ЦЕЛЬ ЛЕКЦИИ

- Единый базис знаний
- Задать вектор развития

ПЛАН ЛЕКЦИИ

- Вступление
- История
- Краткий обзор JavaScript
- Примитивные типы данных
- Операторы
- Типы операторов
- Объекты
- Массивы

ECMASCRIPT

ES - аббревиатура для ECMAScript

- 1997 год - ES1
- 1998 год - ES2
- 1999 год - ES3
- ... - ES4
- 2009 год - ES5
- 2011 год - ES5.1
- 2015 год - ES6/ES2015
- 2016 год - ES7/ES2016
- 2017 год - ES8/ES2017
- 2018 год - ES9/ES2018
- 2019 год - ES10/ES2019
- 2020 год - ES11/ES2020

Ссылка

ОСОБЕНОСТИ JAVASCRIPT

- Динамическая типизация
- Слабая типизация
- Функции - объекты высшего порядка
- Прототипное наследование

ДИНАМИЧЕСКАЯ ТИПИЗАЦИЯ

Переменная связывается с типом в момент присваивания значения, а не в момент объявления переменной

```
1 let a = 'a';
2
3 a = 1;
```

ДИНАМИЧЕСКАЯ ТИПИЗАЦИЯ

Переменная связывается с типом в момент присваивания значения, а не в момент объявления переменной

```
1 let a = 'a';
2
3 a = 1;
```

СЛАБАЯ ТИПИЗАЦИЯ

При некоторых условиях может произойти неяное преобразование (приведение типов) с целью выполнения операции

```
1 const a = '10';
2 const b = 5;
3
4 a / b; // 2
```

СЛАБАЯ ТИПИЗАЦИЯ

При некоторых условиях может произойти неяное преобразование (приведение типов) с целью выполнения операции

```
1 const a = '10';
2 const b = 5;
3
4 a / b; // 2
```

ФУНКЦИИ - ОБЪЕКТЫ ВЫСШЕГО ПОРЯДКА

Функции можно передвать в качестве аргументов другой функции и возращать как результат функци

```
1 function a(){};
2
3 function b(func) {
4     return func;
5 }
6 b(a); // a
```

ПРОТОТИПНОЕ НАСЛЕДОВАНИЕ

Классов в JS не существует, а наследование основано на прототипировании

```
1 function Bird() {}
2 function Duck() {};
3
4 Duck.prototype = new Bird();
5 const duck = new Duck();
```

ПРИМИТИВНЫЕ ТИПЫ ДАННЫХ

Иначе, value types

- String
- Number
- BigInt
- Boolean
- Undefined
- Symbol
- Null (???)

`typeof`

Оператор возвращает строку, которая указывает на тип значения

```
1 typeof 'test'; // 'string'
2
3 typeof 1; // 'number'
4
5 typeof true; // 'boolean'
```

`typeof`

Оператор возвращает строку, которая указывает на тип значения

```
1 typeof 'test'; // 'string'
2
3 typeof 1; // 'number'
4
5 typeof true; // 'boolean'
```

typeof

Оператор возвращает строку, которая указывает на тип значения

```
1 typeof 'test'; // 'string'
2
3 typeof 1; // 'number'
4
5 typeof true; // 'boolean'
```

String

```
1 const a = 'quack123';
2 const b = "quack123";
3 typeof b; // 'string'
4
5 const c = new String(123);
6 typeof c; // 'object'
7
8 const d = String(123);
9 typeof d; // 'string'
```

Важно! Отдельного типа для одного символа не существует.

String

```
1 const a = 'quack123';
2 const b = "quack123";
3 typeof b; // 'string'
4
5 const c = new String(123);
6 typeof c; // 'object'
7
8 const d = String(123);
9 typeof d; // 'string'
```

Важно! Отдельного типа для одного символа не существует.

String

```
1 const a = 'quack123';
2 const b = "quack123";
3 typeof b; // 'string'
4
5 const c = new String(123);
6 typeof c; // 'object'
7
8 const d = String(123);
9 typeof d; // 'string'
```

Важно! Отдельного типа для одного символа не существует.

String

```
1 const a = 'quack123';
2 const b = "quack123";
3 typeof b; // 'string'
4
5 const c = new String(123);
6 typeof c; // 'object'
7
8 const d = String(123);
9 typeof d; // 'string'
```

Важно! Отдельного типа для одного символа не существует.

