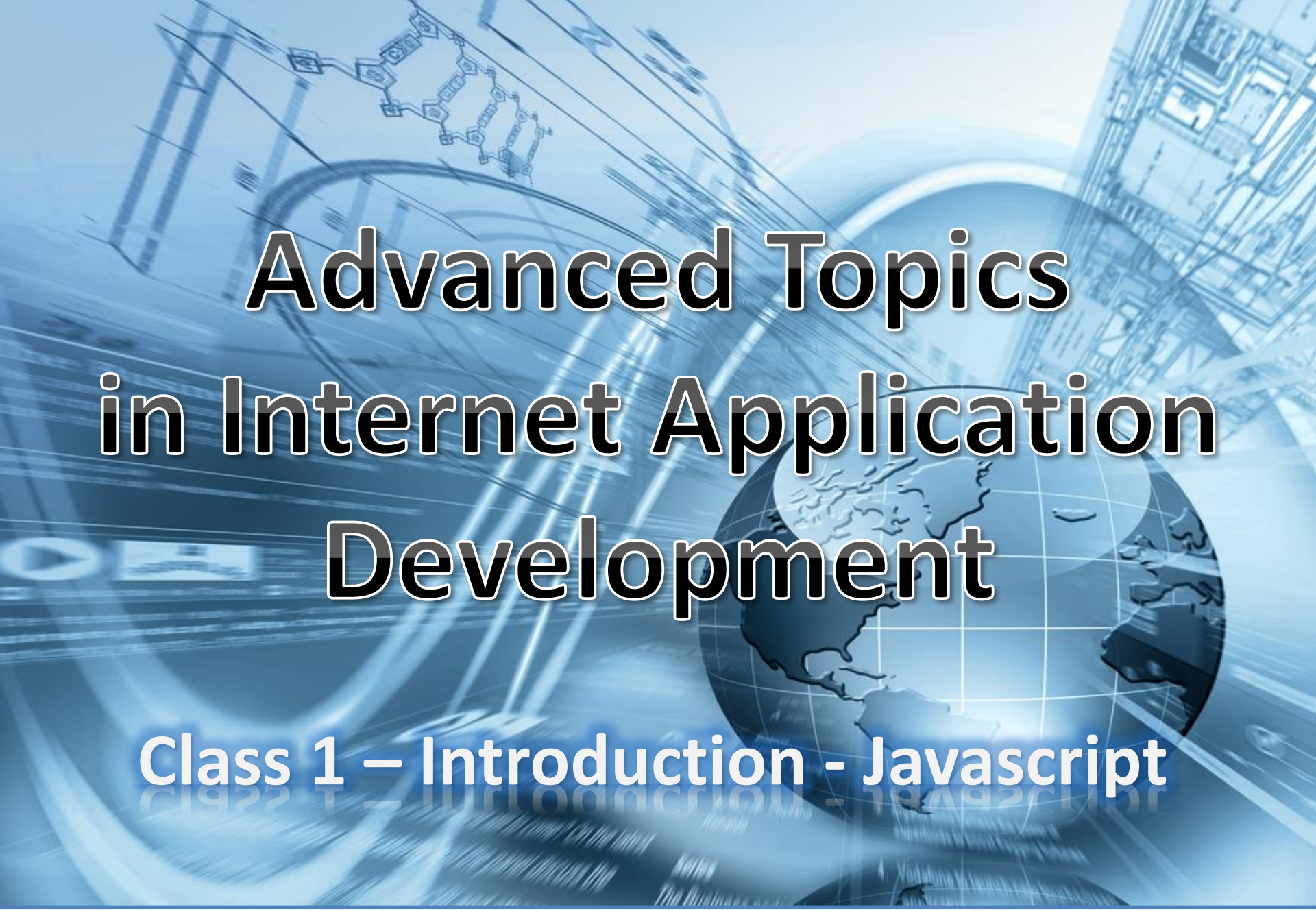


COLMAN CS 2015-B
DR. SHAY HOROVITZ

Today’s lecture

- **Course agenda**
- **Rules**
- **Javascript Recap and new features**

Agenda

- Updates as we go! Order may change.
- Javascript + JQuery + AJAX
- HTML5, CSS3
- MVC (client side) - Angular
- Web APIs / Services
- PHP+MySQL ?
- MEAN Stack (Mongo + Express + Angular + Node.JS)
- WebSockets + WebRTC

Rules!

