Введение в JavaScript

Nartea Nichita, USM, 2026

Содержание

1
НЕМНОГО О КУРСЕ

Немного о курсе и целях изучения

- Курс «**JavaScript**» - вводное изучение одного из ключевых языков программирования
- Предназначен для студентов, начинающих знакомство с JavaScript и программированием в целом
- Темп изучения - **осознанно постепенный.**
- Используются упрощённые примеры, предназначенные для формирования базового понимания

Структура курса

- **Основы синтаксиса JavaScript**
  - базовые конструкции и особенности языка
- **Продвинутые модули JavaScript**
  - ключевые механизмы и принципы работы
- **Прикладное программирование**
  - взаимодействие с HTML и CSS

15 теоретических тем, **6** лабораторных работ

Зачем изучать JavaScript?

- ❏ Один из самых популярных языков программирования
- ❏ 6-е место в рейтинге TIOBE (декабрь 2025)
- ❏ Основа современной фронтенд-разработки
- ❏ Используется в различных IT-направлениях
- ❏ Расширяет профессиональные и карьерные возможности

Популярность языка JavaScript

Dec 2025	Dec 2024	Change	Programming Language	Ratings	Change
1	1		Python	23.64%	-0.21%
2	4	▲	C	10.11%	+1.01%
3	2	▼	C++	8.95%	-1.87%
4	3	▼	Java	8.70%	-1.02%
5	5		C#	7.26%	+2.39%
6	6		JavaScript	2.96%	-1.66%
7	9	▲	Visual Basic	2.81%	+0.85%
8	8		SQL	2.10%	+0.11%

2
O JAVASCRIPT

О JavaScript

- JavaScript — высокоуровневый интерпретируемый язык программирования
- Используется для создания динамических веб-страниц
- **Обеспечивает**:
  - взаимодействие пользователя со страницей
  - обработку событий
  - изменение содержимого без перезагрузки
- Широко применяется в браузерах

Вспомним ...

Интерпретируемые и компилируемые языки

- **Компилируемые языки (C, C++)**
  - код сначала компилируется
  - при ошибке программа не запускается
- **Интерпретируемые языки (JavaScript)**
  - код выполняется построчно
  - строки до ошибки успевают выполниться
- Это объясняет, почему JS может работать частично даже с ошибками

JavaScript и скрипты

- Изначально данный язык программирования создавался, чтобы **сделать веб-страницы "живыми"**, позволяя им *реагировать на пользовательские действия* и *обновлять данные страницы без перезагрузке*.
- Программы на языке JavaScript называются **скриптами**. Они могут встраиваться в HTML и выполняться автоматически при загрузке веб-страницы.

3
ИСТОРИЯ JAVASCRIPT

История появления JavaScript (I)

- Компания **Netscape** хотела сделать **веб "живым"** - таким, чтобы страницы могли реагировать на действия пользователя прямо в браузере, без постоянной перезагрузки и без сложных серверных решений.
- В 1995 году программист **Брендан Айк** разработал первый прототип языка в компании Netscape.
- Язык получил название **JavaScript** и был встроен в браузер Netscape **Navigator**.

Интересный факт: Сначала язык назывался "**Mocha**", затем "**LiveScript**", а затем из-за популярности "Java", был назван "**JavaScript**"

История появления JavaScript (II)

После успеха JavaScript компания Microsoft внедрила собственную версию языка, **JScript**, в Internet Explorer.

- Несмотря на внешнее сходство, **реализации отличались между собой.**
- В результате один и тот же код мог нормально работать в браузере Netscape и ломаться в Internet Explorer.

Это привело к необходимости **стандартизации языка**.

История появления JavaScript (III)

- Чтобы решить описанную выше проблему и добиться того, чтобы код работал одинаково во всех браузерах, компания **Netscape в 1996 году передала JavaScript на стандартизацию в Ecma International**.
- В результате в 1997 году был опубликован первый стандарт ECMA-262, также известный как **ECMAScript** - стандарт скриптового языка общего назначения.

История появления JavaScript (IV)

- Браузеры получили возможность создавать **собственные реализации JavaScript, опираясь на единый стандарт**.
- Эта спецификация определяет, какие возможности должен поддерживать язык и как именно они должны работать, обеспечивая **единообразие выполнения кода на разных платформах и в разных браузерах**.

ECMAScript - это спецификация, которая описывает, как язык должен работать.

