О преподавателе

Басов Денис Алексеевич

- PHP
- JavaScript
- SQL
- HTML
- CSS

JavaScript. Уровень 2. Расширенные возможности

Тема	Часы
Модуль 1. Углубленная работа с функциями *Замыкания, функции высшего порядка, call, bind, apply*	3
Модуль 2. Методы массивов *foreach, map, filter, reduce*	1
Модуль 3. Объектная модель браузера *BOM/DOM, Поиск элементов и коллекции*	4
Модуль 4. DOM *DOM, узлы, связи, работа со стилями, обход коллекции*	5
Модуль 5. События *типы событий, обработчики, отмена действия по умолчанию*	5
Модуль 6. Практическая работа *Обработка данных формы*	4
Модуль 7. Введение в асинхронный JS *async/await, fetch*	4
Модуль 8. Введение в модули и Node.js *модули, экспорт/импорт, npm*	4
Модуль 9. Тестирование и сборка *jest.js, написание и запуск тестов, parsel.js*	5
Модуль 10. Практическая работа *Разработка клиент-серверного приложения*	4

В рамках курса мы научимся:

- Углубленной работе с функциями
- Использовать JS для работы с HTML
- Обрабатывать события в браузере
- Работать с асинхронными запросами на сервер
- Тестировать написанный код

