

# Лекция 2

JavaScript

# Основы JavaScript

Подробнее тут <https://learn.javascript.ru/first-steps>

# Переменные

Мы можем объявить переменные для хранения данных с помощью ключевых слов `var`, `let` или `const`.

- `let` – это современный способ объявления.
- `var` – это устаревший способ объявления. Обычно мы вообще не используем его.
- `const` – похоже на `let`, но значение переменной не может изменяться.

# Типизация

JavaScript **не строго типизированный** язык.

```
1 // Не будет ошибкой  
2 let message = "hello";  
3 message = 123456;
```

# Преобразование типов

```
1 "" == 0 // true
2 "0" == 0 // true
3 "" == "0" // false
4 "0" == false // true
```

# Математические операции

```
1 const a = 3 + 2; // сложение
2 const b = a - 5; // вычитание
3 const c = -b; // отрицание
4 const d = 10 % 2; // остаток от деления
5 const e = d * 2; // умножение
6 const f = d / 2; // деление
7 let a = 0;
8 a += 2; // сложение с присвоением
9 a++; // плюс один
10
```

# Операторы сравнения

```
1 3 > 2
2 4 >= 1
3
4 1 < 3
5 2 <= 2
6
7 false == ""
8 0 == 0
9 false != 1
10 false != true
```

# Оператор условий

```
1 if (year === 2007) {  
2   alert( 'Верните меня туда' );  
3 } else {  
4   alert( 'Эх...' ); // любое значение, кроме 2007  
5 }
```

```
1 const year = (условие) ? 2007 : 2023;
```



# Циклы

```
1 while (condition) {  
2     // код  
3     // также называемый "телом цикла"  
4 }  
5  
6 do {  
7     // тело цикла  
8 } while (condition);  
9  
10 for (начало; условие; шаг) {  
11     // ... тело цикла ...  
12 }
```

# Функции

```
1 function имя(параметры) {  
2     ...тело...  
3 }
```

# Function Expression vs Function Declaration

```
1 // Function Declaration
2 function sum(a, b) {
3   return a + b;
4 }
```

```
1 // Function Expression
2 let sum = function(a, b) {
3   return a + b;
4 };
```

**Function Expression** создаётся, когда выполнение доходит до него, и затем уже может использоваться.

**Function Declaration** может быть вызвана раньше, чем она объявлена.

# Стрелочные функции

```
1 const sum1 = (a, b) => a + b;  
2  
3 const sum2 = (a, b) => { // фигурная скобка, открывающая тело многострочной функции  
4   let result = a + b;  
5   return result; // если мы используем фигурные скобки, то нам нужно явно указать "return"  
6 };  
7  
8 alert(sum1(1, 2)); // 3  
9 alert(sum2(2, 3)); // 5
```

# Объекты

Подробнее тут <https://learn.javascript.ru/object-basics>

# Создать объект

Объект - структура "ключ - значение" встроенная в JavaScript.

```
1 let user = new Object(); // синтаксис "конструктор объекта"  
2 let user = {}; // синтаксис "литерал объекта"
```

# Доступ к полям объекта

```
1 const user = {      // объект
2   name: "John",    // под ключом "name" хранится значение "John"
3   age: 30           // под ключом "age" хранится значение 30
4 };
5
6 console.log(user.name);
7 console.log(user["age"]);
8
```

# Присваивание полей. Вычисляемые свойства

```
1 const user = {};  
2  
3 user.isAdmin = true;  
4 console.log(user.isAdmin); // true  
5  
6 const key = 'age';  
7 user[key] = 18;  
8 console.log(user[key]); // 18  
9 console.log(user.age); // 18  
10  
11 console.log(user.name) // undefined
```



# Сравнение объектов

```
1 let a = {};  
2 let b = a; // копирование по ссылке  
3  
4 alert( a == b ); // true, обе переменные ссылаются на один и тот же объект  
5 alert( a === b ); // true
```

```
1 let a = {};  
2 let b = {}; // два независимых объекта  
3  
4 alert( a == b ); // false
```

# this или магическая область видимости

```
1 let user = { name: "John" };
2 let admin = { name: "Admin" };
3
4 function sayHi() {
5   alert( this.name );
6 }
7
8 // используем одну и ту же функцию в двух объектах
9 user.f = sayHi;
10 admin.f = sayHi;
11
12 // эти вызовы имеют разное значение this
13 // "this" внутри функции - это объект "перед точкой"
14 user.f(); // John (this == user)
15 admin.f(); // Admin (this == admin)
```