JavaScript - это конкретная реализация данного стандарта.

История появления JavaScript (V)

Например,

- В спецификации **ECMAScript** может быть внедрен стандарт, вводящий новый метод **`Number.isEven()`** - проверка числа на четность.
- В Chrome, данная функция будет доступна, а в Firefox эта функция может отсутствовать из-за различий в реализации стандарта.

Движки JavaScript

- Выполнение JavaScript-кода обеспечивает **JavaScript-движок браузера**.
- Каждый браузер использует собственный движок, реализующий стандарт ECMAScript.
- Новые возможности языка **сначала описываются в стандарте, затем реализуются в движках** и только после этого появляются в браузерах.
- Для проверки поддержки функциональности используются таблицы совместимости, например сайт **caniuse.com**.

Как появляется новая функциональность?

- Сначала новая возможность описывается в стандарте **ECMA-262**. На этом этапе это формальное описание, а не готовый код.
- Далее разработчики **JavaScript-движков изучают спецификацию** и **решают, как реализовать её** внутри своего движка.
- Каждый движок реализует новую функциональность самостоятельно.
- Внутренняя реализация может отличаться, но внешнее поведение обязано соответствовать стандарту.

Совместимость браузеров

- Новые возможности JavaScript появляются в браузерах постепенно.
  - Одна и та же функция может работать в Chrome,
  - но ещё не поддерживаться в Firefox.
- Для проверки поддержки используются таблицы совместимости, например сайт caniuse.com.
- Фундаментальные конструкции языка поддерживаются во всех браузерах.

Безопасность JavaScript

- Javascript - **«безопасный» язык**, он не предоставляет низкоуровневый доступ к памяти или процессору
  - JavaScript на веб-страницах ограничен в своих возможностях для обеспечения безопасности пользователей.
  - Современные браузеры предоставляют JavaScript ограниченный доступ к файлам только после определенных действий пользователя, таких как "перетаскивание" файла в окно браузера
  - Каждая веб-страница с поддержкой JavaScript работает в изолированном окружении, и скрипты на одной странице не могут несанкционированно взаимодействовать с другими страницами

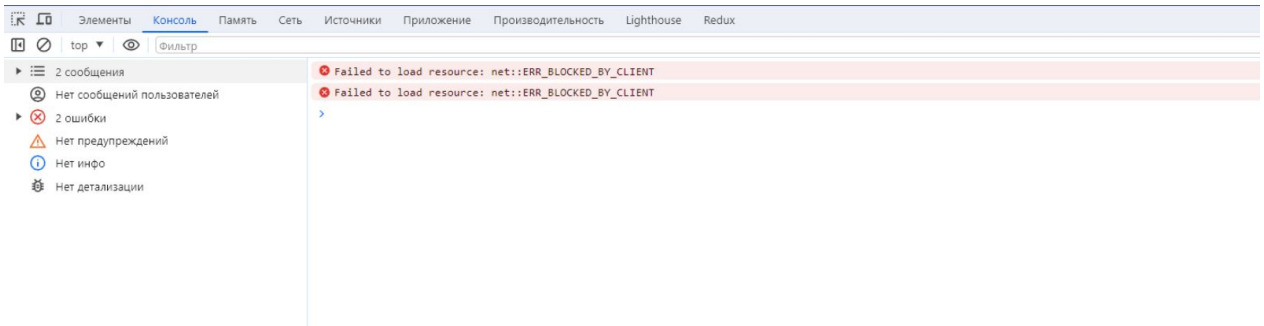
4
ВЫПОЛНЕНИЕ JAVASCRIPT КОДА

Инструменты разработчика

- При разработке на JavaScript ошибки неизбежны. **Чем сложнее программа, тем выше вероятность ошибки**.
- Для этого во всех браузерах есть инструмент под названием "**инструменты разработчика**", в котором есть вкладка с консолью.

Консоль разработчика

- Открываем вашу HTML страницу и нажимаем клавишу **F12** (если используете *Mac Ctrl+Opt+J*). Переходим во вкладку **Консоль** (**Console**), там будут показаны все ошибки и другие сообщения JavaScript.



Выполнение JavaScript кода

Выполнять код JavaScript можно в различных средах и используя различные инструменты

- **Браузеры**. Выполнять код Javascript можно в вышеописанной консоли.
- **Локальные среды разработки**. Чтобы выполнять код JS на вашем компьютере, необходимо скачать NodeJS

Выполнение JavaScript в браузере

- JavaScript можно **выполнять прямо в браузере**.
  - Один из способов - использование консоли разработчика.
- Код вводится вручную и выполняется сразу после нажатия Enter.
- Результат выполнения отображается непосредственно в консоли.

