ES-2015

Стандарт ES-2015

http://www.ecma-international.org/publications/standards/Ecma-262.htm
https://kangax.github.io/compat-table/es6/
"use strict" - не кроссбраузерно

Babel.js

https://babeljs.io/
транспайлер (конвертер между языками)
полифилл https://en.wikipedia.org/wiki/Polyfill
webpack/grunt/…

1
2 let z

Пример в браузере

```
<div id="output"></div>
<!-- Load Babel -->
<script src="https://unpkg.com/babel-standalone@6/babel.min.js"></script>
<!-- Your custom script here -->
<script type="text/babel">
const getMessage = () => "Hello World";
document.getElementById('output').innerHTML = getMessage();
</script>
```

Переменные const и let

Область видимости let - { }
let видна только после объявления
let нельзя переопределять
с let в цикле для каждой итерации создаётся своя переменная
const - переменная, которую нельзя менять
Если const - объект, то св-ва менять можно

Деструктуризация массива

```
let [test1,test2] = ["test111", "test222"];
//console.log(test1,test2);

let [,test3,] = "Ехал Грека через реку".split(" ");
console.log(test3);
```

Оператор spread

```
let [test4, ... test5] = "Ехал Грека через
реку".split(" ");
console.log(test5);
```

должен стоять только последним

Значения по умолчанию

```
let [test6, test7] = ['Строка'];
console.log(test6, test7);//'Строка' undefined

let [test8, test9 = "по умолчанию"] = ['Строка'];
console.log(test8, test9);// 'Строка' "по умолчанию"

let [test10, test11 = (function(){return 123})()] = ['Строка'];
console.log(test10, test11);// 'Строка' 123
```

Деструктуризация объекта

```
let user = {
  name: "Вася",
  age: 20
};
let {name, age} = user;
console.log(name, age);
```

Присвоение свойств в другую переменную

```
let {name: firstname, age: userage } = user;
```

```
console.log(firstname, userage, 1);

let {name: firstname2, age: userage2="Гость",  test12=
"test1212"} = user;
console.log(firstname2, userage2, test12);
```

Деструктуризация без объявления

```
let d1, d2;
({d1, d2} = {d1:5, d2:6});
console.log(d1, d2);
```

Вложенная деструктуризация

```
let user2 = {
  name: "Вася",
  age: 20,
  job: {
    title: 'front-end',
    salary: 4e4
  }
};
let {name:n, age:a, job: {title:about, salary}} = user2;
console.log(n, a, about, salary);
```

Тестирование

https://goo.gl/forms/mbbTcR6rQU6jLIlZ2

Практическая работа

1. Задание 1. Дан массив объектов со свойствами имя и возраст. Отсортировать массив по именам.
   ```
   let names = [ {name:"Яна",age: 23},
     {name:"Коля",age: 29},
     {name:"Аня",age: 25},]
   ```
2. Задание 2. Напишите функцию, которая будет принимать на вход две строки, а на выходе возвращать объект с полями состоящими из искомой строки и количества её вхождение в другую строку. Например,
   ```
   let{str,num}=str_search("ре","Ехал Грека через
реку");//"ре" 3
   ```

Функции: параметры по умолчанию

```
function ipoteka(S = 1e5, p = 12, n = 12){
  return S + "" + p + n;
}
console.log(ipoteka());
console.log(ipoteka(undefined,99,undefined));

function showDate(date = (new Date()).toLocaleDateString()){
  console.log( date );
}
showDate();
showDate((new Date(2017,5)).toLocaleDateString());
```

Оператор spread с функциями

```
function some(param1, param2, ...params){
  console.log(param1, param2, params);
}
some("т е с т", "т о р т", 34, "п р о б а");
```

примечание: spread должен быть в конце функции, а params - массив

Оператор spread для передачи

```
//И с п о л ь з о в а н и е  о п е р а т о р а spread д л я
п е р е д а ч и  д а н н ы х
let nums = [100,200,150];
let min = Math.min(...nums);
console.log(min);
```

Деструктуризация в параметрах

```
let user3 = {
  name: "П е т я",
  age: 30
};
function showUser({name, age}){
```

```
  console.log(name, age);
}
showUser(user3);
```