# Псевдо-ООП

```
1 function User(name) {  
2   this.name = name;  
3  
4   this.sayHi = function() {  
5     alert( "Меня зовут: " + this.name );  
6   };  
7 }  
8  
9 let john = new User("John");  
10  
11 john.sayHi(); // Меня зовут: John
```

# Типы данных

Подробнее тут <https://learn.javascript.ru/data-types>

# Примитивы

- string
- number
- boolean
- symbol
- null
- undefined
- bigint

# Примитив как объект

Работа с методами примитива:

- Примитивы остаются примитивами
- Для доступа к методу создается “объект-обертка”, который предоставляет нужную функциональность, а потом удаляется

Объекты обертки примитивов:

- String
- Number
- Boolean
- Symbol
- BigInt

# Примитив как объект

- Строка `str` - примитив
- В момент обращения к `toUpperCase` создается “объект-обертка”
- Запускается метод, возвращает новую строку
- Специальный “объект-обертка” удаляется, остается примитив `str`

```
1 let str = "Привет";  
2  
3 alert( str.toUpperCase( ) ); // ПРИВЕТ
```

# Примитив как объект

- null/undefined не имеют “объект-обертку”

```
1 alert(null.test); // ошибка
```

- Использовать конструкторы “объектов-оберток” нежелательно

```
1 alert( typeof 0 ); // "число"  
2  
3 alert( typeof new Number(0) ); // "object"!
```

- Использование функцию без new - преобразование типов

```
1 let num = Number("123"); // превращает строку в число
```



# Числа

- 1000000000
- 1\_000\_000\_000 - 1000000000
- 1e9 - 1000000000
- 1e-9 - 1000000000
- 0b11111111 - 255 (двоичная)
- 0o377 - 255 (восьмеричная)
- 0xFF - 255 (шестнадцатеричные)

# Числа (методы)

- `num.toString(base)` - число в строку

```
1 let num = 255;  
2  
3 alert( num.toString(16) ); // ff  
4 alert( num.toString(2) );  // 11111111
```

- `num.toFixed(digits)` - округление в меньшую сторону

```
1 let num = 12.345;  
2  
3 alert( num.toFixed(1) ); // "12.3"  
4 alert( num.toFixed(2) ); // "12.34"
```

# Числа (округление)

- `Math.floor(num)` - округление в меньшую сторону

```
1 alert( Math.floor(3.1) ) // 3
2 alert( Math.floor(-1.1) ) // -2
```

- `Math.ceil(num)` - округление в большую сторону

```
1 alert( Math.ceil(3.1) ) // 4
2 alert( Math.ceil(-1.1) ) // -1
```

- `Math.round(num)` - округление до ближайшего целого

```
1 alert( Math.round(3.1) ) // 3
2 alert( Math.round(3.6) ) // 4
3 alert( Math.round(-1.1) ) // -1
```

- `Math.trunc(num)` - удаление дробной части

```
1 alert( Math.trunc(3.1) ) // 3
2 alert( Math.trunc(-1.1) ) // -1
```

# Числа (неточные выражения)

- Слишком большое число

```
1 alert( 1e500 ); // Infinity
```

- Сложение дробных чисел

```
1 alert( 0.1 + 0.2 ); // 0.30000000000000004
```

- Потеря точности

```
1 alert( 9999999999999999 ); // покажет 10000000000000000
```

# Числа (проверки)

- `isNaN(value)` - проверка на NaN

```
1 alert( isNaN(NaN) ); // true
2 alert( isNaN("str") ); // true
3 alert( NaN === NaN ); // false
```

- `isFinite(value)` - преобразование в число, проверка что это число

```
1 alert( isFinite("15") ); // true
2 alert( isFinite("str") ); // false, потому что специальное значение: NaN
3 alert( isFinite(Infinity) ); // false, потому что специальное значение: Infinity
```

# Числа (проверки)

- **Number.isNaN(value)** - строгая проверка isNaN

```
1 alert( Number.isNaN("str") ); // false, так как "str" является строкой, а не числом
2 alert( isNaN("str") ); // true, так как isNaN сначала преобразует строку "str" в число и в результате преобразования получает NaN
```