Зайдите в консоль разработчика и **введите следующий код:**

```
console.log('Hello, World!');
```

Выполнение JavaScript локально

Для локального выполнения JavaScript, используется среда Node.js.

- После установки Node.js компьютер может исполнять JS-файлы.
- Код запускается через терминал и результат выводится в обычную консоль.

В данном случае, в директории с файлом пропишите в консоль:

```
~ node index.js
```

5
КАК ЗАСТАВИТЬ БРАУЗЕР ГОВОРИТЬ?

Подключение JS к HTML странице (I)

- JavaScript изначально разрабатывался с целью обеспечения взаимодействия с веб-страницами, особенно с использованием HTML и CSS. Таким образом, ключевой функциональной частью стало взаимодействие JavaScript с HTML и CSS
- Создадим простую HTML страницу **`index.html`** и попробуем вставить наш JS-скрипт

```
<!doctype html>
<html lang="en">
  <head>
    <title>Document</title>
  </head>
  <body>
    <div>Эй, а ты точно знаешь JS?</div>
    <script>
      alert('Hello, world');
    </script>
  </body>
</html>
```

- Теперь, если мы откроем HTML страницу наш скрипт, находящийся на странице выполниться и вам будет показано "окно" с надписью **Hello, world!**

Подключение JS к HTML странице (II)

- Тег **<script> -** это HTML-элемент, предназначенный для встраивания или подключения JavaScript-кода на страницу. Код внутри него выполняется автоматически в момент, когда браузер обрабатывает этот тег.
- Тег **<script>** можно размещать в HTML-разделах, код, внутри этого тега, будет выполняться в момент обработки браузером соответствующей части HTML-документа.

Подключение JS к HTML странице (III)

- Обычно скрипты, относящиеся к настройкам и загрузке страницы, помещают в раздел **<head>**.
- Скрипты, влияющие на визуальное отображение или взаимодействие с пользователем, размещают в конце **<body>**.

Что делать если кода JS **слишком много**?

Стоит ли его встраивать в страницу?

Внешние JavaScript скрипты (I)

- В ситуации, когда имеется значительное количество JavaScript-кода, рекомендуется **избегать его встраивания непосредственно в код HTML**, поскольку это может снизить читаемость и обслуживаемость кода.
- Предпочтительным подходом является написание скрипта в отдельном файле формата **JavaScript** (`.js`) и последующее **подключение этого файла к HTML-странице**.

Внешние JavaScript скрипты (II)

- Рассмотрим использование функции prompt для демонстрации взаимодействия с пользователем.

index.js

```
// Используем функцию prompt для запроса данных у пользователя
// и сохраняем введенный текст в переменной text
const text = prompt("Впиши сюда, что хочешь!");
// Выводим введенный текст в консоль разработчика
console.log(text);
```

index.html

```
<!doctype html>
<html lang="en">
  <head>
    <!-- Подключение внешнего JS-файла -->
    <script src="index.js"></script>
  </head>
  <body>
    <p>Посмотри консоль разработчика!</p>
  </body>
</html>
```

** С помощью атрибута `src` тэга `<script>` возможно **подключать внешние JS-файлы**.

Основы взаимодействия JS с HTML

- Для начала изучения материала не обязательно углубляться в детали взаимодействия JS с HTML. **Этот аспект будет рассмотрен в отдельной теме "Работа с DOM"**. Однако, иногда возникает необходимость вставки текста в HTML.
- В этом контексте рассмотрим метод **`getElementById()`**.