Бауманский учебный центр
Специалист.ru
Мировые стандарты обучения

Модуль 1. Углубленная работа с функциями

Модуль 2. Методы массивов

Модуль 3. Объектная модель браузера

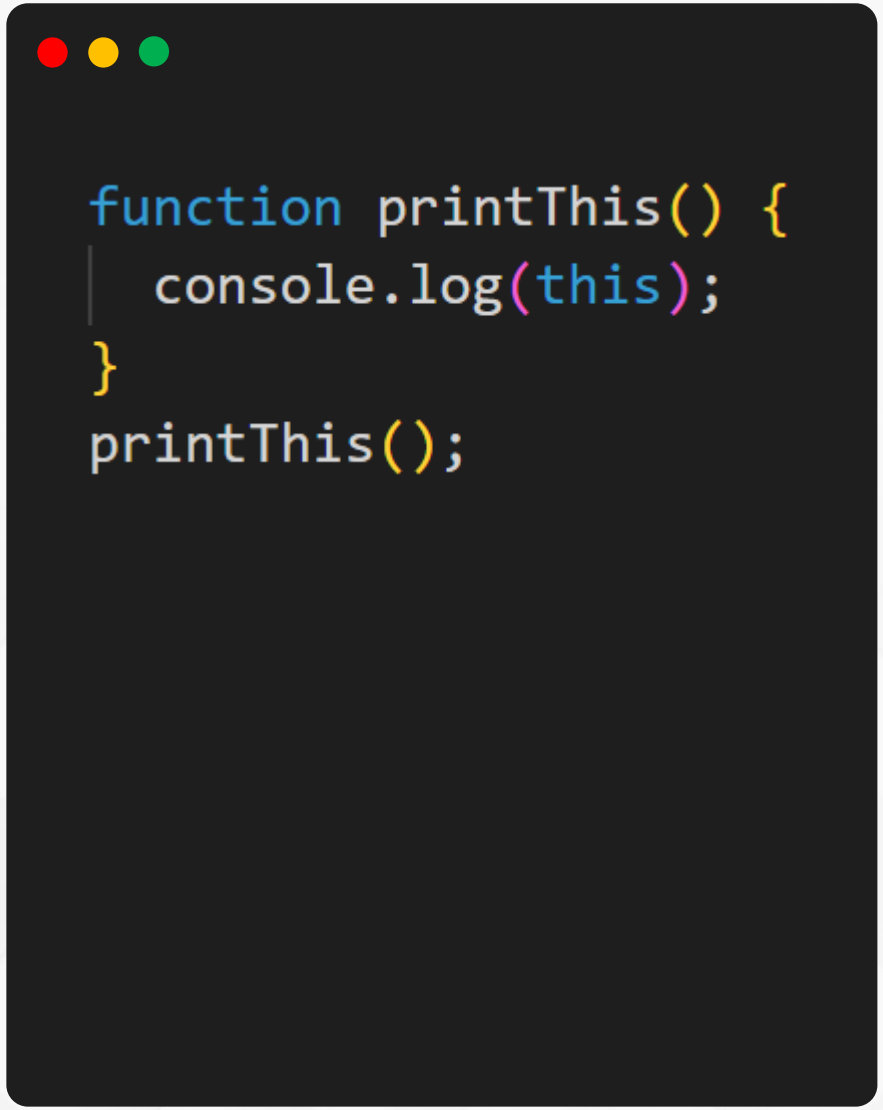
Басов Денис
specialist.ru

В модуле 1, 2, 3 мы рассмотрим:

- Методы apply, call, bind. Замыкания. Рекурсия
- Функции высшего порядка. Каррирование
- Методы map, filter, reduce ...
- Введение в BOM/DOM
- Событийные HTML-атрибуты
- Таймеры и requestAnimationFrame
- Интерфейсы
- Поиск элементов и коллекции

this – это объект, свойством которого является вызванный метод

This в простых функциях



```
function printThis() {
  console.log(this);
}
printThis();
```

Глобальный объект window

```
function printThis() {
  console.log(this);
}
window.printThis();
```

Строгий режим

```
function printThis() {
  "use strict";
  console.log(this);
}
printThis();
```

This и объекты

```
const student = {
  name: "Sarah",
  greet() {
    return `My name is ${this.name}`;
  },
};

console.log(student.greet());
```

This и объекты

```
const student = {
  name: "Sarah",
  greet() {
    return `My name is ${this.name}`;
  },
  newGreet() {
    return `${this.greet()} !!!`;
  },
};

console.log(student.newGreet());
```

Функции + объекты

```
function greet() {
  return `My name is ${this.name}`;
}
const name = "Irina";
const user1 = {
  name: "Ivan",
  greet: greet,
};
const user2 = {
  name: "Bob",
  greet: greet,
};

console.log(greet());
console.log(user1.greet());
console.log(user2.greet());
```

this и вложенные объекты

```
function a() {
  console.log("a", this);
  function b() {
    console.log("b", this);
    const c = {
      hi: function () {
        console.log("c", this);
      },
    };
    c.hi();
  }
  b();
}
a();
```

Функции в методах объектов

```
const student = {
  name: "Anna",
  a() {
    console.log("a", this);
    function b() {
      console.log("b", this);
    }
    b();
  },
};
student.a();
```

Стрелочная функция и this

```
const student = {
  name: "Anna",
  a() {
    console.log("a", this);
    // стрелочная функция
    const b = () => {
      console.log("b", this);
    };
    b();
  },
};
student.a();
```

Еще один способ исправления

```
const student = {
  name: "Anna",
  a() {
    console.log("a", this);
    const self = this;
    function b() {
      console.log("b", self);
    }
    b();
  },
};
student.a();
```

Function.prototype.call()

```
let wizard = {
  name: "Merlin",
  health: 50,
  heal() {
    return (this.health = 100);
  },
};

let archer = {
  name: "Robin Hood",
  health: 30,
};

console.log(wizard.heal());
console.log(archer.heal());
```

```
let wizard = {
  name: "Merlin",
  health: 50,
  heal() {
    return (this.health = 100);
  },
};

let archer = {
  name: "Robin Hood",
  health: 30,
};

console.log(1, archer);
wizard.heal.call(archer);
console.log(2, archer);
```

Function.prototype.call()

```
let wizard = {
  name: "Merlin",
  health: 50,
  heal(value) {
    return (this.health += value);
  },
};

let archer = {
  name: "Robin Hood",
  health: 30,
};

console.log(1, archer);
wizard.heal.call(archer, 10);
console.log(2, archer);
```

```
let wizard = {
  name: "Merlin",
  health: 50,
  heal(value1, value2) {
    return (this.health += value1 - value2);
  },
};

let archer = {
  name: "Robin Hood",
  health: 30,
};

console.log(1, archer);
wizard.heal.call(archer, 50, 20);
console.log(2, archer);
```

