

Лекция 2

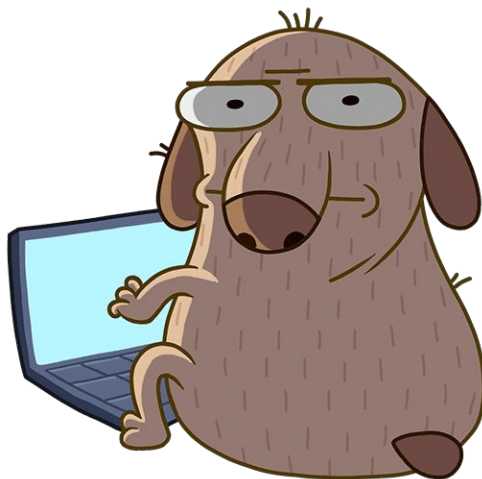
Введение во Frontend.

ОСНОВЫ JS.



Поставь класс и подпишись на канал

- Не забываем отмечаться на лекции
- Очень ждем обратную связь :)



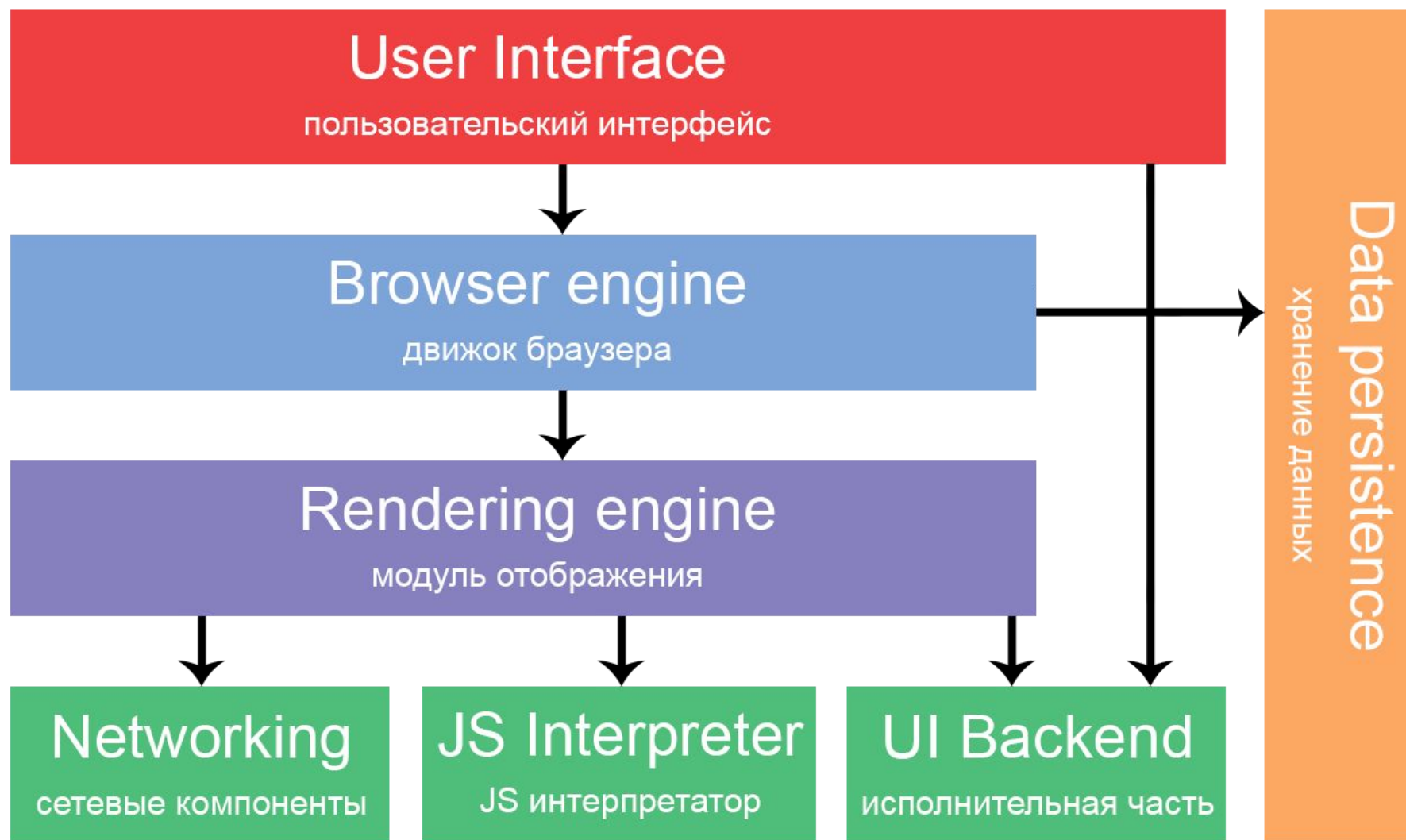
План на сегодня

- Немного о работе браузера
- Современные инструменты разработчика
- Основы JS, CSS
- Web Components 🙄
- Cookies
- Local/Session storage

