



# JavaScript

## Basics

**JavaScript** — это кросс-платформенный, объектно-ориентированный скриптовый язык, являющийся небольшим и легковесным. Внутри среды исполнения JavaScript может быть связан с объектами данной среды и предоставлять программный контроль над ними.

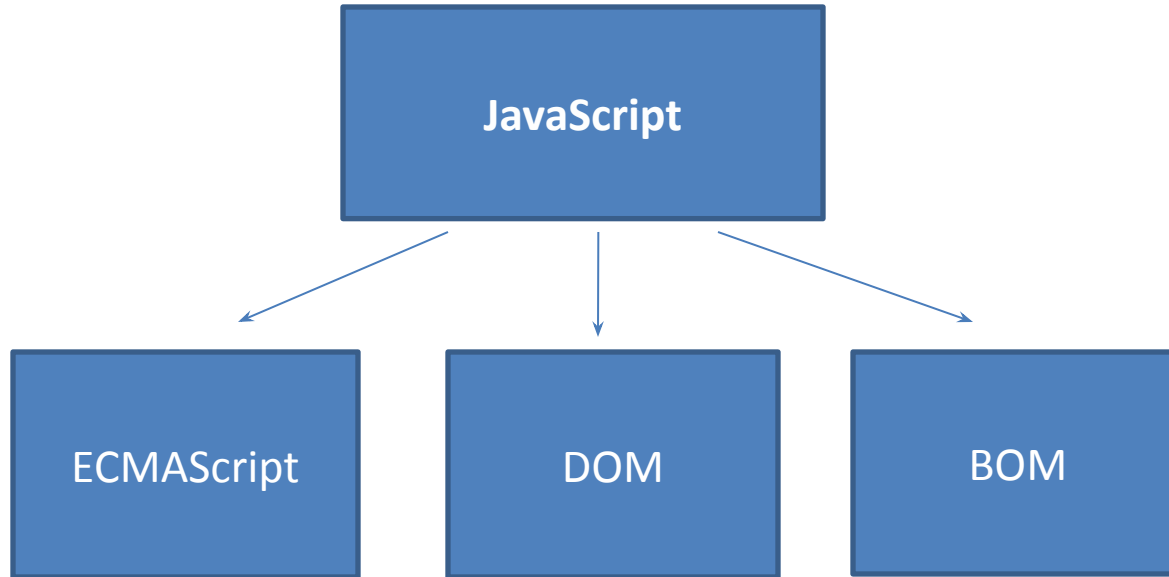
# История JavaScript

- 1995 г. Брендан Айк Netscape LiveScript
- 1997 - стандартизация JavaScript 1.1
- 1998 - ECMAScript 2
- 1999 - ECMAScript 3
- 2007 - ECMAScript 4 (abandoned)
- 2009 - ECMAScript 5
- 2011 - ECMAScript 5.1
- 2015 - ECMAScript 6.0
- 2016 – ECMAScript 2016 (ECMAScript 7)

# JavaScript движки

- SpiderMonkey (Mozilla)
- Rhino (Mozilla)
- V8 (Chrome)
- JavaScriptCore/Nitro (Safari)

# Структура JavaScript



# JavaScript встроенный в разметку

```
<body>  
  <a href="javascript: alert('script')">Click!</a>  
</body>  
<body>  
  <button onclick="alert('script')">Button</button>  
</body>
```

# Использование тега <script>

```
<script type="text/javascript">  
    //Some code  
</script>
```

```
<script>  
    alert('script');  
</script>
```

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

```
<script src="http://www.somewhere.com/script.js"></script>
```

## Расположение тега <script>

### В заголовке

```
<head>  
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>  
  <title>Заголовок</title>  
  <script src="script.js"></script>  
</head>
```

### В теле документа

```
<body>  
  <div> Тело документа</div>  
  <script src="script.js"></script>  
</body>
```

 Лучше располагать скрипты в конце <body>



# Расположение тега <script>

```
<!DOCTYPE html>
<html>
  <head>
    <!--some code-->
  </head>

  <body>
    <!--some code-->

    <script type="text/javascript" src="1.js"></script>

    <script type="text/javascript" src="2.js"></script>
  </body>
</html>
```

# Объявление переменных. Идентификаторы

```
var message1;  
var message2 = "строка";  
message3 = 10;  
var $message = 1,  
    _message10 = 3;  
var a = {}
```

