нетология
университет интернет-профессий

ВОЗМОЖНОСТ И JAVASCRIPT В БРАУЗЕРЕ

ИЛЬЯ МЕДЖИДОВ / RageMarket, CEO

ИЛЬЯ МЕДЖИДОВ

RageMarket, CEO

fb.me/ilya.medzhidov

ПЛАН ЗАНЯТИЯ

1. JavaScript и браузеры
   - Браузеры
   - Отладка и `console.log()`
   - Тэг `<script>`
   - Глобальный объект `window`
2. DOM и HTML-теги
   - `document.getElementById()`
   - Атрибуты тегов
3. События

JAVASCRIPT И БРАУЗЕРЫ

А«

Браузер *– это программное обеспечение для отображения веб-страниц. Браузеры очень широко используются в различных устройствах: не только в привычных компьютерах и смартфонах, но и в телевизорах, игровых автоматах и даже терминалах оплаты.*

ВСТРОЕННЫЙ ИНТЕРПРЕТАТОР

Для исполнения скриптов в любой современный браузер встроен *интерпретатор* JavaScript-кода. И это делает браузеры основной средой выполнения приложений, написанных с использованием **JavaScript**, **HTML** и **CSS**.

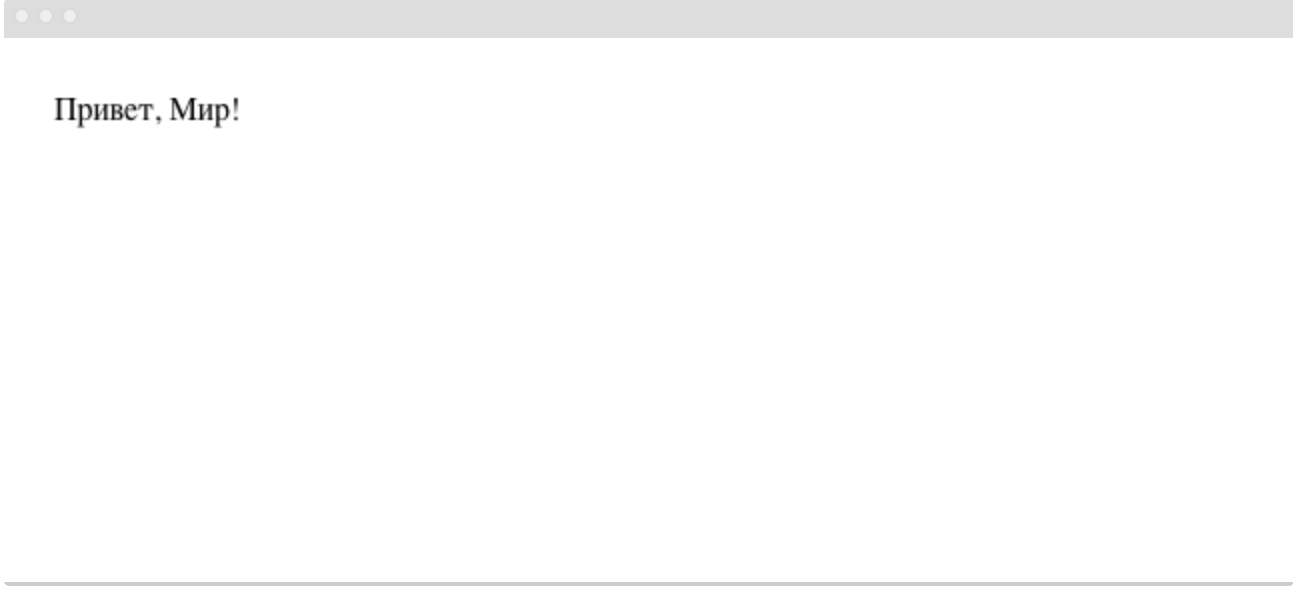
ПРИМЕР ВЕБ-СТРАНИЦЫ

Рассмотрим пример страницы с простой HTML-структурой:

```
<html>
<head>
  <title>Заголовок</title>
</head>
<body>
  Привет, Мир!
</body>
</html>
```

РЕЗУЛЬТАТ ОБРАБОТКИ КОДА

В этом примере нет ничего связанного с JavaScript, поэтому браузер просто отобразит строчку `Привет, Мир!`.



