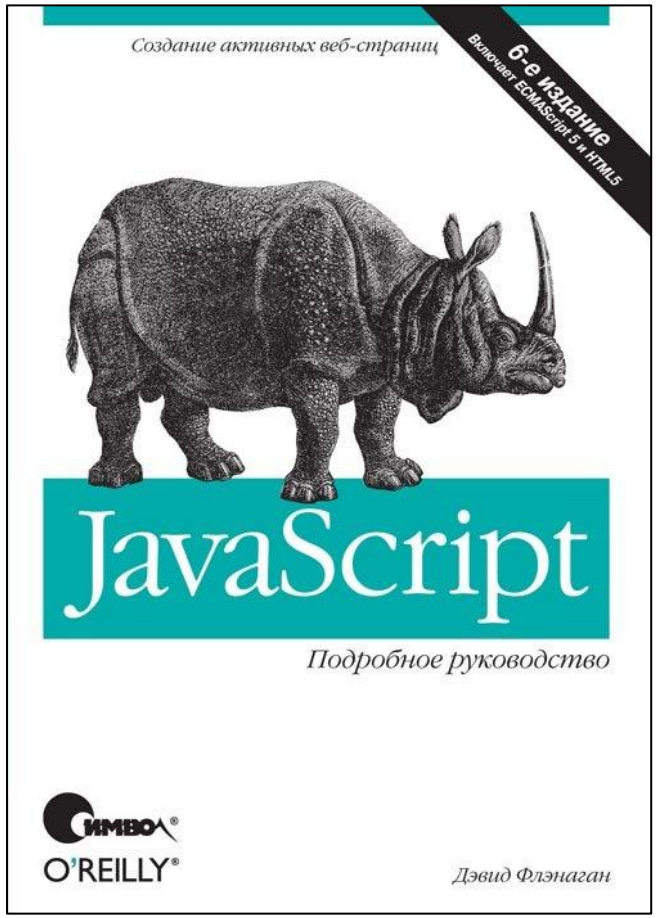
Интернет-технологии и распределённая обработка данных

Лекция 10

Общая характеристика JavaScript

1. История JavaScript
2. Лексическая структура скрипта JavaScript, строгий режим
3. Идентификаторы и ключевые слова
4. Тип данных в JavaScript – общее описание
5. Литералы
6. Преобразование типов

Книги


Создание активных веб-страниц
6-е издание
Включает ECMAScript 5 и HTML5
JavaScript
Подробное руководство
O'REILLY
Дэвид Флэнаган


для профессионалов
JavaScript
для профессиональных
веб-разработчиков
Николас Закас


Используй лучшее в JavaScript
JavaScript
сильные стороны
Дуглас Крокфорд
O'REILLY
YAHOO! PRESS
ПИТЕР

Книги


THE PRINCIPLES OF
OBJECT-ORIENTED
JAVASCRIPT
NICHOLAS C. ZAKAS
Создание веб-приложений с быстрым интерфейсом
Оптимизация
JavaScript
производительности
СИМВОЛ
O'REILLY
YAHOO! PRESS
Николас Закас
O'REILLY
Speaking
JavaScript
AN IN-DEPTH GUIDE FOR PROGRAMMERS
Dr. Axel Rauschm
Шаблоны программирования и проектирования
высококачественных приложений
JavaScript
Шаблоны
СИМВОЛ
O'REILLY
YAHOO! PRESS
Стоян Стефанов
Effective Software Development Series
Scott Meyers, Consulting Editor
Effective
JAVASCRIPT
68 Specific Ways to Harness the Power of JavaScript
David Herman
Foreword by Brendan Eich

Ссылки

1. https://learn.javascript.ru/ – учебник
2. http://es5.javascript.ru/index.html – стандарт ECMAScript 5 на русском языке
3. http://javascript.ru/manual – справочник по JavaScript
   http://www.ecma-international.org/ecma-262/7.0/index.html – стандарт ECMAScript 7
4. https://github.com/uprock/javascript – советы по стилю написания кода
5. http://jslint.com/ – проверка корректности кода
6. http://code.tutsplus.com/articles/resources-for-staying-on-top-of-javascript--cms-21369

Что такое «JavaScript»?

JavaScript – прототипно-ориентированный сценарный язык программирования.

Обычно используется как встраиваемый язык для программного доступа к объектам приложений.

Наиболее широкое применение находит в браузерах для придания интерактивности веб-страницам.

История JavaScript

1995 – Брендан Айк (Brendan Eich) в Netscape создаёт встроенный скриптовый языка для браузера Netscape Navigator.
Сотрудничество: Netscape Communications (Марк Андрессен), Sun Microsystems (Билл Джой)
Основа: функциональные языки, ООП, синтаксис C, автоматическое управление памятью.

09.1995 – **LiveScript** (NN 2.0 beta)

12.1995 – **JavaScript** (NN 2.0B3). В мае 1995 появился язык Java, и это было модное слово!

История JavaScript

- 07.1996 – **JScript** (Internet Explorer 3.0) – аналог **JavaScript** (Microsoft) . Клиентские скрипты + автоматизация администрирования систем Microsoft Windows, создание страниц ASP .

Первым браузером, поддерживающим эту реализацию, был Internet Explorer 3.0.

История JavaScript

В конце 1996 года Ecma International стандартизирует JavaScript.

06.1997 – первая редакция: стандартизированная версия ECMA-262, язык **ECMAScript**,

Ей соответствуют JavaScript версии 1.1, а также языки JScript и ScriptEasy.

История JavaScript

Дальше дело было так:

1. Периодически выходили новые редакции ECMA-262 (новые версии ECMAScript).

2. Компании выпускали новые версии браузеров – так появлялись новые версии JavaScript, JScript,...

Конечно, эти версии опирались на стандарт! Но всё-таки немного отличались от него. Их принято называть *диалектами* ECMAScript.