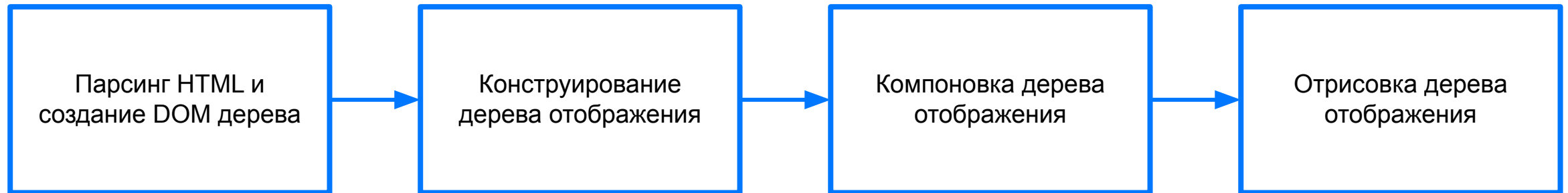
0 браузеров



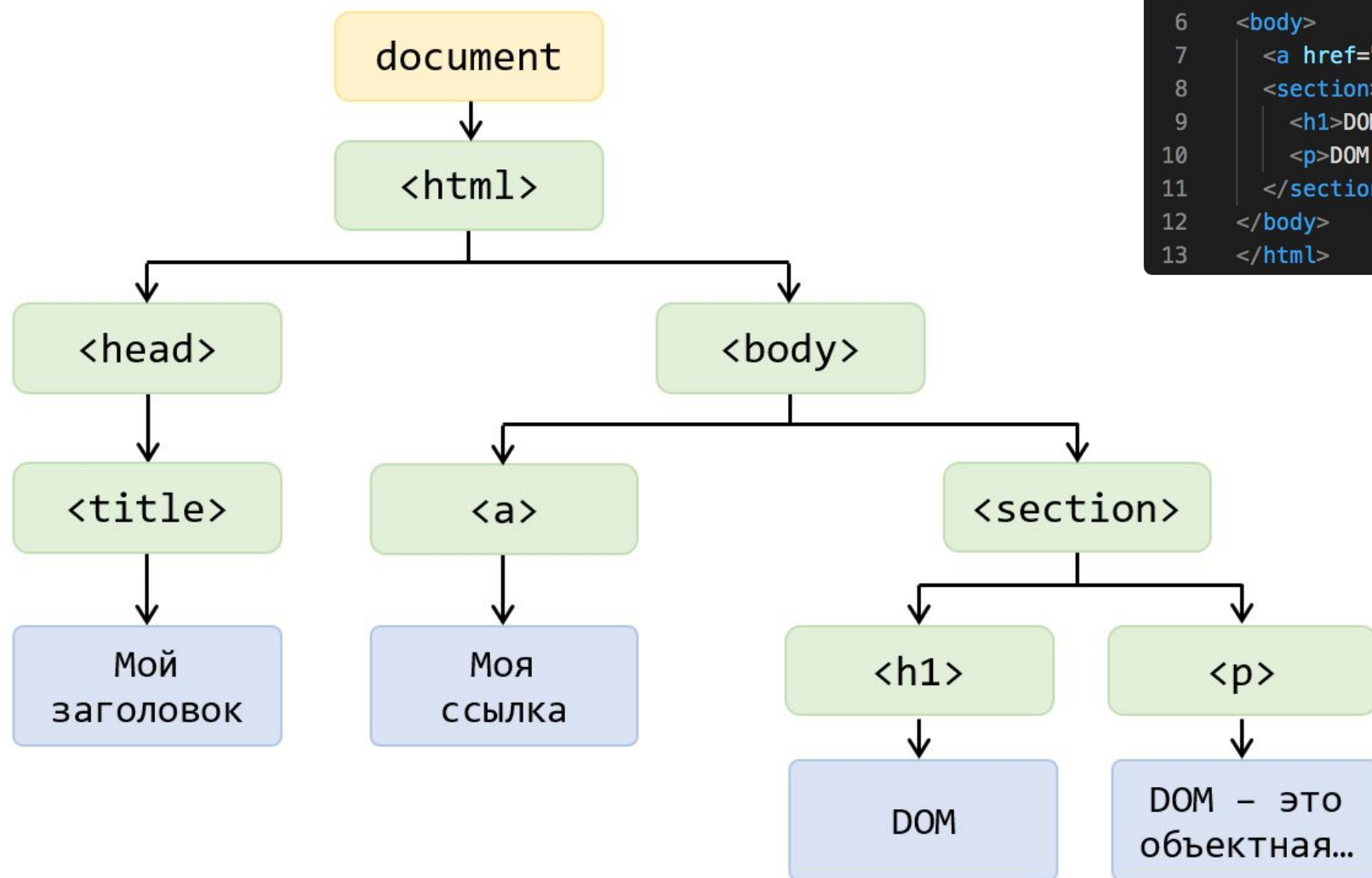
Из чего состоит браузер



Как происходит рендеринг

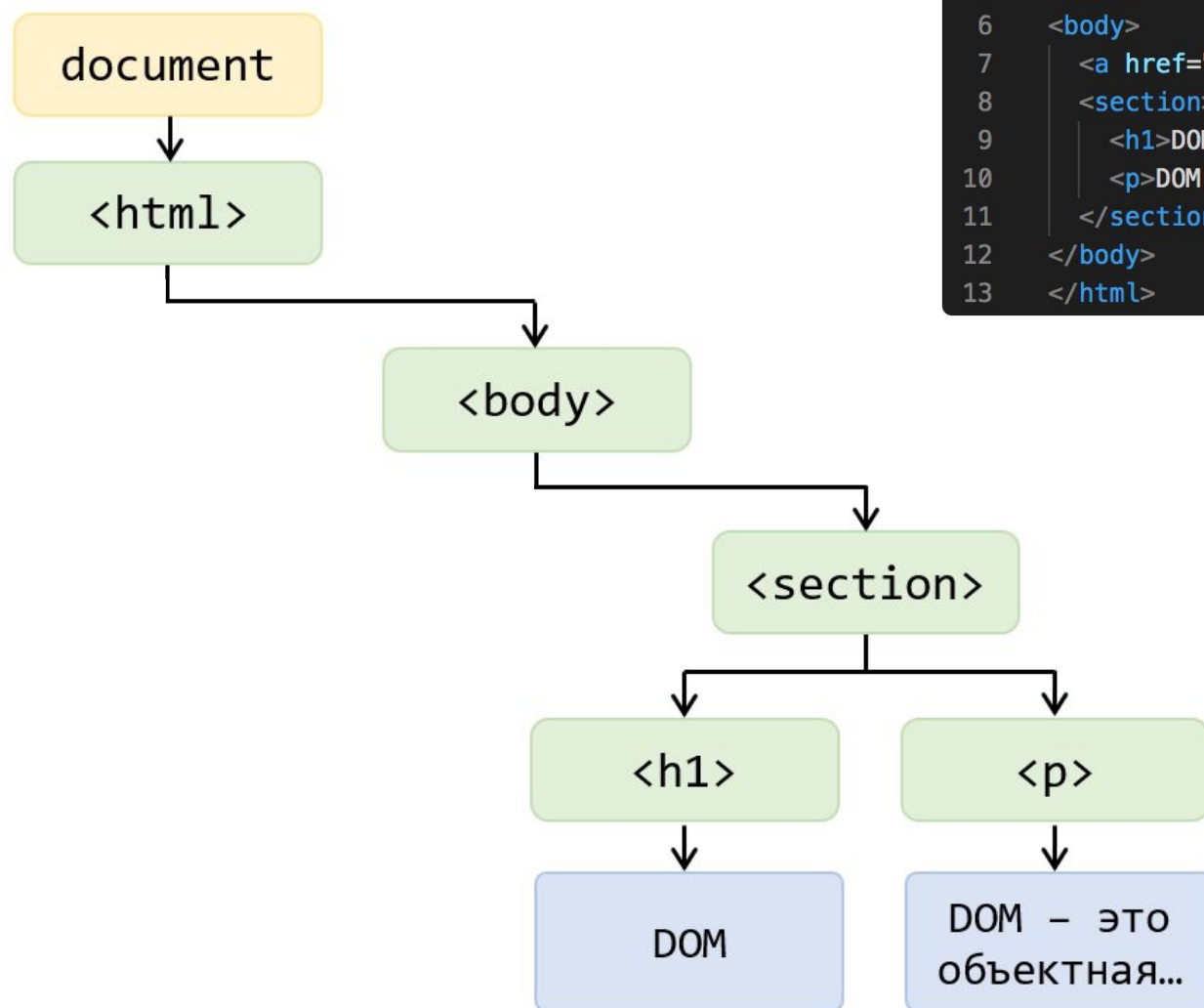


