Лекция 3
JS. Современные возможности

Мартин Комитски

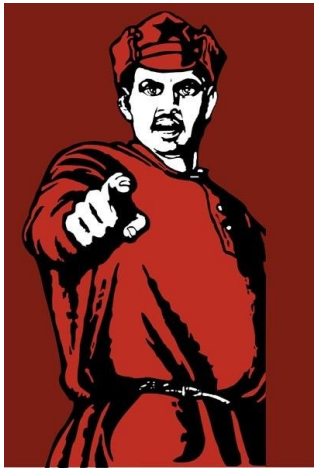
@ mail.ru group

План на сегодня

- Стандарты
- Что есть?
- Что будет?
- Теория
- Еще теория
- А что еще?
- А что еще? [2]
- А что еще? [555]
- Конец

Минутка бюрократии

- Внимание
- Отметки о посещении занятий
- Обратная связь о лекциях



JS. Стандарты

JS. Что есть JavaScript?

- **ECMAScript** — спецификация скриптового языка программирования
- **JavaScript** — язык программирования, одна из реализаций спецификации ECMAScript (наряду с JScript и ActionScript), их ещё называют диалектами ECMAScript
- **ECMA-262** — стандарт компании Ecma International, по которому разрабатывается спецификация ECMAScript
    - последняя версия: 9-е издание в июне 2018 года
    - последний черновик: ECMAScript® 2019 Language Specification
- ISO/IEC 16262:2011(E) — *другой стандарт*, разрабатываемый ISO (в настоящее время активно не развивается)

Подробнее про термины

JS. Как это было

- **Brendan Eich** разработал прототип языка в мае 1995 за 10 дней под кодовым названием Mocha
- В **сентябре 1995** в бета-версии браузера Netscape Navigator 2.0 он был выпущен под названием LiveScript
- В декабре его переименовали в **JavaScript**
- Июнь 1997 — организация **Ecma International** выпустила первую версию стандарта ECMA-262, в котором описывала спецификацию ECMAScript
- Июнь 1998 — спецификация **ECMAScript 2** и международный стандарт **ISO/IEC 16262**

Подробнее про историю развития

JS. Как это было

- Декабрь 1999 — ECMAScript 3
- ECMAScript 4 (abandoned) — заброшенная версия
- Декабрь 2009 — ECMAScript 5
- Июнь 2011 — ECMAScript 5.1 (ISO/IEC 16262:2011)
- Июль 2015 — ECMAScript 2015 (ECMAScript 6th edition) — ES6 Harmony
- Июль 2016 — ECMAScript 2016 (ECMAScript 7th edition)
- Июнь 2017 — ECMAScript 2017 (ECMAScript 8th edition)
- Июнь 2018 — ECMAScript 2018 (ECMAScript 9th edition)
- Лето 2019 — ECMAScript 2019 (и так далее)

JS. Современное состояние JavaScript

- let/const, шаблонные строки, Promise, стрелочные функции
- ES6-модули *и* ES6-классы
- Деструктуризация
- Спреды
- Итераторы *и* генераторы
- Асинхронные функции
- Reflect *и* Proxy
- SharedArrayBuffer *и* Atomics
- WebAssembly
- ...

JS. Что есть на сегодня?

Определение переменных

JS. Всплытие определения переменных

```
1.  console.log(typeof foo);
2.  console.log(typeof bar);
3.
4.  var foo = 'bar';
5.
6.  if (false) {
7.      var bar = ‘sas’;
8.  }
9.
```

JS. Всплытие определения переменных

```
1.  console.log(typeof foo); // undefined
2.  console.log(typeof bar); // undefined
3.
4.  var foo = 'bar';
5.
6.  if (false) {
7.      var bar = ‘sas’;
8.  }
9.
```

JS. Всплытие определения переменных. let / const

```
1.  if (true) {
2.      let bar = 1;
3.  }
4.  console.log(typeof foo);  // ReferenceError
5.  console.log(typeof bar);  // ReferenceError
6.
7.  const foo = 'bar';
8.  foo = 'baz';  // TypeError
9.
```

String

JS. Шаблонные строки

```
const name = 'Muller';

const res = `Hallo, sehr geehrter Herr ${name}!`;
console.log(res);  // Hallo, sehr geehrter Herr Muller!

const multiline = `First line
Second line
Third line`;

multiline.split('\n').length === 3; // true

```

JS. Поддержка Юникода

```
1.  // unicode support
2.  console.log('😀'.length); // 2
3.  console.log('\u{1F600}'); // 😀
4.  console.log('\uD83D\uDE00'); // 😀
5.
6.  String.prototype.charAt(index);        '😀'.charAt(0) === 
7.  String.prototype.charCodeAt(index);    '😀'.charCodeAt(0) === 55357
8.  String.prototype.codePointAt(index);   '😀'.codePointAt(0) === 128512
9.
```

JS. Поддержка Юникода

```
1.  const возраст = 44;
2.  const ФИО = {
3.      имя: 'Антон',
4.      фамилия: 'Павлович Чехов',
5.      возраст
6.  };
7.
8.  function распечатать (пользователь) {
9.      console.log(`${пользователь.имя} ${пользователь.фамилия}`);
10.     console.log(`Возраст ${пользователь.возраст} лет`);
11. }
12.
13. распечатать(ФИО);
```

Для упарывания - [русскuu-loader]

JS. Дополнительные методы строк

```
1.  // ECMAScript 2015 features
2.  String.prototype.includes(searchString, position = 0);
3.  String.prototype.endsWith(searchString, position = length);
4.  String.prototype.startsWith(searchString, position = 0);
5.  String.prototype.repeat(times);
6.
7.  // ECMAScript 2017 features
8.  String.prototype.padStart(maxLength, fillString=' ');
9.  String.prototype.padEnd(maxLength, fillString=' ');
10.
11. // ECMAScript 2019 features
12. String.trimStart();
13. String.trimEnd();
14.
```

https://alligator.io/js/es2019/

Promise

JS. Promise

```
1.  getJSON('/data/books.json')
2.      .then(function (books) {
3.          books.forEach(function (book) {
4.              console.log(book.title);
5.          });
6.      })
7.      .catch(function (error) {
8.          console.error(error);
9.      });
10.     .finally(function () {
11.         console.log('Promise ends');
12.     });
```

JS. Promise

```
1.  // Вернёт промис в состоянии fulfilled («выполнено успешно»)
2.  Promise.resolve( ... );
3.
4.  // Вернёт промис в состоянии rejected («выполнено с ошибкой»)
5.  Promise.reject( ... );
6.
7.  // Вернёт промис, когда выполнятся все промисы
8.  Promise.all( [ ... ] );
9.
10. // Вернёт промис, выполнившийся раньше всех
11. Promise.race( [ ... ] );
12.
```