# Ключевые и зарезервированные слова

break	do	instanceof	typeof		
case	else	new	var		ES3 Keywords
catch	finally	return	void		
continue	for	switch	while		
debugger	function	this	with		
default	if	throw			
delete	in	try			
abstract	enum	import	protected	transient	
boolean	export	int	public	volatile	
byte	extends	interface	short		
char	final	long	static		
class	float	native	super		ES3 Reserved words
const	goto	package	synchronized		
double	implements	private	throws		
class	enum	extends	super		
const	export	import			
implements	package	public			ES5 Reserved words
interface	private	static			
let	protected	yield			

// однострочный комментарий

/\*

\* Это многострочный

\* комментарий

\*/

## 'use strict';

Differences from non-strict to strict:

- function f(a, b, b){}
- {a: 1, b: 3, a: 7}
- var n = 023;
- with
- function() {b = 1;}
- delete Object.prototype

## ОПЕРАТОРЫ

# Операторы.

Оператор	Назначение
-	Изменение знака на противоположный
+	Унарный +
!	Дополнение. Используется для реверсирования значения логических переменных
++	Увеличение значения переменной. Может применяться как префикс переменной или как ее суффикс
--	Уменьшение значения переменной. Может применяться как префикс переменной или как ее суффикс

# Операторы. Арифметические операторы

Оператор

-

+

\*

/

%

Назначение

Вычитание

Сложение

Умножение

Деление

Вычисление остатка от деления



# Операторы. Битовые операторы

Оператор	Логическая операция
&	И
	ИЛИ
^	ИСКЛЮЧАЮЩЕЕ ИЛИ
~	НЕ
>>	Сдвиг в правую сторону
<<	Сдвиг в левую сторону
>>>	Сдвиг в правую сторону с заполнением освобождаемых разрядов нулями

# Операторы. Операторы сравнения

Оператор	Условие
>	Больше
>=	Больше или равно
<	Меньше
<=	Меньше или равно
==	Равно ('1' == 1) - true
!=	Не равно ('1' != 1) - false
===	Идентично ('1' === 1) - false
!==	Не идентично ('1' !== 1) - true

# Операторы. Логические операторы.

Оператор	Описание
	Оператор ИЛИ.
&&	Оператор И.

# Операторы. Логические операторы.

- Логические операторы применяются не только к логическим переменным

10 && 15 = 15

15 && 10 = 10

10 && 0 = 0

10 || 15 = 10

15 || 10 = 15

10 || 0 = 10

"строка1" && "строка2" = "строка2"

"строка1" && 10 = 10

10 && null = null

10 || false = 10

null || "строка2" = "строка2"

"строка2" || 10 = "строка2"

# Операторы. Операторы присваивания.

Оператор	Описание
=	Простое присваивание
+=	Увеличение численного значения или слияние строк
-=	Уменьшение численного значения
*=	Умножение
/=	Деление
%=	Вычисление остатка от деления
>>=	Сдвиг вправо
>>>=	Сдвиг вправо с заполнением освобождаемых разрядов нулями
<<=	Сдвиг влево
=	ИЛИ
&=	И
^=	ИСКЛЮЧАЮЩЕЕ ИЛИ

# Приведение типов

Если при сложении хоть один оператор строка, то результат тоже будет строкой.

$10 + 10 + "14" = "2014"$

Если присутствует минус, то результат число

$"10" + 10 + 10 - 1 = 101009$

Или NaN

$"blue" - 2 = NaN$

# Приведение типов

```
undefined + 123 // NaN
```

```
null + 1 //1
```

```
true + 5 //6
```

```
80 * "7" //560
```

## Логические преобразования

- Пустая строка, 0, NaN, null и undefined приводятся к false
- Всё остальное приводится к true

```
if ("text"){  
  //some code  
}
```



# Сравнение

Выражение	Значение
<code>null == undefined</code>	<code>true</code>
<code>false == 0</code>	<code>true</code>
<code>true == 1</code>	<code>true</code>
<code>null == 0</code>	<code>false</code>
<code>true == 2</code>	<code>false</code>
<code>undefined == 0</code>	<code>false</code>
<code>"5" == 5</code>	<code>true</code>