Основы взаимодействия JS с HTML. getElementById

- Функция **`getElementById`**
  - Используется для выбора элемента по его уникальному идентификатору (**id**)
  - Пример:

    `document.getElementById('example-id')` выберет элемент:

    `<element id=”exampleid”></example>`

Основы взаимодействия JS с HTML. Изменение текста

После выбора элемента можно вставить текст с помощью свойства **`textContent`** или **`innerHTML`:**

Пример с `textContent`:

```
document.getElementById('example-id').textContent = 'Новый текст';
```

Пример с `innerHTML`:

```
document.getElementById('example-id').innerHTML = 'Новый текст';
```

Основы взаимодействия JS с HTML. Полный пример

```html
<!-- index.html -->
<!doctype html>
<html lang="en">
  <head> </head>
  <body>
    <div>Привет, как твои дела?</div>
    <div id="answer"></div>
    <!-- Подключение внешнего JS-файла -->
    <script src="index.js"></script>
  </body>
</html>
```

```js
// index.js

const element = document.getElementById('answer');

const randomNumber = Math.floor(Math.random() * 2) + 1;

if (randomNumber === 1) {
  element.innerHTML = 'Отлично!';
} else {
  element.innerHTML = 'Плохо';
}
```

6
НАПИСАНИЕ JAVASCRIPT КОДА

Начало написания JavaScript-кода

- После изучения способов выполнения JavaScript начинается знакомство с самим языком. На этом этапе важно понять, **из каких базовых элементов состоит код**.
- В JavaScript выделяют **три синтаксические сущности**, которые важно понимать с самого начала:
  - Выражения
  - Инструкции
  - Выражения-инструкции

Выражения

Выражение - это фрагмент кода, который возвращает значение. Проще говоря, это то, что **можно вычислить**.

Примеры выражений:

```
10; // результат выражения будет равен числу 10
20 + 30; // Результат выражения будет равен числу 50.
'Hello' + 'World'; // Результат выражения будет строкой 'Hello World'.
x = 3; // Результат выражения - значение, которое вы присваиваете переменной, то есть число 3.
func(a); //  // Результат выражения зависит от возвращаемого значения функции func с аргументом a.
v ⩾ a; // Результат выражения будет логическим значением (true или false)
```

Инструкции

Инструкции представляют собой синтаксические конструкции и команды, которые выполняют определенные действия, но сами по себе ничего не возвращают

Примеры инструкций:

```
alert('Hello'); // вызов функции `alert` для отображения сообщения
let a; // создание переменной
let b = 3; // создание константы b и присвоение ей значения 3
if (3 > 2) {
  // условная конструкция if для проверки условия
  console.log('Hello, world');
}
```

Выражение-инструкция

Выражение-инструкция - это выражение, использованное как самостоятельная инструкция. Проще говоря, это выражение, которое, помимо вычисления, выполняет дополнительное действие.

Примеры инструкций:

```
b = 1; // выполняет действие присваивания и изменяет состояние программы
console.log('Hello'); // выполняет вывод информации в консоль
```

Задание ...

Выражения, Инструкции, Выражения-Инструкции

Определите, где *выражения, инструкции и выражения-инструкции*:

- ❏ `let x;`
- ❏ `a + b;`
- ❏
```
for (let i = 0; i < 5; i++) {
  console.log(i);
}
```
- ❏ `let y = Math.sqrt(x);`

Один из самых частых споров на просторах интернета:
Ставить ли точку с запятой в конце строки в JavaScript?

В языке JS точка с запятой является необязательной, но...

Точка с запятой (I)

- Парсер JavaScript использует механизм **Automatic Semicolon Insertion (ASI)**, который **автоматически вставляет точки с запятой** там, где это необходимо для корректного выполнения кода.
- Проще говоря, если вы не поставили точку с запятой, **движок в большинстве случаев сделает это за вас**.
- Однако, существуют ситуации, когда парсер может пропустить точку с запятой там, где ее ожидание может привести к ошибкам в интерпретации кода.

Точка с запятой (II)

Плюсы использования точек с запятой в JavaScript	Минусы применения точек с запятой в JavaScript
Предсказуемость кода: Явное указание на конец инструкции делает код более предсказуемым и уменьшает вероятность ошибок.	**Избыточность**: В некоторых случаях точки с запятой могут показаться избыточными, особенно когда автоматическая вставка работает корректно.
Совместимость: Некоторые инструменты и среды разработки могут лучше взаимодействовать с кодом, в котором точки с запятой использованы явно.	**Эстетика**: Для некоторых разработчиков отсутствие явных точек с запятой делает код более читаемым и эстетичным.
Читаемость (дела вкуса)	**Лишний символ**: экономия времени и места
Дело привычки: Ты привык пользоваться точкой с запятой.	**Автоматическая вставка**: точка запятой и так вставляется автоматически

Keep Learning... А что дальше?

- В презентации рассказаны лишь **основы данной темы**.
- Для более глубокого изучения **используйте пособие по курсу и другие материалы на платформе Moodle**: Link