Arrow Functions

JS. Стрелочные функции

```
1.  const hello = () => console.log('Hello, World!');
2.  const sqr = num => num * num;
3.
4.  [1, 2, 3, 4].map(sqr); // [1, 4, 9, 16]
5.
6.  const compare = (left, right) => {
7.      if (left.length === right.length) {
8.          return left.localeCompare(right);
9.      }
10.     return left.length - right.length;
11. }
12.
```

JS. Стрелочные функции

- Короткий синтаксис
- Не являются "*настоящими*" функциями, не имеют своего `this` и своего `arguments`: берут их из LexicalEnvironment
- Нельзя использовать с оператором `new`

ES6. Class

JS. ES6-классы

```
1.  class User {
2.      constructor(login, password) {
3.          this._login = login;
4.          this._password = password;
5.      }
6.
7.      hello() {
8.          console.log('Hello, ' + this._login);
9.      }
10. }
11.
```

JS. ES6-классы. Статические методы

```
1.  class MathUtils {
2.      static sqr(number) {
3.          return number * number;
4.      }
5.
6.      static abs(number) {
7.          return number < 0 ? -number : number;
8.      }
9.  }
10.
```

JS. ES6-классы. Использование

```
1.  const flash = new User('Barry', 'qwerty123');
2.  const reverseFlash = new User('Eobarth', 'passw0rd');
3.
4.  flash.hello();              // Hello, Barry
5.  reverseFlash.hello();       // Hello, Eobarth
6.
7.  MathUtils.sqr(6);     // 36
8.  MathUtils.abs(-33);   // 33
```

JS. ES6-классы. Геттеры и сеттеры

```
class Shape {
    constructor(width, height) {
        this._width = width;
        this._height = height;
    }
    get Square() { return this._width * this._height; }
    set SideLength(value) {
        this._width = this._height = value;
    }
}

// main.js
const shape = new Shape(6, 12);
console.log(shape.Square);   // 72

shape.SideLength = 7;
console.log(shape.Square);   // 49
```

JS. ES6-классы. Наследование

```
1.  class LoginView extends View {
2.      constructor() {
3.          super(document.getElementById('login'));
4.          this._form = this._el.querySelector('.login__form');
5.      }
6.
7.      hide() {
8.          super.hide();
9.          this._form.clear();
10.     }
11. }
12.
```

ES6. Module

JS. ES6-модули

```
1.  // экспортируем значения
2.  export const PI = 4;
3.  export function square(number) { return number * number; }
4.  export default class User {
5.      constructor() { ... }
6.  }
7.
8.  const name = 'Barry Allen', years = 24;
9.  export { name, years as age };
10.
```

JS. ES6-модули

```
1.  // импортируем значения
2.  import { PI, square } from '../module.js';
3.  import { name as login } from '../module.js';
4.  import UserClass from '../module.js';
5.  import * as Utils from '../module.js';
6.  import '../module.js';
7.
8.  // ре-экспорт
9.  export { PI, login as username } from '../module.js';
10. export * from '../module.js';
11.
```

Работа с объектами

JS. Объявление литералов

```
1.  // es6 features
2.  const login = 'Barry Allen', age = 24;
3.  const User = {login, age}; // {login: 'Barry Allen', age: 24}
4.
5.  const Obj = {
6.      func: function () { ... },          // old way
7.      func() { ... },                     // inline methods
8.      get Arr() { return this.array; }, // object getters & setters
9.      ['foo' + 'bar']: value,             // computed property names
10.     array: [1, 2, 3, 4, 5,],            // trailing commas
11.     prop: value,                        // trailing commas
12. }
13.
```

JS. Объявление функций

```
1.  // es8 features
2.  function UseFull (
3.      param1 ,
4.      param2 ,
5.      param3 ,   // trailing commas
6.  ) { return param1 + param2 + param3; }
7.
8.  UseFull (
9.      42 ,
10.     100500 ,
11.     -200600 ,   // trailing commas
12. ); // -100058
13.
14.
```

JS. Дополнительные методы

```
// проверка двух выражений на совпадение
Object.is(value1, value2);
Object.is(1, 1);                // true
Object.is(1, '1');              // false
Object.is(false, false);        // true
Object.is({a: 42}, {a: 42});  // false

Object.is(NaN, NaN);    // true   (NaN === NaN) === false
Object.is(0, -0);       // false  (-0 === 0) === true
// копирование свойств
Object.assign(target, source, source, source, ...);

const s1 = {a: 'Barry'}, s2 = {b: 24};
const result = Object.assign({}, s1, s2);
// result: {
//     a: 'Barry',
//     b: 24
// }

```

JS. Дополнительные методы

```
1.  // Запаковывание объектов
2.  Object.seal(target);    // можно изменить значение имеющихся свойств,
3.                          // но нельзя добавить или удалить их
4.  // Заморозка объектов
5.  Object.freeze(target);  // нельзя изменять значения имеющихся свойств,
6.                          // удалять их, добавлять новые
7.
8.  Object.isFrozen(target);   Object.isSealed(target);

9.  // Перебор ключей, значений и свойств
10. const user = {login: 'Oliver Queen', age: 42};
11.
12. Object.keys(user);     // ['login', 'age']
13. Object.values(user);   // ['Oliver Queen', 35]
14. Object.entries(user);  // [['login', 'Oliver Queen'], ['age', 35]]
```

Новые коллекции

JS. Map — хэш-таблица

```
const map = new Map();
map.set(key, value);   // добавить значение
map.get(key);          // получить значение
map.has(key);          // проверить наличие ключа
map.delete(key);
map.clear();

map.size;              // размер Map
map.forEach(callback); // перебор ключей, свойств, значений
map.values();
map.keys();
map.entries();

```

JS. Set — набор значений без повторения

```
const set = new Set();
set.add(value);           // добавить значение
set.has(value);           // проверить наличие значения
set.delete(value);   set.clear();

set.size;               // размер Set
set.forEach(callback); // перебор ключей, свойств, значений
set.values();
set.keys();
set.entries();
```

Reflect

JS. Reflect

Встроенный JavaScript объект, предоставляющий методы для перехвата взаимодействий с объектами и работы с рефлексией в JavaScript

```
1.  Reflect.apply(target, thisArgument, argumentsList)
2.  Reflect.construct(target, argumentsList)
3.
4.  Reflect.get(target, propertyKey)
5.  Reflect.has()
6.
7.  Reflect.getPrototypeOf(target)
8.  Reflect.setPrototypeOf(target, prototype)
9.
10.
```

