# ВВЕДЕНИЕ В JAVASCRIPT

# BRENDAN EICH

# 1995

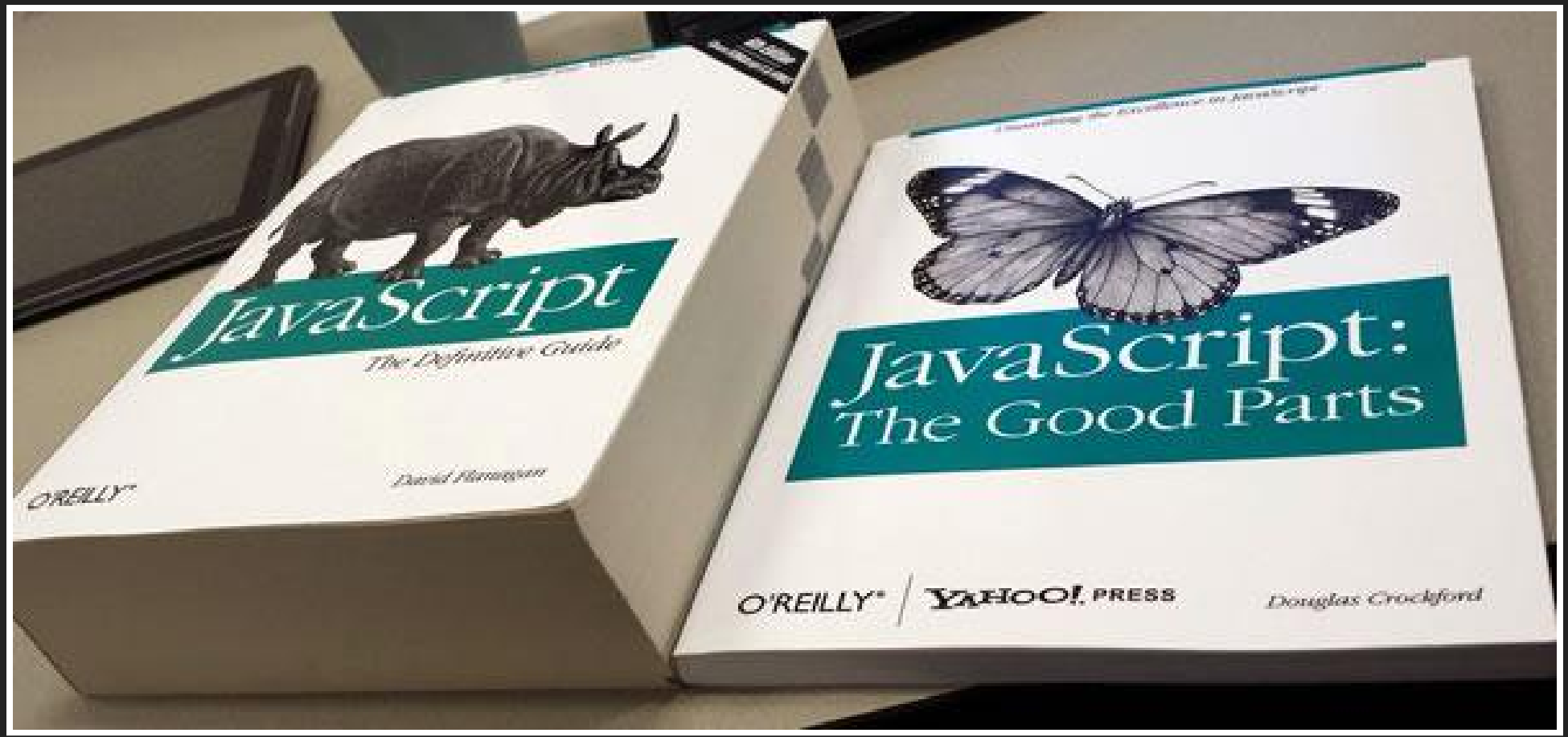
| Scheme | Self | Java |
| --- | --- | --- |
| функции первого класса | прототипы (наследование) | ошибки :) подобный синтаксис |

ECMAScript

# ECMASCRIPT (ECMA-262)

- июнь 1997г.: ECMA-262 1 издание
- июнь 1998г.: ECMA-262 2 издание
- декабрь 1999г.: ECMA-262 3 издание
- ~~июль 2008г.: ECMA-262 4 издание~~
- декабрь 2009г.: ECMA-262 5 издание
- июнь 2011г.: ECMA-262 5.1 издание
- июнь 2015г.: ECMA-262 6 издание ('harmony' или ECMAScript 2015)

http://www.ecma-international.org/ecma-262/6.0/


JavaScript
The Definitive Guide
O'REILLY
JavaScript:
The Good Parts
O'REILLY
YAHOO! PRESS
Douglas Crockford

# TC39
# (ECMA INTERNATIONAL TECHNICAL COMMITTEE 39)

https://github.com/orgs/tc39/people

# ИНСТРУКЦИИ И ВЫРАЖЕНИЯ (STATEMENTS AND EXPRESSIONS)

```
statement;
statement
statement;statement;
```

```
statementstatement
```

# БЛОК ИНСТРУКЦИЙ
# (BLOCK STATEMENT)

```
{
  statement;
  statement;
  statement;
}
```

# ПУСТАЯ ИНСТРУКЦИЯ (EMPTY STATEMENT)

```
;
;;;
```

```
;(function() {
  // код программы
});
```

# ВЫРАЖЕНИЯ

возвращают значения

```
1 + 1
// -> 2
```

Выражениями могут быть:

- Литерал

```
600613;
// -> 600613
```

- Идентификатор

```
levelUp;
// -> "levelUp"
```

- Некоторые ключевые слова

```
this;
// -> {key: "value"}
```

# ОПЕРАТОРЫ

- Унарные
- Бинарные
- Тернарный

- Арифметические
- Логические
- Побитовые (поразрядные)
- Операторы отношения (связей)
- Операторы сравнения
- и т.д.