ОБРАТНЫЕ КАВЫЧКИ

```
1  const c = `quack${122+1}`;
2
3  const c1 = `qua
4  ck${122+1}`; // quac\nk123
5
6  const person = 'world';
7  function tag(strings, name) {
8      var str0 = strings[0];
9      return `${strings[0]}${name.toUpperCase()}`;
10 }
11 console.log(tag`Hello ${ person }`); // Hello WORLD
```

Шаблонные строки

ОБРАТНЫЕ КАВЫЧКИ

```
1  const c = `quack${122+1}`;
2
3  const c1 = `qua
4  ck${122+1}`; // quac\nk123
5
6  const person = 'world';
7  function tag(strings, name) {
8      var str0 = strings[0];
9      return `${strings[0]}${name.toUpperCase()}`;
10 }
11 console.log(tag`Hello ${ person }`); // Hello WORLD
```

Шаблонные строки

ОБРАТНЫЕ КАВЫЧКИ

```
1  const c = `quack${122+1}`;
2
3  const c1 = `qua
4  ck${122+1}`; // quac\nk123
5
6  const person = 'world';
7  function tag(strings, name) {
8      var str0 = strings[0];
9      return `${strings[0]}${name.toUpperCase()}`;
10 }
11 console.log(tag`Hello ${ person }`); // Hello WORLD
```

Шаблонные строки

ОБРАТНЫЕ КАВЫЧКИ

```
1  const c = `quack${122+1}`;
2
3  const c1 = `qua
4  ck${122+1}`; // quac\nk123
5
6  const person = 'world';
7  function tag(strings, name) {
8      var str0 = strings[0];
9      return `${strings[0]}${name.toUpperCase()}`;
10 }
11 console.log(tag`Hello ${ person }`); // Hello WORLD
```

Шаблонные строки

Number

```
1 const a = 123;
2 typeof a; // 'number'
3
4 const b = new Number(123);
5 typeof b; // 'object'
6
7 const c = Number('123');
8 typeof c; // 'number'
```

Тип данных позволяет безопасно представлять числа в диапазоне от -(2^53 - 1) до 2^53 - 1

Number

```
1 const a = 123;
2 typeof a; // 'number'
3
4 const b = new Number(123);
5 typeof b; // 'object'
6
7 const c = Number('123');
8 typeof c; // 'number'
```

Тип данных позволяет безопасно представлять числа в диапазоне от -(2^53 - 1) до 2^53 - 1

Number

```
1 const a = 123;
2 typeof a; // 'number'
3
4 const b = new Number(123);
5 typeof b; // 'object'
6
7 const c = Number('123');
8 typeof c; // 'number'
```

Тип данных позволяет безопасно представлять числа в диапазоне от -(2^53 - 1) до 2^53 - 1

Number

```
1 const a = 123;
2 typeof a; // 'number'
3
4 const b = new Number(123);
5 typeof b; // 'object'
6
7 const c = Number('123');
8 typeof c; // 'number'
```

Тип данных позволяет безопасно представлять числа в диапазоне от -(2^53 - 1) до 2^53 - 1

NUMBER. СПЕЦИАЛЬНЫЕ ЗНАЧЕНИЯ

- Infinity
- -Infinity
- NaN

```
typeof NaN // "number"

1 + NaN // NaN

isNaN(NaN) // true
```

Вопрос, в чем разница между isNaN и Number.isNaN?

NUMBER. СПЕЦИАЛЬНЫЕ ЗНАЧЕНИЯ

- Infinity
- -Infinity
- NaN

```
typeof NaN // "number"

1 + NaN // NaN

isNaN(NaN) // true
```

Вопрос, в чем разница между isNaN и Number.isNaN?

NUMBER. СПЕЦИАЛЬНЫЕ ЗНАЧЕНИЯ

- Infinity
- -Infinity
- NaN

```
1 typeof NaN // "number"
2
3 1 + NaN // NaN
4
5 isNaN(NaN) // true
```

Вопрос, в чем разница между isNaN и Number.isNaN?