- **Exercises**
- **Final project**:
  - Submitted in **lesson no' 13**
  - Groups are allowed (3-4 is optimal)

COLMAN CS 2015 B

DR. SHAY HOROVITZ

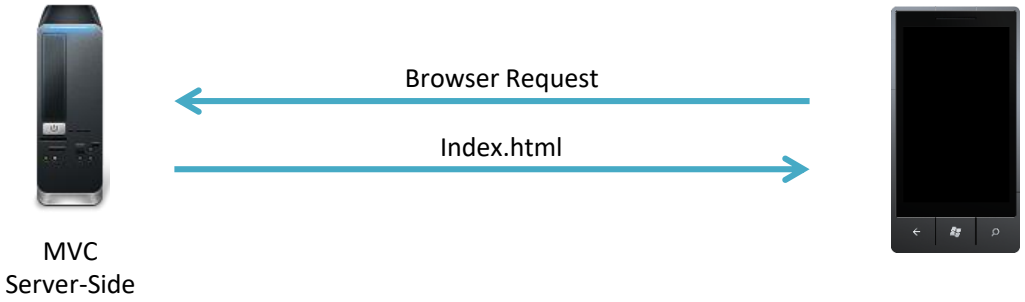
JavaScript Language

COLMAN Modified slides. Original source:

- Eyal Vardi
- CEO E4D Solutions LTD
  Microsoft MVP Visual C#
  site: www.e4d.co.il