DOM



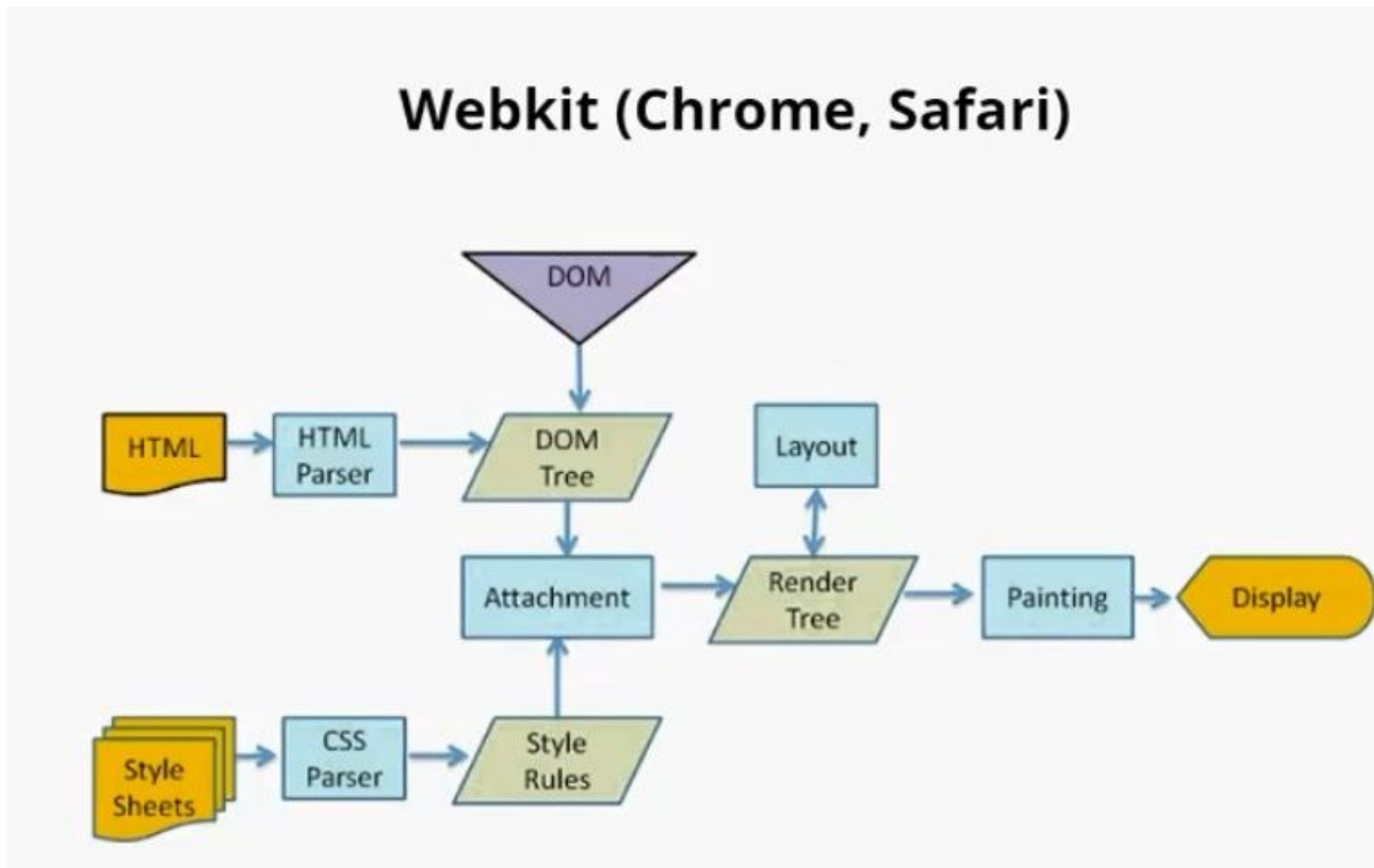
```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4  | <title>Мой заголовок</title>
5  </head>
6  <body>
7  | <a href="#" style="display: none;">Моя ссылка</a>
8  | <section>
9  | | <h1>DOM</h1>
10 | | <p>DOM - это объектная...</p>
11 | </section>
12 </body>
13 </html>
```