JS. Работа с дескрипторами

```
1.  // Reflect.defineProperty(target, propertyKey, attributes)
2.  const object = {};
3.  Reflect.defineProperty(object, 'foo', {
4.      enumerable: false,    // разрешает перечисление
5.      writable: false,      // разрешает перезапись
6.      configurable: false,  // разрешает изменение дескриптора
7.      value: undefined      // значение свойства
8.      get: undefined        // геттер
9.      set: undefined        // сеттер
10. }
11.
```

Дополнительные нововведения

JS. Оператор возведения в степень

```
1.  console.log(3 ** 4);     // 81
2.  console.log(49 ** 0.5);  // 7
3.
```

JS. Новые возможности регулярок

```
1.  // новый флаг s (dotAll)
2.  /foo.bar/.test('foo\nbar');      // false
3.  /foo.bar/s.test('foo\nbar');     // true
4.
5.  // именованные группы в RegExp
6.  let re = /(?<year>\d{4})-(?<month>\d{2})-(?<day>\d{2})/u;
7.  let result = re.exec('2018-04-06');
8.  // result.groups.year === '2018';
9.  // result.groups.month === '04';
10. // result.groups.day === '06';
11.
12.
```

SharedArrayBuffer и Atomics

JS. Про Web Workers

```
1.  // main
2.  const worker = new Worker(scriptUrl);            // создали воркер
3.  // const sharedWorker = new SharedWorker(scriptUrl);
4.
5.  worker.postMessage({hello: 'world'});            // отправили данные
6.  worker.onmessage = function(e) { e.data ... };  // обработчик onmessage
7.
8.  // worker
9.  self.onmessage = function(e) { e.data... };    // обработчик onmessage
10. self.postMessage({hello: 'world'});             // отправили данные

11.
```

JS. SharedArrayBuffer и Atomics

```
1.  const buffer = new ArrayBuffer(15 * 1024 * 1024);  // 15 MB RAM
2.  worker.postMessage(buffer);                        // клонирует данные
3.
4.  const shared = new SharedArrayBuffer(length);      // разделяемая
    память
5.  worker.postMessage(shared);                        // клонирует данные
6.
7.  Atomics.add(typedArray, pos, val);     // потокобезопасное сложение
8.  Atomics.sub(typedArray, pos, val);     // потокобезопасное вычитание
9.  Atomics.store(typedArray, pos, val);   // потокобезопасная запись
10. Atomics.load(typedArray, pos);         // потокобезопасное чтение
11. Atomics.wait(typedArray, pos, val[, timeout]);  // like as Linux
    futexes
12. ...

13.
```

Деструктуризация

JS. Деструктуризация массивов

```
1.  // деструктуризация
2.  const [name, family] = 'Barry Allen'.split(' ');
3.  console.log(name);    // Barry
4.  console.log(family);  // Allen
5.
6.  // пропуск элементов
7.  const [, , var3, var4] = [1, 2, 3, 4];
8.  const [num1, , num3] = [1, 2, 3, 4];
9.
10. // значения по умолчанию
11. const [name, family = 'Black'] = ['Barry'];
12. console.log(name);    // Barry
13. console.log(family);  // Black
14.
15. // swap переменных
16. let title1 = 'Book 1', title2 = 'Book 2';
17. ([title1, title2] = [title2, title1]);   // title1 === 'Book 2'
18.                                          // title2 === 'Book 1'
19.
```

JS. Деструктуризация объектов

```
const person = {login: 'Barry Allen', age: 42, alive: true};

// деструктуризация
const {login, age} = person;

// значения по умолчанию
const {login, age, password = 'qwerty123'} = person;

// переименование свойств
const {login: username, age} = person;    // username === 'Barry Allen'

// комбинация
const {login: username = 'Anonymous', age} = person;

// вложенная деструктуризация
const element = {tagName: 'DIV', size: {width: 300, height: 200}};
const {
    tagName: tag,
    size: {width: w = 250, height: h = 250} = {},
    color = 'red',
    childs: [first, second, , last] = []
} = element;
console.log({tag, w, h, color, first, second, last});
```

JS. Деструктуризация параметров функций

```
1.  function square({width: w, height: h = 100}) {
2.      return w * h;
3.  }
4.
5.  square({width: 20, height: 50, color: 'red'});    // 1000
6.  square({width: 42});                              // 4200
7.
8.  // obj prop destruct
9.  const value = 5;
10. const value2;
11. const obj = {
12.     value,
13.     value2
14. };
15.
```

Spread/rest оператор

JS. Spread/rest оператор

```
1.  // размазывание массивов
2.  const arr = ['a', 'b', 'c', 'd'];
3.  const arr2 = [1, 2, ...arr, 3]; // spread
4.  console.dir(arr2); // [1, 2, 'a', 'b', 'c', 'd', 3];
5.
6.  // используется при деструктуризации
7.  const scoreboard = ['Barry', 'Cisco', 'Caitlin', 'Harrison'];
8.  const [first, second, ...rest] = scoreboard; // rest
9.
10. // first === 'Barry'
11. // second === 'Cisco'
12. // rest === ['Caitlin', 'Harrison']
13.
```

JS. Spread/rest оператор

```
1.  // передача параметров в функцию
2.  const numbers = [1, 2, 42, 532, -3.14, -Infinity];
3.  const maximum = ... ?
4.  const minimum = ... ?
5.
6.  const maximum = Math.max.apply(null, numbers);   // 532
7.  const minimum = Math.min.apply(null, numbers);   // -Infinity
8.
9.  // передача параметров в функцию
10. const numbers = [1, 2, 42, 532, -3.14, -Infinity];
11. const maximum = Math.max(...numbers);   // 532
12. const minimum = Math.min(...numbers);   // -Infinity
13.
```

JS. Rest-параметры функции

```
1.  // сбор праметров функции
2.  function summ(...nums) {
3.      // можно работать не с arguments, а с настоящим массивом nums
4.      return nums.reduce((sum, current) => sum + current, 0);
5.  }
6.
7.  console.log(summ(1, 2, 3, 4, 5));    // 15
8.
```

JS. Spread для объектов

```
const name = { first: 'Barry', last: 'Allen' };
const address = {
    city: 'Central',
    country: 'USA',
    street: '11',
};

const profile = {
    age: 20,
    ...name,
    ...address,
};