- **Number.isFinite(value)** - строгая проверка isFinite

```
1 alert( Number.isFinite("123") ); // false, так как "123" является строкой, а не числом
2 alert( isFinite("123") ); // true, так как isFinite сначала преобразует строку "123" в число 123
```

# Числа (преобразование)

- +value или Number(value) - преобразование в число

```
1 alert( Number('100') ); // 100
2 alert( Number('100px') ); // NaN
```

- parseInt(value) - преобразование в число (только число из строки)

```
1 alert( parseInt('100px') ); // 100
2 alert( parseInt('12.3') ); // 12, вернётся только целая часть
```

- parseFloat(value) - преобразование в число (только число из строки)

```
1 alert( parseFloat('12.5em') ); // 12.5
2 alert( parseFloat('12.3.4') ); // 12.3, произойдёт остановка чтения на второй точке
```

# Строки

- Одинарные кавычки
- Двойные кавычки
- Обратные кавычки

```
1 let single = 'single-quoted'; // single-quoted
2 let double = "double-quoted"; // double-quoted
3
4 let backticks = `backticks ${single} ${double}`; // backticks single-quoted double-quoted
```



# Строки (доступ к символам)

## Доступ к символам как у массива

```
1 let str = `Hello`;
2
3 // получаем первый символ
4 alert( str[0] ); // H
5 alert( str.at(0) ); // H
6
7 // получаем последний символ
8 alert( str[str.length - 1] ); // o
9 alert( str.at(-1) );
```

# Строки (неизменяемы)

```
1 let str = 'Hi';  
2  
3 str[0] = 'h'; // ошибка  
4 alert( str[0] ); // не работает
```

```
1 let str = 'Hi';  
2  
3 str = 'h' + str[1]; // заменяем строку  
4  
5 alert( str ); // hi
```

# Строки (методы)

- `str.length` - длина строки
- `str.indexOf(str, pos)` - получение индекса наличия подстроки (с начала)
- `str.lastIndexOf(str, pos)` - получение индекса наличия подстроки (с конца)
- `str.toLowerCase()` - преобразование в нижний регистр
- `str.toUpperCase()` - преобразование в верхний регистр
- `str.includes(str, pos)` - проверка наличия подстроки (с позиции)
- `str.startsWith(str)` - проверка наличия подстроки (с начала)
- `str.endsWith(str)` - проверка наличия подстроки (с конца)

# Строки (получение подстроки)

- `str.slice(start, end)` - возвращает часть строки от `start` (не включая) до `end`

```
1 let str = "stringify";
2 // 'strin', символы от 0 до 5 (не включая 5)
3 alert( str.slice(0, 5) );
4 // 's', от 0 до 1, не включая 1, т. е. только один символ на позиции 0
5 alert( str.slice(0, 1) );
```

- `str.substring(start, end)` - возвращает часть строки между `start` и `end`

```
1 let str = "stringify";
2
3 // для substring эти два примера – одинаковы
4 alert( str.substring(2, 6) ); // "ring"
5 alert( str.substring(6, 2) ); // "ring"
```

- `str.substr(start, length)` - возвращает часть строки от `start` длины `length`

```
1 let str = "stringify";
2 // ring, получаем 4 символа, начиная с позиции 2
3 alert( str.substr(2, 4) );
```

# Массивы

```
1 let arr = new Array();
2 let arr = [];
3
4 let fruits = ["Яблоко", "Апельсин", "Слива"];
5
6 alert( fruits[0] ); // Яблоко
7 alert( fruits[1] ); // Апельсин
8 alert( fruits[2] ); // Слива
9
10 // разные типы значений
11 let arr = [ 'Яблоко', { name: 'Джон' }, true, function() { alert('привет'); } ];
```

# Массивы (доступ)

- `arr[]`
- `arr.at(index)`

```
1 let fruits = ["Apple", "Orange", "Plum"];
2 alert( fruits[fruits.length-1] ); // Plum
3
4 let fruits = ["Apple", "Orange", "Plum"];
5 // то же самое, что и fruits[fruits.length-1]
6 alert( fruits.at(-1) ); // Plum
```