№	Дата
1	06.1997
2	06.1998
3	12.1999
4	Не вышла
5	12.2009
5.1	**06.2011**
6	06.2015
7	06.2016
8	06.2017

Редакции ECMAScript

Поддержка версий в браузерах
http://kangax.github.io/compat-table/es2016plus/

Поддержка ECMAScript 6 в Mozilla
https://developer.mozilla.org/ru/docs/Web/JavaScript/New_in_JavaScript/ECMAScript_6_support_in_Mozilla

ES3, ES5, ES6, ES7, ES8, ES2015, ES2016, ES2017, ES2018, ES2019, ECMAScript 2015, ECMAScript 2016, ECMAScript 2017, ECMAScript 2018, ECMAScript 2019

Движок JavaScript

Движок JavaScript (JavaScript engine) – виртуальная машина, транслирующая и выполняющая JavaScript-код (как правило, в браузере).

Популярные движки: V8, SpiderMonkey, Chakra, Nitro, JavaScriptCore.

Бесплатные игровые движки на HTML5 и JavaScript

https://tproger.ru/digest/free-game-engines-js/

SpiderMonkey

Особенности: исторически первый, открытый код, содержит интерпретатор JS в байт-код, JIT-компилятор этого байт-кода (IonMonkey), сборщик мусора.

Реализует: **JavaScript 1.8.5** (ECMAScript 5.1 + некоторые возможности ECMAScript 6, 7).

Используется: Firefox, Adobe Acrobat.

V8

Особенности: открытый код, компиляция JavaScript в машинный код, эффективная сборка мусора. Очень быстрый! 70% рынка

Реализует: **JavaScript** (ECMAScript 5.1 + некоторые возможности ECMAScript 6, 7).

Используется: Chromium, Chrome, Opera, Maxthon, Яндекс браузер и других.

Название – мощный двигатель внутреннего сгорания под названием V8.

Chakra

Особенности: JIT-компиляция в параллельном потоке, сборщик мусора.

Реализует: **JScript 9.0** (ECMAScript 5).

Используется: Internet Explorer 9.

Nitro (JavaScriptCore)

Особенности: прямая компиляция в машинный код (ранее – интерпретатор и JIT-компилятор).

Реализует: **ECMAScript** (ECMAScript 5).

Используется: Apple Safari.

Выводы по движкам

С точки зрения синтаксиса языка и API – у каждого движка свои (мелкие) особенности.

Интерпретация или компиляция – зависит от движка (чаще всего используется гибридный подход).

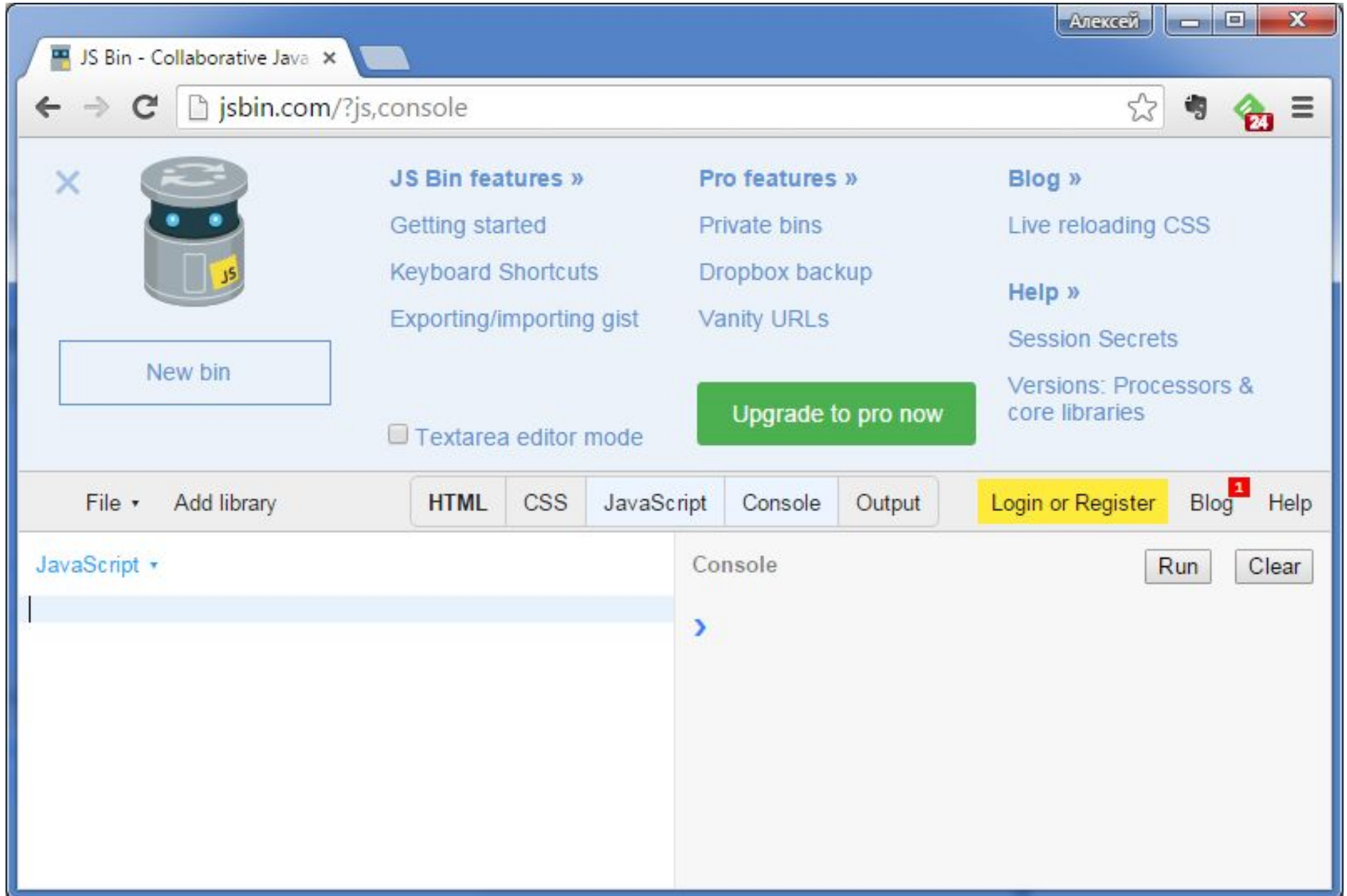
ECMAScript 5 работает в Firefox 4, Chrome 19, Safari 6, Opera 12.10 и Internet Explorer 10.