BigInt

Позволяет работать с числами больше чем `2^53-1`

```
1 const a = 1n;
2 typeof a // "bigint"
3 1 + 1n; // Cannot mix BigInt and other types
4
5 const b = BigInt('1');
6 typeof b; // 'bigint'
```

BigInt

Позволяет работать с числами больше чем `2^53-1`

```
1 const a = 1n;
2 typeof a // "bigint"
3 1 + 1n; // Cannot mix BigInt and other types
4
5 const b = BigInt('1');
6 typeof b; // 'bigint'
```

Boolean

Примитив логического типа

```
1 const a = true;
2 typeof a // "boolean"
3
4 const b = new Boolean(1);
5 typeof b; // 'object'
6
7 const c = Boolean(1);
8 typeof c; // 'boolean'
```

Boolean

Примитив логического типа

```
1 const a = true;
2 typeof a // "boolean"
3
4 const b = new Boolean(1);
5 typeof b; // 'object'
6
7 const c = Boolean(1);
8 typeof c; // 'boolean'
```

Boolean

Примитив логического типа

```
1 const a = true;
2 typeof a // "boolean"
3
4 const b = new Boolean(1);
5 typeof b; // 'object'
6
7 const c = Boolean(1);
8 typeof c; // 'boolean'
```

Undefined

Специальный тип значения, который получает переменная/метод/свойство объекта после объявления, но до присвоения значения

```
1 let a;
2 typeof a; // "undefined"
3
4 const test = {};
5 typeof test.derp; // "undefined"
```

Null

Ссылка на несуществующий объект

```
1 const a = document.getElementById('i-never-use-this-id');
2 console.log(a); // null
3 typeof a // "object"
```

Null

Ссылка на несуществующий объект

```
1 const a = document.getElementById('i-never-use-this-id');
2 console.log(a); // null
3 typeof a // "object"
```

Symbol

Тип данных, значения которого всегда уникальны.

```
1 const a = Symbol("foo");
2
3 const b = Symbol("foo");
4
5 a === b; // false
```

Используются как имена свойств в объекте, но об этом позже...

Symbol

Тип данных, значения которого всегда уникальны.

```
1 const a = Symbol("foo");
2
3 const b = Symbol("foo");
4
5 a === b; // false
```

Используются как имена свойств в объекте, но об этом позже...

ОПЕРАТОРЫ

Оператор - команда, которая сообщает о необходимости выполнения некоторых действий

```
1 const a = 'a';
```

Присвоение (=) является оператором

ОПЕРАНДЫ

Операнд – значение, к которому применяется оператор

```
1 5 + 6;
```

Значения 5 и 6 являются операндами

БИНАРНЫЙ ОПЕРАТОР

Оператор называется бинарным, если он применяется к двум операндам

```
1 5 + 6;
```

УНАРНЫЙ ОПЕРАТОР

Оператор называется унарным, если он применяется к одному операнду

```
1 typeof "str";
```

АРИФМЕТИЧЕСКИЕ ОПЕРАТОРЫ

- Сложение +
- Вычитание -
- Инкремент ++
- Декремент --
- Умножение *
- Деление /
- Взятие остатка от деления %
- Возведение в степень **

СЛОЖЕНИЕ

Оператор + может быть бинарным и унарным.

```
5 + 6; // 11
5 + '6'; // '56'

+'6'; // 6
+'foo'; // NaN
```

СЛОЖЕНИЕ

Оператор + может быть бинарным и унарным.

```
5 + 6; // 11
5 + '6'; // '56'

+'6'; // 6
+'foo'; // NaN
```

ВЫЧИТАНИЕ

Оператор `-` также может быть бинарным и унарным.

```
5 - 4; // 1
5 - '4'; // 1

-6; // -6
-'6'; // -6
```

ВЫЧИТАНИЕ

Оператор `-` также может быть бинарным и унарным.

```
5 - 4; // 1
5 - '4'; // 1

-6; // -6
-'6'; // -6
```

ИНКРЕМЕНТ И ДЕКРЕМЕНТ

Существуют постфиксные и префиксные формы данных операторов

```
1 let a = 10;
2 ++a; // 11
3 --a; // 10
4
5 const b = ++a; // b = 11
6 const c = a++; // c = 11 why???
```

ИНКРЕМЕНТ И ДЕКРЕМЕНТ

Существуют постфиксные и префиксные формы данных операторов

```
1 let a = 10;
2 ++a; // 11
3 --a; // 10
4
5 const b = ++a; // b = 11
6 const c = a++; // c = 11 why???
```