ПОДКЛЮЧАЕМ СКРИПТ

Чтобы браузер начал исполнять JavaScript-код, его необходимо поместить внутрь специального тега `<script>` и добавить этот тег на страницу. Например, вот так:

```
<html>
<head>
  <title>Заголовок</title>
</head>
<body>
  Привет
  <script>
    console.log('Мир!');
  </script>
</body>
</html>
```

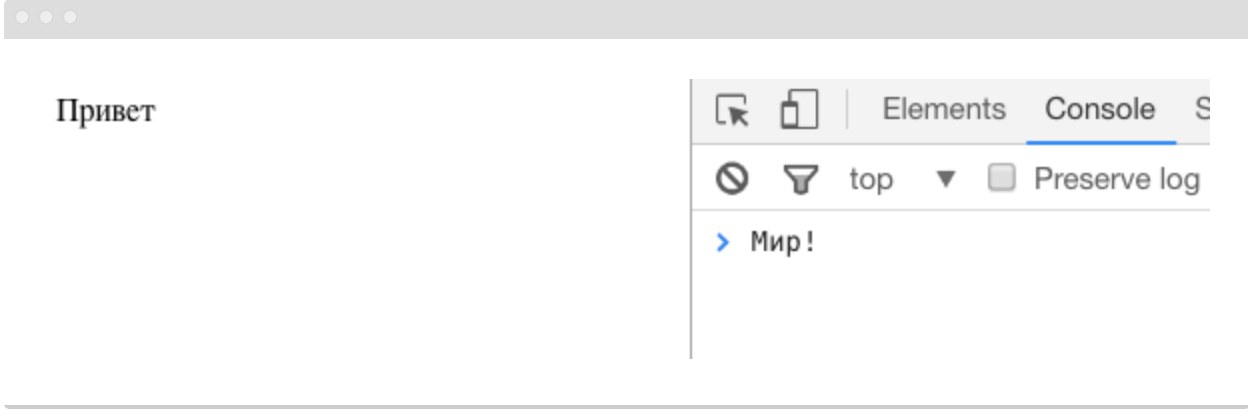
ПОРЯДОК ДЕЙСТВИЙ

Что будет делать браузер:

1. Отобразит содержимое HTML-документа до тега `<script>`;
2. Исполнит содержимое тега `<script>` как JavaScript-код;
3. Продолжит отображать содержимое пока не встретит следующий тег `<script>` или пока не дойдет до конца.

КУДА ДЕЛСЯ «МИР»?

Обычный пользователь увидит лишь строчку `Привет` в результате прошлого примера, потому что результат вызова `console.log('Мир!')` нужно смотреть в специальном интерфейсе – в инструментах разработчика.



ОТКРЫВАЕМ «ИНСТРУМЕНТЫ РАЗРАБОТЧИКА»

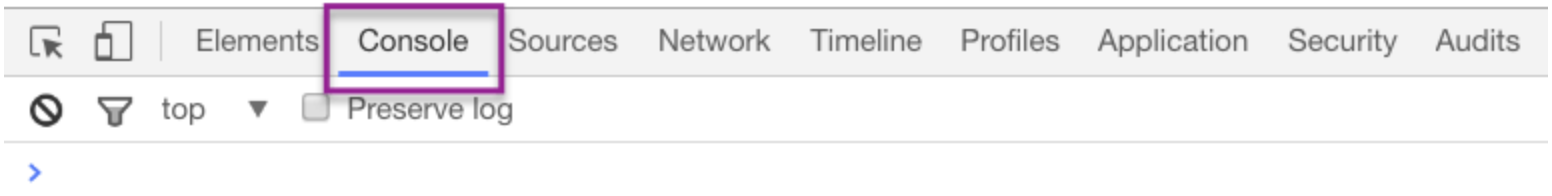
Чтобы быстро открывать «Инструменты разработчика» в разных браузерах, удобно держать под рукой *горячие клавиши*:

- Chrome/Firefox/Opera/Safari:
  - Windows/Linux: `Ctrl` + `Shift` + `I`
  - Mac: `Cmd` + `Opt` + `I`
- IE/Edge: `F12`

ВКЛАДКА «CONSOLE»