IDE для JavaScript

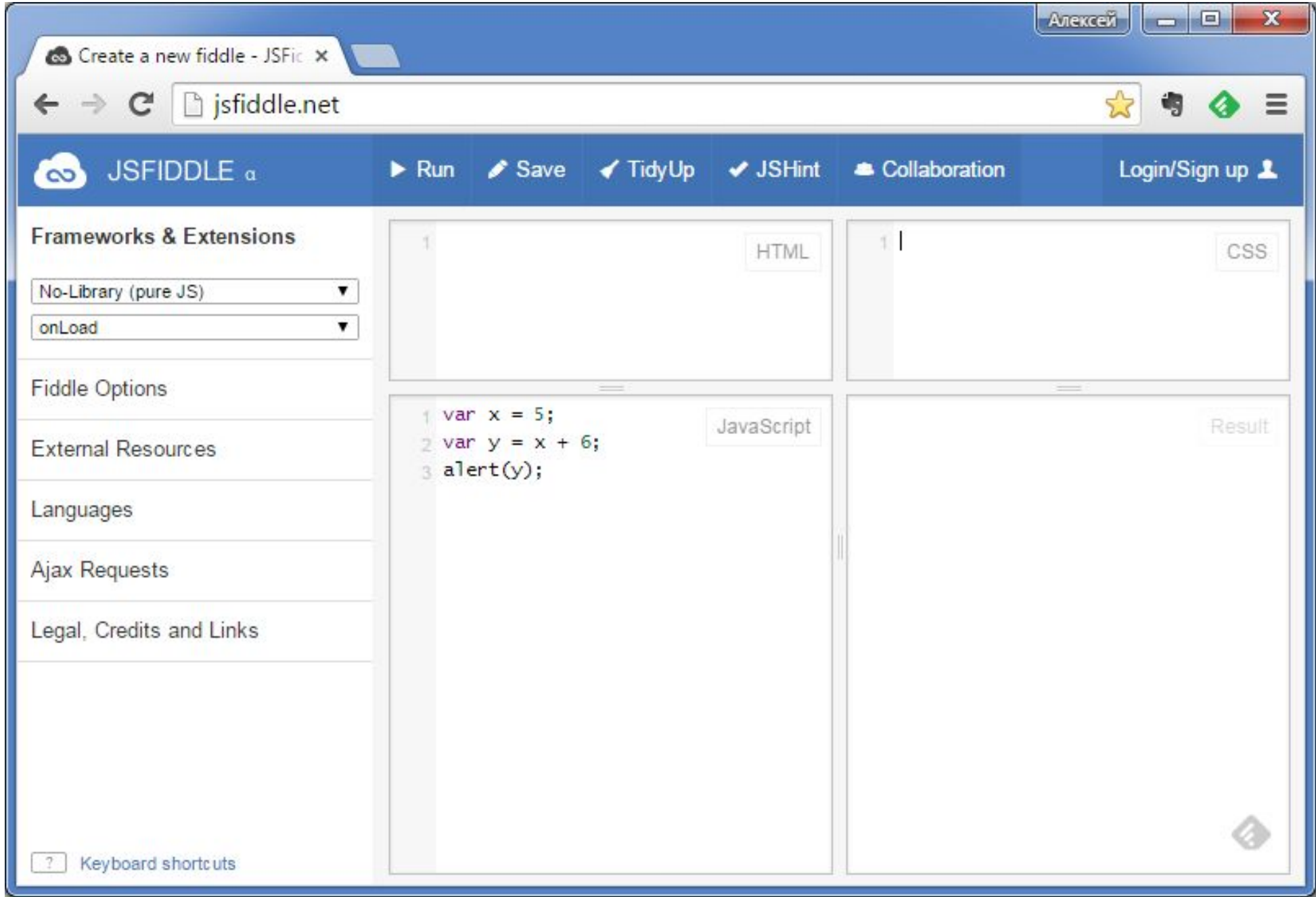
1. Notepad++
2. Sublime Text 3 (http://www.sublimetext.com/3)
3. Visual Studio 2015
4. WebStorm 10
5. Online IDE (Cloud9, jsbin.com, jsfiddle.net)

https://habrahabr.ru/post/159999/ Koding.com, CodePen, Dabblet, Pastebin.me, CSSDesk, jsdo.it, Tinker, Tinkerbin, CSSDeck

jsbin.com


Алексей
JS Bin - Collaborative Java
jsbin.com/?js,console
JS Bin features »
Getting started
Keyboard Shortcuts
Exporting/importing gist
Pro features »
Private bins
Dropbox backup
Vanity URLs
Blog »
Live reloading CSS
Help »
Session Secrets
Versions: Processors & core libraries
New bin
Textarea editor mode
Upgrade to pro now
File
Add library
HTML
CSS
JavaScript
Console
Output
Login or Register
Blog
Help
JavaScript
Console
Run
Clear

jsfiddle.net



```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Document</title>
</head>
<body>
   <p>Начало документа...</p>
  <script>
    alert( 'Привет, Мир!' );
  </script>
  <p>...Конец документа</p>
</body>
</html>
```

Подключение JavaScript:

1) вставить в любое место HTML при помощи тега **script**

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Document</title>
</head>
<body>
 <p>Начало документа...</p>
   <script src="scripts/example.js"></script>
  <p>...Конец документа</p>
</body>
</html>
```

2) JavaScript-код вынести в отдельный файл, который подключается в HTML (абс. путь или полный URL)

scripts\example.js

```
var x = 10;
alert(x);
```

подключить несколько скриптов

```
<script src="/js/script1.js"></script>
<script src="/js/script2.js"></script> ...
```

Если указан атрибут `src`, то содержимое тега игнорируется.

