JavaScript

WS Slides #2

Scripting Languages

"A **scripting language** is a programming language that is used to manipulate, customise, and automate the facilities of an existing system …
… the existing system is said to provide a host environment of objects and facilities, which completes the capabilities of the scripting language."

Existing system for us: The browser

JavaScript

JavaScript is a client (browser) side* scripting language
- Allow authors to create interactive web pages
- API's for DOM manipulation and many more ...

Implementation of the ECMAScript 5.1 specification

*) JS has started to go server side, ...

Executing JavaScript

Script(s) file downloaded from server (cached!)
at page request, typically

```
<!-- In HTML Page (if many scripts, order matters !!) -->
 <head>
   <!-- Will download script -->
   <script type="text/javascript" src="js/myjavascript.js" />
 </head>
 <body>
      ...
 </body>
```

- All script elements are executed by the browser in found order (as the document is loaded)
-JavaScript handles events, Normally only "event handling set up" executed during page load (after DOM creation!)
-Possibilities for "code on demand" (security)
-Script APIs http://www.w3.org/html/wg/drafts/html/master/webappapis.html#webappapis

Developing with JavaScript

Must have a JS debugger

Chrome (Tools > Developer tools)

In Firefox, addon Firebug (Tools menu). Installed in school

Possible to step code, inspect values etc.
- Debugging dynamically downloaded JS in debugger needs tweak

```
//Last in file products.js
//@ products.js
```

JavaScript vs Java

Looks close to Java but ...

- Java and Javascript are similar like Car and Carpet are similar

Language more related to functional languages
Scheme

- Heavy use of functions

Criticism: JS has (too?) many design flaws

- Author, Brendan Eich, .. says he implemented JS in 10 days.. we believe him...