ОПЕРАТОРЫ `*, /, %, **`

- Не имеют унарной формы
- Преобразуют операнды к числу
- Пытаются выполнить арифметическую операцию

```
1 10 * 5; // 50
2
3 10 / '5'; // 2
4
5 11 % '5'; // 1
6
7 2 ** '3'; // 8
```

ОПЕРАТОРЫ ПРИСВАИВАНИЯ

Операнду слева от оператора (=) устанавливается значение, которое берётся из правого операнда

```
1 let a = 'a'; // a
2
3 let b = c = 1 + 1 // b и c = 2
```

Запись	Результат
`x += y`	`x = x + y`
`x -= y`	`x = x - y`
`x *= y`	`x = x * y`
`x /= y`	`x = x / y`
`x %= y`	`x = x % y`
`x <<= y`	`x = x << y`
`x >>= y`	`x = x >> y`
`x >>>= y`	`x = x >>> y`
`x &= y`	`x = x & y`

ОПЕРАТОРЫ СРАВНЕНИЯ

Сравнивает операнды и возвращает логическое значение

Типы равенств:

- Строгое (===)
- Нестрогое (==)

НЕСТРОГОЕ РАВЕНСТВО

```
1 1 == '1' // true
2 1 == true // true
```

СТРОГОЕ РАВЕНСТВО

```
1 1 === '1' // false
2 1 === 1 // true
```

СРАВНЕНИЕ СТРОК

Строки сравниваются в лексикографическом порядке

```
1 '10' < '12' // true
2 '100' < '12' // true
```

СРАВНЕНИЕ ОБЪЕКТОВ

Две переменные равны, если содержат ссылку на один и тот же объект

```
1 const a = {};
2 const b = a;
3
4 console.log({} === {}); // false
5 console.log(a === b); // true
```

Запись	Результат
`!=`	`true`, если операнды не равны
`!==`	`true`, если операнды не равны и/или имеют разный тип
`>`	`true`, если операнд слева больше
`>=`	`true`, если операнд слева больше или равен операнду справа
`<`	`true`, если операнд слева меньше
`<=`	`true`, если операнд слева меньше или равен операнду справа

ЛОГИЧЕСКИЕ ОПЕРАТОРЫ

Выполняет операцию над булевыми операндами и позволяет получить новый операнд булевого типа

Оператор	Результат
`x && y`	Логическое И вернет true, если оба операнда true
`x || y`	Логическое ИЛИ вернет true, если один из операндов true
`!x`	Логическое НЕ вернет true, если операнд false

СОКРАЩЕННОЕ ВЫЧИСЛЕНИЕ

Суть: следующий операнд вычисляется только в том случае, если предыдущего недостаточно для вычисления выражения

||

```
1 const a = false;
2 const b = false;
3 const c = 'c';
4 const d = a || b || c || dummy; // d = 'c'
```

Поиск первого истинного значения

||

```
1 const a = false;
2 const b = false;
3 const c = 'c';
4 const d = a || b || c || dummy; // d = 'c'
```

Поиск первого истинного значения

&&

```
1 const a = 'a';
2 const b = false;
3 const c = 'c';
4 const d = a && b && c; // d = false
```

Поиск первого ложного значения

ТЕРНАРНЫЙ ОПЕРАТОР

условие ? выражение1 : выражение2

```
1 5 > 6 ? console.log('5 > 6') : console.log('5 < 6'); // 5 < 6
```

ПРИОРИТЕТ ОПЕРАТОРОВ

Определяет порядок, в котором операторы выполняются.

```
1 let a = 5;
2 6 < a++ * 2; // true
3 // 6 < 6 * 2
4 // 6 < 12
5 // true
```

ИСПОЛЬЗОВАНИЕ СКОБОК

На приоритет можно влиять скобками

```
1 let a = 5;
2 (6 < a++) * 2; // 0
3 // (6 < 6) * 2
4 // false * 12
5 // 0
```

АССОЦИАТИВНОСТЬ

Указывает на последовательность выполнения операторов при отсутствии явных указаний на очерёдность при равном приоритете