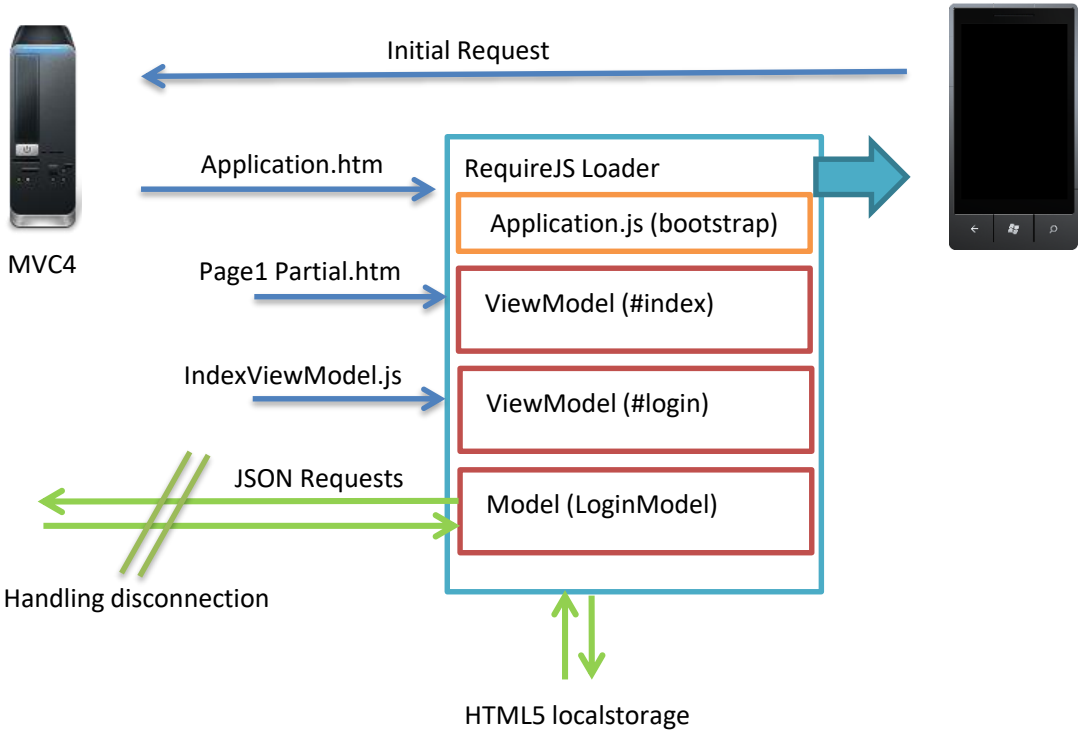
The Challenge

Traditional apps

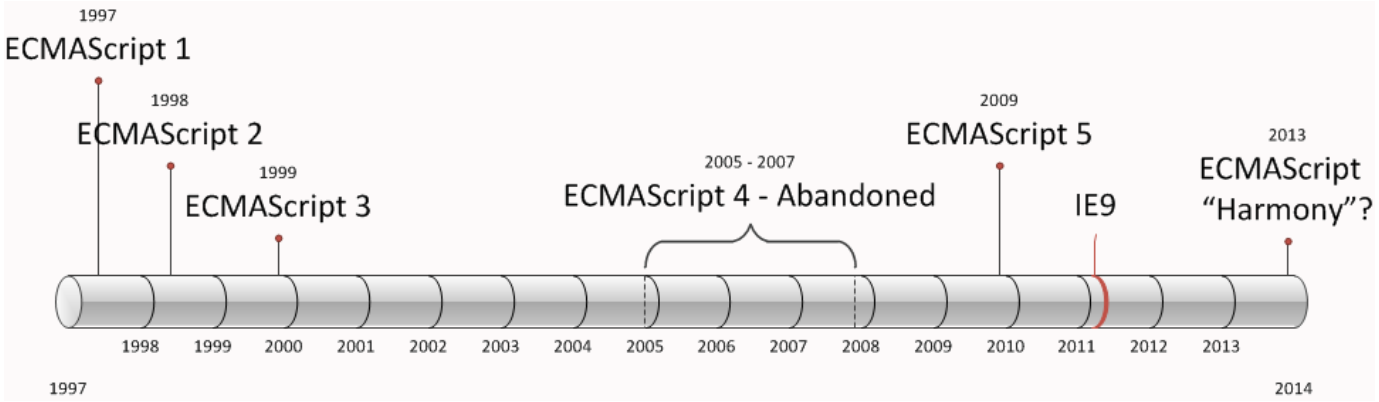


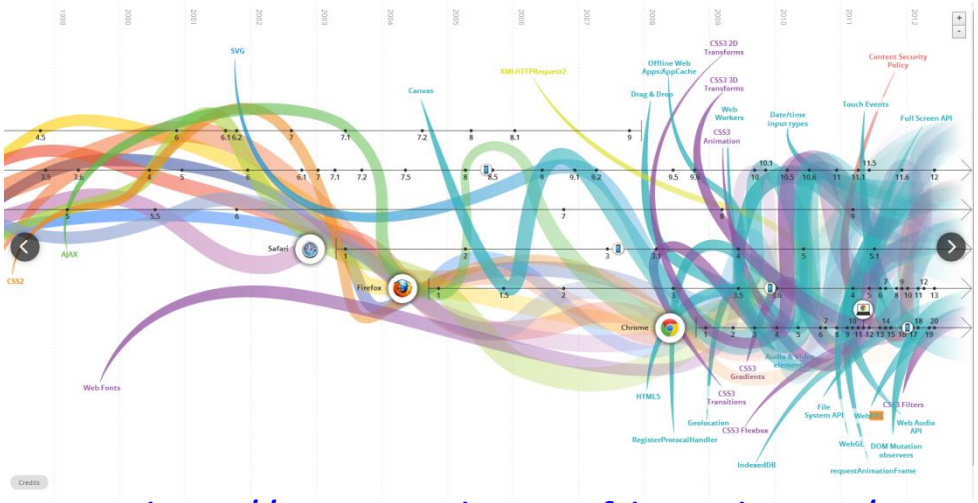
Traditional Request / Response for ALL **rendered** content and assets