Деструктуризация в параметрах со значениями по умолчанию

```
let user4 = {   name: "Петя" };
function showUser({name, age = 45, salary = 5e4}){
  console.log(name, age, salary);
}
showUser(user4); //обязательно вызвать с аргументом

function showUser({name, age = 45, salary = 5e4} = {}){
  console.log(name, age, salary);
}
showUser();
```

Свойство name функции содержит имя функции

```
function foo(){}
let doo = function doo(){};
console.log(foo.name, doo.name);
```

Объявление функции в блоке

```
if( true ){
   next();
   function next(){ console.log("next") }
}
//next();//ошибка, нет такой функции - видна только в блоке
```

Стрелочная функция

```
let step = d => d + 1;
console.log( step( 10) );
```

```
//это почти что
let step2 = function(d) { return d + 1; };
console.log( step2( 12) );
```

Стрелочная с аргументами

```
let sub = (d1,d2) => d1 - d2;
console.log( sub( 12,4) );

//сортировка на лету
console.log([123,25,65].sort((d1,d2)=>d1 - d2));
```

Стрелочная без аргументов

```
const somefunc = () => somefunc.name;
console.log(somefunc.name);

//для большой функции
const somefunc2 = () => {
  //т у т  д р у г о й  к о д
  return somefunc2.name;
}
console.log(somefunc2.name);
```

Стрелочные функции не имеют своего this

```
let complex = {
  courses: ["JavaScript","PHP","MySQL"],
  showCourses: function(){
    this.courses.forEach(
      course => console.log("К у р с : " + course + " В с е г о : " +
this.courses.length)
    )
  }
}
complex.showCourses();
```

```
нельзя использовать в качестве конструктора (не работают со "своим this")
```

Отсутствие arguments у функций-стрелок

```
function outer(){
  let inner = () => console.log(arguments[0]);
  inner(3);
}
outer(4);
```

удобно для декорирования (спросите препода, расскажет на следующем занятии!)

Строки шаблоны

```
let str = `я просто строка`;
let str2 = `а я строка  с символом перевода строки`;
let str3 = `я строка с подстановкой выражения ${str} `;

console.log(str,str2,str3);
```

Функции для шаблонизации строк

```
function funcTemplate(str, ...params){
  console.log(str);
  console.log(str.raw);
  console.log(params);
}
let ddd = 345;
let func1 = funcTemplate`некоторая \n строка ${ddd} пример`;

function pre(str){
  return `<pre>${str}</pre>`;
}
let text = pre`слово      второе слово`;
```

Строковые методы

```
let g = "Ехал Грека через реку";
console.log( g.includes("через") );//входит ли одна строка в другую?
console.log( g.endsWith("реку") );//завершается ли одна строка в другой?
console.log( g.startsWith("Е") );
console.log( g.repeat(3) );//повторение строки
```

Тестирование

https://goo.gl/forms/ZtwGj8wy1ahPN5ph1

Практическая работа

1. Задание 1. Дан код
   ```
   let d = [45,78,10,3];d[7] = 100;
   ```
   Найти сумму значений элементов массива, используя стрелочные функции.

3. Задание 2. Познакомьтесь с методом массивов reduce и выполните предыдущее задание с использованием этого метода.

4. Задание 3. Дан массив объектов со свойствами имя и возраст. Отсортировать массив по именам.
   ```
   let names = [
     {name:"Яна",age: 23},
     {name:"Коля",age: 29},
     {name:"Аня",age: 25},
   ]
   ```
   Рекомендуется подумать над этим заданием самостоятельно, а в случае затруднения разобрать один из нижерасположенных вариантов

   ```
   //console.log(names.sort(({name:n1},{name:n2})=>n1 > n2 ? 1: 0 ))
   //console.log(names.sort((a1,a2)=>a1.age > a2.age ? 1: -1 ))
   //console.log(names.sort((...a)=>a[0].age > a[1].age ? 1: -1 ))
   //console.log(names.sort((a1,a2,p="age")=>a1[p] > a2[p] ? 1: -1 ))
   ```

   Или посмотреть пример замыканий в JS

5. Приступайте к решению задач http://htmllab.ru/zadachi-po-javascript-function/

Короткое свойство

```
let model = "н о в а я";
let power = 120;

let car = {
  model,
  power
};
console.log(car);
```