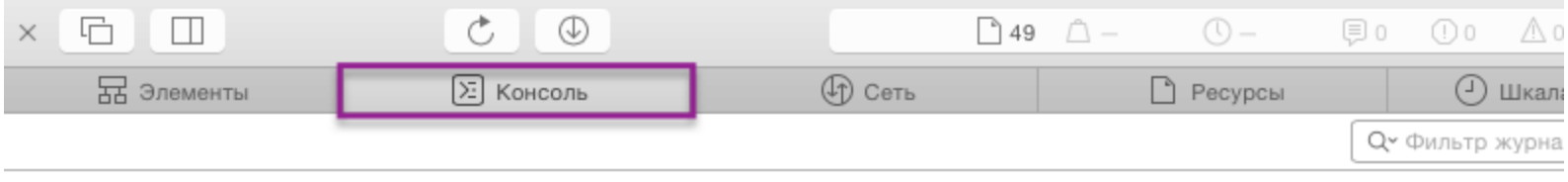
В инструментах разработчика нас интересует вкладка **Console** – именно в ней отображается вывод вызовов `console.log()`.

– Chrome



– Firefox

– Safari



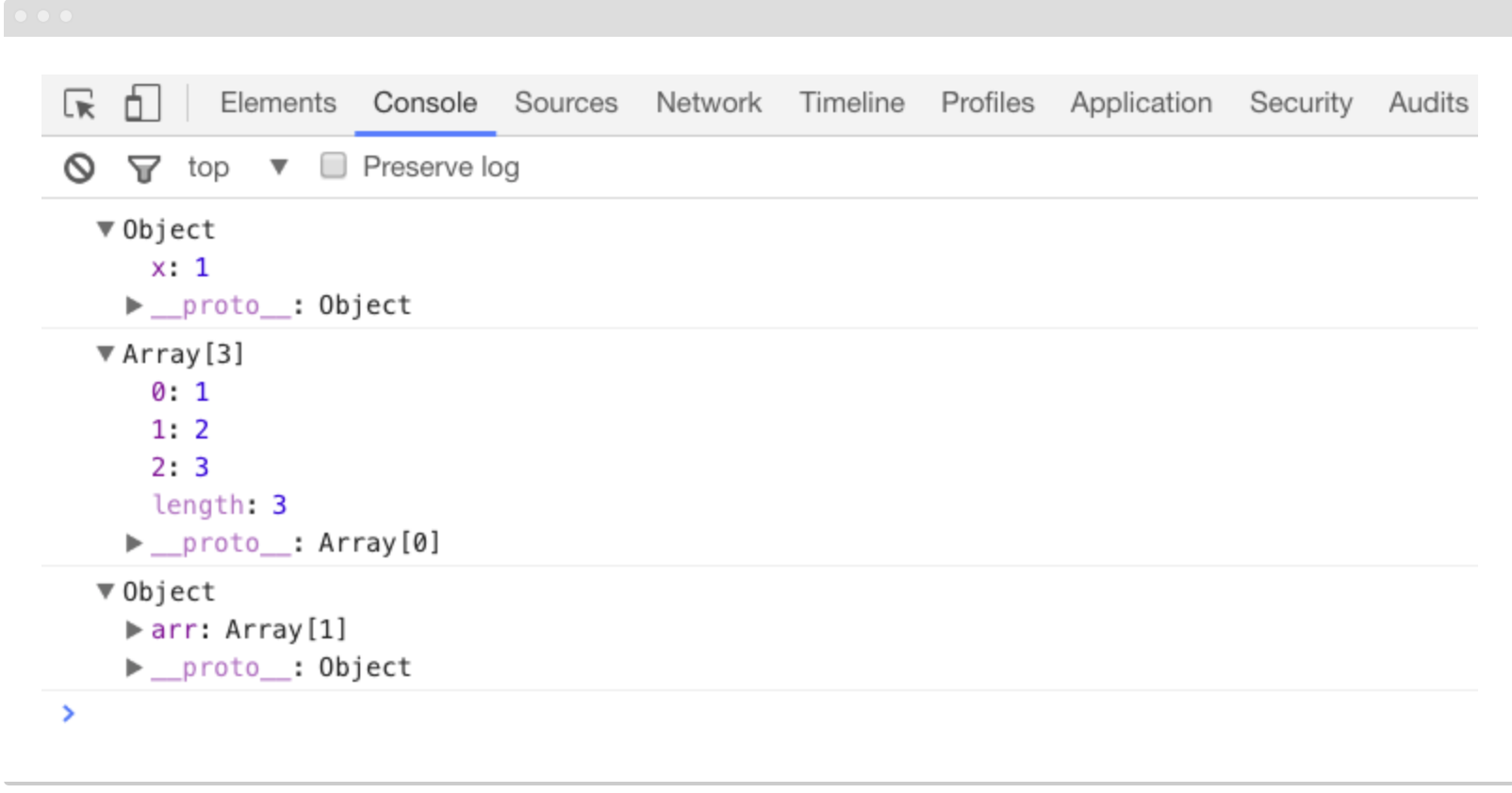
ВЫВОД СЛОЖНЫХ ОБЪЕКТОВ В КОНСОЛЬ

Причем с помощью `console.log()` можно выводить не только строки и числа, но и более сложные объекты:

```
console.log({ x: 1 });  // Объект
console.log([1, 2, 3]); // Массив
console.log({
  arr: [
    { s: 's' }
  ]
}); // объект с глубокой вложенностью
```