# Массивы (методы)

- `arr.length` - длина массива
- `arr.pop()` - удаление последнего элемента
- `arr.push(el)` - добавление в конец массива
- `arr.shift()` - удаление первого элемента
- `arr.unshift(el)` - добавление в начало массива

```
1 let fruits = ["Яблоко", "Груша"];  
2 fruits.pop(); // удаляем "Груша" ("Яблоко")  
3 fruits.push("Груша"); // добавляем "Груша" ("Яблоко", "Груша")  
4 fruits.shift(); // удаляем "Яблоко" ("Груша")  
5 fruits.unshift("Яблоко"); // добавляем "Яблоко" ("Яблоко", "Груша")
```

# Массивы (добавление/удаление значений)

- `arr.slice(start, end)` - создание нового массива с `start` до `end` (не включая)

```
1 let arr = ["t", "e", "s", "t"];  
2 alert( arr.slice(1, 3) ); // e,s (копирует с 1 до 3)  
3 alert( arr.slice(-2) ); // s,t (копирует с -2 до конца)
```

- `arr.concat(arg1, arg2...)` - создание нового массива с добавлением `argN`

```
1 let arr = [1, 2];  
2 alert( arr.concat([3, 4]) ); // 1,2,3,4  
3 alert( arr.concat([3, 4], [5, 6]) ); // 1,2,3,4,5,6
```



# Массивы (перебор элементов)

Цикл for

```
1 let arr = ["Яблоко", "Апельсин", "Груша"];  
2  
3 for (let i = 0; i < arr.length; i++) {  
4   alert( arr[i] ); // Яблоко, Апельсин, Груша  
5 }
```

# Массивы (перебор элементов)

## Цикл for of

```
1 let arr = ["Яблоко", "Апельсин", "Груша"];  
2  
3 for (let el of arr) {  
4   alert( el ); // Яблоко, Апельсин, Груша  
5 }
```

# Массивы (перебор элементов)

## Метод forEach

```
1 let arr = ["Яблоко", "Апельсин", "Груша"];  
2  
3 arr.forEach((el) => alert ( el ));
```

# Массивы (поиск)

- `arr.indexOf(el, start)` - получение индекса элемента (с начала)
- `arr.lastIndexOf(el, start)` - получение индекса элемента (с конца)
- `arr.includes(el, start)` - проверка наличия элемента

# Массивы (поиск)

- `arr.find(function(el, index))` - получение элемента по условию
- `arr.findIndex(function(el, index))` - получение индекса по условию (с начала)
- `arr.findLastIndex(function(el, index))` - получение индекса по условию (с конца)

```
1 let users = [  
2   {id: 1, name: "Вася"},  
3   {id: 2, name: "Петя"},  
4   {id: 3, name: "Маша"}  
5 ];  
6  
7 let user = users.find((item) => item.id == 1); // {id: 1, name: "Вася"}  
8 users.findIndex((user) => user.name == 'Вася') // 0  
9 users.findLastIndex((user) => user.name == 'Вася') // 3
```

# Массивы (преобразование массива)

- `arr.filter(function(el, index))` - получение нового массива по условию

```
1 let users = [  
2   {id: 1, name: "Вася"},  
3   {id: 2, name: "Петя"},  
4   {id: 3, name: "Маша"}  
5 ];  
6  
7 let someUsers = users.filter((item) => item.id < 3); // первые 2
```

- `arr.map(function(el, index))` - преобразование каждого элемента

```
1 let lengths = ["Бильбо", "Гэндальф", "Назгул"].map((item) => item.length);  
2 alert(lengths); // 6,8,6
```

# Массивы (преобразование массива)

- `arr.sort(function(el1, el2))` - сортировка массива
- `arr.reverse()` - меняет порядок элементов на обратный
- `str.split(delim)` - преобразование строки в массив
- `arr.join(glue)` - преобразование массива в строку

# Массивы (преобразование массива)

`arr.reduce(function(acc, el, index))`

```
1 let arr = [1, 2, 3, 4, 5];  
2  
3 let result = arr.reduce((sum, current) => sum + current, 0);  
4  
5 alert(result); // 15
```



# Массивы (проверка)

`Array.isArray(arr)` - проверка, что элемент массив

```
1 alert(typeof {}); // object
2 alert(typeof []); // тоже object
3 alert(Array.isArray({})); // false
4 alert(Array.isArray([])); // true
```