Render tree



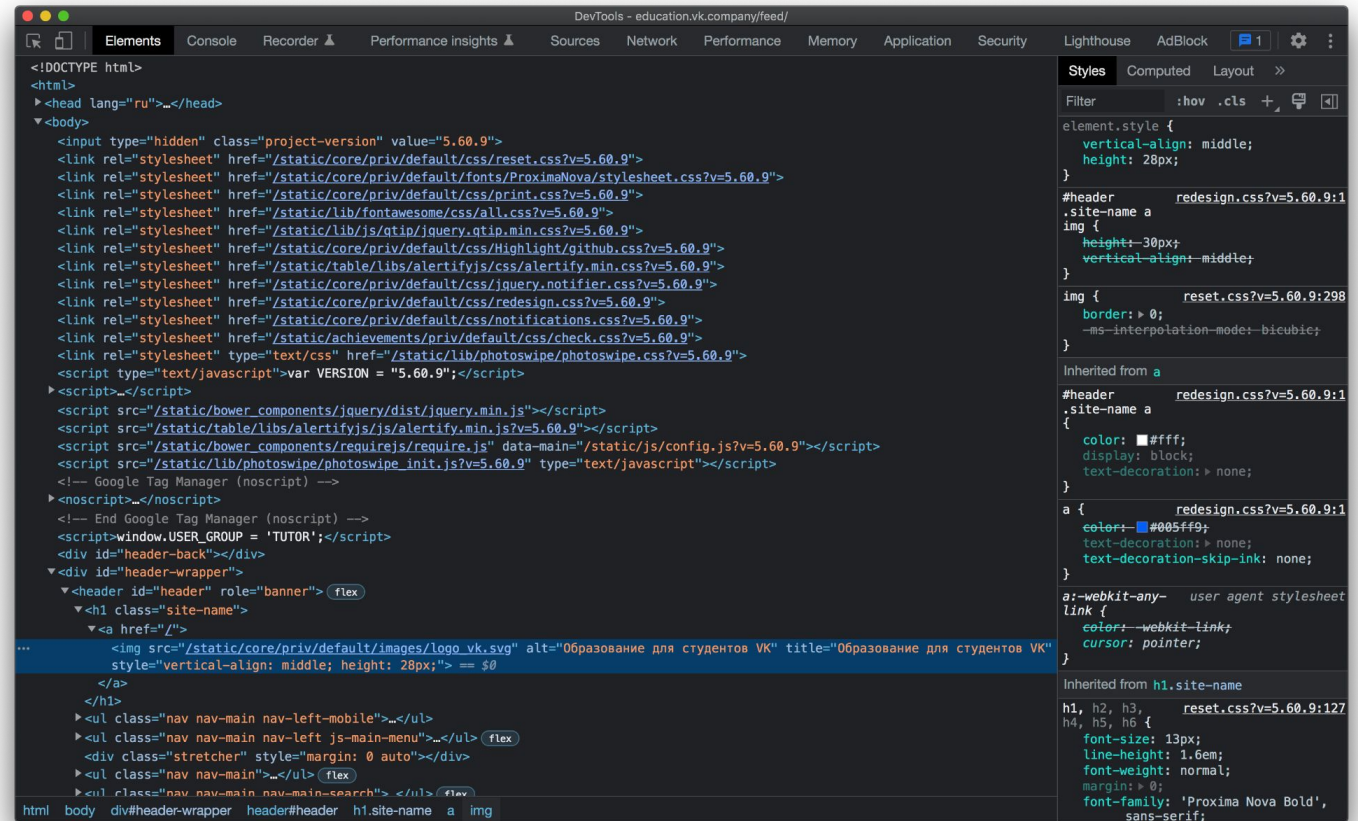
```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <title>Мой заголовок</title>
5  </head>
6  <body>
7    <a href="#" style="display: none;">Моя ссылка</a>
8    <section>
9      <h1>DOM</h1>
10     <p>DOM - это объектная...</p>
11   </section>
12 </body>
13 </html>
```


Как происходит рендеринг в Webkit



Chrome developer tools

- Инспектор DOM и CSS
- Дебаггер JS
- Инспектор сетевых запросов
- Профилировщик
- Просмотр Cookie, Local/Session storage
- Установка расширений для разработки под фреймворки (React/Vue/...)