РЕЗУЛЬТАТ ВЫВОДА

В результате обработки кода с предыдущего слайда в консоль будет выведена следующая информация:



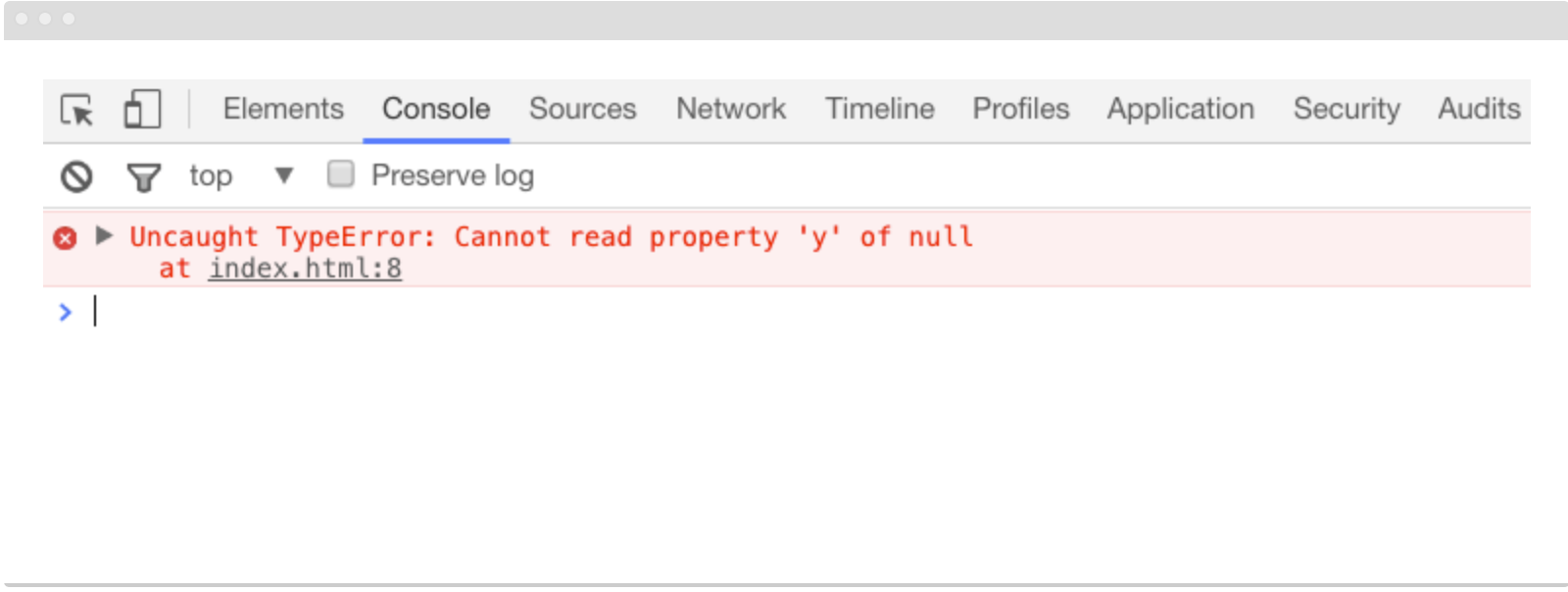
ВЫВОД ОШИБОК

В консоли инструментов разработчика отображаются ошибки, которые могут случиться во время работы скрипта. Создадим HTML-станицу с тегом `<script>` с некой ошибкой, например обращение к свойству `null`:

```
<html>
<head>
  <title>Документ с ошибкой</title>
</head>
<body>
  <script>
    const x = null;
    x.y += 1; // ошибка
  </script>
</body>
</html>
```

