

JavaScript

a.k.a. ECMAScript



Motivation

- All modern browsers support JavaScript for client-side HTML scripting...
- but it is also used in other environments
 - Microsoft's Active Scripting and .NET (JScript)
 - Adobe's PDF
 - Macromedia/Adobe's Flash
 - ...

Hello World!

```
<html>
  <head><title>Hello world!</title></head>
  <body>
    <script type="text/javascript">
      document.write("Hello world!")
    </script>
  </body>
</html>
```

Brief history

- **1995:** JavaScript 1.0 introduced by Netscape and Sun
- **1996:** Microsoft introduces JScript
- **1997:** ECMAScript, 1st edition, standardized as ECMA-262 ~ JavaScript 1.3
- **1999:** ECMA-262 3rd edition ~ JavaScript 1.5
- **2004:** E4X (ECMAScript for XML)
- **2006:** JavaScript 1.7
- **20??:** ECMA-262 4th edition ~ JavaScript 2.0

Overview

- The JavaScript language
 - Language highlights and inspiration
 - Types, objects, and functions
 - Dynamic code evaluation, exceptions, regular expressions
 - Prototype-based inheritance
 - E4X – XML in JavaScript
- JavaScript in a browser environment
 - DOM
 - AJAX
 - JSON

focus on **dynamic** and other “unusual” aspects of the language

Highlights

Inspired by

- Java:
 - syntax, name 😊
- Self:
 - class-less object-oriented, prototype-based inheritance
 - dynamic typing
- Perl:
 - regular expressions
- functional programming languages:
 - higher-order functions, closures

Dynamic typing

- JavaScript has **no static type checking**
- Every value has a type at runtime
- Automatic **type conversions**, rather than runtime errors
- The type of any variable may change during execution!

Example

declares a variable (recommended, but not required!)

```
var x = 42  
document.writeln(x)  
x = "foo"  
document.writeln(x)
```

automatic
conversion from
number to string

type of x changes!

42
foo

semicolons (statement terminators)
are inserted automatically

Types

Built-in types:

- boolean (true, false)
- number (IEEE)
- string (UTF-16)
- null (null)
- undefined (undefined)
- **object**



“primitive”

Objects

- An **object** is an identity with a collection of named properties
- Each **property** is a primitive value or a reference to an object

i.e. an object is a finite map from strings to values
and it can be changed in any conceivable way!
(also called an *associative array*)

- Some properties have special **attributes**:
ReadOnly / DontEnum / DontDelete / Internal

Tricky example (using Rhino)

```
js> var x = { a: 42 }
```

```
js> x.a
```

```
42
```

```
js> x["a"]
```

```
42
```

```
js> "a" in x
```

```
true
```

```
js> x.b
```

```
js> print(x.b)
```

```
undefined
```

```
js> x["undefined"] = 7
```

```
7
```

```
js> x[x.b]
```

```
7
```

```
js> x.b.c
```

```
[...] uncaught JavaScript runtime exception:
```

```
TypeError: Cannot read property "c" from undefined
```

```
js> delete x.a
```

```
js> "a" in x
```

```
false
```

declares a new variable
and constructs an object

properties are really stored
in a map from strings to values

the value undefined
is converted to a string

type conversions
can be surprising

exceptions do exist

properties can be deleted dynamically

Object manipulation statements

```
var obj = { name: "John Doe", phone: "(202) 555-1414" }
```

```
function dump_properties1(x) {  
  result = ""  
  for (var i in x) {  
    result += i + ": " + x[i] + "\n"  
  }  
  return result  
}
```

iterates through property names

```
name: John Doe  
phone: (202) 555-1414
```

```
function dump_properties2(x) {  
  result = ""  
  for each (var p in x) {  
    result += p + "\n"  
  }  
  return result  
}
```

iterates through property values

```
John Doe  
(202) 555-1414
```

Predefined objects

- Boolean
- Number
- String
- Function
- Object
- Array
- Date
- Math
- RegExp
- ...