Web Application (SPA)



ECMAScript Versions & Web Evolution





http://www.evolutionoftheweb.com/

Syntax

C-style syntax

Similar to C, C++, Java, C#, but with…

```
function everything() {
  var x = {a:10}, y = [1,2], z = function() { };
  for(var p in x) {
    lbl: for(var i = 0; i < 10; i++) {
      switch(x[p]) {
        case 0: continue; default: break lbl;
      }
    }
    if(x instanceof String || 'a' in x) delete x.a;
  }
  while(true) {
    if(true) {
      this.x = 3 / (-this.f()) ? /foo/g : new String("hello")
    } else {
      do { try { throw 3; } catch(e) { debugger; return; } finally {}}
      while(false);
    }  } }
```

for..in

Object, Array, and Function literals

Regular expressions

Automatic semicolon insertion

Case-sensitivity

- Everything is case-sensitive:
  - Variables
  - Function names
  - Operators

Identifiers

- An identifier is the name of a variable, function, property, or function argument.
  - The **first** character must be a **letter**, an **underscore** (_), or a **dollar** sign ($).
  - All other characters may be letters, underscores, dollar signs, or numbers.

- Traditionally, ECMAScript identifiers use camel case (i.e. **myCar**, **doTest**)

Comments

- ECMAScript uses C-style comments for both single-line and block comments.

```
//single line comment

Or

/*
* This is a multi-line
* Comment
*/
```

Statements

- Statements in JS are terminated by a semicolon.

```
//valid – preferred.
var diff = a - b;
```

- Omitting the semicolon makes the parser determine where the end of a statement occurs.

```
//valid even without a semicolon - not recommended.
var sum = a + b
```

Strict Mode

- ECMAScript 5 introduced the concept of strict mode.
- Strict mode helps out in a couple ways:
  - It catches some common coding bloopers, throwing exceptions.
  - It prevents, or throws errors, when relatively “unsafe” actions are taken (such as gaining access to the global object).
  - It disables features that are confusing or poorly thought out.

```
"use strict";

function doSomething(){
    "use strict";
    //function body
}
```

Keywords

Keywords

```
break
case
catch
continue
debugger*
default
delete
do
else
finally
for
function
if
in
Instanceof
new
return
switch
this
throw
try
typeof
var
void
while
with
```

Reserved Words (3 Edition)

```
abstract
boolean
byte
char
class
const
debugger
double
enum
export
extends
final
float
goto
implements
import
int
interface
long
native
package
private
protected
public
short
static
super
synchronized
throws
transient
volatile
```

Reserved Words (5 Edition)

- Strict Mode

```
implements      package      public
interface       private      static
let             protected    yield
```

- None strict Mode

```
class      enum      extends      super
const      export    import
```

Variables

Variables

- Loosely typed

```
var message = "hi";
message = 100; //legal, but not recommended
```

- The **var** operator to define a variable makes it **local to the scope** in which it was defined.

```
function test(){
    var message = "hi"; //local variable
}

test();

alert(message); //error!
```

Global Variable

- Define a variable globally by simply omitting the `var` operator as follows:

```
function test(){
    message = "hi"; // global variable
}

test();

alert(message);     // “hi”
```

Multi Variable

- If you need to define more than one variable, you can do it using a single statement, separating each variable with a comma like this:

```
var message = "hi",
    found   = false,
    age     = 29;
```

Data Types

Data Types

- There are five simple data types (also called primitive types) in ECMAScript:
  - Undefined
  - Null
  - Boolean
  - Number
  - String
- Object (unordered list of name-value pairs)

The typeof Operator

- A typeof operator determine the data type of a given variable.
  - The typeof can't distinguish between objects.

```
typeof undefined          // "undefined"
typeof 0                  // "number"
typeof true               // "boolean"
typeof "foo"              // "string"
typeof {}                 // "object"

typeof null               // "object"
typeof function () { }    // "function"
typeof NaN                // "number"
```

official mistake

Functions are still objects.