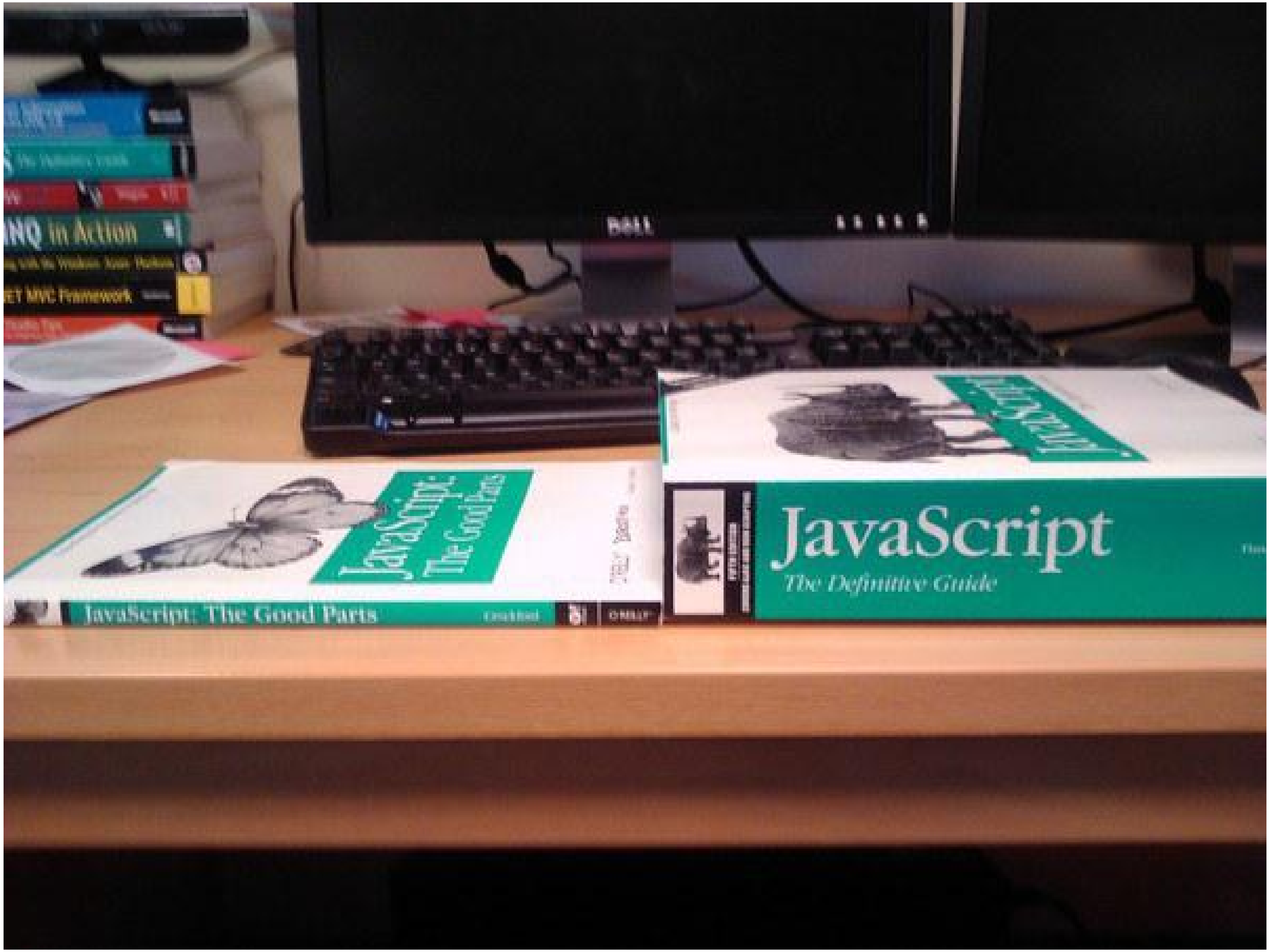
JavaScript vs Java, cont

Anyway, many thing are comfortable for a Java programmer
- Case sensitive

These details probably no problem
- Numbers 1,2, ...3.5
- Strings like “a string” or ‘a string’ (+ for concatenation)
- Operators, +, -, >, <=, NOTE: use ===, !== not ==, !=)
- Statements, if, for, while, ...

IO normally by GUI but possible to use console.log in debugger

The Good Parts



Awful Parts

Non exhaustive list ...

- Untyped, all errors runtime (typical: Nothing happens..!) Spelling!
- Global variables, introducing (using) an undeclared variable makes it global (i.e. omitting the "var" keyword)
- All compilation units loaded into a common global namespace, name clashes
- No modularization mechanisms (packages..)
- Program semantics very strange: semi-colon insertions, variable reordering?!
- **Many gotchas!!** Link on course page. NetBeans will warn for some …

There's a "strict" mode trying to reduce design flaws

JS Language Overview

- Object-based scripting language
- **Object** is a dynamic collection of properties each with zero or more attributes (i.e. key, values)
- **Properties** are containers that hold other objects, primitive values, or functions
- A primitive value is a member of one of the following built-in types: **Undefined, Null, Boolean, Number, and String**
- An object is a member of the remaining built-in type **Object**
- A **function** is a callable object
- A function that is associated with an object via a property is a **method**

JS Built-in Objects

"ECMAScript defines a collection of built-in objects that round out the definition of ECMAScript entities. These built-in objects include the **global** object, the **Object** object, the **Function** object, the **Array** object, the **String** object, the **Boolean** object, the **Number** object, the **Math** object, the **Date** object, the **RegExp** object, the **JSON** object, and the Error objects **Error, EvalError**, **RangeError, ReferenceError, SyntaxError, TypeError** and **URIError**."
// ECMA Specification

NOTE

- Objects not classes, there are no classes in JS

JS Host Environment

"A web browser provides an ECMAScript host environment for client-side computation including, for instance, objects that represent windows, menus, pop-ups, dialog boxes, text areas, anchors, frames, history, cookies, and input/output [host objects]. Further, the host environment provides a means to attach scripting code to events such as change of focus, page and image loading, unloading, error and abort, selection, form submission, and mouse actions. Scripting code appears within the HTML and the displayed page is a combination of user interface elements and fixed and computed text and images. The scripting code is reactive to user interaction and there is no need for a main program."
// ECMA Specification

In browser the window object = the global object

References

Objects manipulated via references like Java
- References have no type, language untyped (= not statically typed
- Values have types not references (variables))

```
// Using  an untyped reference
var myRef = ....
```

Objects

Object are <u>mutable</u> maps like; Map<Property, Value> in Java

- Object <u>structure</u> possible change during lifetime
- No constructor
- Everything public
- Can contain sub-objects
- Internal properties, pops up (not possible to manipulate with language, but need understanding)
- Object literal (create object in line): { … } ← an objec

Functions

Functions are (close to) objects with the additional capability of being callable

- Functions are first class members; Functions as parameters/result, references to, callbacks
- Anonymous functions common
- Inner functions (function inside function)
- Functions may have properties (data..!)
- Functions as information hiding. Objects inside function not accessible from outside. JS has function scope not block scope
- Has property "arguments". Gives access to all supplied arguments and more...
- If return ... in function, function will return value. Else void

Function cont.