JavaScript



В двух словах

- Выполняется на клиенте
- Интерпретируемый
- Прототипно-ориентированный
- Динамическая типизация
- Интеграция с HTML и CSS
- Поддерживается всеми современными браузерами и включен по умолчанию

Типы данных

Примитивы

- number
- string
- boolean
- null
- undefined
- symbol
- BigInt

Объекты

- все остальное

Типы данных

Оператор **typeof** возвращает строку с типом аргумента. Это полезно, когда мы хотим обрабатывать значения различных типов по-разному или просто хотим сделать проверку.

```
1  typeof undefined // "undefined"
2
3  typeof 0 // "number"
4
5  typeof 10n // "bigint"
6
7  typeof true // "boolean"
8
9  typeof "foo" // "string"
10
11 typeof Symbol("id") // "symbol"
12
13 typeof Math // "object" (1)
14
15 typeof null // "object" (2)
16
17 typeof alert // "function" (3)
```

Преобразование типов

Логическое

```
Boolean(0)           // false
Boolean(null)        // false
Boolean(undefined)   // false
Boolean(NaN)         // false
Boolean('')          // false
// Все остальное - true
```

Численное

```
Number(undefined)    // NaN
Number(null)          // 0
Number(false)         // 0
Number(true)          // 1
Number('')            // 0
Number(' 10 кошек')   // NaN
Number('    10    ')  // 10
Number('собака')       // NaN
```

Строковое

```
String(42)           // "42"
String(true)          // "true"
String(undefined)     // "undefined"
```

Операторы сравнения

- >

- <

- >=

- <=

- ==

- !=

- ===

- !==

Использование нестрогого сравнения `==` может вызывать проблемы. Например, оно не отличает **0** от **false**. Это происходит из-за того, что операнды разных типов преобразуются оператором `==` к числу. В итоге, и пустая строка, и **false** становятся нулём.

Используйте `===` и `!==`, чтобы сравнивать без приведения типов!

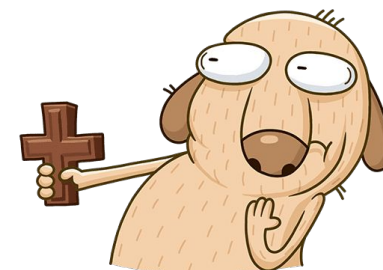


Перерыв! (10 минут)

Препоd (с)

Достаем листочки, пишем контрольную

- `console.log(4 + 3 + 2 + "1");`
- `console.log(typeof typeof 1);`
- `console.log(1 < 2 < 3);`
- `console.log(3 > 2 > 1);`
- `console.log(null > 0);`
- `console.log(null == 0);`
- `console.log(null >= 0);`



Операторы

- Условные операторы: **if**, **'?'**
- Циклы: **for**, **while**
- Конструкция **switch**

Замыкания

```
var i = 10;
var array = [];

while (i--) {
    array.push(function() {
        return i + i;
    });
}

console.log([
    array[0](),
    array[1](),
]);
```

Замыкания

```
var i = 10;
var array = [];

while (i--) {
    array.push(function() {
        return i + i;
    });
}

console.log([
    array[0](),
    array[1](),
]);
```

```
var i = 10;
var array = [];

while (i--) {
    (function (i) {
        array.push(function() {
            return i + i;
        });
    })(i)
}

console.log([
    array[0](),
    array[1](),
]);
```

Оператор new, ключевое слово this

```
function myFunction () {  
    console.log(this)  
}  
  
myFunction()  
new myFunction()
```

- Ключевое слово **this** ссылается на текущий контекст
- Оператор **new** вызывает функцию в контексте нового объекта

Прототипное наследование

- Все объекты имеют скрытое свойство **[[Prototype]]**, которое является либо другим объектом, либо null.
- Когда мы хотим прочитать свойство из Object, а оно отсутствует, JavaScript автоматически берёт его из прототипа.
- Операции записи/удаления работают непосредственно с объектом, они не используют прототип.
- Свойство **Function.prototype** (не путать с **[[Prototype]]**) устанавливает **[[Prototype]]** для новых объектов при использовании оператора **new**

Классы

```
class Animal {  
    constructor(name) {  
        this.name = name;  
    }  
  
    walk() {  
        console.log("I walk: " + this.name);  
    }  
}  
  
class Rabbit extends Animal {  
    walk() {  
        super.walk();  
        console.log("...and jump!");  
    }  
}  
  
new Rabbit("Вася").walk();
```

- Оператор **extend** позволяет описывать новые классы на основании существующих.

