

G22.3033-002
Scripting Languages
6/26/2008
Client-Side Scripting (JavaScript)

© Martin Hirzel G22.3033-002 NYU 6/26/2008 1

Outline

- JavaScript Basics
- Prototypes
- Scopes
- At 7:30pm: Quiz 2

© Martin Hirzel G22.3033-002 NYU 6/26/2008 2

JavaScript

About JavaScript

- Prototype-based object oriented language
 - There are no classes (unlike Java, C++, etc.)
 - Object inherits from prototype object
 - Similar to Self programming language
- Event-driven: callbacks for browser actions
- Mostly used for client-side scripting

```

sequenceDiagram
    participant User
    participant Browser
    participant Server
    User->>Browser: render HTML
    Browser->>Server: click or type input
    Server->>Browser: serve HTML+script
    Browser->>User: run JavaScript event handler
    User->>Browser: update rendering
  
```

© Martin Hirzel G22.3033-002 NYU 6/26/2008 3

JavaScript

Related Languages

- JavaScript is not related to Java
 - Originally LiveScript, by Brendan Eich at Mozilla
 - Name change is Netscape/Sun marketing ploy
- 1999: ECMAScript v.3 official specification
 - JavaScript 1.5 (many browsers) = JScript 5.5 (IE)
- In progress: ECMAScript v.4 = JavaScript 2.0
- JavaScript is the only client-side scripting language that works in all browsers
 - VBScript only works in Internet Explorer

© Martin Hirzel G22.3033-002 NYU 6/26/2008 4

JavaScript

Lexical Peculiarities

- Embedded in HTML `<script>...</script>`
- Case sensitive
- No sigils, no interpolation
- Semicolon optional at end of line, that can cause bug when prefix valid statement
- Single-line comments: `//...`, or at start `! --...`
- Multi-line comments: `/*...*/`
- Literals: `"s"`, `'s'`, `true`, `null`, `RegExp /.../`, Object `{x:1, y:2}`, Array `[1,2,3]`

© Martin Hirzel G22.3033-002 NYU 6/26/2008 5

JavaScript

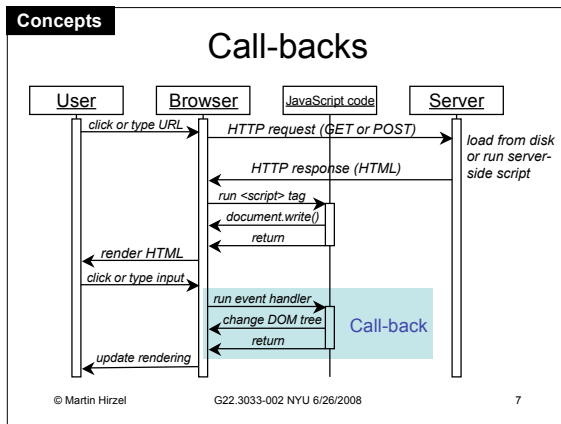
How to Write + Run Code

- Put in local file, then File→Open in web browser:

```

<html><head>
  <meta http-equiv="Content-Script-Type" content="text/javascript" />
  <script>
    function greet() {
      var who = document.question.who.value;
      var answer = document.getElementById("answer");
      answer.innerHTML = "Hi, " + who + "!";
    }
  </script>
</head><body>
  <span class="result" id="answer"></span>
  <form name="question">
    Who shall be greeted:
    <input type="text" name="who" onchange="greet();" />
    <input type="button" value="Greet" onclick="greet();" />
  </form>
</body></html>
  
```

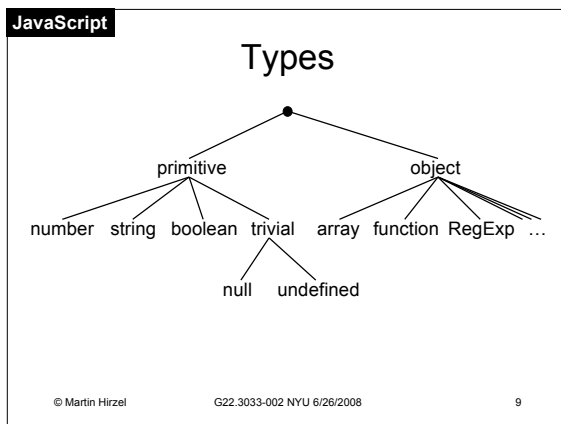
© Martin Hirzel G22.3033-002 NYU 6/26/2008 6