Call conventions same as Java (by value)

- Some primitives are like objects (boxed)!
- Passing in a function is different (although it's an object)?!
- Different invocation patterns (!), more to come...

Methods

A function as a property of an object (literal here)
- Methods, <u>will belong to object</u>, not shared between objects

```
var o = { // Start object literal

          ...

          doIt: function() {  // Method (reference to anon. function)
              return … ;
          }

}; // End object literal

// Call
o.doIt();
```

Closures

“A closure is formed when one of those inner functions is made accessible outside of the function in which it was contained, so that it may be executed after the outer function has returned. At which point it still has access to the local variables, parameters and inner function declarations of its outer function. Those local variables, parameter and function declarations (initially) have the values that they had when the outer function returned and may be interacted with by the inner function.”
// http://www.jibbering.com/faq/notes/closures/

More to come...

Funny comment on web
“Closures are not hard to understand once the core concept is grokked. However, they are impossible to understand by reading any academic papers or academically oriented information about them!”

Prototype

All objects have an implicit reference to a "parent object"

- This is the **object prototype**
- Prototype object used to share properties between objects (remember; methods belong to individual object)
- Displayed in Chrome debugger as **__proto__**

The prototype chain

- All objects have a link to it's "parent", a chain of references. If asking for some property not found in actual object, the prototype chain is searched
- Final object in chain is Object object
- Possible to alter new children by assigning new values to the prototype for some parent (old children not affected)

Constructor Function

Function used to create and initialize new objects in conjunction with operator **new** (what's "new" in Java?)

- Can't distinguish from normal function, except using leading uppercase for name (an idiom, not enforced)
- Imagine a freestanding constructor (outside object)

A constructor function has a not hidden reference to an automatically created unnamed prototype object

- This is called the **prototype property**
- Accessed with NameOfContructorFunction.prototype
- The prototype objects has a reference back to constructor

Object Creation and Prototype

Using constructor function and new

"When a constructor creates an object, that object implicitly references the constructor's prototype property for the purpose of resolving property references."