Function.prototype.apply()

```
let wizard = {
  name: "Merlin",
  health: 50,
  heal(value) {
    return (this.health += value);
  },
};

let archer = {
  name: "Robin Hood",
  health: 30,
};

console.log(1, archer);
wizard.heal.apply(archer, [50]);
console.log(2, archer);
```

```
let wizard = {
  name: "Merlin",
  health: 50,
  heal(value1, value2) {
    return (this.health += value1 - value2);
  },
};

let archer = {
  name: "Robin Hood",
  health: 30,
};

console.log(1, archer);
wizard.heal.apply(archer, [50, 10]);
console.log(2, archer);
```

Function.prototype.bind()

```
let wizard = {
  name: "Merlin",
  health: 50,
  heal(value) {
    return (this.health += value);
  },
};

let archer = {
  name: "Robin Hood",
  health: 30,
};

console.log(1, archer);
wizard.heal.bind(archer, 50);
console.log(2, archer);
```

```
let wizard = {
  name: "Merlin",
  health: 50,
  heal(value) {
    return (this.health += value);
  },
};

let archer = {
  name: "Robin Hood",
  health: 30,
};

console.log(1, archer);
let healArcher = wizard.heal.bind(archer, 50);
healArcher();
console.log(2, archer);
```

Currying - каррирование

```
function multiply(num1, num2) {
  return num1 * num2;
}

const multiplyByTwo = multiply.bind(this, 2);
console.log(multiplyByTwo(4));
console.log(multiplyByTwo(8));

const multiplyByFive = multiply.bind(this, 5);
console.log(multiplyByFive(10));
console.log(multiplyByFive(5));
```

Рекурсия

```
function powerOf(num, pow) {
  if (pow === 1) {
    return num;
  }
  return num * powerOf(num, pow - 1);
}

console.log(powerOf(2, 3));
```

```
function powerOf(num, pow) {
  let result = 1;

  for (let i = 0; i < pow; i++) {
    result *= num;
  }
  return result;
}

console.log(powerOf(2, 3));
```

Closures - замыкания

```
let closure = function () {
  let count = 0;
  return function increment() {
    count++;
    console.log(count);
  };
};

const increment = closure();
increment();
increment();
increment();
```

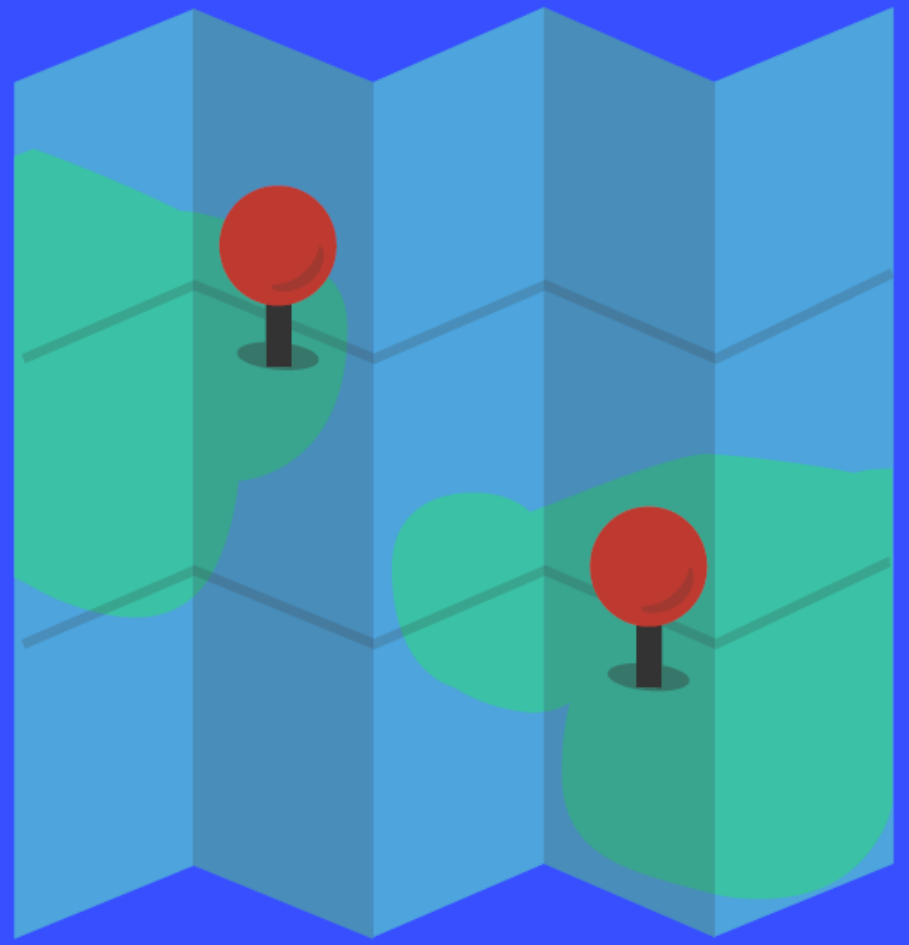
FOREACH

```
const nums = [9, 8, 7, 6, 5, 4, 3, 2, 1];

nums.forEach(function (n) {
  console.log(n * n)
  //prints: 81, 64, 49, 36, 25, 16, 9, 4, 1
});

nums.forEach(function (el) {
  if (el % 2 === 0) {
    console.log(el)
    //prints: 8, 6, 4, 2
  }
})
```