JavaScript Input and Output

- Input:
 - Call-backs triggered by events, when user interacts with HTML `<form>` etc.
 - Reading information from DOM tree
- Output
 - `document.write("...")`
 - At HTML parse time: in-place insert
 - Later from event handler: overwrite document(!)
 - DOM tree manipulation: `document.getElementById("...").innerHTML = "..."`
 - Browser interaction, e.g., pop-up: `alert("...")`

© Martin Hirzel G22.3033-002 NYU 6/26/2008 8



JavaScript Variable Declarations

Implicit	<code>b = 5;</code>	Reading an undeclared variable is a runtime error
Explicit	<code>var x;</code> <code>var i=0, msg="hi";</code>	Declaration without assignment is undefined value

- Scope of implicit declaration is global
- Scope of explicit declaration is local to function (there is no block scope)

© Martin Hirzel G22.3033-002 NYU 6/26/2008 10

JavaScript Type Conversions

Value	Boolean	Number	String	Object
undefined	false	NaN	"undefined"	Error
null	false	0	"null"	Error
Boolean	false	Identity	0	Boolean object
	true		1	
Number	0	Identity	"0"	Number object
	NaN		"NaN"	
	Other	true	String	
String	" "	Number or NaN	Identity	String object
	Other		true	
Object	true	<code>valueOf()</code> , <code>toString()</code> , or NaN	<code>toString()</code>	Identity

© Martin Hirzel G22.3033-002 NYU 6/26/2008 11

JavaScript Operators

<code>..</code> , <code>[]</code> , <code>()</code> , <code>new</code>		Member/array/call access, creation
<code>++</code> , <code>--</code> , <code>+</code> , <code>-</code> , <code>!</code> , <code>delete</code> , <code>typeof</code> , <code>void</code>	1	<code>delete</code> : remove property from object; <code>typeof</code> : return type; <code>void</code> : discard value
<code>*</code> , <code>/</code> , <code>%</code>	2	L Multiplicative
<code>+</code> , <code>-</code>	2	L Additive; <code>+</code> : add numbers/concat strings
<code><<</code> , <code>>></code> , <code>>>></code>	2	L Bitwise shift
<code><</code> , <code><=</code> , <code>></code> , <code>>=</code> , <code>instanceof</code> , <code>in</code>	2	L Comparison; <code>in</code> : check membership
<code>==</code> , <code>!=</code> , <code>===</code> , <code>!==</code>	2	L Identity; <code>===</code> , <code>!==</code> : equal value+type
<code>&</code>	2	L Bitwise and
<code>^</code>	2	L Bitwise xor
<code> </code>	2	L Bitwise or
<code>&&</code>	2	L Logical and
<code> </code>	2	L Logical or
<code>?:</code>	3	R Conditional
<code>=</code> , <code>+=</code> , <code>-=</code> , ...	2	R Assignment
<code>,</code>	2	L Multiple evaluation

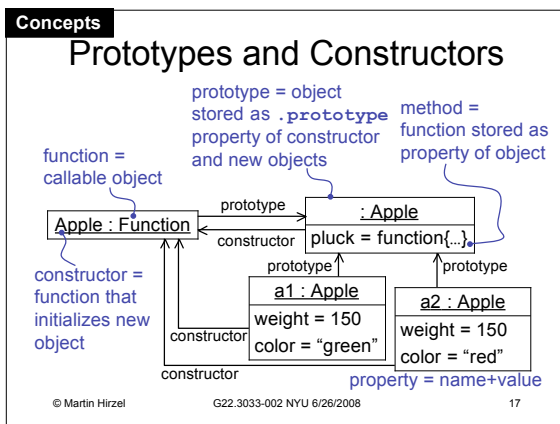
© Martin Hirzel G22.3033-002 NYU 6/26/2008 12

JavaScript	
<h3>Control Statements</h3>	
Conditional	<code>if (expr) ... [else ...]</code> <code>switch (expr) { case expr: ... default: ... }</code>
Fixed-iteration loops	<code>for (expr; expr; expr) ...</code> <code>for (var in expr) ...</code>
Indefinite loops	<code>while (expr) ...</code> <code>do ... while (expr);</code>
Unstructured control	<code>break [labelName];</code> <code>continue [labelName];</code> <code>return [expr];</code> <code>throw expr;</code>
Other	<code>try {...} [catch (id) {...}] [finally {...}]</code> <code>with (expr) ...</code>
© Martin Hirzel G22.3033-002 NYU 6/26/2008 13	