РЕЗУЛЬТАТ КОДА С ОШИБКОЙ

Тогда в консоли отобразится текст ошибки и где именно она произошла:



`<script>`

Во всех примерах до этого мы помещали JavaScript-код непосредственно внутри тега `<script>`.

Но есть возможность написать скрипт в отдельном файле, например, **index.js** и потом подключить его к HTML-странице.

ПОДКЛЮЧЕНИЕ ВНЕШНЕГО СКРИПТА

Для подключения внешних скриптов необходимо использовать атрибут `src` тега `<script>`. Значением этого атрибута является путь до файла со скриптом. Пути бывают разных видов.

ОТНОСИТЕЛЬНЫЙ ПУТЬ

Этот скрипт мы загрузили с использованием *относительного пути*, то есть **index.js** должен быть расположен в той же директории, что и загруженная HTML-страница.

```
<script src="index.js"></script>
```

АБСОЛЮТНЫЙ ПУТЬ

Здесь показан *абсолютный путь*. Он начинается с `/` и отсчитывается от корня сайта.

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

ПОЛНЫЙ URL

В этом примере указан *полный URL* до некоего скрипта, находящегося на другом сайте. Он начинается с **http://** или **https://**, далее идет доменное имя, например **ajax.googleapis.com**, а затем уже абсолютный путь до файла.

```
<script src="https://ajax.googleapis.com/ajax/libs/
jquery/3.1.1/jquery.min.js"></script>
```

ПОЛНЫЙ URL БЕЗ УКАЗАНИЯ ПРОТОКОЛА

Так можно загрузить внешний скрипт с другого сайта, но без точного указания протокола. Скрипт будет загружен по **http**, если текущая страница открыта с помощью **http**. Если же текущий протокол **https**, то и загрузка внешнего скрипта пойдет по **https**.

```
<script src="//ajax.googleapis.com/ajax/libs/
jquery/3.1.1/jquery.min.js"></script>
```

ОДНОВРЕМЕННОЕ СОДЕРЖИМОЕ И `src`

Содержимое `<script>` и атрибут `src` одновременно использовать не нужно, потому что содержимое будет проигнорировано.

```
<script src="index.js">
  console.log('test'); // никогда не будет вызвано
</script>
```

А«

Глобальный объект *– обязательная составляющая в любом окружении, в котором исполняется JavaScript-код. Это требование описано в спецификации ECMA-262.*

В браузере глобальный объект – это `window`.

СВОЙСТВА ГЛОБАЛЬНОГО ОБЪЕКТА

Любые переменные, объявленные при помощи `var` вне какой-либо функции, становяться свойствами глобального объекта `window`.

Например:

```
var name = 'Иван';
console.log(window.name); // Иван
// или
window.lastname = 'Иванов';
console.log(lastname); // Иванов
```

`let`, `const` И `window`

Другой эффект при использование одноименных свойства в `window` и переменной `let` или `const`:

```
const age = 1;
window.age = 99;
console.log(age); // 1
```

DOM И HTML-ТЕГИ

A«

HTML-документ *– это набор определенным образом структурированных HTML-тегов.*

Он является инструкцией для браузера о том, как отображать веб-страницу.

HTML-документ является текстовым файлом и может быть просмотрен и отредактирован в любом текстовом редакторе.

ОБЯЗАТЕЛЬНАЯ СТРУКТУРА

Для создания HTML-документа недостаточно добавить любому текстовому файлу расширение **.html**. Требуется соблюсти определенную структуру:

1. В начале всегда идет тег `html`.
2. В него вложены теги `head` и `body`.
   - В `head` находится служебная информация, которая не отображается на странице. Например, `title` с заголовком страницы.
   - Внутри `body` находится все то, что будет видно на экране.