Принимает функцию обратного вызова. Вызывает функцию один раз для каждого элемента массива.

MAP

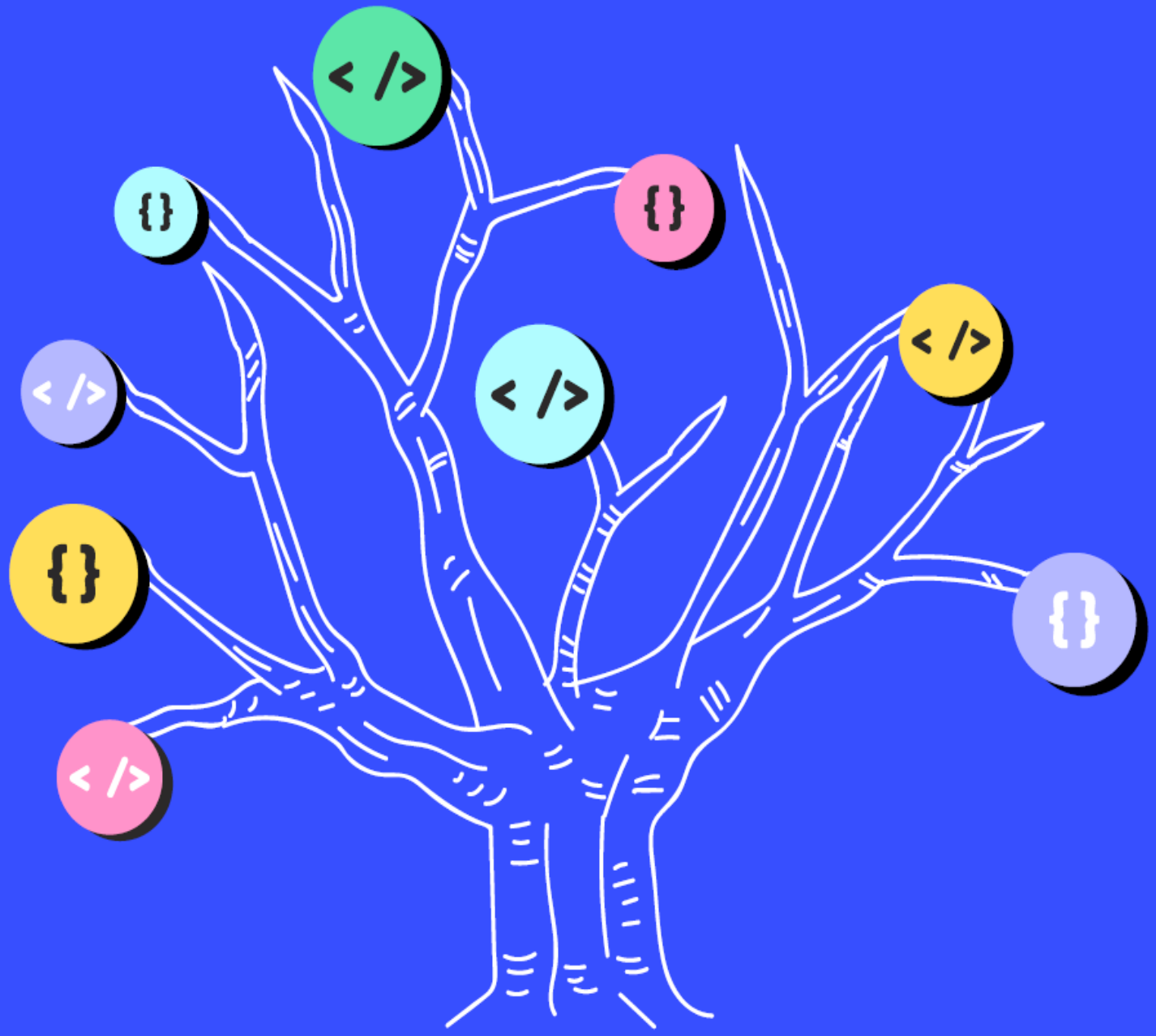
Создает новый массив с результатами вызова функции обратного вызова на каждом элементе массива