Современный JavaScript

- get/set/defineProperty
- let/const
- map/reduce/filter
- деструктуризация

```
let [first, second] = 'Hello world!'.split(' ');  
//first - 'Hello', second - 'world'  
  
let [first, second, ...rest] = 'Hello world! My name is JS'.split(' ');  
//first - 'Hello', second - 'world', rest - массив
```

- стрелочные функции

```
element.addEventListener('click', event => event.preventDefault());  
['one', 'two'].map(item => item.toUpperCase());
```

HTML и CSS



HTML, CSS

```
<!doctype html>
<html>
  <head>
    <meta charset='utf-8' />
    <title>Hello world!</title>
    <link rel="stylesheet" href="./style.css">
  </head>
  <body>
    <h1>Статья</h1>
    <article>
      <p>Текст статьи.</p>
    </article>
    <footer>
      «Подвал» страницы
    </footer>
    <script src="./bundle.js"></script>
  </body>
</html>
```

Подключение CSS

Атрибут style

```
<body>
  <div style="border: 1px solid red">text</div>
</body>
```

Ter style

```
<style>
  div {
    border: 1px solid red;
  }
</style>
```

CSS файл

```
<link rel="stylesheet" href="style.css">
```

CSS селекторы

Простые

- `.class`
- `#id`
- `*`
- `[attribute]`

Комбинаторы

- `div p`
- `div > p`
- `div + p`
- `div ~ p`
- `col.selected || td`

CSS селекторы. Специфичность

- 1 если правило определено в атрибуте style, 0 иначе (= a)
- число атрибутов id в селекторе (= b)
- количество других атрибутов и псевдоклассов в селекторе (= c)
- количество имен элементов и псевдоэлементов в селекторе (= d)

```
* {} /* a=0 b=0 c=0 d=0 -> specificity = 0,0,0,0 */
li {} /* a=0 b=0 c=0 d=1 -> specificity = 0,0,0,1 */
li::first-line {} /* a=0 b=0 c=0 d=2 -> specificity = 0,0,0,2 */
ul li {} /* a=0 b=0 c=0 d=2 -> specificity = 0,0,0,2 */
ul ol+li {} /* a=0 b=0 c=0 d=3 -> specificity = 0,0,0,3 */
h1 + *[rel=up] {} /* a=0 b=0 c=1 d=1 -> specificity = 0,0,1,1 */
ul ol li.red {} /* a=0 b=0 c=1 d=3 -> specificity = 0,0,1,3 */
li.red.level {} /* a=0 b=0 c=2 d=1 -> specificity = 0,0,2,1 */
#x34y {} /* a=0 b=1 c=0 d=0 -> specificity = 0,1,0,0 */
style="" /* a=1 b=0 c=0 d=0 -> specificity = 1,0,0,0 */
```