Not-A-Number

The Undefined Type

- When a variable is declared using var but not initialized, it is assigned the value of undefined

```
var message;
alert( message == undefined ); //true
alert(message); //"undefined“

//make sure this variable isn’t declared
//var age
alert(age); //causes an error

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

The Null Type

- The Null type is the second data type that has only one value: the special value null.
- A null value is an empty object pointer, which is why typeof returns “object”
- The value undefined is a derivative of null, so ECMA-262 defines them to be superficially equal.

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

alert(null == undefined); //true
```

The Boolean Type

- Note that the Boolean literals true and false are case-sensitive.
- To convert a value into its Boolean equivalent, the special Boolean() casting function is called.
- The if statement, automatically perform this Boolean conversion.

```
var message = "Hello world!";
if (message) {
    alert("Value is true");
}
```

The Number Type

- Number uses the IEEE-754 format to represent both integers and floating-point values.

```
var intNum    = 55;   //integer
var octalNum1 = 070;  //octal for 56
var hexNum1   = 0xA;  //hexadecimal for 10
var floatNum2 = 0.1;
var floatNum3 = .1;   //valid, but not recommended
var floatNum1 = 1.;   // interpreted as integer 1
var floatNum2 = 10.0; //whole number - interpreted as integer 10
```

Floating-Point Values

- Accurate up to 17 decimal places but are far less accurate in arithmetic computations than whole numbers.
  - For instance, adding 0.1 and 0.2 yields 0.30000000000000004 instead of 0.3.

```
if (a + b == 0.3){ //avoid!
    alert("You got 0.3.");
}
```

Range of Values

- Number.MIN_VALUE = 5e-324
- Number.MAX_VALUE = 1.7976931348623157e+308
- Out of range = Infinity

```
var result = Number.MAX_VALUE + Number.MAX_VALUE;
alert( isFinite(result) ); //false
```

- NaN - Not a Number , which is used to indicate when an operation intended to return a number has failed

```
alert( NaN == NaN );          //false
alert( isNaN(NaN) );          //true
alert( isNaN(10) );           //false
alert( isNaN("10") );         //false
alert( isNaN("blue") );       //true
alert( isNaN(true) );         //false
```

Number Conversions

- There are three functions to convert nonnumeric values into numbers:
  - Number() --> Any data type to number
  - parseInt() --> string to number
  - parseFloat() --> string to number

```
var num1 = Number("Hello world!");   //NaN
var num2 = Number("");               //0
var num3 = Number("000011");         //11
var num4 = Number(true);             //1

var num1 = parseInt("1234blue");     //1234
var num2 = parseInt("");             //NaN
var num3 = parseInt("0xA");          //10 - hexadecimal
var num4 = parseInt(22.5);           //22
var num5 = parseInt("70");           //70 - decimal
var num6 = parseInt("0xf");          //15 - hexadecimal
```

The String Type

- Represents a sequence of zero or more 16-bit Unicode characters.

```
var firstName = "Nicholas";
var lastName  = 'Zakas';
```

- Strings are immutable in ECMAScript, meaning that once they are created, their values cannot change.

```
"a string" == "a string"
```

The Object Type

- Everything else, Including functions.
- Each Object instance has the following properties and methods:
  - Constructor
  - hasOwnProperty(propertyName)
  - isPrototypeOf(object)
  - propertyIsEnumerable(propertyName)
  - toLocaleString()
  - toString()
  - valueOf()

Operators

Increment / Decrement

```
var num1 = 2;
var num2 = 20;
var num3 = --num1 + num2;  //equals 21
var num4 = num1   + num2;  //equals 21


var num1 = 2;
var num2 = 20;
var num3 = num1-- + num2;  //equals 22
var num4 = num1   + num2;  //equals 21
```

Unary Plus and Minus

- The unary plus is represented by a single plus sign (+) placed before a variable and does nothing to a numeric value.

```
var num = 25;
num = +num; //still 25
```

- When the **unary plus** is applied to a nonnumeric value, **it performs the same conversion as the Number() casting function**.