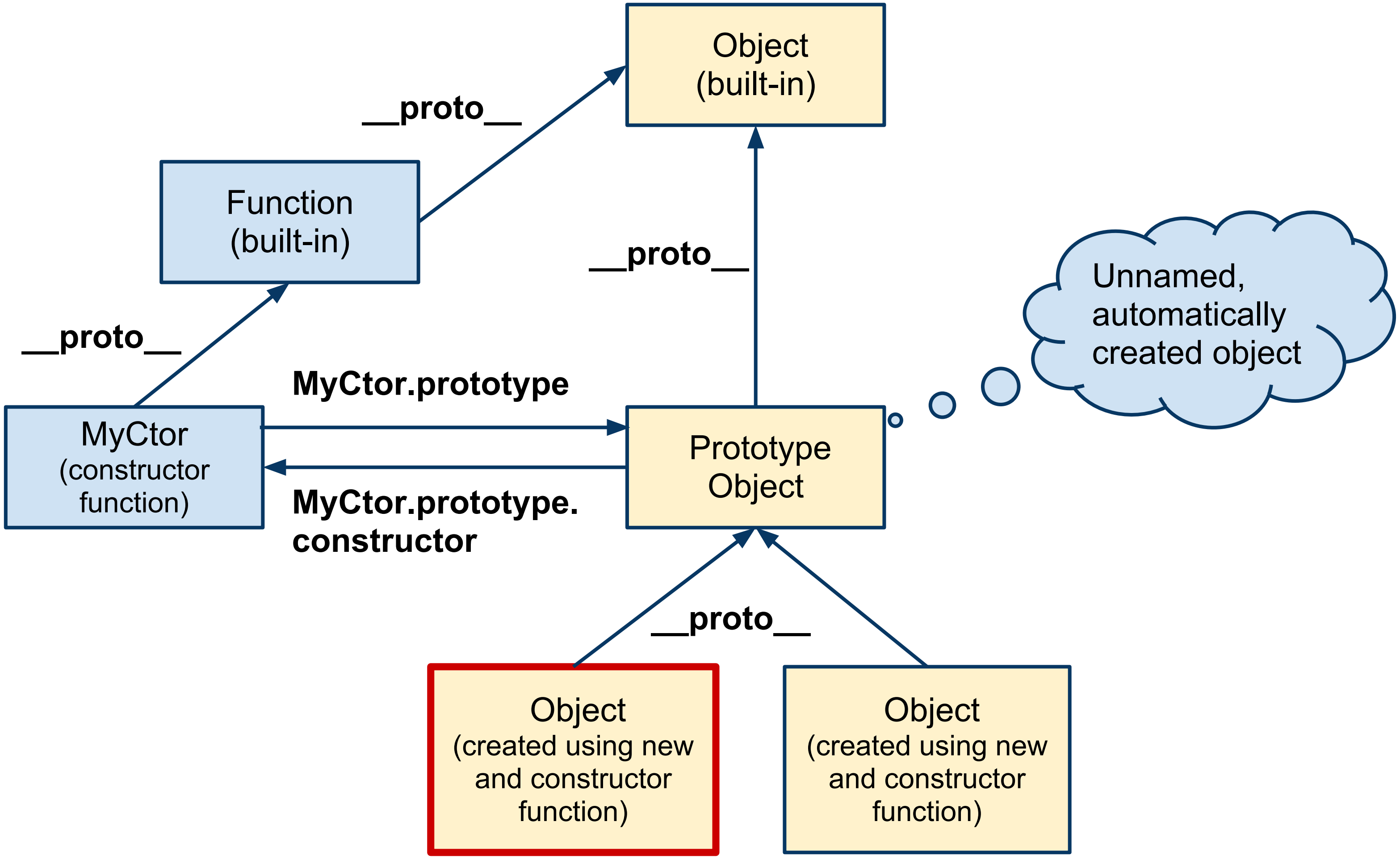
// ECMA Spec

An alternative to create objects (possible better) is Object.create(...) (not used by us)

Object Creation with Constructor

Executing

`Object o = new MyCtor();`


Object
(built-in)
__proto__
Function
(built-in)
__proto__
Unnamed, automatically created object
__proto__
MyCtor.prototype
MyCtor
(constructor function)
Prototype Object
MyCtor.prototype. constructor
__proto__
Object
(created using new and constructor function)
Object
(created using new and constructor function)

The “this” keyword

Different from Java

In global scope **this** is the window object (the global object)

In function scope

- The value of **this** in a function context is provided by the caller and determined by the current form of a call expression (how the function call is written syntactically!!!)
- this not statically bound (may vary)
- Complicated: We simplify by identifying fairly correct invocation patterns ...

“This” invocation patterns

Global function invocation: In function **this** is the global object

Method invocation: In method **this** is the object the method belongs to

Constructor function invocation (in conjunction with new) : In function **this** is the newly created object

Preserving “this”

Inner functions doesn’t share **this** with outer

To use **this** in inner function, have to store before executing inner

```
var me = this;
// .. this changes, use me
```

Sometimes handled automatically by some higher level libraries, JQuery, ...

Emulating Java

We use the JS "pseudo-classical" style + module pattern to emulate classes

- Class is = constructor function (for object data) + prototype holding shared methods
- Put "class" in matching file like Java (class name = file name)

Pseudo-Classical JavaScript

```
// Emulate a class (this is in file person.js)
var Person = function( name ){   // Constructor function
    // Set attributes here, "this" is the actual object!
    this.name = name;
};

Person.prototype = (function(){ // Shared by all objects
    // Public API here
    return { // Anonymous object
        setName : function( name ){
            this.name = name;      // "this" is the actual object
        },

        getName : function (){
            return this.name;
        }
    }
})();   // Call function immediately
```

Pseudo-Classical JavaScript cont.

```
// MUST use new else disaster (this bound to global object)
var p = new Person( "Sara" );
```

Now…

… p is a reference to the returned anonymous object with methods setName and getName (previous slide)

```
// Call
var name = p.getName();
```

getName not found in p, so search prototype, method found! … call it, "this" supplied by caller (the p object), so this is really the actual object

Miscellaneous

Comments as in Java /* */ and //

Declaring variables using var keyword (optional **but use**!)

```
var a = { … }// a reference to object literal
    // Forget var -> Global scope (even in functions)
```

Ending ';' optional **but use!**

Watch out for conversions and comparison, always use === for comparison

JavaScript is **single threaded**, … ! More to come…

Exceptionhandling …

JavaScript API's

Language very tightly connected to some API's

The most basic are DOM, DOM Event, CSS, and AJAX (XMLHttpRequest object) …

- …will not be used directly!
- We'll use the higher level library JQuery, more to come…
- Many new API's in HTML5, not covered …

Window and Document

This is part of the DOM API, we don't use but some knowledge needed

- The window object represents the browser window itself (automatically created)
- Each tab contains its own window object i.e. not shared between tabs in the same window (except possible some global properties)
- The document property of a window points to the DOM for the document loaded in that window

```
 // Example native JS DOM API call
var e = document.getElementById( ...);
```

- Also document.cookie