# Object (методы)

- `Object.keys(obj)` - получение массива ключей
- `Object.values(obj)` - получение массива значений
- `Object.entries(obj)` - получение массива ключ/значение

```
1 let user = {  
2   name: "John",  
3   age: 30  
4 };  
5  
6 Object.keys(user) // ["name", "age"]  
7 Object.values(user) // ["John", 30]  
8 Object.entries(user) // [ ["name", "John"], ["age", 30] ]
```

# Деструктуризация (массив)

```
1 // у нас есть массив с именем и фамилией
2 let arr = ["Ilya", "Kantor"];
3
4 // деструктурирующее присваивание
5 // записывает firstName = arr[0]
6 // и surname = arr[1]
7 let [firstName, surname] = arr;
8
9 alert(firstName); // Ilya
10 alert(surname); // Kantor
```

# Деструктуризация (массив)

```
1 let [name1, name2] = ["Julius", "Caesar", "Consul", "of the Roman Republic"];
2
3 alert(name1); // Julius
4 alert(name2); // Caesar
5 // Дальнейшие элементы нигде не присваиваются
```

```
1 // значения по умолчанию
2 let [name = "Guest", surname = "Anonymous"] = ["Julius"];
3
4 alert(name); // Julius (из массива)
5 alert(surname); // Anonymous (значение по умолчанию)
```

# Деструктуризация (объект)

```
1 let options = {  
2   title: "Menu",  
3   width: 100,  
4   height: 200  
5 };  
6  
7 let {title, width, height} = options;  
8  
9 alert(title); // Menu  
10 alert(width); // 100  
11 alert(height); // 200
```

# Деструктуризация (объект)

```
1 let options = {  
2   title: "Menu",  
3   height: 200,  
4   width: 100  
5 };  
6  
7 // title = свойство с именем title  
8 // rest = объект с остальными свойствами  
9 let {title, ...rest} = options;  
10  
11 // сейчас title="Menu", rest={height: 200, width: 100}  
12 alert(rest.height); // 200  
13 alert(rest.width);  // 100
```

## Деструктуризация (объект)

```
1 let options = {  
2   title: "Menu"  
3 };  
4  
5 let {width: w = 100, height: h = 200, title} = options;  
6  
7 alert(title);    // Menu  
8 alert(w);        // 100  
9 alert(h);        // 200
```

# Деструктуризация (вложенные)

```
1 let options = {
2   size: {
3     width: 100,
4     height: 200
5   },
6   items: ["Cake", "Donut"],
7   extra: true
8 };
9
10 // деструктуризация разбита на несколько строк для ясности
11 let {
12   size: { // положим size сюда
13     width,
14     height
15   },
16   items: [item1, item2], // добавим элементы к items
17   title = "Menu" // отсутствует в объекте (используется значение по умолчанию)
18 } = options;
19
20 alert(title); // Menu
21 alert(width); // 100
22 alert(height); // 200
23 alert(item1); // Cake
24 alert(item2); // Donut
```



# Деструктуризация (функции)

```
1 let options = {
2   title: "My menu",
3   items: ["Item1", "Item2"]
4 };
5
6 function showMenu({
7   title = "Untitled",
8   width: w = 100, // width присваиваем в w
9   height: h = 200, // height присваиваем в h
10  items: [item1, item2] // первый элемент items присваивается в item1, второй в item2
11 }) {
12   alert( `${title} ${w} ${h}` ); // My Menu 100 200
13   alert( item1 ); // Item1
14   alert( item2 ); // Item2
15 }
16
17 showMenu(options);
```

# Map

Map - коллекция ключ/значение

```
1 let map = new Map();
2
3 map.set("1", "str1");    // строка в качестве ключа
4 map.set(1, "num1");      // цифра как ключ
5 map.set(true, "bool1");  // булево значение как ключ
6
7 // помните, обычный объект Object приводит ключи к строкам?
8 // Map сохраняет тип ключей, так что в этом случае сохранится 2 разных значения:
9 alert(map.get(1)); // "num1"
10 alert(map.get("1")); // "str1"
11
12 alert(map.size); // 3
```