HTML + JAVASCRIPT

Для работы с HTML-документами в JavaScript существует *объектная модель документа* – **DOM**.

Согласно этой модели каждому тегу соответствует *объект-узел*.

Всему HTML-документу соответствует глобальный объект `document`.

С помощью свойств и методов `document` и объектов-узлов осуществляется взаимодействие с HTML-документом.

ИЩЕМ ТЕГ ПО `id`

Для получения объекта DOM, соответствующего нужному тегу, существует несколько разных способов.

Самым простым является метод объекта `document` для поиска по атрибуту `id` – `getElementById()`.

```
<div id="find_me">Найди меня</div>
<script>
  const div = document.getElementById('find_me');
  console.log(div);
  // <div id="find_me">Найди меня</div>
</script>
```

ВОЗМОЖНЫЕ ОШИБКИ

`document.getElementById()` возвращает `null` если ничего не найдено.

Это может случиться по двум причинам:

- тэга с искомым атрибутом `id` нет в документе
- не совпадает регистр символов

```
<div id="FIND_ME">Найди меня</div>
<script>
  const div1 = document.getElementById('div1');
  console.log(div); // null

  const findMe = document.getElementById('find_me');
  console.log(findMe); // null
</script>
```

`id` В ГЛОБАЛЬНОМ ОБЪЕКТЕ

Тег с заданым атрибутом `id` создает одноименное свойство в глобальном объекте `window`.

Давайте это проверим:

```
<div id="find_me">Найди меня</div>
<script>
  const div = document.getElementById('find_me');
  console.log(div === window.find_me); // true
</script>
```

АТРИБУТЫ HTML-ТЕГОВ

Атрибут `id` может быть применим к любому тегу. Но есть и атрибуты, которые применяются только к определенным тегам. Например, у тега `img` есть специальные атрибуты: `src` – адрес картинки, `width` и `height` – ширина и высота.

```
<img id="heart" src="http://bit.ly/2nyH0tY"
width="150" height="150">
```

ЧИТАЕМ АТРИБУТЫ

JavaScript позволяет нам получить значение любого атрибута у любого тега. Попробуем получить значение атрибута `src` у тега `img`:

```
<img id="heart" src="http://bit.ly/2nyH0tY"
width="150" height="150">
<script>
  const img = document.getElementById('heart');
  console.log(img.src); // http://bit.ly/2nyH0tY
</script>
```

ЗАДАЕМ ИНТЕРВАЛ

В JavaScript существует инструмент, позволяющий задавать интервал времени, через который функция будет срабатывать. Посмотрим на пример работы тймера `setInterval`:

```
<script>
  const img = document.getElementById('heart');
  function showSrc() {
    console.log(`${img.src}. Кстати, прошло 10 секунд`);
  }
  setInterval(showSrc, 10000);
</script>
```

МЕНЯЕМ АТРИБУТЫ

Можно поменять атрибуты во время исполнения скрипта. Все атрибуты тега являются свойствами соответствующего объекта-узла.

ПРИМЕР РАБОТЫ С АТРИБУТАМИ

Добавим небольшой скрипт на страницу с картинкой. Live Demo

```
const img = document.getElementById('heart');
let step = 0;
setInterval(() => {
  if (step % 2 === 0) {
    img.width *= 2;
    img.height *= 2;
  } else {
    img.width /= 2;
    img.height /= 2;
  }
  step += 1;
}, 500);
```

СОБЫТИЯ

ПРИЛОЖЕНИЕ ДЛЯ ПОЛЬЗОВАТЕЛЯ

Приложение в браузере чаще всего построено на взаимодействии с пользователем.

Часто встречается кейс, когда человек прокручивает страницу вниз и скрипт загружает новые записи.

Или пользователь нажимает на кнопку и появляется выпадающее меню.

СОБЫТИЯ И СВОЙСТВА ОБЪЕКТОВ-УЗЛОВ

С точки зрения JavaScript взаимодействие с пользователем построено на основе событий и обработчиков этих событий - функций.

Существует несколько различных способов установки связи между событиями и функциями-обработчиками.

Сегодня мы познакомимся с наиболее простым вариантом установки обработчиков событий - на основе свойств объектов-узлов.