Unary Plus and Minus

```
var s1 = "01";
var s2 = "1.1";
var s3 = "z";
var b  = false;
var f  = 1.1;
var o  = {
    valueOf: function() {
        return -1; }
};
s1 = +s1; //value becomes numeric 1
s2 = +s2; //value becomes numeric 1.1
s3 = +s3; //value becomes NaN
b  = +b;  //value becomes numeric 0
f  = +f;  //no change, still 1.1
o  = +o;  //value becomes numeric -1
```

Bitwise Operators

- NOT - ~
- AND - &
- OR - |
- XOR - ^
- Left Shift <<
- Right Shift >>
- Unsigned Right Shift >>>

Bitwise Operators

```
var num = -18;
alert(num.toString(2)); //”-10010”


// Not
var num1 = 25;     //binary 00000000000000000000000011001
var num2 = ~num1; //binary 11111111111111111111111100110
alert(num2);       //-26

// And
var result = 25 & 3;
alert(result); //1

// OR
var result = 25 | 3;
alert(result); //27

// XOR
var result = 25 ^ 3;
alert(result); //26

// Left Shift
var oldValue = 2;
//binary 1000000 which is decimal 64
var newValue = oldValue << 5;
```

Boolean Operators

- Not
- And - &&
- OR - ||

```
alert(!false);  //true
alert(!"blue"); //false
alert(!0);      //true
alert(!NaN);    //true
alert(!"");     //true
alert(!12345);  //false
```

Equality Operators

- Performed conversions into like types before doing a comparison.

```
7 == "7" // true
```

- Not equal because different data types

```
7 === "7" // false
```

Statement

Statements

- If
- Do-while
- For
- For – in
- Labeled statements
- Break & Continue
- With
- Switch

Functions

Function

- First class object
- Can be defined inside other functions.
- Functions stored in objects
- Functions that don't specify a return value actually return the special value undefined.

```
function foo() {}

// expands to
var foo= function foo() {};
```

Arguments

- An ECMAScript function doesn't care how many arguments are passed in, nor does it care about the data types of those arguments.
  - This happens because arguments in ECMAScript are represented as an array internally.
  - An arguments object that can be accessed while inside a function to retrieve the values of each argument that was passed in.

```
function sayHi() {
    alert("Hello " + arguments[0] + ", " + arguments[1] );
}
```

Arguments Example

```
function doAdd() {
    if (arguments.length == 1) {
        alert(arguments[0] + 10);
    } else if (arguments.length == 2) {
        alert(arguments[0] + arguments[1]);
    }
}
doAdd(10);      //20
doAdd(30, 20);  //50
```

No Overloading

- ECMAScript functions cannot be overloaded in the traditional sense.

```
function addSomeNumber(num) {
    return num + 100;
}
function addSomeNumber(num) {
    return num + 200;
}
var result = addSomeNumber(100); //300
```

Function Internals

- Two special objects exist inside a function: **arguments** and **this**.
- the arguments object also has a property named **callee**, which is a pointer to the function that owns the arguments object.

```
function factorial(num) {
    if (num <= 1) {  return 1; }
    else { return num * factorial(num - 1); }
}

function factorial(num) {
    if (num <= 1) { return 1; }
    else { return num * arguments.callee(num - 1); }
}
```

deprecated

this

- Is a reference to the **context object** that the function is operating on.
- A function is called in the **global scope** of a web page, the this object points to **window**.
- The value of **this** is not determined until the function is **called**, so its value may **not be consistent** throughout the code execution.
- ECMAScript 5 also formalizes an additional property on a function object: **caller**.

this Example

```
var name = "Global";

function Demo1() {
    var func1 = function () { return this.name; };
    var ObjA = {
        name: "A",
        getName: function() { return this.name; }
    };
    var ObjB = {
        name: "B",
        getName: function () { return this.name; }
    };

    var a = this;

    alert( func1() );
    alert( ObjA.getName() );
    alert( ObjB.getName() );
}
Demo1();
```

“Global” , “A” , “B”