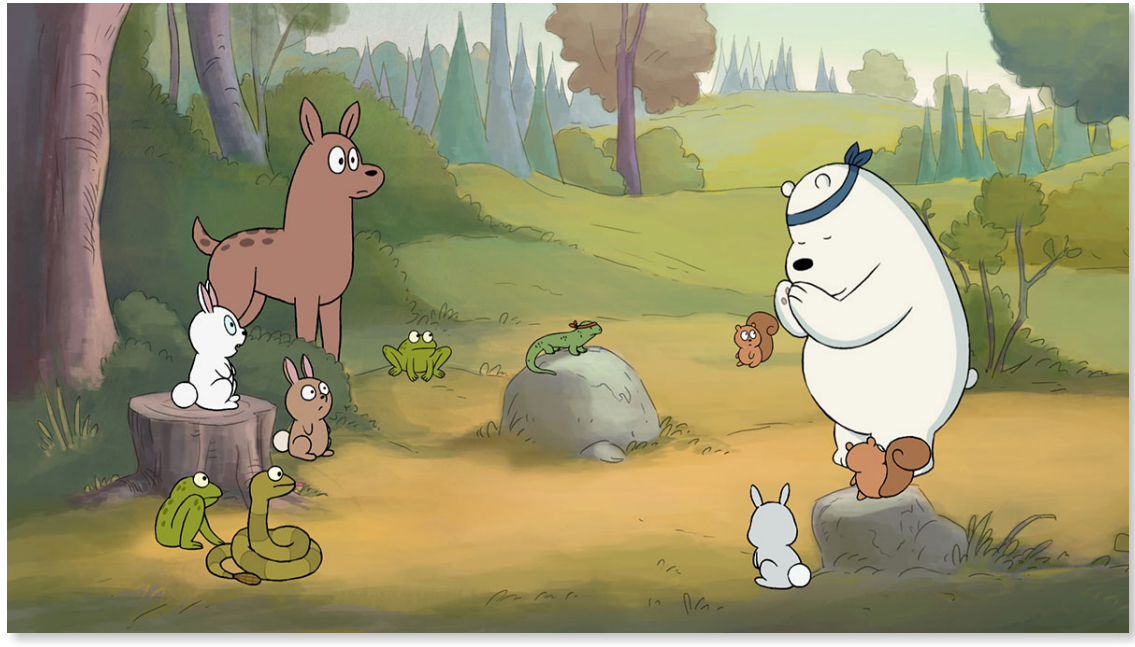
Frontend технологии

Привет



О чем этот курс

— JavaScript Core

— Browser API

— HTTP, Restful API

— React

— Программирование

История JS

— май 1995 - первая версия LiveScript by Brendan Eich

— 1997 - первый стандарт ECMAScript (ES1)

— 1998/1999 - ES2/3 - новые конструкции и типы данных

— 2006 - библиотека JQuery

— 2008 - Google v8 engine

— 2009 - ES5 - strict mode, JSON, новые возможности

— 2009 - среда NodeJS

— 2015 - ES2015/6/Harmony

— 2016 - ES2016

— 2017 - ES2017

JavaScript

— Язык высокого уровня

— Интерпретируемый

— Без системы ввода-вывода (sic!)

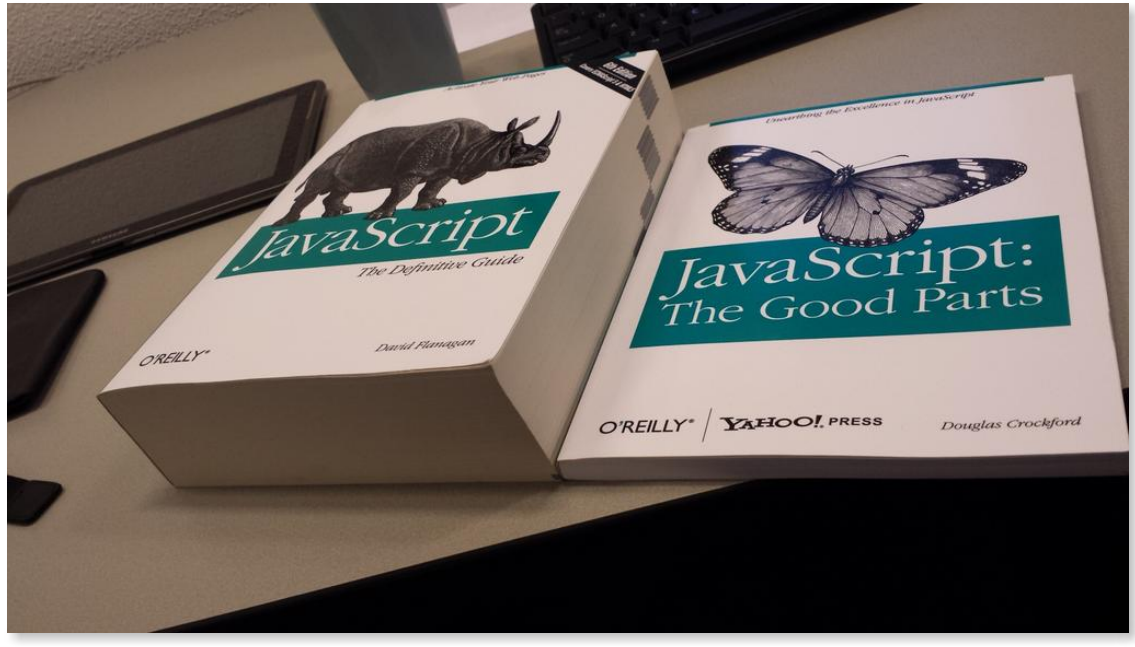
— Динамическая типизация

— Слабая типизация

— Прототипное ООП

— Функциональное программирование (ФП)