```
<script src="file.js">
alert(1);
// указан src – внутренняя часть тега игнорируется
</script>
```

<script type="text/javascript"> ... </script>

Атрибут async

кроме IE9-.

при обнаружении <script async src="..."> браузер не останавливает обработку страницы . Когда скрипт будет загружен – он выполнится.
Первым выполнится тот, который раньше загрузится.

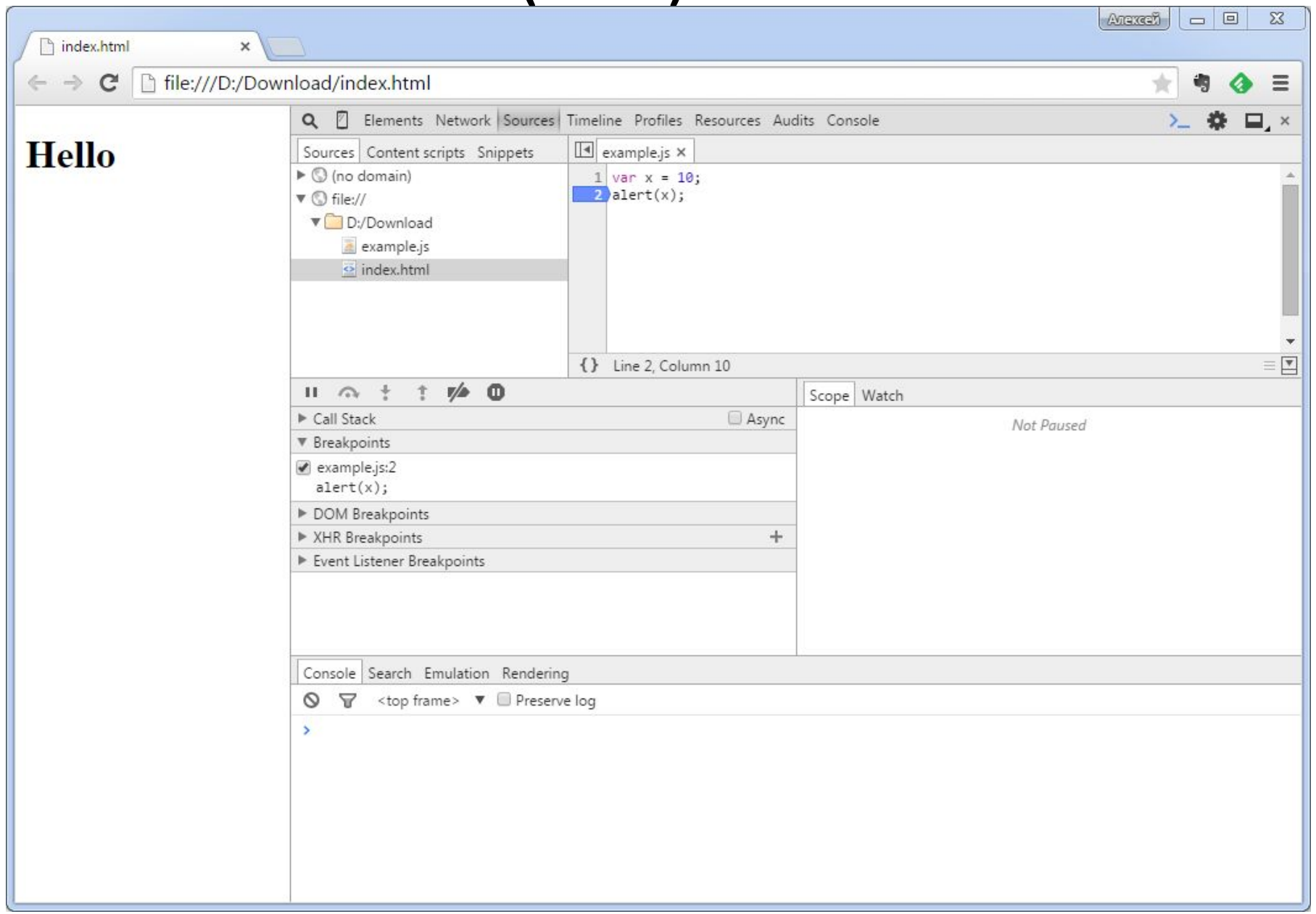
```
<script src="1.js" async></script>
<script src="2.js" async></script>
```

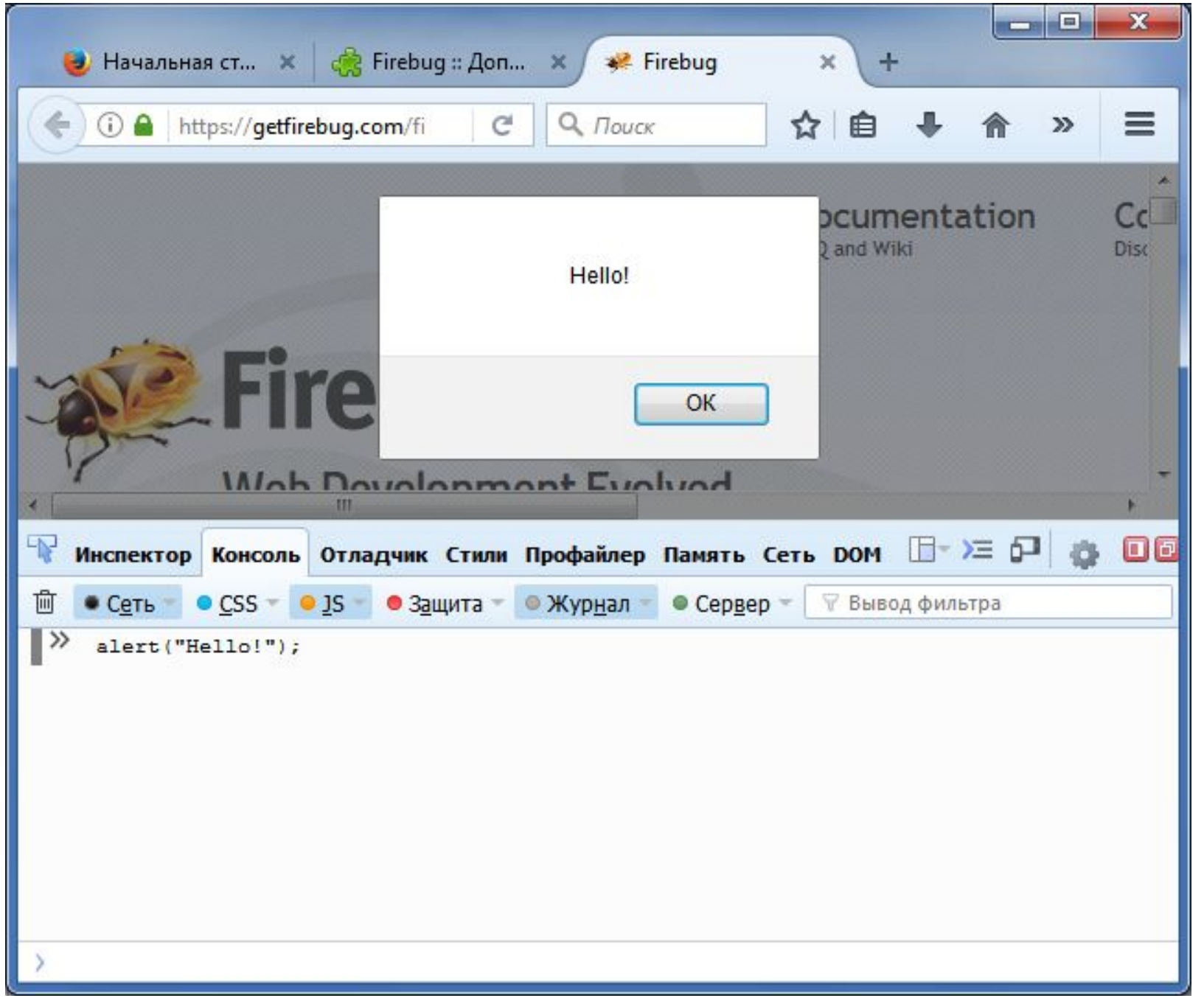
Атрибут defer

выполняется после обработки браузером всего документа HTML, относительный порядок скриптов сохраняется.