```

Symbol

JS. Типы данных в JavaScript

1. Number
2. Boolean
3. String
4. Object
5. null
6. undefined
7. Symbol

JS. Создание символов

```
1.  // без new
2.  const symbol1 = Symbol();
3.  const symbol2 = Symbol('label');
4.  const symbol3 = Symbol('label');
5.
6.  console.log(typeof symbol1);        // symbol
7.  console.log(symbol2 == symbol3);    // false
8.  console.log(symbol2 === symbol3);   // false
9.
10. console.log(symbol1);               // 'Symbol()'
11. console.log(symbol2);               // 'Symbol(label)'
12.
```

JS. Глобальные символы

```
1.  // берутся из реестра глобальных символов
2.  // если символа нет в реестре - создаётся новый символ
3.  const symbol1 = Symbol.for('label');
4.  const symbol2 = Symbol.for('label');
5.  console.dir(symbol1 == symbol2);        // true
6.
7.  const symbol3 = Symbol('label');
8.  console.dir(Symbol.keyFor(symbol1));    // 'label'
9.  console.dir(Symbol.keyFor(symbol3));    // undefined
10.
```

JS. Символы в качестве имён новых свойств

```
1.  const User = {
2.      name: 'Barry Allen',
3.      [Symbol.for('hello')]() {
4.          console.log(`Hello, ${this.name}!`);
5.      }
6.  }
7.
8.  User[Symbol.for('hello')]();   // 'Hello, Barry Allen!'
9.
10. const helloSymbol = Symbol.for('hello');
11. User[helloSymbol]();           // 'Hello, Barry Allen!'
12.
```

JS. Системные символы

1. Symbol.hasInstance
2. Symbol.iterator
3. Symbol.replace
4. Symbol.search
5. Symbol.toPrimitive
6. Symbol.toStringTag
7. Symbol.match
8. ...
9.

Итераторы

JS. Итераторы

Итераторы *— расширяющая понятие «массив» концепция. Итерируемые или, иными словами, «перебираемые» объекты — это те, содержимое которых можно перебрать в цикле.*

JS. Итерируемые объекты

- Массивы
- Псевдомассив arguments
- Строки
- Коллекции DOM-нод в браузере
- Генераторы
- Map, Set...
- Пользовательские итерируемые объекты

JS. Итераторы

В общем смысле, итератор — это объект, предоставляющий метод next(), который возвращает следующий элемент определённой последовательности. Для перебора итераторов существует специальный цикл for ... of

JS. Перебор итераторов

```
1.  const numbers = [2, 3, 5, 7, 11, 13];
2.
3.  for (const prime of numbers) {
4.      console.log(`Prime number ${prime}!`);
5.  }
6.
```

JS. Связь со спредами

```
1.  // оператор расширения итерируется по итератору
2.  // и возвращает массив из элементов итератора
3.
4.  function arrayUniq()
5.      const source = [...arguments];
6.      // Set.prototype.values() возвращает итератор по
    элементам коллекции
7.      return [...new Set(source).values()];
8.  }
9.
```

JS. Symbol.iterator

```
1.  const iterable = [1, 2, 3, 4];
2.  const iterator = iterable[Symbol.iterator]();
3.
4.  console.log(iterator.next());    { value: 1, done: false }
5.  console.log(iterator.next());    { value: 2, done: false }
6.  console.log(iterator.next());    { value: 3, done: false }
7.  console.log(iterator.next());    { value: 4, done: false }
8.  console.log(iterator.next());    { value: undefined, done: true }
9.  console.log(iterator.next());    { value: undefined, done: true }
10.
```

JS. Кастомные итераторы

```
1.  const iterable = {
2.      current: 0,
3.      [Symbol.iterator]() { return this; },
4.      next() {
5.          if (this.current) {
6.              return { value: this.current--, done: false };
7.          }
8.          return { value: undefined, done: true }
9.      }
10. };
11.
```

JS. Кастомные итераторы

```
1.  iterable.current = 7;
2.  const elements = [...iterable]; // [ 7, 6, 5, 4, 3, 2, 1 ]
3.
4.  iterable.current = 5;
5.  let summ = 0;
6.  for (const n of iterable) {
7.      summ += n;
8.  }
9.  console.log(summ);    // 15
```

Генераторы

JS. Объявление функции-генератора

Генераторы *– новый вид функций в современном JavaScript. Они отличаются от обычных тем, что могут приостанавливать своё выполнение, возвращать промежуточный результат и далее возобновлять его позже, в произвольный момент времени.*

JS. Объявление функции-генератора

```
1.  // можно так
2.  function * generator() {
3.      yield 1;
4.      yield 2;
5.      return 3;
6.  }
7.
8.  // можно и так
9.  const generator = function * () {
10.     yield 1;
11.     yield 2;
12.     return 3;
13. };
```

JS. Объявление функции-генератора

```
1.  const gen = generator();
2.
3.  console.log(gen.next());   // { value: 1, done: false }
4.  console.log(gen.next());   // { value: 2, done: false }
5.  console.log(gen.next());   // { value: 3, done: true }
6.  console.log(gen.next());   // { value: undefined, done:
    true }
7.  console.log(gen.next());   // { value: undefined, done:
    true }
8.
9.  const gen2 = generator();  // console.log([...gen2])
```

JS. "Бесконечные" генераторы

```
1.  function * fibonacci() {
2.      let prev = 1, curr = 0;
3.      while (true) {
4.          let now = prev + curr;
5.          prev = curr; curr = now;
6.          yield now;
7.      }
8.  }
```

JS. Взаимодействие с генераторами

```
1.  function * rand(length) {
2.      while (length--) {
3.          yield Math.random();
4.      }
5.  }
6.
7.  console.log([...rand(3)]);   // [ 0.216, 0.39, 0.555 ]
8.  console.log([...rand(5)]);   // [ 0.782, 0.806, 0.294,
    0.228, 0.755 ]
9.
```

JS. Взаимодействие с генераторами

```
1.  function * simple() {
2.      let num = yield 'line 2';
3.      return num;
4.  }
5.  const gen = simple();
6.  console.log(gen.next());     // { value: 'line 2',
    done: false }
7.  console.log(gen.next(42));   // { value: 42, done: true
    }
8.  console.log(gen.next());     // { value: undefined,
    done: true }
9.
```

JS. Взаимодействие с генераторами

```
1.  function * wow() {
2.      let num = 0, sum = 0;
3.      while (num = yield sum) {
4.          sum += num;
5.      }
6.      return sum;
7.  }
8.
```