} *wrapper objects*

similar to the
core classes
in Java

Type conversion

- Performed “as needed”
- Primitive values ↔ wrapper objects
(e.g. `new String(...)`, `.valueOf()`)
- like in Java, but more aggressive

Some interesting operators

- assignment: `a=b`
- equality: `a==b` (allows coercion)
- identity: `a===b`

- `"1" == true` evaluates to true
- `null == undefined` evaluates to true
- `NaN === NaN` evaluates to false

Arrays

```
var x = [ 1.1, true, "a" ]  
var y = new Array(1.1, true, "a")  
  
x[10000] = 42 // arrays may be sparse  
  
var obj = { name: "John Doe", phone: "(202) 555-1414" }  
obj[10000] = 42 // normal objects are array-like!!!
```

- Arrays are a special kind of objects
 - Array has a lot of array-specific methods
 - the `length` property is automatically updated
 - setting `length` expands or truncates

Dynamic code evaluation

- `eval(s)` parses and evaluates the code in the string `s` using the current scope
- `new Function(arg1, ..., argN, s)` creates a new function object where the string `s` becomes the function body
- Can be used for executing dynamically generated code
- *Use with care!*

Exceptions and runtime errors

- JavaScript has the usual mechanism with **throw** and **try – catch – finally** (but catch catches *all* exceptions)
- Runtime errors (Error or its “subtypes”):
 - function required at function call and new
 - numbers required at arithmetic operators
 - can’t access properties of null or undefined
 - can’t access non-existing variable
 - argument to eval must be syntactically correct
 - ...

Regular expressions

- Useful for text processing (pattern match, replace)
- Syntax and functionality inspired by Perl

```
js> var re = /^\\d{8}$/  
js> re.test("20041577")  
true
```

- See the JavaScript guide for details...

Fun with functions (1/7)

Functions as **lambda expressions**:

```
inc = function(x) { return x+1 }
```

(this is a function *expression*, not a function *declaration* !)

```
fibonacci = function f(x) {  
  if (x <= 1)  
    return 1  
  else  
    return f(x-1) + f(x-2)  
}
```

f is only visible inside the function body

Fun with functions (2/7)

Number of parameters need not match:

```
js> function inc(x) { return x+1 }
```

```
js> inc(7)
```

```
8
```

```
js> inc()
```

```
NaN
```

```
js> inc("x",42)
```

```
x1
```

x is set to undefined

+ is overloaded

(inc is a function *declaration*, not a function *expression*)

Fun with functions (3/7)

Parameter list as arguments object:

```
function list(separator) {  
  var result = ""  
  var first = true  
  for (var i = 1; i < arguments.length; i++) {  
    if (first)  
      first = false  
    else  
      result += separator  
    result += arguments[i]  
  }  
  return result  
}
```

```
list(";", "foo", "bar", "baz")
```

```
foo;bar;baz
```

Fun with functions (4/7)

A **method** is a function that is associated with an object:

```
js> var counter = {  
  value: 0,  
  inc: function() { this.value++ }  
}  
js> counter.value  
0  
js> counter.inc()  
js> counter.inc()  
js> counter.value  
2
```

inc is now a method

this is bound to the current object

Fun with functions (5/7)

Functions are objects:

```
function inc(x) { return x+1 }
```

```
inc.description = "adds 1 to the given value"
```

Fun with functions (6/7)

Temporary methods using `call`:

```
f.call(x, 1, 2);
```

has the same effect as

```
x.foo = f;  
x.foo(1, 2);  
delete x.foo;
```

(`apply` is a variant where the arguments are specified as an array)

Fun with functions (7/7)

