()`;
- добавлен оператор возведения в степень `**`;

## Что нового в ES8:

- добавлены методы `"".padStart()` и `"".padEnd()`;
- добавлены методы `Object.values(variable)` и `Object.entries(variable)`;
- добавлена асинхронность и ключевые слова `async` и `await`;

# JavaScript: ES9

## Что нового в ES9:

- асинхронный цикл `for await (let val of asyncIterable) { ... };`
- добавлена функция `Promise(...).finally();`
- добавлена поддержка `look behind`, именованные группы и флаг `s` (при установке `.` будет учитывать и символы переноса строк) в регулярных выражениях;

# JavaScript: ES10

Что нового в ES10:

- добавлены функции `[].flat()` и `[].flatMap()`;
- улучшен метод  `[].sort()`;
- добавлена функция `Object.fromEntries()`;
- добавлены функции `"".trimStart()` и `"".trimEnd()`;
- добавлена функция `(() => {}).toString()`;
- некоторые исправления совместимости с JSON;
- в `catch` больше не обязательно указывать контейнер аргумента.

# JavaScript: ES11

Что нового в ES11:

- добавлена функция `"".matchAll()`;
- тип данных `BigInt`;
- добавлен динамический импорт.

# JavaScript: ES12

## Что нового в ES12:

- добавлена функция `"".replaceAll();`
- добавлена функция `Promise.any();`
- тип ошибки `AggregateError`;
- новые типы операторов присваивания: `??=`, `&&=`, `||=`;
- разделители для чисел `_`.

# JavaScript: ES13

Что нового в ES13:

- добавлены приватные методы и поля в классы (#);
- добавлен флаг d (добавляет индексы, на которых встречены совпадения) в регулярные выражения;
- добавлено свойство `Error.cause`;
- добавлены методы `String.at()`, `Array.at()`;
- добавлен метод `Object.hasOwn()`.



# Версии JavaScript

Как уже известно, старый код будет работать в новых браузерах. Но обратное неверно. Новый код в старых браузерах может не работать.

Чтобы обойти это ограничение, используются так называемые полифиллы. Это такие части кода, которые могут реализовать новую фичу старыми средствами.

# Версии JavaScript

**Есть специальные инструменты, которые позволяют переводить новый код в старую версию. Например, это можно сделать при помощи Babel.**

# Работа из JavaScript с HTML и CSS

Как встроить скрипт в страницу

```
<script src="scripts/main.js"></script>
```

Сказать, что скрипт должен быть после элементов, с которыми он будет взаимодействовать (чтобы они точно существовали на момент выполнения скрипта).

# Полезные ссылки

- <https://developer.mozilla.org/ru/docs/Web/JavaScript/Guide> – **обучалка по JS;**
- <https://262.ecma-international.org/> – **спецификация ECMAScript, тут можно посмотреть, что нового в последней версии JS;**
- <https://jsdoc.app/> - **документация по JSDoc;**
- [https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Strict\\_mode](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Strict_mode) – **подробнее о strict mode;**
- <https://developer.mozilla.org/ru/docs/Web/API/Event> – **события, которые передаются в различные обработчики.**

**Дальше NPM!**