Особенности современного JavaScript



Strict Mode

— Особый режим работы интерпретатора

— Исправляет большинство проблем и багов JS

— Включается директивой `"use strict";`

— ES6 вводит поддержку Strict Mode по умолчанию в классах и модулях

```
"use strict";
// весь код файла в строгом режиме
/* ---------------------------------------- */
function f() {
    "use strict";
    // код функции в строгом режиме
}
/* ---------------------------------------- */

import x from "y";
// модуль ES2015, строгий режим по умолчанию

class x {
    // класс ES2015, строгий режим по умолчанию
}
```

Неявное приведение типов

— *приведение типа* - преобразование из одного типа данных в другой

— неявное преобразование типа выполняет интерпретатор JS в операциях, где смешаны данные разных типов

— всегда используйте явное приведение типов в местах, где возможны разночтения

```
5 == "5";   // true
undefined == null;  // true

5 + "1";    // "51"
5 - "1";    // 4

String(5) > "111"; // true
5 > Number("111"); // false

5 > "111";  // ? не знаю, не хочу знать и вам не советую!
```

Лексическая структура

— Кодировка UTF-16

— Комментарии

— Case-sensitive идентификаторы

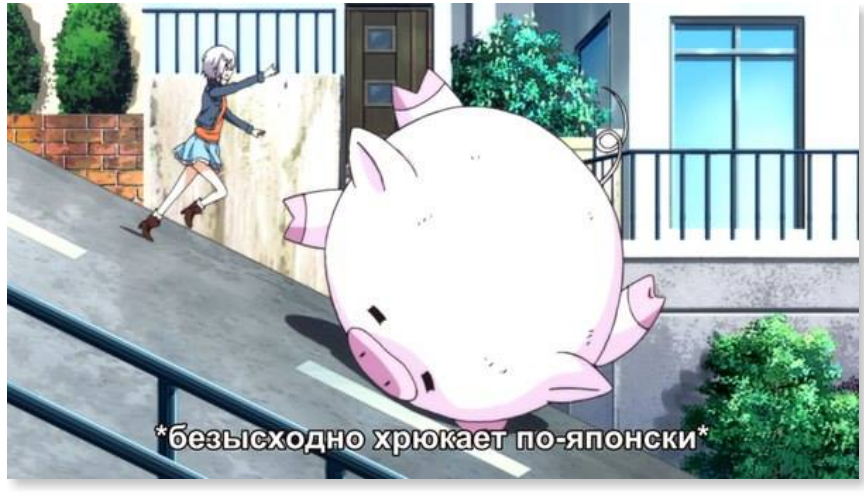
— Ключевые слова

— Точки с запятой

```
/* Это
комментарий */

Abc = 5;
abc = "5";
ABC = { x: 1, y: 2 };
// Abc, abc и ABC - разные переменные
// (кстати, и это комментарий!)

ï = "I'm a variable too!";
こんにちは = "Kon'nichiwa!";
💩 = "poop"; // не работает :(
```



Объявление переменных: `var`, `let`, `const`

— Тип переменной не указывается (динамическая типизация)

— Неинициализированная переменная имеет значение `undefined`

— Область видимости `var` - функция

— Область видимости `let`, `const` - блок `{ ... }`

— Объявление `var` всплывает ("hoisting")

```
var i;
console.log(i);  // undefined

// переменная существует (hoisting), но не инициализирована:
console.log(x);  // undefined
var x = "abc";
console.log(x);  // "abc"

console.log(y);  // ReferenceError, у let нет hoisting'a
let y = 15;
const z;         // SyntaxError, константа должна быть инициализирована
```

`var`, `let`, `const`: лучшие практики

— Используйте `let/const` вместо `var`

— Используйте `const` для переменных, чье значение не планируется менять

Цепочки областей видимости

— Локальные переменные - свойства некоего скрытого объекта

— Каждый фрагмент кода имеет цепочку областей видимости (*scope chain*)

— Scope chain используется при разрешении имени переменной

```
let a = "a0", b = "b0", c = "c0"; // глобальные переменные

function f1() {
    let a = "a1", b = "b1";       // локальные переменные функции

    if (true) {
        let a = "a2";             // локальная переменная блока if
        console.log(a, b, c);     // -> "a2", "b1", "c0"
    }
}
f1();
```