JS. Взаимодействие с генераторами

```
1.  const gen = wow();
2.  gen.next();         // { value: 0, done: false }
3.  gen.next(1);        // { value: 1, done: false }
4.  gen.next(2);        // { value: 3, done: false }
5.  gen.next(3);        // { value: 6, done: false }
6.  gen.next(4);        // { value: 10, done: false }
7.  gen.next(5);        // { value: 15, done: false }
8.
9.  gen.next(0);        // { value: 15, done: true }
10.
```

JS. Взаимодействие с генераторами

```
1.  const gen = wow();
2.  gen.next();       // { value: 0, done: false }
3.  gen.next(1);      // { value: 1, done: false }
4.  gen.throw(new Error('kek'));  // Error: kek
5.
6.  gen.next(0);      // до этого места выполнение не дойдёт
7.
```

JS. Композиция генераторов

```
1.  function * twicer(element) {
2.      yield element; yield element;
3.  }
4.  function * test() {
5.      yield * twicer(42);
6.      yield * twicer('test');
7.  }
8.
9.  console.log([...test()]);    // [ 42, 42, 'test',
    'test' ]
10.
```

Асинхронные функции (async/await)

JS. async/await

Ключевое слово `async` *позволяет объявлять асинхронные функции, которые возвращают промис. Внутри таких функций возможна "синхронная" работа с промисами с помощью ключевого слова* `await`.

JS. Объявление функций

```
1.  async function good() {
2.      return 42;
3.  }
4.
5.  good()
6.      .then(res => console.log('Good: ', res);
7.
```

JS. Объявление функций

```
1.  async function bad() {
2.      throw new Error('kek');
3.  }
4.
5.  good()
6.      .then(res => console.log('Good: ', res));
7.      .catch(err => console.error(err));
8.
```

JS. Объявление функций

```
async function luck(num) {
    if (Math.random() < 0.5) {
        return num * 2;
    }
    throw new Error('kek');
}

luck(21)
    .then(res => console.log('Good: ', res));    // may be 42
    .catch(err => console.error(err));           // or may be an Error

```

JS. Использование

```
1.  async function loadJSON(url) {
2.      const response = await fetch(url, {method: 'GET'});
3.      if (response.statusCode !== 200) {
4.          throw new Error(`Can not load json ${url}`);
5.      }
6.      const json = await response.json();
7.      return json;
8.  }
9.
```

JS. Использование

```
1.  async function load(query) {
2.      const list = await fetchList(query);
3.      const result = await Promise.all(list.map(item => loadItem(item)));
4.      return result;
5.  }
6.
7.  async function load(query) {
8.      const list = await fetchList(query);
9.      return Promise.all(list.map(item => loadItem(item)));
10. }
11.
```

JS. Асинхронные итераторы

```
 1.  function readFiles(names) {
 2.      return names.map(
 3.          filename => promiseRead(filename)
 4.      )
 5.  }
 6.
 7.  for (const pRead of readFiles([...])) {
 8.      const source = await pRead;
 9.      // logic ...
10.  }
11.
```

JS. Новый цикл for-await-of

```
1.  for await (const source of readFiles([...])) {
2.      console.log(source)
3.      // logic ...
4.  }
5.
```

JS. Асинхронные генераторы

Незаменимо, когда заранее не известно количество итерируемых элементов

```
async function* readLines(path) {
    let file = await fileOpen(path);
    try {
        while (!file.EOF) {
            yield await file.readLine();
        }
    } finally {
        await file.close();
    }
}

```

JS. Асинхронные генераторы

```
1.  for await (const line of readLines(filePath)) {
2.      console.log(line)
3.      // logic ...
4.  }
5.
```

Proxy

JS. Proxy

Объект Proxy (Прокси) — особый объект, смысл которого — перехватывать обращения к другому объекту и, при необходимости, модифицировать их.

```
1.  // создание Proxy
2.  const proxy = new Proxy(target, handler);
3.
4.  // target - объект, обращения к которому надо перехватывать
5.  // handler - объект с функциями-перехватчиками для операций к target
6.
```

JS. Создание Proxy

```
const user = {};
const proxy = new Proxy(user, {
    get (target, property, receiver) {
        console.log(`Чтение ${property}`);
        return target[property];
    },
    set (target, property, value, receiver) {
        console.log(`Запись ${property} = ${value}`);
        target[property] = value;
        return true;
    },
});

```

JS. Использование Proxy

```
1.  proxy.name = 'Barry Allen';                 // Запись name = Barry Allen
2.  proxy.age = 22;                             // Запись age = 22
3.  proxy['long property'] = 'qux';             // Запись long property = qux
4.
5.  const name = proxy.name;                    // Чтение name
6.  const age = proxy.age;                      // Чтение age
7.  const long = proxy['long property'];        // Чтение long property
8.
9.
```

JS. Конфигурация Proxy

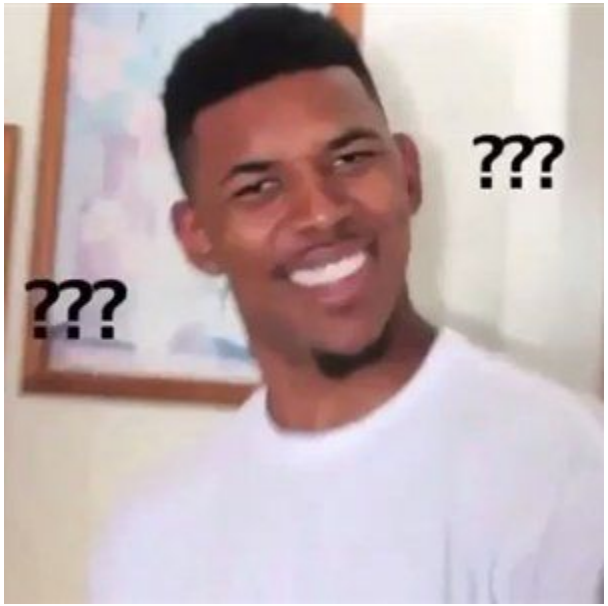
```
1.  const handler = {
2.      get (target, name, receiver);          // получение свойств
3.      set (target, name, val, receiver);     // установка свойства
4.      apply (target, thisValue, args);       // вызовы функции
5.      construct (target, args);              // вызовы конструктора с new
6.      has (target, name);                    // оператор in
7.      defineProperty (target, property, descriptor);
8.                                             // метод
    Object.defineProperty()
9.      deleteProperty (target, property);     // оператор delete
10.     ...
11. };
12.
```

JS. Применение Proxy

```
1.  const original = {};
2.  const magic = wrapWithProxy(original);
3.
4.  magic.data.elements[0].attributes.color = 'black';
5.  magic.country.map.shops = [ ... ];
6.
```

JS?

Вопросы?



Конец

Конец?