```
<script src="1.js" defer></script>
<script src="2.js" defer></script>
```

Отладка – Chrome DevTools (F12)




Начальная ст...
Firebug :: Доп...
Firebug
https://getfirebug.com/fi
Поиск
Hello!
OK
Инспектор
Консоль
Отладчик
Стили
Профайлер
Память
Сеть
DOM
Сеть
CSS
JS
Защита
Журнал
Сервер
Вывод фильтра
alert("Hello!");

Лексическая структура

Лексическая структура языка программирования – набора **элементарных правил**, определяющих как пишутся программы на этом языке.

Примечание: программы на JavaScript будем по традиции называть *скриптами*.

Для записи скриптов используются символы Unicode

Чувствительность к регистру

Исходный код – набор *инструкций* и *комментариев*

Комментарии бывают *строчные* и *блочные*

Пробелы и пробельные символы игнорируются

Комментарии игнорируются

```
// это значит "переменная" (на китайском языке)
var 变量 = 10;

/*
  пробелы, табуляция - игнорируются
*/
      alert(变量);
```

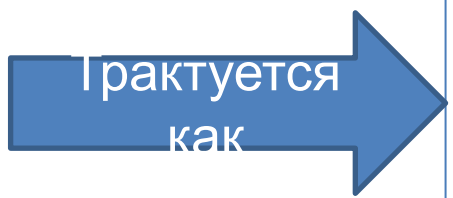
Для отделения инструкций используется *точка с запятой*.

JavaScript трактует *переход на новую строку* как точку с запятой:

- сразу после ключевых слов `return`, `break`, `continue`
- перед операторами ++ и --
- если следующий непробельный символ не может быть интерпретирован как продолжение текущей инструкции

```
var x
x
=
1
alert(x)


return
true
```




```
var x;
x = 1;


alert(x);


return;
true;
```

Вывод: **всегда** используйте точку с запятой для разделения инструкций.

Строгий режим

В ECMAScript 5 появился «строгий режим» (strict mode).

Программирование в строгом режиме накладывает **ряд ограничений**, чтобы оградить программиста от опасных частей языка (сформированных в процессе его развития) и снизить вероятность ошибки.

Включение строгого режима

`"use strict";` (или `'use strict';`)

В первой строке скрипта – действует на весь скрипт.

В первой строке функции – действует внутри функции.

Не в первой строке – ни на что не влияет.
Нет директивы возврата в старый режим
Проблемы: IE9-, библиотеки без учета "use strict";

Идентификаторы в JavaScript

Состоят из:

букв,

цифр,

символов _ (подчеркивание) и **$** (доллар)

Первый символ не должен быть цифрой.

Идентификаторы в JavaScript

myFunction OK
K1 OK
$hello OK
变量 OK
$ OK
_ OK

apple
AppLE

1abc BAD
Ab*cd BAD
my-name BAD

Правила именования

- Только английский (не транслит) myGoods, price, link
- короткие имена только для переменных «местного значения»
- переменные из нескольких слов пишутся вместеВотТак: borderLeftWidth
- имя переменной должно максимально чётко соответствовать хранимым в ней данным
- если ищем переменную с одним именем, а находим – с другим, то переименовать переменную, чтобы имя было тем, которое искали.

Ключевое слово

Ключевое слово – идентификатор, зарезервированный для нужд языка

То есть, мы **не можем** использовать ключевое слово как имя для своих нужд

например: `var`, `class`, `return`, `export` и др.

Ключевые слова в ECMAScript

Список ключевых слов включает следующие категории:

- **Зафиксированы в текущем стандарте**
- Зарезервированы для использования в будущем
- Были ключевыми в предыдущих версиях стандарта
- Являются ключевыми только в строгом режиме

Настоящими ключевые слова – только в первой категории, но остальное не рекомендуется

abstract	arguments	boolean	break	byte
case	catch	char	class	const
continue	debugger	default	delete	do
double	else	enum	eval	export
extends	false	final	finally	float
for	function	goto	if	implements
import	in	instanceof	int	interface
let	long	native	new	null
package	private	protected	public	return
short	static	super	switch	synchronized
this	throw	throws	transient	true
try	typeof	var	void	volatile
while	with	yield		

Зарезервированы для использования в будущем

Являются ключевыми только в строгом режиме

Были ключевыми в ECMAScript 3

Тип данных в JavaScript

number число

string строка

boolean для хранения булевых значений

null значение null

undefined значение undefined

symbol (ECMAScript 6) символы

object любые объекты

number

Это 64-битовое число с плавающей запятой, сохранённое в формате IEEE 754.

JavaScript определяет две глобальные переменные `Infinity` (положительная бесконечность) и `NaN` (для хранения «не-числа»).