# Map (методы)

- `map.set(key, value)`
- `map.get(key)`
- `map.has(key)`
- `map.delete(key)`
- `map.clear()`
- `map.size()`
- `map.keys()` - получение массива ключей
- `map.values()` - получение массива значений
- `map.entries()` - получение массива ключ/значение

# Set

## Set - коллекция значений

```
1 let set = new Set();
2
3 let john = { name: "John" };
4 let pete = { name: "Pete" };
5 let mary = { name: "Mary" };
6
7 // считаем гостей, некоторые приходят несколько раз
8 set.add(john);
9 set.add(pete);
10 set.add(mary);
11 set.add(john);
12 set.add(mary);
13
14 // set хранит только 3 уникальных значения
15 alert(set.size); // 3
16
17 for (let user of set) {
18   alert(user.name); // John (потом Pete и Mary)
19 }
```

# Set (методы)

- `set.add(value)`
- `set.has(value)`
- `set.delete(value)`
- `set.clear()`
- `set.size()`
- `set.keys()` - получение массива значений
- `set.values()` - получение массива значений
- `set.entiers()` - получение массива значение/значение

# JSON (stringify)

```
1 let student = {
2   name: 'John',
3   age: 30,
4   isAdmin: false,
5   courses: ['html', 'css', 'js'],
6   wife: null
7 };
8
9 let json = JSON.stringify(student);
10
11 alert(typeof json); // мы получили строку!
12
13 alert(json);
14 /* выведет объект в формате JSON:
15 {
16   "name": "John",
17   "age": 30,
18   "isAdmin": false,
19   "courses": ["html", "css", "js"],
20   "wife": null
21 }
22 */
```

# JSON (parse)

```
1 // строковый массив
2 let numbers = "[0, 1, 2, 3]";
3
4 numbers = JSON.parse(numbers);
5
6 alert( numbers[1] ); // 1
```

```
1 let user = '{ "name": "John", "age": 35, "isAdmin": false, "friends": [0,1,2,3] }';
2
3 user = JSON.parse(user);
4
5 alert( user.friends[1] ); // 1
```

# Классы

Подробнее тут <https://learn.javascript.ru/classes>



# Класс

```
1 class User {  
2  
3   constructor(name) {  
4     this.name = name;  
5   }  
6  
7   sayHi() {  
8     alert(this.name);  
9   }  
10  
11 }  
12  
13 // Использование:  
14 let user = new User("Иван");  
15 user.sayHi();
```

## Класс (что это?)

```
1 class User {  
2   constructor(name) { this.name = name; }  
3   sayHi() { alert(this.name); }  
4 }  
5  
6 // доказательство: User - это функция  
7 alert(typeof User); // function
```

# Классы (геттеры/сеттеры)

```
1 class User {  
2  
3   constructor(name) {  
4     // вызывает сеттер  
5     this.name = name;  
6   }  
7  
8   get name() {  
9     return this._name;  
10  }  
11  
12  set name(value) {  
13    if (value.length < 4) {  
14      alert("Имя слишком короткое.");  
15      return;  
16    }  
17    this._name = value;  
18  }  
19  
20 }  
21  
22 let user = new User("Иван");  
23 alert(user.name); // Иван  
24  
25 user = new User(""); // Имя слишком короткое.
```

## Классы (статические свойства)

```
1 class Article {  
2     static publisher = "Илья Кантор";  
3 }  
4  
5 alert( Article.publisher ); // Илья Кантор
```

# Классы (статические методы)

```
1 class Article {
2   constructor(title, date) {
3     this.title = title;
4     this.date = date;
5   }
6
7   static compare(articleA, articleB) {
8     return articleA.date - articleB.date;
9   }
10 }
11
12 // использование
13 let articles = [
14   new Article("HTML", new Date(2019, 1, 1)),
15   new Article("CSS", new Date(2019, 0, 1)),
16   new Article("JavaScript", new Date(2019, 11, 1))
17 ];
18
19 articles.sort(Article.compare);
20
21 alert( articles[0].title ); // CSS
```

# Классы (наследование)

```
1 class Animal {
2   constructor(name) {
3     this.speed = 0;
4     this.name = name;
5   }
6   run(speed) {
7     this.speed = speed;
8     alert(`${this.name} бежит со скоростью ${this.speed}.`);
9   }
10  stop() {
11    this.speed = 0;
12    alert(`${this.name} стоит неподвижно.`);
13  }
14 }
15
16 class Rabbit extends Animal {
17   hide() {
18     alert(`${this.name} прячется!`);
19   }
20 }
```