Вычисляемые свойства

```
let prop = "salary";
let human = {
  [prop]: 1e5,
  [prop+"Gross"]: 1e5*1.13
};
console.log(human.salary);
console.log(human["salaryGross"]);
```

get/setPrototypeOf

ES5 Object.getPrototypeOf(obj)
ES6 Object.setPrototypeOf(obj, newProto)
Свойство __proto__ работает

Object.assign(target, src1, src2...)

```
let programmist = {programmist: true};
let hobby = {type: "sport"};
let human2 = {name: "П е т я", type: "л ю б л ю  п о е с т ь"};
Object.assign(human2,programmist,hobby);
console.log(human2);
```

Копирование объекта без объектных свойств

```
let kolya = {name: "К о л я", type: "л ю б л ю  п о е с т ь"};
```

```
let petya2 = Object.assign({},kolya,programmist);
console.log(petya2);
```

Методы объекта

```
let firstName = "В а с и л и й";
let coder = {
  firstName,
  say(){
    return `П р и в е т , я ${this.firstName}`;
  }
}
console.log(coder.say());
```

Геттеры и сеттеры

```
let firstName = "В а с и л и й";
let coder = {
  firstName,
  say(){     console.log(`П р и в е т , я ${this.firstName}`);   },
  set setName(n){     this.firstName = n;   },
  get getName(){     return this.firstName;   }
}
coder.setName = "В а с я";
console.log(coder.getName);
```

Методы с вычисляемыми названиям

```
let firstName = "В а с и л и й";
let prof = "programmer";
let coder = {
  firstName,
  [prof](){
    return prof;
  }
}
console.log(coder.programmer());
```

super

```
let tree = {   getinfo(){     console.log("я  д е р е в о");   } };
let spruce = {
  __proto__: tree,
  getinfo(){
    console.log(super.getinfo);
    super.getinfo();
    console.log("х в о й н о е");
  }
};
console.log(spruce.getinfo());
```

Особенности super

не работает с стрелочными функциями и обычными методами

```
let tree = {   getinfo(){     console.log("я  д е р е в о");   } };
let spruce = {
  __proto__: tree,
  getinfo(){
    setTimeout(()=>super.getinfo(),1000);
  }
};
console.log(spruce.getinfo());
```

Методы привязаны к объекту навсегда

```
let tree = {    getinfo(){   console.log("я  д е р е в о");   } };
let spruce = {
  __proto__: tree,
  getinfo(){
    setTimeout(()=>super.getinfo(),1000);
  }
};
let getinfo2 = spruce.getinfo;
console.log(getinfo2());
```

obj.__proto__ – ссылка на прототип

Классы. class

```
class Book {
  constructor (title) {
    this.title = title;
  }
  getInfo(){
    console.log(this.title);
  }
}
let book = new Book("JS д л я  н а ч и н а ю щ и х");
console.log(book.getInfo());
```

А раньше…

```
function Book(title){
  this.title = name;
}
Book.prototype.getInfo = function(){
  console.log(this.title);
}
```

Классы: важно помнить

Book нельзя вызвать без new
Объявление класса ведёт себя как let
Метод getInfo() является именно методом, имеет доступ к super
Все методы класса работают в строгом режиме
Все методы класса не перечислимы!

class Expression/Класс выражение

```
let Book = class {
  getInfo() {
   console.log("я  к н и г а");
  }
};
console.log( new Book().getInfo() );
```

```
let Journal = class BaseBook {
  getInfo() {
   console.log("я в т о р о е  и з д а н и е");
  }
};
console.log( new Journal().getInfo() );
//console.log( new BaseBook().getInfo() ); О ш и б к а
```

Геттеры, сеттеры, вычисляемые свойства

```
class Coder  {
  constructor(firstName, lastName){
    this.firstName = firstName;
    this.lastName  = lastName;
  }
  say(){
    console.log(`П р и в е т, я ${this.firstName}`);
  }
  set setName(n){
    [this.firstName,this.lastName] = n.split(' ');
  }
  get getName(){
    return `${this.firstName} ${this.lastName}`;
  }
  ["test" + 123](){
    console.log("м е т о д  с  в ы ч и с л я е м ы м
н а з в а н и е м");
  }
}
let coder = new Coder("В а с я","И в а н о в");
coder.setName = "В а с и л и й  И в а н о в";
console.log(coder.getName);
```