## Бинарный оператор присваивания

```
a = 5;
a = b = c = 5;
```

## Инструкция с использованием операторов сложения, вычитания и присваивания

```
let a;
let b = 5;
a = 10 + 2 - b;
```

# ПЕРЕМЕННЫЕ И ТИПЫ ДАННЫХ

```
var someVariable;
let anotherVar;
const dontChangeMe = 10;
```

# ИНСТРУКЦИЯ ОБЪЯВЛЕНИЯ (DECLARATION STATEMENT)

```
var someVar;
var anotherVar,
    thirdVar,
    theLastVar;
```

```
// a-zA-Z, _, $, \u
let variable,
    _variable,
    $variable,
    var1,
    \u0061;     // не рекомендуется
```

```
let 1var; // Uncaught SyntaxError: Unexpected number
```

## Чувствительность к регистру (Case sensetivity):

```
sameVar vs SameVar;
```

## Соглашения об именовании (naming conventions):

```
var someVariable,
    VARIABLE,
    __variable__,
    Variable,
    $variable;
```

# СПИСОК ЗАРЕЗЕРВИРОВАННЫХ СЛОВ

```
// используются
default  delete  do  else  export  extends  finally
for  function  if  import  in  instanceof  let  new
return  super  switch  this  throw  try  typeof
var  void  while  with  yield

// не используются
enum  implements  package  protected
static  interface  private  public  await
```

# ИНИЦИАЛИЗАЦИЯ ПЕРЕМЕННЫХ

```
var firstVar = 10;
let firstLet = "some string",
    secondLet = true;
const someConst = 3.14;
```

```
const someConst = 3; // Uncaught SyntaxError: Unexpected number
```

# ТИПЫ ДАННЫХ

- Простые типы (primitives)
- Объектные типы (objects)

# ПРОСТЫЕ ТИПЫ (PRIMITIVES)

- Число (Number)
- Строка (String)
- Логический (Boolean)
- Null
- Undefined
- Символ (Symbol)

# ПРОСТЫЕ ТИПЫ (PRIMITIVES)

- Число (Number)

```
var num = 10,
    decimalNum = 3.14,
    hexadecimal = 0x66aa,
    exponentNum = 1e6;
```

- Строка (String)
- Логический (Boolean)
- Null
- Undefined
- Символ (Symbol)

# ПРОСТЫЕ ТИПЫ (PRIMITIVES)

- Число (Number)
- Строка (String)

```
var double = "some string"
    single = 'some string'
    template = `Hello you`
```

- Логический (Boolean)
- Null
- Undefined
- Символ (Symbol)

# ПРОСТЫЕ ТИПЫ (PRIMITIVES)

- Число (Number)
- Строка (String)
- Логический (Boolean)

```
var boolTrue = true,
    boolFalse = false;
```

- Null
- Undefined
- Символ (Symbol)

# ПРОСТЫЕ ТИПЫ (PRIMITIVES)

- Число (Number)
- Строка (String)
- Логический (Boolean)
- Null

```
var empty = null;
```

- Undefined
- Символ (Symbol)

# ПРОСТЫЕ ТИПЫ (PRIMITIVES)

- Число (Number)
- Строка (String)
- Логический (Boolean)
- Null
- Undefined

```
let declaredVar;
var undef = undefined;
```

- Символ (Symbol)

# ПРОСТЫЕ ТИПЫ (PRIMITIVES)

- Число (Number)
- Строка (String)
- Логический (Boolean)
- Null
- Undefined
- Символ (Symbol)

```
var symb = Symbol();
```

# ОБЪЕКТНЫЕ ТИПЫ (OBJECTS)

- Объект (Object)

```
let obj = {someKey: “value”};
```

- Массив (Array)

```
var arr = [1, 3, 6];
```

- Функция (Function)

```
function someFunc() {}
```

- Регулярное выражение (Regular expression)

```
var regExp = /\d/g;
```

- другие

# ОПЕРАТОР TYPEOF

```
typeof 1              // number
typeof ""             // string
typeof true           // boolean
typeof undefined      // undefined
typeof null           // object
var sym = Symbol();
typeof sym            // symbol
```

# ДИНАМИЧЕСКАЯ ТИПИЗАЦИЯ

— приём, широко используемый в языках программирования и языках спецификации, при котором переменная связывается с типом в момент присваивания значения, а не в момент объявления переменной. Таким образом, в различных участках программы одна и та же переменная может принимать значения разных типов. Примеры языков с динамической типизацией — Smalltalk, Python, Objective-C, Ruby, PHP, Perl, JavaScript, Lisp, xBase, Erlang.

# КОММЕНТАРИИ В КОДЕ

- Строчные

```
// строчный комментар
```

- Блочные

```
/*
Блочный комментарий
на нескольких строках
*/
```

# ПОЛЕЗНЫЕ ССЫЛКИ

- devdocs.io
- developer.mozilla.org
- jstherightway.org
- shamansir.github.io/JavaScript-Garden
- stackoverflow.com
- learn.javascript.ru
- Esprima Parser

# ЗАДАНИЕ

Изучить инструмент отладки javascript кода в браузере Google Chrome

- Использование консоли Google Chrome
- Отладка JavaScript в Chrome Developer Tools. Часть 1 — консоль
- Chrome DevTools
- Debugging JavaScript
- Explore and Master Chrome DevTools (курс на codeschool)