```
alert( 1 / 0 ); // Infinity
alert( "нечисло" * 2 ); // NaN, ошибка
```

Семантика примитивных типов при присваивании (копируется значение, не ссылка).

string

Строка Unicode-символов (UTF16).

Неизменяемый.

Семантика примитивных типов при присваивании одинарные и двойные кавычки равноправны.

В ECMAScript 6 появился тип данных для одиночных символов **symbol**

boolean

Два булевых значения (`true` и `false`).

Семантика примитивных типов при присваивании.

null и undefined

Это два так называемых *тривиальных типа*, ибо представляют по одному значению: тип null представляет значение `null (ничего)`, тип undefined – значение `undefined (переменная объявлена, но в нее ничего не записано)`.

JavaScript определяет глобальную переменную `undefined` для представления значения типа undefined.

```
var age = null; //возраст age неизвестен
```

object

Значениями этого типа являются любые объекты.

Объект представляет собой набор свойств.

Классов нет, *прототипная модель*.

Массивы, функции – это тоже объекты.

Семантика ссылочного типа при присваивании.

```
var user = { name: "Вася" };
```

Оператор typeof

Синтаксис оператора: `typeof x` или `typeof(x)`

- `typeof undefined // "undefined"`
- `typeof 0 // "number«`
- `typeof true // "boolean"`
- `typeof "foo" // "string"`
- `typeof {} // "object«`
- `typeof null // "object"` — официальная ошибка языка, сохр. для совместимости
- `typeof function(){} // "function"` — является подвидом объектов, а не базовым типом

Объявления

- **var** – переменная локальная или глобальная, инициализация переменной значением является необязательной. `var x = 42;`
- **let** – локальная переменная в области видимости блока, инициализация переменной значением является необязательной. `let y = 13;`
- просто присвоить значение `y = 13;` (глобальная, не рекомендуется – `strict mode`)
- **const** – именованная константа, доступная только для чтения.

Присваивание значений

Переменная, объявленная через var или let без присвоения начального значения, – значение undefined.

```
var a; console.log("The value of a is " + a);
//Значение переменной a undefined
var input;    if (input === undefined) { . . .} //true
```

Доступ к необъявленной переменной или переменной до её объявления – исключение ReferenceError:

```
console.log("The value of b is " + b);
```

Что такое «литерал»?

Литерал – последовательность символов в исходном коде скрипта, которая представляет фиксированное значение некоторого типа данных.

Иными словами, литерал – это константа, непосредственно включённая в текст программы.

Почему это важно для JavaScript?

Вот так в JavaScript выглядит обычное объявление и инициализация переменной:

```
var x = "This is a string variable";
```

Единственный способ определить тип x – по литералу справа.

Литерал целого числа

Целые числа могут быть записаны в десятичной, шестнадцатеричной, восьмеричной и двоичной системах счисления.

Десятичные числовые литералы

Целые **десятичные** – используем цифры `0,1,…,9`.
Впереди можно поставить знак + или –

```
var x = 123;
```

не начинать десятичное число с незначащего нуля :

```
var y = 0999; // не самая лучшая идея!
```

Восьмеричные числовые литералы

Целые **восьмеричные** – начинаем с **0** и используем цифры 0,1,…,7. Впереди можно поставить знак + или –

```
var x = 0123;
```

Три настораживающих факта о восьмеричных числах:

- если в числе встретиться 8 или 9 – уже как десятичное
- не допустимы в строгом режиме
- в стандарте указаны как *возможное расширение*

Шестнадцатеричные литералы

Целые **шестнадцатеричные** – начинаем с **`0x`** или **`0X`** и используем цифры `0,1,…,9,a,b,c,d,e,f,A,B,C,D,E,F`. Впереди можно поставить знак + или –

```
var x = 0x123abc;
var y = -0XFFF;
```

Двоичные литералы

Ведущие символы 0b (или 0B) – число двоичное.
включают в себя только цифры 0 и 1.

```
var FLT_SIGNBIT = 0b10000000000000000000000000000000; // 2147483648
var FLT_EXPONENT = 0b01111111100000000000000000000000;
                                                    // 2139095040
var FLT_MANTISSA = 0B00000000011111111111111111111111; // 8388607
```

Числовые литералы

Максимальное целое число, хранимое **точно** = 2^{53}:

```
var biggestInt = 9007199254740992;

// попробуйте вот так - будет забавно:
var biggestIntAndOne = 9007199254740993;
alert(biggestIntAndOne);
```

Литералы вещественных чисел

Синтаксическая форма литерала **вещественных чисел** (используются десятичные *цифры*):

[*цифры*][.*цифры*][(E|e)[(+|-)]*цифры*]

Впереди можно указать знак + или – (но только если следом за ним не точка)

```
var x = 3.1415926;  // OK
var y = -.33333333;  // BAD!
var z = 4.08e12;      // OK
```

Строковые литералы

Последовательность Unicode-символов в парных одинарных или двойных кавычках:

```
var st1 = "Normal";
var st2 = 'Also normal';
var oneChar = "A";
var empty = "";
```

Управляющие символы внутри строки

`\0` Символ NUL

`\b` «Возврат»

`\t` Горизонтальная табуляция

`\n` Перевод строки

`\v` Вертикальная табуляция

`\f` Перевод или прогон страницы

`\r` Возврат каретки

`\"` Двойная кавычка

`\'` Одинарная кавычка

`\\` Обратный слэш

`\xXX` Символ Latin-1, заданный двумя шестнадцатеричными цифрами XX от 00 до FF. Например, \xA9 (символ ©).

`\uXXXX` Символ Unicode, четыре шестнадцатеричных цифры XXXX

кавычки

```
var quote = "He read \"The Cremation of Sam McGee\" by R.W.
Service.";console.log(quote);
// He read "The Cremation of Sam McGee" by R.W. Service.
```

косая черта