MAP

```
const texts = ['rofl', 'lol', 'omg', 'ttyl'];
const caps = texts.map(function (t) {
  return t.toUpperCase();
})
texts; //["rofl", "lol", "omg", "ttyl"]
caps; //["ROFL", "LOL", "OMG", "TTYL"]
```

THE DOM



УБЕДИТЕСЬ, ЧТО ВЫ ЗНАКОМЫ С ОСНОВАМИ **HTML И CSS!**

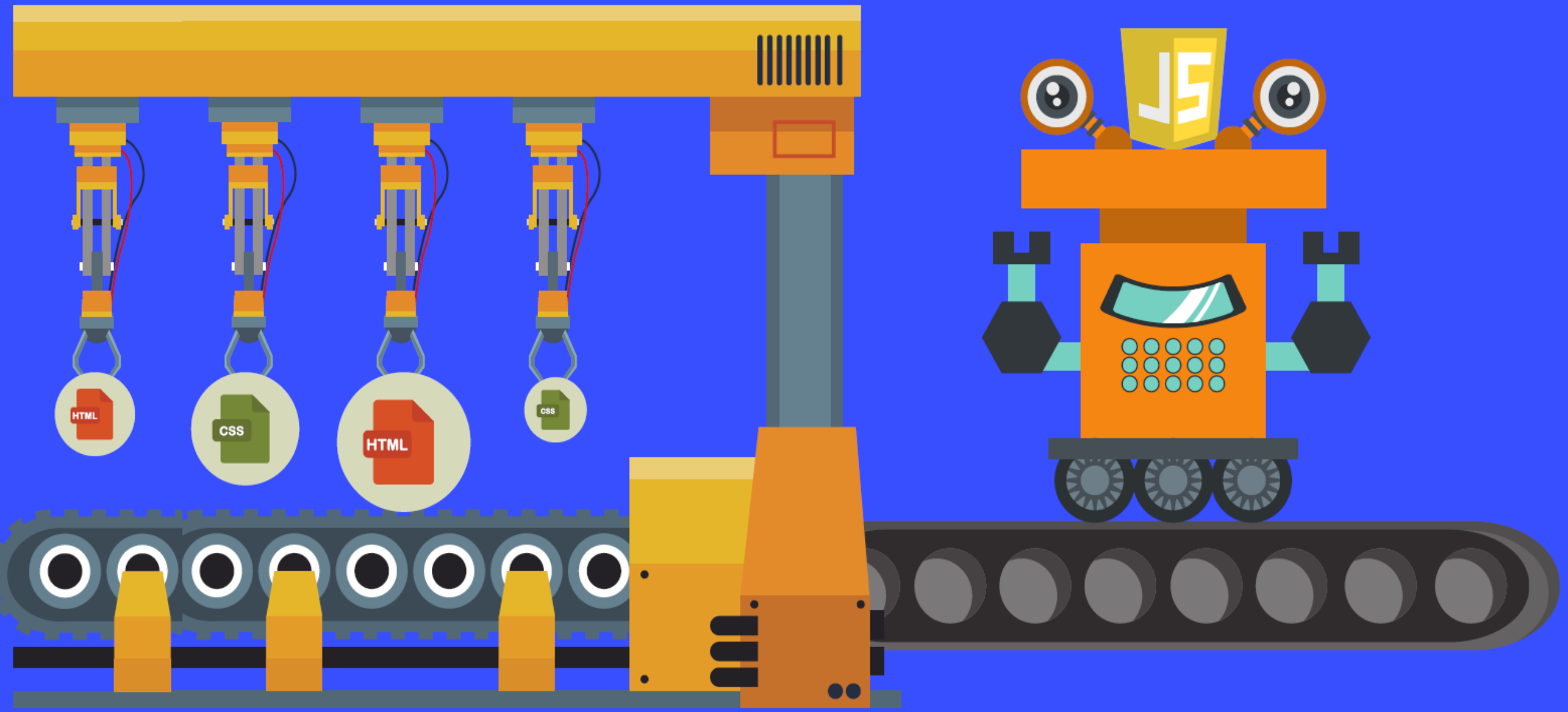
DOM

Document Object Model

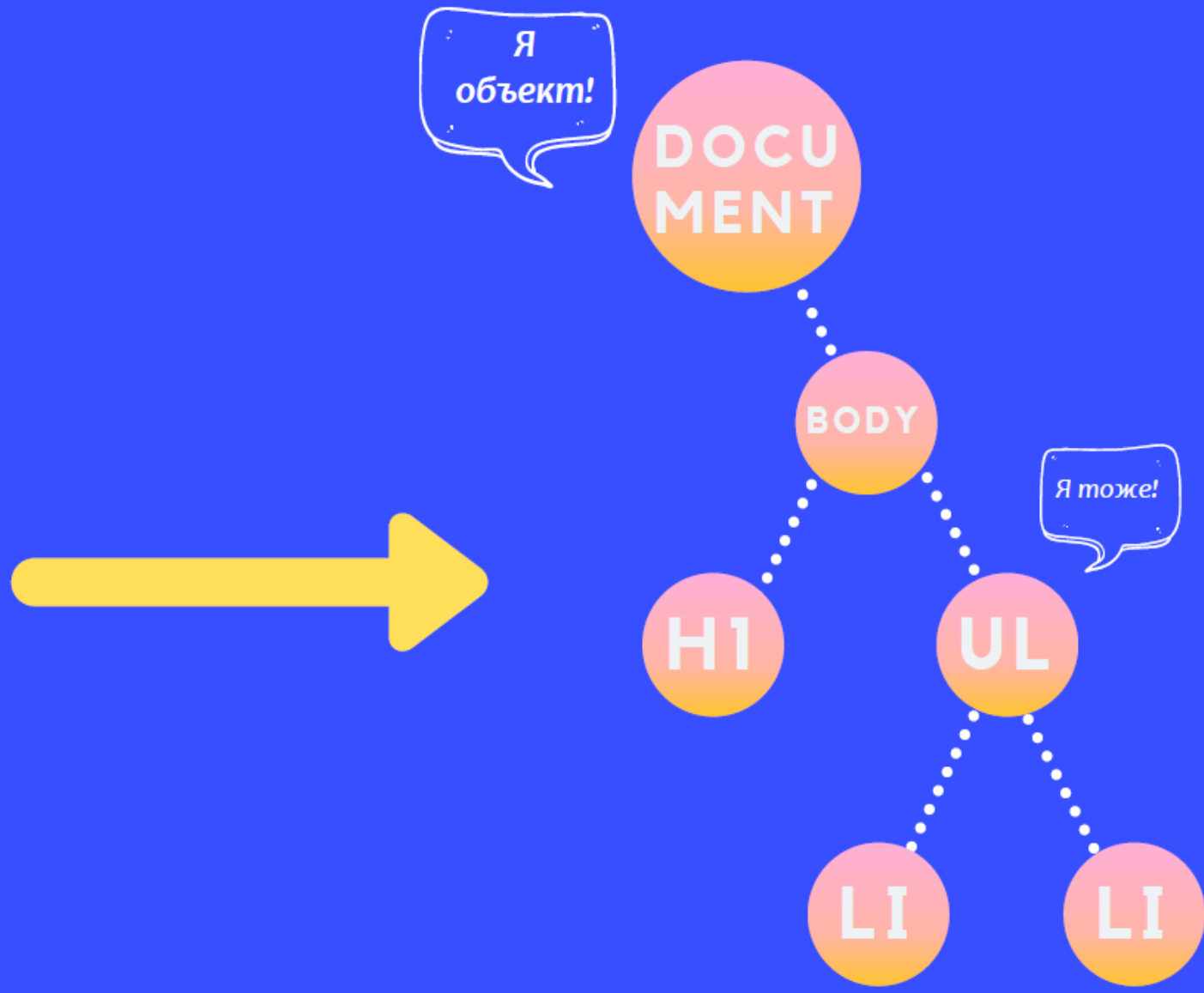
ЧТО ЭТО ТАКОЕ?