ПРИМЕР: ВОЗВЕДЕНИЕ В СТЕПЕНЬ

Порядок: справа-налево

```
2 ** 2 ** 3; // 256
// 2 ** 3 = 8; 2 ** 8 = 256
```

ПРИМЕР: ДЕЛЕНИЕ И УМНОЖЕНИЕ

Порядок: слева-направо

```
10 / 2 * 2; // 10
```

Таблица приоритетов

СПИСОК ОПЕРАНДОВ ДЛЯ ОЗНАКОМЛЕНИЯ

- Логические операторы
- Операторы присваивания
- Операторы сравнения
- Побитовые операторы
- Строковые операторы
- Арифметические операторы
- Условный/Тернарый оператор
- delete
- function
- in
- instanceof
- new
- this
- typeof
- void
- Аксессоры
- Запятая

Источник

ОБЬЕКТ КАК ТИП ДАННЫХ

Совокупность свойств и методов для работы с ними

```
1 const duck = {
2     sound: 'quack',
3     makeSomeNoise: function() {
4         return this.sound;
5     }
6 }
7
8 duck.makeSomeNoise(); // 'quack'
```

Массивы и функции в JS являются обьектами.
Также в наличии объекты-обертки для примитивов (String, Number и т.д.)

СОЗДАНИЕ ОБЪЕКТОВ

```
1  const duck = {
2      sound: 'quack'
3  };
4
5  function Duck() {
6      this.sound = 'quack';
7  }
8  const another_duck = new Duck(); // Duck {sound: "quack"}
9
10 const and_another_duck = Object.create(duck);
```

СОЗДАНИЕ ОБЪЕКТОВ

```
1  const duck = {
2      sound: 'quack'
3  };
4
5  function Duck() {
6      this.sound = 'quack';
7  }
8  const another_duck = new Duck(); // Duck {sound: "quack"}
9
10 const and_another_duck = Object.create(duck);
```

СОЗДАНИЕ ОБЪЕКТОВ

```
1  const duck = {
2      sound: 'quack'
3  };
4
5  function Duck() {
6      this.sound = 'quack';
7  }
8  const another_duck = new Duck(); // Duck {sound: "quack"}
9
10 const and_another_duck = Object.create(duck);
```

СВОЙСТВА ОБЪЕКТОВ

```
1 const duck = {
2     sound: 'quack'
3 };
4
5 duck.sound === duck['sound']; // true
6
7 const property = 'wings';
8 duck[property] = 2;
```

Имена свойств могут быть строки, или тем, что может быть сконвертировано в строку. Также именем поля может быть притив Symbol.

СВОЙСТВА ОБЪЕКТОВ

```
1 const duck = {
2     sound: 'quack'
3 };
4
5 duck.sound === duck['sound']; // true
6
7 const property = 'wings';
8 duck[property] = 2;
```

Имена свойств могут быть строки, или тем, что может быть сконвертировано в строку. Также именем поля может быть притив Symbol.

ОТЛИЧИЯ ОБЪЕКТОВ ОТ ПРИМИТИВОВ

Объекты изменяемы (mutable)

```
1 const duck = { sound: 'quack' };
2 duck.swim = function() {};
```

ОТЛИЧИЯ ОБЪЕКТОВ ОТ ПРИМИТИВОВ

Объекты изменяемы (mutable)

```
1 const duck = { sound: 'quack' };
2 duck.swim = function() {};
```

ОТЛИЧИЯ ОБЪЕКТОВ ОТ ПРИМИТИВОВ

Объекты сравниваются по ссылке

```
1 const duck = { sound: 'quack' };
2 duck.swim = function() {};
3 const another_duck = duck;
4 duck === another_duck // true
```

ОТЛИЧИЯ ОБЪЕКТОВ ОТ ПРИМИТИВОВ

Объекты сравниваются по ссылке

```
1 const duck = { sound: 'quack' };
2 duck.swim = function() {};
3 const another_duck = duck;
4 duck === another_duck // true
```

МАССИВЫ

Совокупность элементов, доступ к которым осуществляется по индексу.

```
1 const ducks1 = [1,2,3,4];
2
3 const ducks2 = new Array(1,2,3,4);
4 const ducks3 = new Array(4); //[empty × 4]
```

В отличии от Object, Array содержит методы для операций обхода и изменения массива (push, forEach).

МАССИВЫ

Совокупность элементов, доступ к которым осуществляется по индексу.

```
1 const ducks1 = [1,2,3,4];
2
3 const ducks2 = new Array(1,2,3,4);
4 const ducks3 = new Array(4); //[empty × 4]
```

В отличии от Object, Array содержит методы для операций обхода и изменения массива (push, forEach).