# Классы (наследование, переопределение методов)

```
1 class Animal {
2
3   constructor(name) {
4     this.speed = 0;
5     this.name = name;
6   }
7
8   run(speed) {
9     this.speed = speed;
10    alert(`${this.name} бежит со скоростью ${this.speed}.`);
11  }
12
13  stop() {
14    this.speed = 0;
15    alert(`${this.name} стоит неподвижно.`);
16  }
17
18 }
19
20 class Rabbit extends Animal {
21   hide() {
22     alert(`${this.name} прячется!`);
23   }
24
25   stop() {
26     super.stop(); // вызываем родительский метод stop
27     this.hide();  // и затем hide
28   }
29 }
```

# Классы (наследование, переопределение конструктора)

```
1 class Animal {  
2     constructor(name) {  
3         this.speed = 0;  
4         this.name = name;  
5     }  
6     // ...  
7 }  
8  
9 class Rabbit extends Animal {  
10  
11     constructor(name, earLength) {  
12         this.speed = 0;  
13         this.name = name;  
14         this.earLength = earLength;  
15     }  
16  
17     // ...  
18 }
```



# Классы (приватные свойства)

```
1 class CoffeeMachine {
2   _waterAmount = 0;
3
4   set waterAmount(value) {
5     if (value < 0) throw new Error("Отрицательное количество воды");
6     this._waterAmount = value;
7   }
8
9   get waterAmount() {
10    return this._waterAmount;
11  }
12
13  constructor(power) {
14    this._power = power;
15  }
16
17 }
18
19 // создаём новую кофеварку
20 let coffeeMachine = new CoffeeMachine(100);
21
22 // устанавливаем количество воды
23 coffeeMachine.waterAmount = -10; // Error: Отрицательное количество воды
```

# Классы (приватные свойства)

```
1 class CoffeeMachine {
2     #waterLimit = 200;
3
4     #checkWater(value) {
5         if (value < 0) throw new Error("Отрицательный уровень воды");
6         if (value > this.#waterLimit) throw new Error("Слишком много воды");
7     }
8 }
9
10 let coffeeMachine = new CoffeeMachine();
11
12 // снаружи нет доступа к приватным методам класса
13 coffeeMachine.#checkWater(); // Error
14 coffeeMachine.#waterLimit = 1000; // Error
```

## Классы (проверка)

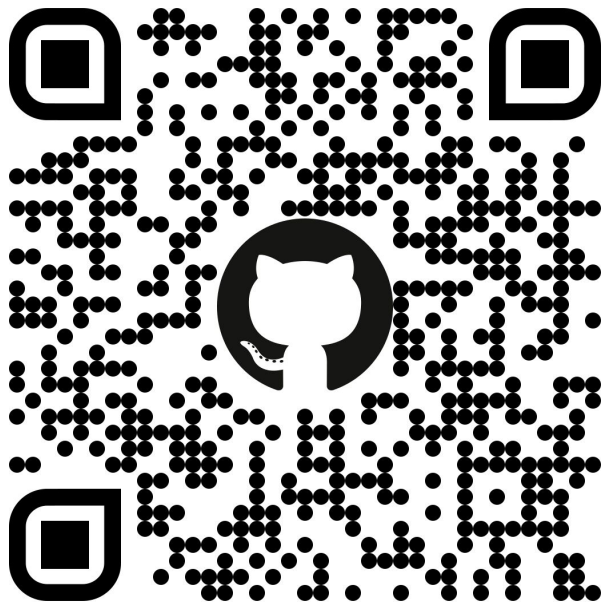
```
1 class Animal {}  
2 class Rabbit extends Animal {}  
3  
4 let rabbit = new Rabbit();  
5 alert( rabbit instanceof Rabbit ); // true  
6 alert(rabbit instanceof Animal); // true
```

# Документ

Подробнее тут <https://learn.javascript.ru/document>

# Обратная связь по курсу

Issue на GitHub - <https://github.com/iu5git/JavaScript/issues>



# Обратная связь по лекции

Google формы - <https://forms.gle/nQ1iqgiHAnXYkycS8>