Статические свойства

```
class User{
  constructor(firstName, lastName){
    this.firstName = firstName;
    this.lastName  = lastName;
```

```
  }
}
class UserFabric {
  static createUser(){
    return new User("Тестовый", "пользователь");
  }
}
let guest = UserFabric.createUser();
console.log(guest.firstName);
```

Константы через статические св-ва

```
class Physics {
  static get electronCharge(){
    return 1.6e-19;
  }
}
console.log(Physics.electronCharge);
```

Наследование

```
class Tree  {
  constructor(age){
    this.age = age
  }
  show(){
    console.log(`Дерево. Возраст: ${this.age}`);
  }
};
class Spruce extends Tree{
  show(){
    console.log("Сосна");
    super.show();
  }
};
let tree = new Tree(34);
let spruce = new Spruce(14);
console.log(tree.show());
```

Особенности наследования

Конструктор родителя наследуется автоматически.
Если в потомке он не указан, используется родительский
Если у потомка есть свой constructor, то вызвать конструктор родителя можно через super(), но только внутри конструктора потомка и (!) до обращения к this (он появляется после вызова super)

Тестирование

https://goo.gl/forms/IbhgoCkDvE5Rq9lu1

Практическая работа

1. Игровой пример. Создаем четыре классы космических тел и оцениваем получаемые от них ресурсы. Создайте класс Solid, хранящий координаты x и y, resources - с начальным значением 0, square - площадь поверхности космического объекта, геттер для ресурсов, и метод увеличения количества ресурсов. На основе Solid, создайте классы
    a. Asteroid со своим методом добавления ресурса
    b. Planet метод производства ресурсов должен отличаться от астероидного
    c. Star
2. * Создайте класс MyList с свойством хранящим массив слов и методами: (1) методом устанавливающим формат экспорта текста и (2) методом вызывающим экспорт

Модули в EcmaScript

файл с кодом
export - помечаем переменные и функции для использования вне модуля
import - подключение других модулей
встроенной поддержки нет, но есть webpack и тд..

export

```
export let user = {name: "В а с и л и й"};

let user2 = {name: "В а с и л и й"};
let user3 = {name: "А н н а"};
export {user2}; // и л и export {user2, user3};
```

export под псевдонимом

```
export {user2 as programmer, user3 as user};
```

export классов и функций

```
export function summ(a,b) {return a + b;} //и м я
о б я з а т е л ь н о

export class Tree {
  constructor(height){
    this.height = height;
  }
}
```

export default

```
export default class Tree {
  constructor(height){
    this.height = height;
  }
}
```

import

```
import {user2, user3} from "./lib"; //путь к файлу/модулю
user2, user3 - импортируемые переменные (были объявлены в export)
```

import всех значений

```
import * as  props from "./lib";
console.log(props.user2);
```

Итераторы и генераторы

```
итерируемые объекты (объекты, которые можно перебирать в цикле)
"use strict";
let names = ["Петя", "Даша", "Ваня"];

for (let value of names) {console.log(value);}
for (let value of names[0]) {console.log(value);}
```

Аналог итератора в ECMAScript5

```
"use strict";
let dayNames =
["воскресенье","понедельник","вторник","среда","четверг","пятница","суббота" ];

function nextDay(arr, start){
  return {
    next: function(){
      return arr[start++];
    }
  }
}
let iteratorDays = nextDay(dayNames,1);
for(let i = 0; i < 6; i++)
  console.log(iteratorDays.next());
```

Генераторы

```
Генератор - функция, возвращающая итератор

"use strict";
let dayNames =
["воскресенье","понедельник","вторник","среда","четверг","пятница","суббота" ];

function* nextDay(arr, start){
  for(let i = 0; i < arr.length; i++) yield arr[i];
```

```
}
let iteratorDays = nextDay(dayNames,1);
for(let i = 0; i < 7; i++)
  console.log(iteratorDays.next());
for(let i of iteratorDays)
  console.log(i);
```

Тест

https://goo.gl/forms/cxOtA2wU6EJpRIA63

Практическая работа

1. Задание 1. Напиши функцию создания генератора sequence(start, step). Она при вызове возвращает другую функцию-генератор, которая при каждом вызове дает число на 1 больше, и так до бесконечности. Начальное число, с которого начинать отсчет, и шаг, задается при создании генератора. Шаг можно не указывать, тогда он будет равен одному. Начальное значение по умолчанию равно 0. Генераторов можно создать сколько угодно