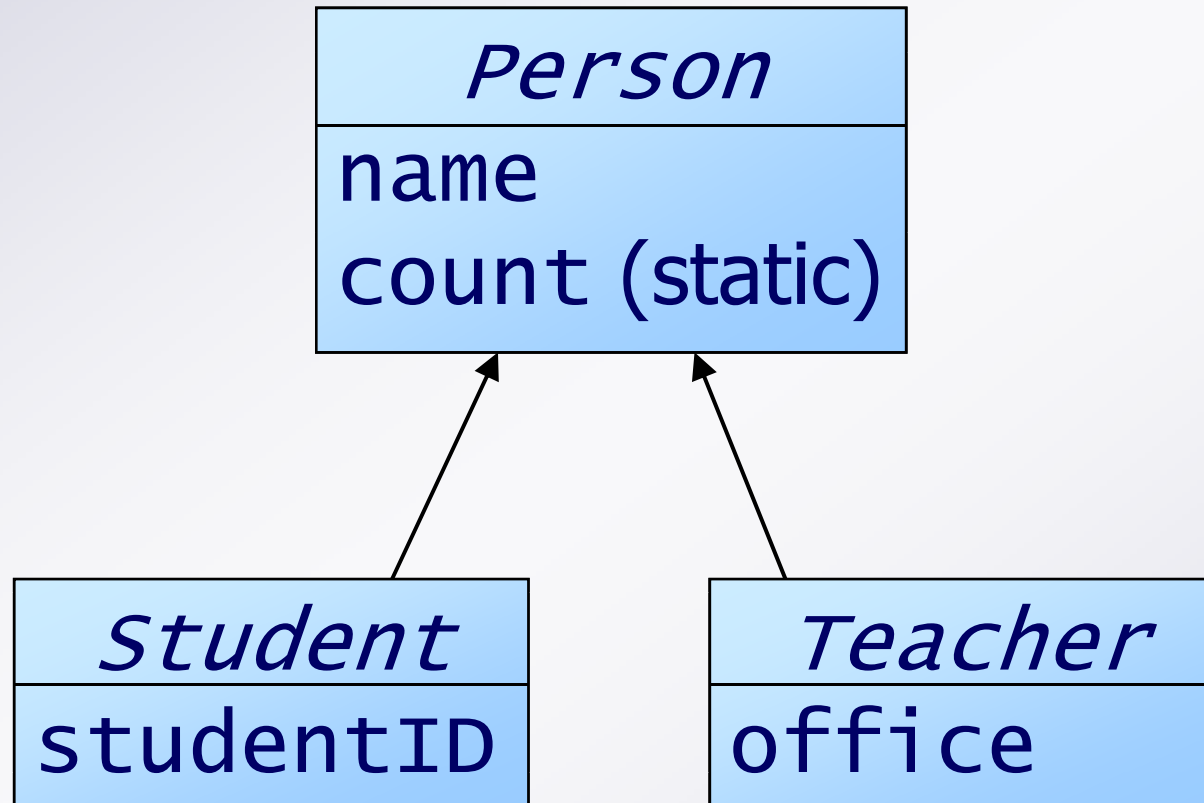
Functions can be used as **constructors**:

```
function Person(n) {  
  this.name = n  
}  
  
var x = new Person("John Doe")
```



now x is a new object
with a name property

Prototype-based inheritance



Prototype-based inheritance

– *Inheritance without classes!*

- Every function objects has a **prototype** property (aka. the *explicit prototype link*) for sharing information between instances
- `new Foo(...)` constructs a new empty object, sets the `__proto__` property (aka. the *internal prototype link*) to the prototype of `Foo`, and invokes `Foo` as a constructor
- Property lookup searches via the internal prototype link



Prototype-based inheritance

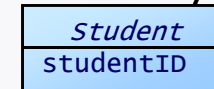
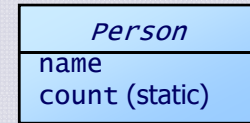
```
function Person(n) {  
  this.name = n || "???"  
  Person.prototype.count++  
}
```

```
Person.prototype.count = 0
```

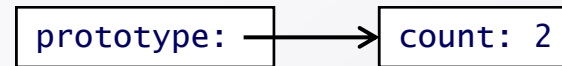
```
function Student(n,s) {  
  this.base = Person  
  this.base(n)  
  //delete this.base  
  this.studentid = s  
}
```

```
Student.prototype = new Person
```

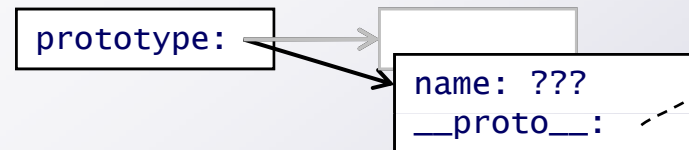
```
var x = new Student("Joe Average", "100026")  
print(x.count) // returns 2
```



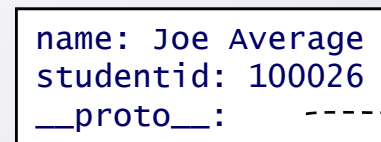
Person



Student



x



Links

- JavaScript guide and DOM reference:
<http://developer.mozilla.org/en/docs/JavaScript>



- Rhino (interpreter, open-source):
<http://www.mozilla.org/rhino/>

- ECMAScript (ECMA-262 3rd edition):
<http://www.ecma-international.org/publications/standards/Ecma-262.htm>