== и ===

"55" == 55 //true

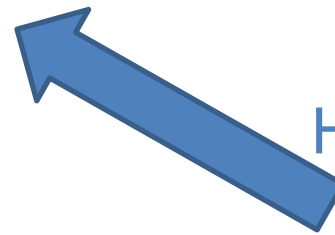
"55" === 55 //false

true == 1 //true

true === 1 // false

# typeof

```
var message;  
alert(typeof message); // undefined  
alert(typeof age); // undefined
```



Необъявленная  
переменная

```
var text = "text";  
alert(typeof text); // string  
var car = null;  
alert(typeof car); // object
```

## ВЫРАЖЕНИЯ

# Выражения. If else

- Выражение `if else` имеет следующий синтаксис:

```
if (condition) statement1 else statement2
```

- Наиболее часто употребляемая конструкция:

```
if (i > 25)
    alert("Greater than 25."); // line statement
else if ((i <= 25) && (i > 15)) {
    alert("Less than 25 and greater than 15."); // block statement
}
else {
    alert("Less than or equal to 15."); // block statement
}
```



**АНТИШАБЛОН**

## BEST PRACTICE



```
if (i > 25) {  
    alert("Greater than 25.");           //block  
}  
else if ((i <= 25) && (i > 15)) {  
    alert("Less than 25 and greater than 15."); //block  
}  
else {  
    alert("Less than or equal to 15."); //block  
}
```

# Выражения. Расположение фигурной скобки



JavaScript фигурная скобка **ВСЕГДА** должна располагаться на строчке сразу после конструкции :

```
if (i > 25) {  
    alert("Greater than 25.");  
}  
else {  
    alert("Less than or equal to 15.");  
}
```

<http://jsfiddle.net/paullasarev/fv6jhupe/>

# Расположение фигурной скобки

```
function func() {  
  return  
  {  
    a: "name"  
  };  
}
```

```
console.log(func());  
// undefined  
console.log(func2());  
// { a: 'name' }
```

```
function func2() {  
  return {  
    a: "name"  
  };  
}
```

[Standard ECMA-262: 7.9 Automatic Semicolon Insertion](#)



# Выражения. do while

- Цикл `do while` имеет следующий синтаксис:

```
do {  
    statement;  
} while (expression);
```

- Пример конструкции:

```
var i = 0;  
do {  
    i += 2;  
} while (i < 10);
```

# Выражения. while

- Цикл `while` имеет следующий синтаксис:

```
while(expression) statement ;
```

- Пример конструкции:

```
var i = 0;  
while (i < 10) {  
    i += 2;  
}
```

# Выражения. for

- Цикл `for` имеет следующий синтаксис:  
`for (initialization; expression; post-loop-expression)  
statement;`

- Пример конструкции:

```
var count = 10;  
for (var i = 0; i < count; i++) {  
    alert(i);  
}
```

- Конструкция без условий:

```
for ( ; ; ) {           //infinite loop  
    doSomething();  
}
```

# Выражения. for in

- Цикл `for in` имеет следующий синтаксис:  
`for (property in expression) statement`
- Пример конструкции:

```
for (var propName in window) {  
    document.write(propName);  
}
```



- Выдаст `error` при переборе свойств в `null` или `undefined`
- Порядок обхода не определен

# Выражения. label

- Выражение `label` имеет следующий синтаксис:  
`label: statement`

- Пример конструкции:

```
start: for (var i = 0; i < count; i++) {  
    alert(i);  
}
```

Используется с операторами `break` и `continue`!



# Выражения. break и continue

- Пример конструкции `break`:

```
var num = 0;
for (var i = 1; i < 10; i++) {
    if (i % 5 == 0) {
        break
    }
    num++;
}
alert(num);
```



Выход из цикла for

← 4

- Пример конструкции `continue`:

```
var num = 0;
for (var i = 1; i < 10; i++) {
    if (i % 5 == 0) {
        continue;
    }
    num++;
}
alert(num);
```



Возврат в начало  
цикла

← 8

# Выражения. break и continue использование label

- Пример конструкции `break` и `continue` с `label` :

```
var num = 0;
outermost:
for (var i=0; i < 10; i++) {
    for (var j=0; j < 10; j++) {
        if (i == 5 && j == 5) {
            break outermost;
        }
        num++;
    }
}
alert(num);
```