Оператор эквивалентности (a === b)

```
5 === "5";        // false, разные типы
null === null;    // true
false === false; // true
NaN === NaN;      // false (!) нужно использовать Number.isNaN()
16.94 === 16.94; // true, числа с одинаковыми значениями
"abc" === "abc"; // true; строки одинаковой длины с одинаковыми 16-битными значениями

[1, 2, 3] === [1, 2, 3];  // false, ссылки на разные объекты
let a = [1, 2, 3], b = a;
a === b;                  // true, ссылки на один и тот же объект

16.94 !== 16.94; // false, !== это прямая противоположность ===
```

Инструкции ветвления

— `if (cond) statement1 [else statement2]`

— `switch (cond) { statements }`

— `switch` сравнивает значения по правилам `===`

```
if (x < 0)
    return -x;
else
    return x;

switch (animal) {
    case "Cow":
    case "Giraffe":
    case "Dog":
    case "Pig":
        console.log("Will go on Noah's Ark.");
        break;
    case "Dinosaur":
    default:
        console.log("Will not.");
}
```

Условный (тернарный) оператор ? :

— Синтаксис `cond ? expr1 : expr2`;

— Вычисляется одно из выражений в зависимости от истинности условия

— В отличие от `if` тернарный оператор возвращает значение

```
return x > 0 ? x : -x;

let greeting = "hello " + (username ? username : "there");

value = value === undefined ? "default value" : value;
```

Циклы `while`

— `while (cond) statement`

— `do statement while (cond)` - выполнится минимум один раз

— `continue` переходит к следующей итерации цикла

```
x = 1;
while (x > 10) {
    console.log("x > 10!");
}

do {
    console.log("x > 10!"); // выполнится один раз
} while (x > 10);
```

Циклы `for`

— `for ([init]; [cond]; [final-expr]) statement` - old-school for

— `for (variable in obj) statement` - проход по свойствам объекта

```
let arr = [1, 3, 5, 7, 9];
for (let i = 0; i < arr.length; i++) {
    console.log(arr[i]);
}

let obj = {a: 1, b: 2};
for (let prop in obj) {
    console.log(prop, obj[prop]);  // => "a" 1; "b" 2;
}
```

Цикл `for...of`

— `for (variable of iterable) statement`

— Проход по значениям *итерируемого объекта* или *итератора*

— Примеры итерируемых объектов: `Array`, `String`, `Map`, `Set`

— Существуют различные итераторы: `values()`, `keys()`, `entries()`

— Итератор по умолчанию - свойство `Symbol.iterator`

```
let arr = ["a", "b", "c"];  // Массив - итерируемый объект

// Итератор по умолчанию
arr[Symbol.iterator] === arr.values; // true

for (let x of arr)           console.log(x);  // "a", "b", "c"; итератор по умолчанию
for (let x of arr.values())  console.log(x);  // "a", "b", "c"; то же самое
for (let x of arr.keys())    console.log(x);  // 0, 1, 2; проход по ключам
for (let x of arr.entries()) console.log(x);  // [0, "a"], [1, "b"], [2, "c"];
```

Исключительные ситуации

— Код, который может вызвать ошибку, заключается в блок `try`

— При ошибке управление передается в блок `catch`

— Блок `finally` выполняется всегда

— `catch/finally` - опциональны

— 8 встроенных типов ошибок, можно объявлять свои

```
try {
    ...
    throw new Error("whoopsie");
    ...
} catch (e) {
    if (e instanceof Error)
        console.log("it's an Error!");
    console.error(ex.message);
} finally {
    console.log("finally");
}
```

Модули es6

— Имеют собственное пространство имен

— Могут импортировать зависимости из других модулей

— Могут экспортировать свои имена наружу

```
//------ lib.js ------
export const sqrt = Math.sqrt;
export function square(x) { return x * x; }
export function diag(x, y) { return sqrt(square(x) + square(y)); }

//------ main.js ------
import { square, diag } from "lib";
console.log(square(11)); // 121
console.log(diag(4, 3)); // 5

// или
//------ main.js ------
import * as lib from "lib";
console.log(lib.square(11)); // 121
console.log(lib.diag(4, 3)); // 5
```

Модули es6

— Могут экспортировать "значение по умолчанию"

```
//------ myFunc.js ------
export default function () { ... };

//------ main1.js ------
import myFunc from "myFunc";
myFunc();
```

```
//------ underscore.js ------
export default function (obj) {
    ...
};
export function each(obj, iterator, context) {
    ...
}
export { each as forEach };

//------ main.js ------
import _, { each as myEach } from "underscore";

// или даже
import { default as _, each as myEach } from "underscore";
...
```

Глобальный объект

— Разные имена в зависимости от среды (`window`, `global`)

— Поля этого объекта - глобальные переменные

— Содержит:

(*legacy*) глобальные свойства (`NaN`, `undefined`, `Infinity`)

(*legacy*) глобальные функции (`isNaN`, `parseInt`, ...)

стандартные классы (`Number`, `Object`, `String`, ...)

стандартные объекты (`Math`, `JSON`)

объекты среды выполнения (`document`, `console`, `setTimeout`, ...)