```
var home = "c:\\temp";
// c:\temp
```

перевод строки

```
var poem = "Roses are red,\n\
Violets are blue.\n\
I'm schizophrenic,\n\
And so am I."
```

Обратный слэш

Если после него записан «неожиданный» символ, то обратный слэш игнорируется:

```
var x = "\A\L\E\X";  //не рекомендуется
alert(x); // выведет ALEX
```

Литералы для boolean и null

`true false` Два литерала для типа boolean

`null` Один возможный литерал для типа null

Литералы регулярных выражений

Текст между парой символов слэша – *литерал регулярного выражения*. За вторым слэшем может следовать один или более символов, которые модифицируют поведение шаблона:

```
/ab+c/i;
```
//компиляция регулярного выражения при вычислении выражения. Рек., если регулярное выражение неизменно – цикл

```
new RegExp('ab+c', 'i');
```
//компиляция регулярного выражения во время выполнения. . Рек., если шаблон регулярного выражения будет меняться или если вы не знаете шаблон

Литералы регулярных выражений

Отдельного типа для регулярных выражений нет!

При применении литерала создаётся объект, используя конструктор `RegExp()`:

```
// полная форма – запись через конструктор
var expr = new RegExp("pattern", "flags");

// сокращенная форма – литеральная запись
var expr = /pattern/flags;
```

pattern – текст регулярного выражения.
flags – если определён, может принимать любую комбинацию нижеследующих значений:
g – глобальное сопоставление
i -- игнорирование регистра при сопоставлении
m – сопоставление по нескольким строкам; символы начала и конца (^ и $) начинают работать по нескольким строкам (то есть, происходит сопоставление с началом или концом *каждой* строки (строки разделяются символами \n или \r), а не только с началом или концом всей вводимой строки)
y – «липкий» поиск; сопоставление в целевой строке начинается с индекса, на который указывает свойство lastIndex этого регулярного выражения (и не пытается сопоставиться с любого более позднего индекса).

https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/

Следующий скрипт использует метод replace() экземпляра строки String для сопоставления с именем в формате *имя фамилия* и выводит его в формате *фамилия, имя*. В тесте замены скрипт использует заменители $1 и $2, которые заменяются на результаты соответствующих сопоставившихся подгрупп регулярного выражения.

```
var re = /(\w+)\s(\w+)/; //(x) – сопоставляется с x и запоминает
var str = 'John Smith';
var newstr = str.replace(re, '$2, $1');    //$ сопоставляется с концом ввода
console.log(newstr);
 // пример с русскими буквами
var re = /([а-яё]+)\s([а-яё]+)/i;  //[xyz] – набор символов
var str = 'Джон Смит';
var newstr = str.replace(re, '$2, $1');
console.log(newstr);
```

Пример выведет «Smith, John» и «Смит, Джон»

Литералы массивов и объектов

«литералы массивов» и «литералы объектов» – это не литералы (согласно строгой грамматике JavaScript)!

Ибо могут содержать внутри выражения.

Правильное их название: **выражения-инициализаторы**.

Список из нуля или более выражений, каждое из которых представляет элемент массива, заключенный в квадратные скобки ([])

Литералы массивов

инициализируется с помощью переданных значений, которые будут являться его элементами, длина массива будет равна числу переданных аргументов:

```
var a = [];              // пустой массив
var b = [1, 2+3];        // два элемента
var c = [1, "Alex", []];  // три элемента
```

Элементы массива можно «пропускать» (`undefined`):

```
var d = [1,,,4,5];
```

запятая в конце списка элементов игнорируется

```
var myList = ['home', 'school', ]; // ["home", undefined, "school"]
```

Литерал объектов

список из нуля или более пар, состоящих из имен свойств и связанных с ними значений, заключенный в фигурные скобки ({}).

Не использовать литерал объекта в начале выражения, т.к. это приведет к ошибке или к неожиданному поведению – "{" интерпретируется как начало блока.

```
var car = { myCar: "Saturn", getCar: CarTypes("Honda"), special: Sales };
```

свойству myCar объекта car – строка "Saturn",

свойству getCar — результат вызова функции CarTypes("Honda"),
свойству special — значение переменной Sales:

можно использовать числовой или строковой литералы в именах свойств или вкладывать один объект в другой:

```
var car = { manyCars: {a: "Saab", "b": "Jeep"}, 7: "Mazda" };

alert(car.manyCars.b); // Jeep
alert(car[7]); // Mazda
```

Именем свойства объекта может быть любая строка (пустая тоже). Если имя свойства не корректный идентификатор – в кавычки. Для обращения [ ] , а не точку (.):

```
var unusualPropertyNames = { "" : "An empty string",
                             "!": "Bang!" }
alert(unusualPropertyNames."");     // SyntaxError: Unexpected string
alert (unusualPropertyNames[""]);  // "An empty string"
alert (unusualPropertyNames.!);     // SyntaxError: Unexpected token !
alert(unusualPropertyNames["!"]); // "Bang!"
```

Преобразование типов

не нужно указывать тип данных переменной, когда вы ее объявляете, типы данных преобразуются автоматически по мере необходимости во время выполнения скрипта

```
var answer = 42;
answer = "Thanks for all the fish...";
x = "The answer is " + 42 // "The answer is 42"
y = 42 + " is the answer" // "42 is the answer"
"37" - 7 // 30
"37" + 7 // "377"
```

некоторые операции в JavaScript требуют операндов определённых типов.

Пример: операция . («точка», доступ к свойству объекта) требует в качестве операнда объект.

JavaScript может преобразовывать один тип в другой.

Преобразование null, undefined, boolean

	В строку	В число	В boolean	В объект
`undefined`	`"undefined"`	`NaN`	`false`	ошибка `TypeError`
`null`	`"null"`	`0`	`false`	ошибка `TypeError`
`true`	`"true"`	`1`		`new Boolean(true)`
`false`	`"false"`	`0`		`new Boolean(false)`