 55

# Выражения. switch

- Выражение `switch` имеет следующий синтаксис:

// Используется строгое сравнение `===`

```
switch (expression) {  
    case value: statement;  
        break;  
    case value: statement;  
        break;  
    case value: statement;  
        break;  
    default: statement;  
}
```



- Выражения. switch Пример использования:

```
switch (new Date().getDay()) {  
    case 1:  
    case 2:  
    case 3:  
    case 4:  
    case 5:  
        text = "Soon it is Weekend";  
        break;  
    case 0:  
    case 6:  
        text = "It is Weekend";  
    default:  
        text = "Looking forward to the Weekend";  
        break;  
}
```

- Выражения. switch Пример использования:

```
var num = 25;
switch (true) {
  case num < 0:
    alert("Less than 0.");
    break;
  case num >= 0 && num <= 10:
    alert("Between 0 and 10.");
    break;
  case num > 10 && num <= 20:
    alert("Between 10 and 20.");
    break;
  default:
    alert("More than 20.");
}
```

# Выражения. try catch

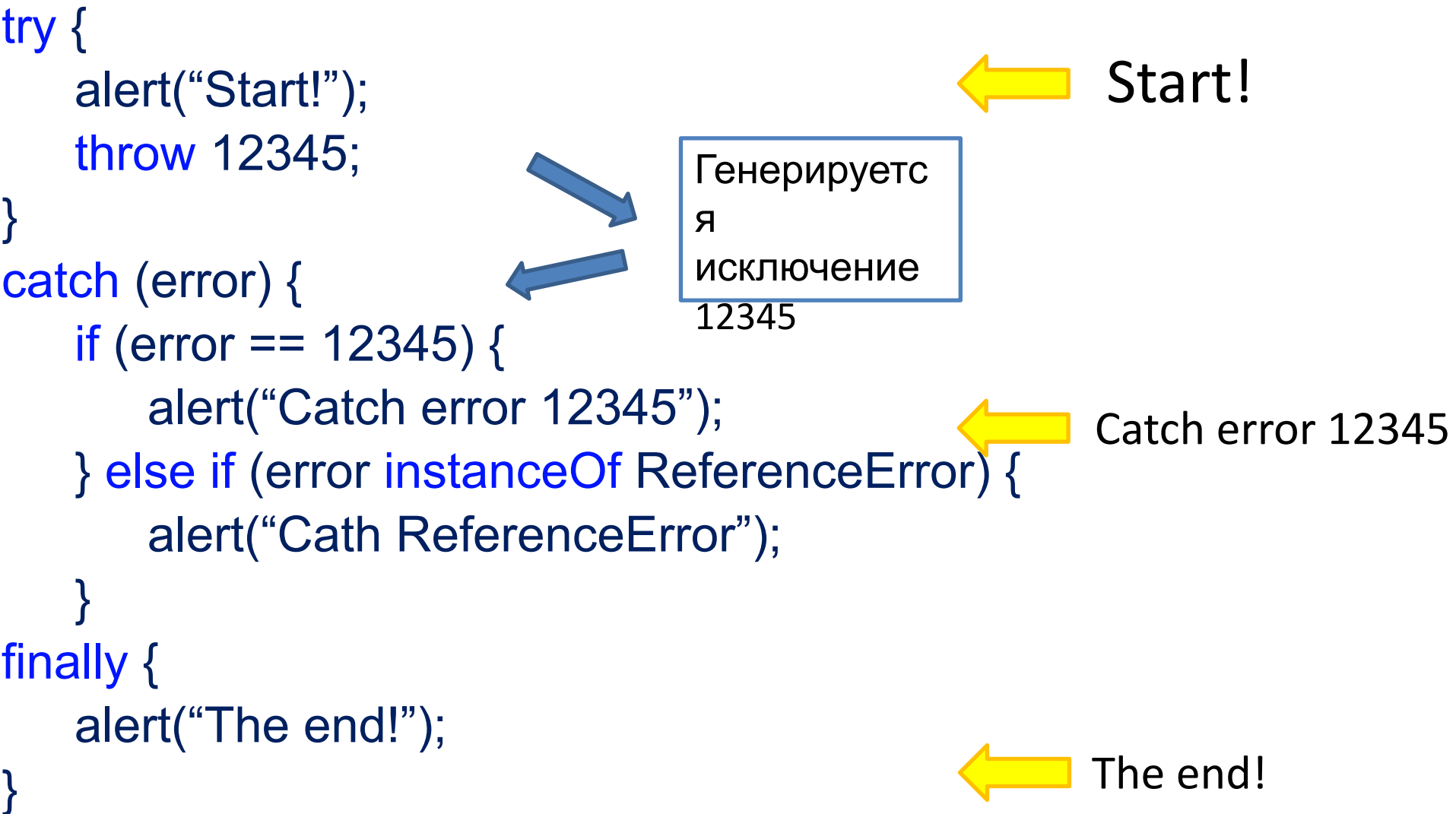
- Конструкция `try catch` имеет следующий синтаксис:

```
try {  
    //code that may  
    //cause an error  
} catch (error) {  
    //what to do when  
    // an error occurs  
} finally {  
    //what to do after  
    // (is always)  
}
```