Перерыв! (10 минут)

Препод (с)

WebAssembly

JS. WebAssembly

Во время создания WebAssembly решалась следующая задача: **быстро исполнять код в браузере**

WebAssembly (wasm) — эффективный низкоуровневый байт-код, предназначенный для исполнения в браузере. WebAssembly представляет собой переносимое абстрактное синтаксическое дерево, обеспечивающее как более быстрый парсинг, так и более быстрое выполнение кода, чем JavaScript — developers.google.com

JS. WebAssembly

WebAssembly — это не полная замена JS, а лишь технология, позволяющая писать критичные к ресурсам модули и компилировать их в переносимый байт-код с линейной моделью памяти и статической типизацией

Применения: редактирование изображений/видео/музыки, криптография, математические вычисления, игры...

JS. Что же такое WebAssembly?

- Бинарный формат
- НЕ язык программирования, а байт-код
- **Загружается в браузер и исполняется в браузере** — формально, WebAssembly исполняется JavaScript-движком, а не самим браузером, поэтому есть и другие варианты исполнения, например, под Node.js
- Исполняется виртуальной машиной
- НЕ имеет ничего общего с WEB, кроме того что общается с внешним миром через JavaScript

JS. WebAssembly

```
1.  // исходник на C
2.  int fib(int n) {
3.      if (n == 0) { return 0; } else {
4.          if ((n == -1) || (n == 1)) { return 1; } else  {
5.              if (n > 0) { return fib(n - 1) + fib(n - 2); }
6.              else { return fib(n + 2) - fib(n + 1); }
7.          }
8.      }
9.  }
10.
```

JS. WebAssembly

```
1.  // текстовое представление WAST
2.  (module
3.      (table 0 anyfunc)
4.      (memory $0 1)
5.      (data (i32.const 12) "\01\00\00\00\00\00\00\01\00\00\00")
6.      (export "memory" (memory $0))
7.      (export "fib" (func $fib))
8.      (func $fib (param $0 i32) (result i32)
9.          (local $1 i32)
10.         (block $label$0
11.             (br_if $label$0
12.                 (i32.ge_u
13.                 // ...
```

JS. WebAssembly

```
// скомпилированный байт-код wasm
const wasmCode = new Uint8Array(
[0,97,115,109,1,0,0,0,1,134,128,128,128,0,1,96,1,127,1,127,3,130,128,
128,128,0,1,0,4,132,128,128,128,0,1,112,0,0,5,131,128,128,128,0,1,0,
1,6,129,128,128,128,0,0,7,144,128,128,128,0,2,6,109,101,109,111,114,
121,2,0,3,102,105,98,0,0,10,203,128,128,128,0,1,197,128,128,128,0,1,
1,127,2,64,32,0,65,1,106,34,1,65,3,79,13,0,32,1,65,2,116,65,12,106,
40,2,0,15,11,2,64,32,0,65,1,72,13,0,32,0,65,127,106,16,0,32,0,65,
126,106,16,0,106,15,11,32,0,65,2,106,16,0,32,1,16,0,107,11,11,146,
128,128,128,0,1,0,65,12,11,12,1,0,0,0,0,0,0,0,1,0,0,0]
);

```

JS. WebAssembly

```
1.  // запускаем wasm-модуль
2.  const wasmCode = new Uint8Array([...]);
3.  const wasmModule = new WebAssembly.Module(wasmCode);
4.  const wasmInstance = new WebAssembly.Instance(wasmModule, []);
5.
6.  console.log(wasmInstance.exports.fib(10));
7.
```

Демо: — Танки, Quake.
Golang на js
Frontend на ts

Поддержка совместимости

JS. Поддержка совместимости

1. Поддержка версий ECMAScript (пример)
2. Возможности браузера
3. Все плохо. Что делать?

JS. Поддержка совместимости. Решение

Полифилл *— это библиотека, которая добавляет в старые браузеры поддержку возможностей, которые в современных браузерах являются встроенными.*

```
1.  if (!Object.is) {
2.      Object.is = function(x, y) {
3.          if (x === y) { return x !== 0 || 1 / x === 1 / y; }
4.          else { return x !== x && y !== y; }
5.      }
6.  }
7.
```

JS. Поддержка совместимости. Решение

Транспайлинг *— это конвертация кода программы, написанной на одном языке программирования в другой язык программирования*

```
1.  // before
2.  const f = num => `${num} в квадрате это ${num ** 2}`;
3.
4.  // after
5.  var f = function (num) {
6.      return num + ' в квадрате это ' + Math.pow(num, 2);
7.  };
8.
```

JS. Поддержка совместимости. Решение

Babel *— многофункциональный транспайлер, позволяет транспиллировать ES5, ES6, ES2016, ES2017, ES2018, ES2019, ES.Next, JSX и Flow*

Babel REPL — бабель-онлайн

- Парсит исходный код и строит AST
- Последовательно вызывает набор функций, которые каким-то образом трансформируют AST программы
- В процессе трансформации части AST, относящиеся к современному синтаксису, заменяются на эквивалентные, но более общеупотребительные фрагменты
- Преобразует модифицированное AST в новый транспилированный код

JS. Текущие версии JavaScript

- Декабрь 1999 — *ECMAScript 3*
- *ECMAScript 4 (abandoned)* — заброшенная версия
- Декабрь 2009 — *ECMAScript 5*
    - Июнь 2011 — *ECMAScript 5.1* (ISO/IEC 16262:2011)
- Июль 2015 — *ECMAScript 2015* (ECMAScript 6th edition) — ES6 Harmony
- Июль 2016 — *ECMAScript 2016* (ECMAScript 7th edition)
- Июнь 2017 — *ECMAScript 2017* (ECMAScript 8th edition)
- Июнь 2018 — ECMAScript 2018 (ECMAScript 9th edition)
- *Лето 2019* — ECMAScript 2019 (ECMAScript 10th edition)
- ES.Next (Будущие реализации)

JS. ES.Next

ES.Next *— так временно называют совокупность новых возможностей языка, которые могут войти в следующую версию спецификации. Фичи из ES.Next правильнее называть “предложения” (proposals) , потому что они всё ещё находятся на стадии обсуждения*

Процесс ТС39

JS. Процесс TC39

TC39 (технический комитет 39) *— занимается развитием* ***JavaScript****. Его членами являются компании (помимо прочих, все основные производители браузеров). TC39 регулярно собирается, на встречах присутствуют участники, представляющие интересы компаний, и приглашенные эксперты.*

Процесс TC39 *— алгоритм внесения изменений в спецификацию ECMAScript. Каждое предложение по добавлению новой возможности в ECMAScript в процессе созревания проходит ряд этапов*