МАССИВЫ

Совокупность элементов, доступ к которым осуществляется по индексу.

```
1 const ducks1 = [1,2,3,4];
2
3 const ducks2 = new Array(1,2,3,4);
4 const ducks3 = new Array(4); //[empty × 4]
```

В отличии от Object, Array содержит методы для операций обхода и изменения массива (push, forEach).

ДОСТУП К ЭЛЕМЕНТАМ

Для получения значения используются квадратные скобки

```
1 const ducks1 = [1,2,3,4];
2 ducks1[0] //1
3 ducks1.1 // Uncaught SyntaxError: Unexpected number
```

ДОСТУП К ЭЛЕМЕНТАМ

Для получения значения используются квадратные скобки

```
1 const ducks1 = [1,2,3,4];
2 ducks1[0] //1
3 ducks1.1 // Uncaught SyntaxError: Unexpected number
```

LENGTH

Некоторые функции (pop, push) изменяют длинну, однако на length можно повлиять иначе.

```
1 const ducks = [1,2,3,4];
2
3 ducks[10] = 10 // [1, 2, 3, 4, empty × 6, 10], length - 11
4
5 ducks.length = 12
6 // [1, 2, 3, 4, empty × 6, 10, empty], length - 12
7
8 ducks.length = 0 // []
```

Вывод: длинная массива не отражает количество реальных значений в массиве

LENGTH

Некоторые функции (pop, push) изменяют длинну, однако на length можно повлиять иначе.

```
1 const ducks = [1,2,3,4];
2
3 ducks[10] = 10 // [1, 2, 3, 4, empty × 6, 10], length - 11
4
5 ducks.length = 12
6 // [1, 2, 3, 4, empty × 6, 10, empty], length - 12
7
8 ducks.length = 0 // []
```

Вывод: длинная массива не отражает количество реальных значений в массиве

LENGTH

Некоторые функции (pop, push) изменяют длинну, однако на length можно повлиять иначе.

```
1 const ducks = [1,2,3,4];
2
3 ducks[10] = 10 // [1, 2, 3, 4, empty × 6, 10], length - 11
4
5 ducks.length = 12
6 // [1, 2, 3, 4, empty × 6, 10, empty], length - 12
7
8 ducks.length = 0 // []
```

Вывод: длинная массива не отражает количество реальных значений в массиве

LENGTH

Некоторые функции (pop, push) изменяют длинну, однако на length можно повлиять иначе.

```
1 const ducks = [1,2,3,4];
2
3 ducks[10] = 10 // [1, 2, 3, 4, empty × 6, 10], length - 11
4
5 ducks.length = 12
6 // [1, 2, 3, 4, empty × 6, 10, empty], length - 12
7
8 ducks.length = 0 // []
```

Вывод: длинная массива не отражает количество реальных значений в массиве

НЕМНОГО О TYPED ARRAYS

Typed arrays - массиво-подобные объекты, которые предоставляют механизм доступа к двоичным данным.

Использование: действия с файлами, обработка изображений, WebGL (графика) и т.д.

ПРИМЕР `typed arrays`

В реализации выделяют:

- Буфер (`ArrayBuffer`)
- Представление данных (`Int8`, `Uint32`, `Float64` и т.д.)

```
1 const buffer = new ArrayBuffer(16); // буфер в 16 байт
2 const int8View = new Int8Array(buffer);
3
4 for (let i = 0; i < int8View.length; i++) {
5     int8View[i] = i; // заполняем буфер 8-ми битными целыми чис
6 }
7 console.log(int8View);
```

Больше информации тут

ПРИМЕР `typed arrays`

В реализации выделяют:

- Буфер (`ArrayBuffer`)
- Представление данных (`Int8`, `Uint32`, `Float64` и т.д.)

```
1 const buffer = new ArrayBuffer(16); // буфер в 16 байт
2 const int8View = new Int8Array(buffer);
3
4 for (let i = 0; i < int8View.length; i++) {
5     int8View[i] = i; // заполняем буфер 8-ми битными целыми чис
6 }
7 console.log(int8View);
```

Больше информации тут

ИСТОЧНИКИ

- MDM web docs (link)
- learn.javascript.ru
- Цикл статей на хабре (link)
- Секреты JavaScript ниндзя (link)
- JavaScript Garden (link)
- Стандарт