- DOM - это JavaScript представление веб-страницы
- Это наше окно к содержимому веб-страницы
- Это просто куча объектов, с которыми можно взаимодействовать через JS.

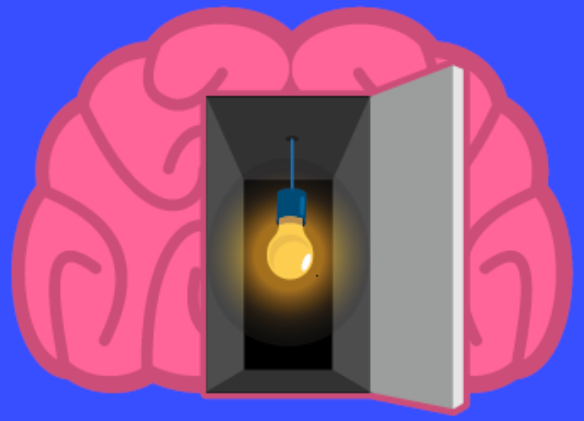
HTML+CSS превращаются в... JS Объекты



```
<body>
  <h1>Hello!</h1>
  <ul>
    <li>Water Plants</li>
    <li>Get Some Sleep</li>
  </ul>
</body>
```

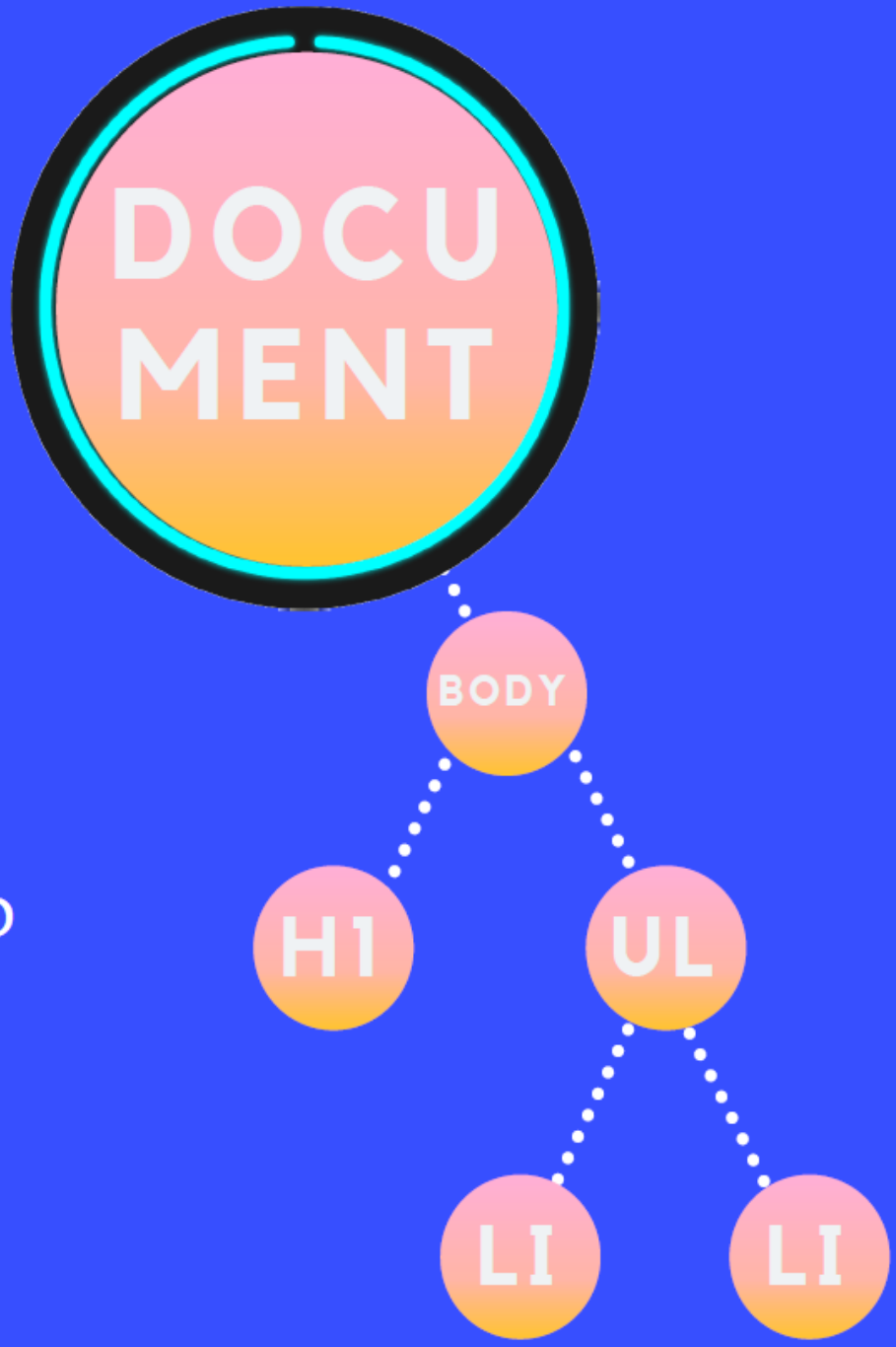


JS Objects



Document

Объект ***document*** — это наша точка входа в мир DOM. Он содержит представления всего содержимого страницы, а также массу полезных методов и свойств.



ВЫБОР

1 ВЫБОР

2 МАНИПУЛИРОВАНИЕ

ВЫБОР

- getElementById
- getElementsByTagName
- getElementsByClassName

querySelector

- Новый комплексный метод выбора одного элемента.
- Принимает селектор CSS

```
//Finds first h1 element:
document.querySelector('h1');

//Finds first element with ID of red:
document.querySelector('#red');

//Finds first element with class of big:
document.querySelector('.big');
```

querySelectorAll

Та же идея, но возвращает **набор** выбранных элементов

Событийные HTML-атрибуты

```html
<button onclick="myFunction()">Click me</button>
<p id="demo"></p>
```

```javascript
function myFunction() {
  const demoEl = getElementById("demo");
  demoEl.innerHTML = "Hello World";
}
```

Установка отложенного запуска

```
function log() {
  console.log("Запуск через 4 сек");
}
setTimeout(log, 4 * 1000);
```

```
function greetUser(fName) {
  console.log(`Привет, ${fName}!`);
}
setTimeout(greetUser, 2 * 1000, "Иван");
```

Периодический запуск

```
function greetUser(fName) {
  console.log(`Привет, ${fName}!`);
}
setInterval(greetUser, 2 * 1000, "Иван");
```

```
let timerEl = document.querySelector("#timer");

let counter = 0;
function timer() {
  timerEl.innerHTML = counter;
  counter++;
}
setInterval(timer, 1000);
```

Спасибо
за внимание!

Ваши вопросы...

Учебный центр «СПЕЦИАЛИСТ» –

Ваш путь к успеху

info@specialist.ru

+7 (495) 232-32-16