- 0 этап: идея (strawman)
- 1 этап: предложение (proposal)
- 2 этап: черновик (draft)
- 3 этап: кандидат (candidate)
- 4 этап: финал (finished)

JS. Будущее JavaScript

- Репозиторий со списком текущих предложений
- Предложения, перешедшие в stage-4
- Наиболее интересные *proposals:*
    - **Optional catch binding (stage-4)** — link
    - BigInt — link
    - Class and Property Decorators — link
    - Nullish Coalescing — link
    - Static public fields — link

Диалекты JavaScript

JS. Транспилляция из ES.Next

```
1.  import { flying } from 'abilities';
2.  class Creature {
3.      constructor({ name, ...rest}) {
4.          console.log(`Привет, ${name}, твои свойства:`, rest);
5.      }
6.  }
7.
8.  @flying
9.  class Dragon extends Creature {
10.     static haveTail = true;
11.     legs = 4;
12.     async *eat(...staff) {
13.         // Eat something...
14.     }
15. }
16.
```

JS. Dart

```
 1.  import 'dart:async';
 2.  import 'dart:math' show Random;
 3.
 4.  Stream<double> computePi({int batch: 1000000}) async* { ... }
 5.
 6.  main() async {
 7.      print('Compute π using the Monte Carlo method.');
 8.      await for (var estimate in computePi()) {
 9.          print('π ≅ $estimate');
10.      }
11.  }
12.
```

JS. CoffeeScript

```
1.  class Human
2.      constructor : (@name) ->
3.
4.  class Baby extends Human
5.      say    : (msg) -> alert "#{@name} говорит '#{msg}'"
6.      saymsg = (msg) -> alert msg
7.      @echo  = (msg) -> console.log msg
8.
9.  matt = new Baby("Матвей")
10. matt.sayHi()
11.
```

JS. ClojureScript

```
1.  (ns hello-world.core
2.    (:require [cljs.nodejs :as nodejs]))
3.
4.  (nodejs/enable-util-print!)
5.
6.  (defn -main [& args]
7.    (println "Hello world!"))
8.
9.  (set! *main-cli-fn* -main)
10.
```

JS. Elm

```
1.  import Html exposing (text)
2.
3.  main =
4.    text (toString (zip ["Tom", "Sue", "Bob"] [45, 31, 26]))
5.
6.  zip : List a -> List b -> List (a,b)
7.  zip xs ys =
8.    case (xs, ys) of
9.      ( x :: xBack, y :: yBack ) ->
10.         (x,y) :: zip xBack yBack
11.
12.     (_, _) ->
13.         []
14.
15.
```

JS. TypeScript

```
1.  interface IPerson {
2.      name: string;
3.      age: number;
4.  }
5.
6.  function meet(person: IPerson) {
7.      return `Привет, я ${person.name}, мне ${parson.age}`;
8.  }
9.
10. const user = { name: 'Jane', age: 21 };
11. console.log(meet(user));
12.
```

TypeScript

JS. TS

TypeScript *— язык программирования, представленный Microsoft в 2012 году. TypeScript является* ***обратно совместимым*** *с JavaScript и компилируется в последний. TypeScript отличается от JavaScript возможностью явного статического назначения типов, а также поддержкой подключения модулей*

Разработчиком языка TypeScript является **Андерс Хейлсберг** (англ. Anders Hejlsberg), создавший ранее Turbo Pascal, Delphi и C#.

TypeScript Deep Dive — крутая книга по TypeScript

JS. Преимущества TypeScript

- Аннотации типов и проверка их согласования на этапе компиляции
- Интерфейсы, кортежи, декораторы свойств и методов, расширенные возможности ООП
- TypeScript — **надмножество JavaScript**, поэтому любой код на JavaScript будет выполнен и в TypeScript
- Широкая поддержка IDE и адекватный автокомплит
- Поддержка ES6-модулей из коробки

JS. Как переписать проект на TS

1. Переименовываем *.js в *.ts
2. ?
3. ??
4. ???
5. ????
6. ??????
7. PROFIT!
8.

JS. tsconfig.json

```
{
    "compilerOptions": {
        "outDir": "cache/",
        "target": "es2016",
        "declaration": false,
        "module": "commonjs",
        "strictNullChecks": true,
        "sourceMap": true
        ...
     }
}

```

JS. Аннотации типов

```
1.  const valid: boolean = true;
2.  const count: number = 42;
3.  const man: string = 'Barry Allen';
4.
5.  console.log(man * 2);
6.  // Error: The left-hand side of an arithmetic
7.  // operation must be of type 'any', 'number' or an enum type
8.
9.
```

JS. Type Inference

Вывод типов *(англ. type inference) — в программировании возможность компилятора самому логически вывести тип значения у выражения.*

```
1.  const valid = true;
2.  const count = 42;
3.  const man = 'Barry Allen';
4.
5.  console.log(man * 2);
6.  // Error: The left-hand side of an arithmetic
7.  // operation must be of type 'any', 'number' or an enum type
8.
9.
```

JS. Аннотации типов

```
1.  const valid = true;
2.  const count = 42;
3.  const name = 'Barry Allen';
4.
5.  const values: number[] = [1, 2, 3, 4, 5];
6.  const tuple: [string, number] = ['Mean of life', 42];
7.
8.  enum Color {Red, Green, Blue};
9.  const c: Color = Color.Green;
10.
11.
```

JS. Аннотации типов

```
1.  let some: any = true; some = 42;
2.  some = 'maybe a string instead'; // типы не проверяются
3.  // приведение типов (“trust me, I know what I’m doing”)
4.  let length: number = (<string>some).length;
5.  length = (some as string).length;
6.
7.  let unusable: void = undefined;
8.  let u: undefined = undefined;
9.  let n: null = null;
10.
```

JS. Функции в TypeScript

```
1.  function sum(x: number, y: number): number {
2.      return x + y;
3.  }
4.
5.  const many: number = sum(40, 2);
6.
7.  const gcd = (a: number, b: number): number =>
8.      (b === 0) ? a : gcd(b, a % b);
9.
10. console.log(gcd(48, 30)); // 6
11.
12.
```

JS. Функции в TypeScript

```
1.  function sum(x: number, y?: number): number {
2.      if (y) {
3.          return x + y;
4.      } else {
5.          return x;
6.      }
7.  }
8.
9.  console.log(sum(34, 8));  // 42
10. console.log(sum(42));     // OK! - 42
11.
```