СВОЙСТВА НА `on`

Для каждого события на объектах-узлах есть соответствующие свойства, начинающиеся на `on`. Если записать в это свойство функцию, то она будет вызвана в момент наступления события.

СОБЫТИЕ `click`

Событие `click` – нажатие левой кнопкой мыши – может быть применимо к любому элементу HTML-документа: `body`, `div`, `img`, `button` и так далее.

Для этого события у объекта-узла есть свойство `onclick`, в которое мы и будем записывать функцию-обработчик.

```
<button id="element_id">Нажми на меня</button>
<script>
  const element = document.getElementById('element_id');
  element.onclick = function() {
    console.log('click'); // click
  };
</script>
```

ЗНАЧЕНИЕ ПО УМОЛЧАНИЮ

Значение свойства `onclick` по умолчанию равно `null`.

```
const element = document.getElementById('element_id');
console.log(element.onclick); // null
```

АНИМАЦИЯ ПО КЛИКУ

Давайте пример с изменением размеров картинки сделаем на основе события `click`. Live Demo

```
const img = document.getElementById("heart");
let step = 0;
function changeSizes() {
   if (step % 2 === 0) {
     img.width *= 2;
     img.height *= 2;
   } else {
     img.width /= 2;
     img.height /= 2;
   }
   step += 1;
};
img.onclick = changeSizes;
```

ОБРАБОТЧИК ДЛЯ НЕСКОЛЬКИХ УЗЛОВ

Функцию-обработчик можно назначить для нескольких объектов-узлов.

До этого мы получали ссылку на объект-узел из замыкания. В этом случае удобно использовать `this`, потому что функция-обработчик вызывается в контексте объекта-узла тега, на котором происходит обработка события.

Стоит отметить, что для этого случая не подходят стрелочные функции, потому что контекст вызова для них недоступен.

НАЖИМАЕМ КНОПКИ

Live Demo

```
<button id="btn1">Кнопка 1</button>
<button id="btn2">Кнопка 2</button>
<button id="btn3">Кнопка 3</button>
<script>
  function logElementId() {
    console.log(this.id);
  }
  for (const btnId of ['btn1', 'btn2', 'btn3']) {
    const btn = document.getElementById(btnId);
    btn.onclick = logElementId;
  }
</script>
```

ПРЕКРАТИТЬ ОБРАБОТКУ СОБЫТИЯ

Как прекратить обработку события? Для этого в соответствующее свойство необходимо записать `null`.

Например, можно отменить обработку события после первого срабатывания. Live Demo

```
<button id="element_id">Нажми на меня</button>
<script>
  const element = document.getElementById('element_id');
  element.onclick = function() {
    console.log("Первое и единственное срабатывание
    функции-обработчика")
    element.onclick = null;
  };
</script>
```

ИТОГИ

ПОДКЛЮЧЕНИЕ СКРИПТА

- Браузеры обрабатывают JavaScript-код автоматически;
- Скрипт подключается к HTML-странице при помощи тега `<script>`;
- Во всех браузерах существуют «Инструменты разработчика»;
- Вывести информацию в консоль можно при помощи `console.log()`;
- В консоли можно посмотреть ошибки, возникшие в ходе работы скрипта;
- Можно писать JavaScript-код не только внутри HTML-разметки, но и в отдельном файле;
- Подключается внешний JavaScript-код с помощью атрибута `src` тега `<script>`

JAVASCRIPT В БРАУЗЕРЕ

- Все переменные, объявленные через `var` за пределами функций являются свойствами глобального объекта `window`;
- Для работы с HTML в JavaScript существует DOM;
- Найти тег по `id` можно при помощи метода `getElementById()`;
- Атрибуты можно читать и изменять через свойства объектов-узлов;
- Любые действия пользователя с точки зрения JavaScript – события;
- Событие `click` - нажатие левой кнопкой мыши по элементу;
- Для события `click` у объекта-узла есть свойство `onclick`
- Одну функцию обработчик можно *повесить* на несколько узлов;
- Событие прекращает отслеживаться, если установить `null` в соответствующее свойство;

нетология
университет интернет-профессий

Задавайте вопросы и напишите отзыв о лекции!

ИЛЬЯ МЕДЖИДОВ

fb.me/ilya.medzhidov