Пример конструкции:

```
try {  
    throw 12345;  
} catch (error){  
    if (error == 12345){  
        //handle type error  
    } else if (error instanceof ReferenceError){  
        //handle reference error  
    } finally {  
        // Some statement  
    }  
}
```

# Выражения. try catch finally в примерах



# Выражения. scope в выражениях

```
var a = 1;
```

```
if (true) {  
    var b = 2;
```

```
    console.log(a);  
    console.log(b);
```

```
}
```

```
console.log(a);  
console.log(b);
```

# Выражения. scope в выражениях

```
var a = 1;
```

```
if (true) {  
    var b = 2;
```

```
    console.log(a); ←  
    console.log(b); ←
```

```
}
```

```
console.log(a); ←  
console.log(b); ←
```



# Выражения. scope в выражениях

```
var a = 1;
```

```
if (true) {  
    var b = 2;
```



**НЕ  
СОЗДАЕТ  
SCOPE**

```
    console.log(a); ← 1
```

```
    console.log(b); ← 2
```

```
}
```

```
console.log(a); ← 1
```

```
console.log(b); ← 2
```

# Типы данных (Data types)



# Типы данных

## Примитивные:

- Number
- String
- Null
- Undefined
- Boolean

## Ссылочные:

- Array
- Object
- Date
- RegExp
- Function

## ПРИМИТИВНЫЕ ТИПЫ ДАННЫХ

# Примитивные типы данных

- Числа `var a = 1, pi = 3.14;`
- Строки `var str = "string";`
- Булевы значения `var a = true,  
b = false;`
- null `var a = null;`
- undefined `var a = undefined;`

## Особенности:

- передаются по значению
- нельзя добавить ни свойства ни методы
- имеют объекты-обертки

# Примитивные типы данных

## Особенности:

- передаются по значению

```
var e11, e12;
```

```
e11 = 1;
```

```
e12 = e11;
```

```
e11 = 3;
```

```
console.log(e11); ← 3
```

```
console.log(e12); ← 1
```

## Особенности:

- нельзя добавить ни свойства ни методы

```
var el = 5 ;
```

```
el.prop = "Hello World";
```

```
console.log(el.prop); // undefined
```

# Примитивные типы данных

## Особенности:

- имеют объекты-обертки

<http://jsfiddle.net/paullasarev/emcmnos0/2/>

```
> "string".__defineGetter__  
search  
slice  
small  
split  
strike  
sub  
substr  
substring  
sup  
toLocaleLowerCase  
toLocaleString  
toLocaleUpperCase  
toLowerCase  
toString  
toUpperCase  
trim  
trimLeft  
trimRight  
valueOf
```

# Undefined

```
var message; //undefined  
alert(message); //"undefined"
```



# Null

```
var person = null;  
if (person != null){  
    //что-нибудь  
}
```

# Null and undefined

**undefined** - any uninitialized value

**null** - explicitly defined value

According to the ECMAScript 5 spec:

- Both **Null** and **Undefined** are two of the six built in types.

## 4.3.9 undefined value

primitive value used when a variable has not been assigned a value

## 4.3.11 null value

primitive value that represents the intentional absence of any object value

<http://www.ecma-international.org/ecma-262/5.1/>

# Null and undefined

```
console.log(typeof undefined)  
// undefined
```

```
console.log(typeof null)  
// object
```

```
console.log(null === undefined)  
// false
```

```
console.log(null == undefined)  
// true
```