JS. Функции в TypeScript

```
1.  function sum(x: number, y: number = 42): number {
2.      return x + y;
3.  }
4.
5.  console.log(sum(34, 8));  // 42
6.  console.log(sum(42));     // OK! - 84
7.
8.
```

JS. Функции в TypeScript

```
1.  function sum(...numbers: number[]): number {
2.      return numbers.reduce((sum: number, current: number):
    number => {
3.          sum += current; return sum;
4.      }, 0);
5.  }
6.
7.  console.log(sum(1, 2, 3, 4, 5));      // 15
8.  console.log(sum(42, 0, -10, 5, 5));   // 42
9.
10.
11.
```

JS. Функции в TypeScript

```
1.  function square(num: number): number;
2.  function square(num: string): number;
3.  function square(num: any): number {
4.      if (typeof num === 'string') {
5.          return parseInt(num, 10) * parseInt(num, 10);
6.      } else {
7.          return num * num;
8.      }
9.  }
10.
```

JS. Функции в TypeScript

```
1.  function square(num: string | number): number {
2.      if (typeof num === 'string') {
3.          return parseInt(num, 10) * parseInt(num, 10);
4.      } else {
5.          return num * num;
6.      }
7.  }
8.
```

JS. Интерфейсы в TypeScript

```
interface Figure {
    width: number;
    readonly height: number;
}

const square: Figure = {width: 42, height: 42};
square.width = 15;    // OK
square.height = 15;   // Cannot assign to read-only property

```

JS. Интерфейсы в TypeScript

```
1.  interface Figure {
2.      width: number;
3.      height: number;
4.  }
5.  interface Square extends Figure {
6.      square: () => number;
7.  }
8.  const sq = {width: 15, height: 20,
9.      square() { return this.width * this.height; } };
10. sq.square();    // 300
11.
```

JS. Классы в TypeScript

```
1.  abstract class Class1 {
2.      abstract func1(): void;  // необходимо определить в
    наследниках
3.  }
4.  class Class2 extends Class1 {
5.      static readonly field3: string = 'hello';
6.      protected name: string;
7.      private field1: number;
8.      constructor() { super(); }
9.      public func1(): void { ... }
10. }
11.
```

JS. Классы в TypeScript

```
1.  interface Squarable {
2.      calcSomething(): number;
3.  }
4.
5.  class Square implements Squarable {
6.      width: number;
7.      height: number;
8.
9.      // Error: Class 'Square' incorrectly implements interface
    'Squarable'.
10.     // Property 'calcSomething' is missing in type 'Square'.
11. }
12.
```

JS. Generics в TypeScript

```
class Queue<T> {
    private data = [];
    push = (item: T) => this.data.push(item);
    pop = (): T => this.data.shift();
}

const queue = new Queue<number>();
queue.push(0);     // OK
queue.push('1');   // Error: cannot push a string

```

JS. Generics в TypeScript

```
1.  function makeKeyValue<K, V>(key: K, value: V): { key: K;
    value: V } {
2.      return {key, value};
3.  }
4.
5.  const pair = makeKeyValue('days', ['ПН', 'ВТ']);
6.  pair.value.push('СР', 'ЧТ', 'ПТ', 'СБ', 'ВС');  // OK
7.  pair.value.push(42);   // Error: cannot push a number
8.
```

JS. Декораторы свойств и методов

```
1.  class Utils {
2.      @memoize
3.      static fibonacci (n: number): number {
4.          return n < 2 ? 1 : Utils.fibonacci(n - 1) +
    Utils.fibonacci(n - 2)
5.      }
6.  }
7.  console.time('count');
8.  console.log(Utils.fibonacci(50));
9.  console.timeEnd('count');     // оооочень долго
10.
11.
```

JS. Декораторы свойств и методов

```
1.  function memoize (target, key, descriptor) {
2.      const originalMethod = descriptor.value;
3.      const cache = {};
4.      descriptor.value = function (n: number): number {
5.          return cache[n] ? cache[n] : cache[n] =
    originalMethod(n);
6.      }
7.  }
8.  console.log(Utils.fibonacci(1000));   //
    7.03303677114227 65e+208
9.  console.timeEnd('count');             // count: 5.668ms
10.
```

JS. Как "типизировать" js-код

TypeScript Declaration Files (`.d.ts`) *— служат для описания интерфейсов, экспортируемых классов и методов для модулей, написанных на обычном JavaScript*

```
 1.  interface JQueryStatic {
 2.      ajax(settings: JQueryAjaxSettings): JQueryXHR;
 3.      (element: Element): JQuery;
 4.      (html: string, ownerDocument?: Document): JQuery;
 5.      (): JQuery;
 6.  }
 7.
 8.  declare var $: JQueryStatic;
 9.  declare module 'jquery' {
10.      export = $;
11.  }
12.
```

JS. Как "типизировать" js-код

Типизация с помощью JSDoc

```
1.  // Пример типизирования функции с помощью JSDoc + TypeScript
2.  /**
3.   * @param p0 {string} - Строковый аргумент объявленный на
    манер TS
4.   * @param {string}  p1 - Строковый аргумент
5.   * @param {string=} p2 - Опциональный аргумент
6.   * @param {string} [p3] - Другой опциональный аргумент
7.   * @param {string} [p4="test"] - Аргумент со значением по-
    умолчанию
8.   * @return {string} Возвращает строку
9.   */
10. function fn3(p0, p1, p2, p3, p4){
11.   // TODO
12. }
13.
```

JS. Как "типизировать" js-код

Flow от Facebook

```
1.  // @flow
2.  function concat(a /*: string */, b /*: string */) {
3.      return a + b;
4.  }
5.
6.  concat('A', 'B'); // Works!
7.  concat(1, 2); // Error!
8.
9.
```

JS. Полезные ссылки

- ECMAScript 2019 Language Specification
- Ecma International
- Отличие между ECMA-262 и ISO/IEC 16262
- Поддержка версий JavaScript на kangax.github.io и возможности браузеров на caniuse.com
- Про полифиллы подробно здесь, сборник полифиллов
- Babel и Babel REPL, а также babel-preset-env
- Подборка крутых книг про JavaScript
- Процесс TC39, репозиторий со списком всех текущих предложений и предложения, перешедшие в stage-4
- Источник информации о TypeScript
- Модуль ts-node: TypeScript execution environment and REPL for node.js
- TypeScript Deep Dive — крутая книга по TypeScript
- Крутая серия книг о JavaScript
- Использование Web Workers — MDN
- Туториал Web Workers — MDN
- WebAssembly — MDN

Домашнее задание № 3

1. Продолжение верстки, новый экран
2. Применение современных возможностей js

Срок сдачи

22 октября

Спасибо за внимание!