CSS селекторы

Псевдо-классы

- :first-of-type
- :visited
- :checked
- :hover

Псевдо-элементы

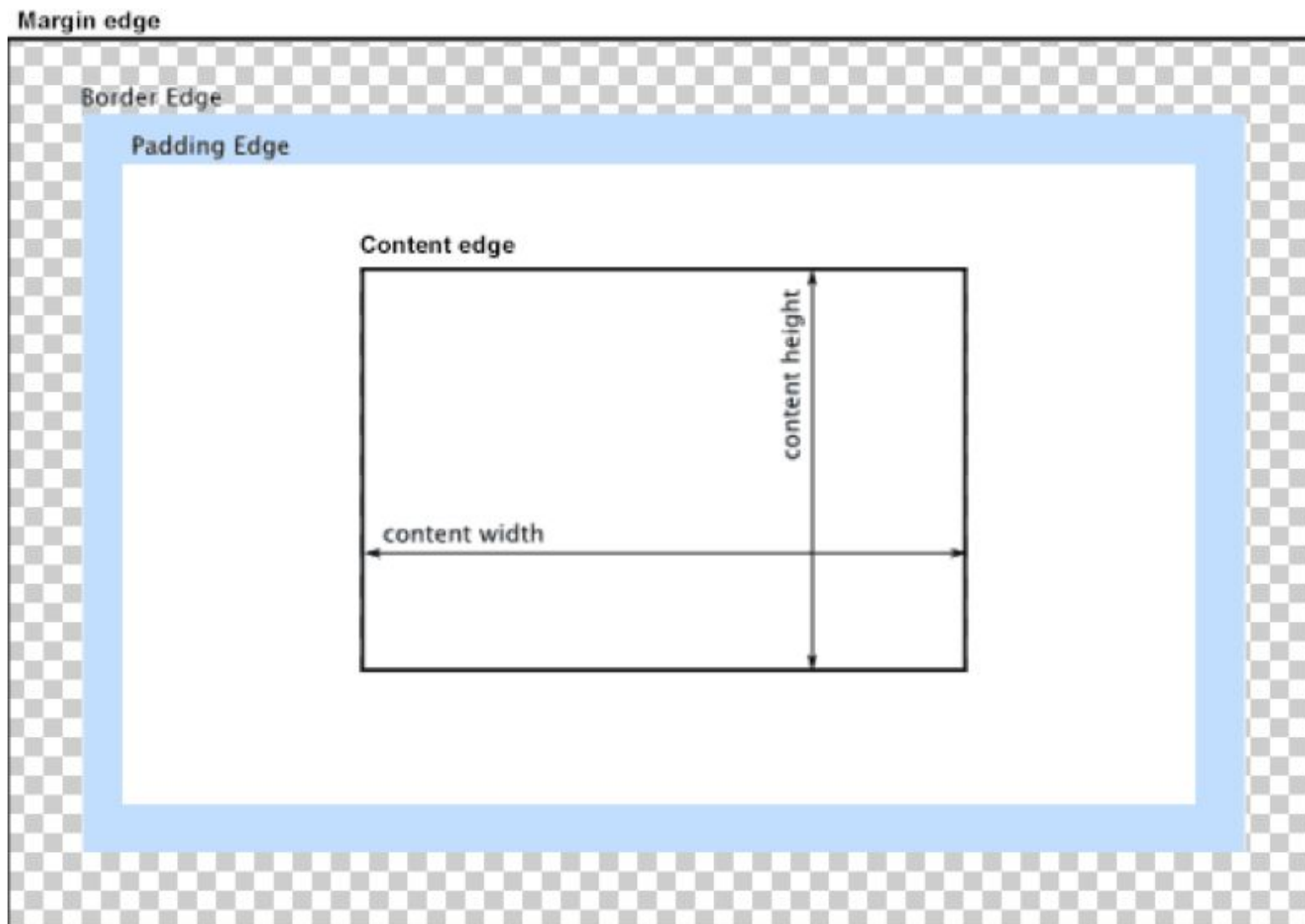
- ::before
- ::after
- ::first-line

CSS и JS

```
<body>
  <style>
    div {
      display: inline;
    }
  </style>
  <div style="border: 1px solid red">text</div>
</body>
```

```
var myDiv = document.body.querySelector('div:first-of-type');
myDiv.style.borderColor; // 'red'
myDiv.style.display; // ""
getComputedStyle(myDiv).display; // "inline"
```

Блочная модель



Flexbox

<https://codepen.io/enxaneta/full/adLPwv>

Cookies

Куки – это небольшие строки данных, которые хранятся непосредственно в браузере. Куки обычно устанавливаются веб-сервером при помощи заголовка *Set-Cookie*. Затем браузер будет автоматически добавлять их в (почти) каждый запрос на тот же домен при помощи заголовка *Cookie*.

- path
- domain
- expires/max-age
- secure
- samesite

LocalStorage/SessionStorage

Объекты веб-хранилища **localStorage** и **sessionStorage** позволяют хранить пары ключ/значение в браузере.

- В отличие от куки, объекты веб-хранилища не отправляются на сервер при каждом запросе. Именно поэтому мы можем хранить гораздо больше данных. Большинство современных браузеров могут выделить как минимум 5 мегабайтов данных (или больше), и этот размер можно поменять в настройках.
- Сервер не может манипулировать объектами хранилища через HTTP-заголовки. Всё делается при помощи JavaScript.

```
LocalStorage.getItem(key)
SessionStorage.setItem(key, value)
Storage.removeItem(key)
Storage.clear()
```

WebComponents

Мы можем создавать пользовательские HTML-элементы, описываемые нашим классом, со своими методами и свойствами, событиями и так далее. Как только пользовательский элемент определен, мы можем использовать его наравне со встроенными HTML-элементами.

```
class MyBestComponent extends HTMLElement {
  connectedCallback() {
    const value = this.getAttribute('value') || undefined,

    const html = `
      <div class="block">
        <h1>${value}</h1>
        <p>This is another text in paragraph</p>
        <button>Click me!</button>
      </div>
    `;

    this.innerHTML = html;
  }
}

customElements.define("my-best-component", MyBestComponent);

<my-best-component value="Hello from WebComponent"></my-best-component>
```

Вставка от Мартина про ДЗ



Домашнее задание №2

1. Разработать первый экран приложения
2. Задеплоить его на Github Pages

Расширенное описание задания, подсказки, а также презентации с лекций всегда есть в [репозитории](#).

Срок сдачи

18 октября

Спасибо за внимание!



Пока!

Присоединяйтесь к сообществам про образование в VK

- [VK Джуниор](#)
- [VK Образование](#)