```
var myArray = [];  if ( !myArray[0] ) myFunction(); //  undefined в false
var  a;  a + 2; // NaN
var n = null;  console.log(n * 32); // В консоль выведется 0
```

Преобразование строк

	В число	В boolean	В объект
`""` (пустая строка)	`0`	`false`	`new String("")`
`"1.2"` (непустая строка, число)	`1.2`	`true`	`new String("1.2")`
`"one"` (непустая строка, не число)	`NaN`	`true`	`new String("one")`

Преобразование чисел

	В строку	В boolean	В объект
`0`	`"0"`	`false`	`new Number(0)`
`-0`	`"0"`	`false`	`new Number(-0)`
`NaN`	`"NaN"`	`false`	`new Number(NaN)`
`Infinity`	`"Infinity"`	`true`	`new Number(Infinity)`
`-Infinity`	`"-Infinity"`	`true`	`new Number(-Infinity)`
`1` (конечное, ненулевое)	`"1"`	`true`	`new Number(1)`

Преобразование массивов

	В строку	В число	В boolean
`[]` (пустой массив)	`""`	`0`	`true`
`[8]` (один элемент, у которого строковое представление преобразуется в число)	`"8"`	`8`	`true`
`["A"]` (любой другой массив)	метод `join()`	`NaN`	`true`

Преобразование объектов в строку

A. Есть `toString()`, который возвращает не объект => преобразовать результат в строку.

Иначе

B. Есть `valueOf()`, который возвращает не объект => преобразовать результат в строку.

Иначе

C. Ошибка `TypeError`.

Преобразование объектов в число

A. Есть `valueOf()`, который возвращает не объект => преобразовать результат в число.

Иначе

B. Есть `toString()`, который возвращает не объект => преобразовать результат в число.

Иначе

C. Ошибка `TypeError`.

Преобразование объектов в boolean

Всегда возвращает `true`.

Неявные и явные преобразования

Для явного преобразования в простые типы используются следующие функции: Boolean(), Number(), String(). При неявном преобразования интерпретатор использует те же функции, что используются для явного преобразования.

Строковое преобразование

```
var a = true;
alert( a ); // "true "

alert( String(null) === "null" ); // true

alert( true + "test" ); // "truetest"
alert( "123" + undefined ); // "123undefined"
```

Функция String() преобразует значения по следующим правилам:

- Для всех значений кроме null и undefined автоматически вызывается метод toString() и возвращается строковое представление значения.
- Для значения null возвращается строка "null".
- Для значения undefined возвращается строка "undefined".

метод `.toString(основание_системы_счисления)`
может использоваться, чтобы конвертировать числа:

```
(255).toString(16)
'ff'

(4).toString(2)
'100'

(8).toString(8)
'10'
```

Численное преобразование

происходит в математических функциях и выражениях, а также при сравнении данных различных типов (кроме сравнений ===, !==).

Альтернативный способ – "унарный плюс":

```
"1.1" + "1.1"       // "1.11.1 "
(+"1.1") + (+"1.1") // 2.2
//  скобки не обязательны, используются для ясности
var a = +"123"; // 123
var a = Number("123"); // 123, тот же эффект
```

parseInt(numString, [radix])

//radix – основание системы счисления

parseFloat(numString)

```
var access = parseInt("11000", 2)
    // переводит строку с двоичной записью числа в число.
var access2 = n.toString(2)
// получает для числа n запись в 2-ной системе в виде строки.
```

Функция Number() преобразует значения по следующим правилам:

- Логические значения true и false –
  в 1 и 0 соответственно.
- Числа – без изменения.
- Значение null в 0.
- Значение undefined в NaN.
- Пробельные символы по краям обрезаются.
  Далее, если остаётся пустая строка, то 0,
- иначе из непустой строки "считывается" число, при ошибке результат NaN.

Логическое преобразование

Преобразование к true/false :

if(value),

при применении логических операторов.

Все значения, которые интуитивно «пусты» : 0, пустая строка, null, undefined и NaN становятся false.

Остальное, в том числе и любые объекты – true.

Для явного преобразования :
двойное логическое отрицание !!value ;
вызов Boolean(value).

```
alert( !!"0" ); // true
alert( !!" " ); // любые непустые строки, даже из пробелов - true!
```

Функция Boolean() преобразует значение в его логический эквивалент

значения null/undefined

- К числу при арифметических операциях и сравнениях >,>=,

`<,<= :  undefined в NaN; null в 0`

- при проверке равенства `==  :`
  что `null` и `undefined` равны `"=="` между собой, но эти значения не равны никакому другому значению

null не подчиняется законам математики – он «больше либо равен нулю»: null>=0, но не больше и не равен

```
alert( null >= 0 ); //true, т.к. null преобразуется к 0
alert( null > 0 ); // false (не больше), т.к. null преобразуется к 0
alert( null == 0 ); // false (и не равен!), т.к. == рассматривает null особо.
```

Значение undefined вообще «несравнимо»:

```
alert( undefined > 0 ); // false, т.к. undefined -> NaN
alert( undefined == 0 ); // false, т.к. это undefined (без преобразования)
alert( undefined < 0 ); // false, т.к. undefined -> NaN
```

Преобразование простых типов в объекты

Для преобразования простых значений в объекты используются конструкторы Boolean(), Number(), String():

```
var oNum = new Number(3);
var oStr = new String("1.2");
var oBool = new Boolean(true);

alert(typeof oNum);   // "object"
alert(typeof oStr);   // "object"
alert(typeof oBool);  // "object"
```

Преобразование объектов в простые значения

Все объекты наследуют два метода преобразования: toString() и valueOf().

Метод toString() возвращает строковое представление объекта.

```
alert({x: 1}.toString());   // "[object Object]«
```

метод toString() у массива преобразует все его элементы в строки и затем объединяет их в одну строку, вставляя запятые между ними:

```
alert([1,2,3].toString());   // "1,2,3"
```

Преобразование объектов в простые значения

метод valueOf() должен преобразовать объект в представляющее его простое значение, если такое значение существует.

Объекты по своей сути являются составными значениями, и большинство объектов не могут быть представлены в виде единственного простого значения, поэтому по умолчанию метод valueOf()возвращает не простое значение, а ссылку на него:

```
alert(typeof {x:2}.valueOf());   // "object"
```