JavaScript	
<h3>Writing Subroutines</h3>	
• Declaration: <code>function id (arg*) {...}</code>	– Declaration creates named function at compile time – <code>id</code> mandatory; visible externally in enclosing scope
• Arguments: <code>arg ::= id</code>	– User may pass more or fewer than declared – Missing arguments have value <code>undefined</code> – All (declared and surplus) arguments are in <code>arguments</code> array-like object
• Literal: <code>function [id] (arg*) {...}</code>	– Expression creates anonymous function at runtime – <code>id</code> optional; only visible internally for recursion
© Martin Hirzel G22.3033-002 NYU 6/26/2008 14	

<h3>Outline</h3>	
• JavaScript Basics	
• Prototypes	
• Scopes	
• At 7:30pm: Quiz 2	
© Martin Hirzel G22.3033-002 NYU 6/26/2008 15	

JavaScript	
<h3>Using Objects</h3>	
<code>a1 = new Apple(150, "green");</code> <code>a2 = new Apple(150, "green");</code>	Constructor calls
<code>a2.color = "red";</code>	Property access
<code>document.write(a1.prepare("slice") + "
\n");</code> <code>document.write(a2.prepare("squeeze") + "
\n");</code>	Method calls
© Martin Hirzel G22.3033-002 NYU 6/26/2008 16	



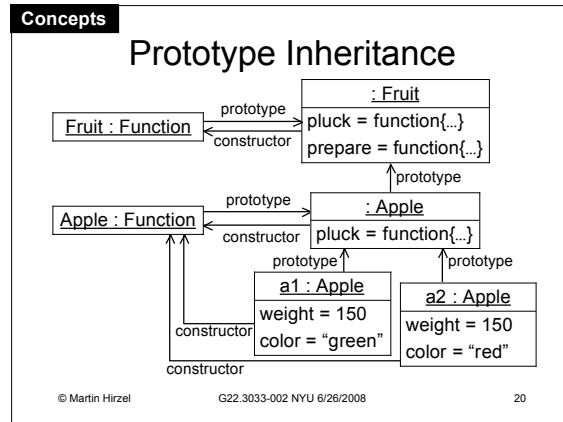
JavaScript	
<h3>Defining Classes Constructors</h3>	
• All functions <code>f</code> are objects with a prototype property	
• <code>f.prototype</code> initially points to an almost empty object	
• <code>f.prototype.constructor</code> points back to <code>f</code>	
<pre>function Apple(weight, color) { this.weight = weight; this.color = color; }</pre>	Convention: Assign object properties (fields) in constructor
<pre>Apple.prototype.pluck = function() { return this.color + " apple"; }</pre>	Assign prototype properties (methods) before any calls to constructor
© Martin Hirzel G22.3033-002 NYU 6/26/2008 18	

JavaScript

Operators on Objects

<code>new C (...)</code>	<ul style="list-style-type: none"> Create new empty object <i>o</i> Set <code>o.prototype = C.prototype</code> Set <code>o.constructor = C.prototype.constructor</code> Call <code>C(...)</code>, pass <i>o</i> as value for <code>this</code> Return result of <code>C(...)</code> or <i>o</i> if none
<code>o.p</code>	<ul style="list-style-type: none"> If object <i>o</i> has property <i>p</i>, return it Otherwise, look in <code>o.prototype</code>
<code>o.p = expr</code>	<ul style="list-style-type: none"> If object <i>o</i> has property <i>p</i>, assign it Otherwise, create it and assign it
<code>o instanceof C</code>	<ul style="list-style-type: none"> If <code>o.constructor</code> is <i>C</i>, return <code>true</code> Otherwise, look in <code>o.prototype</code>

© Martin Hirzel G22.3033-002 NYU 6/26/2008 19



JavaScript

Inheritance in JavaScript

```
function Fruit(weight) {
  this.weight = weight;
}
Fruit.prototype.pluck = function() {
  return "fruit(" + this.weight + "g)";
}
Fruit.prototype.prepare = function(how) {
  return how + "d" + this.pluck();
}
function Apple(weight, color) {
  this.weight = weight;
  this.color = color;
}
Apple.prototype = new Fruit();
delete Apple.prototype.weight;
Apple.prototype.constructor = Apple;
Apple.prototype.pluck = function() {
  return this.color + " apple";
}
```

- Constructor assigns fields
- Top-level assigns methods
- Inherit from `Fruit.prototype`
- Remove spurious property
- Enable `instanceof` checks

© Martin Hirzel G22.3033-002 NYU 6/26/2008 21

Soap-box

Evaluating Prototypes

<p>Strengths</p> <ul style="list-style-type: none"> Orthogonality <ul style="list-style-type: none"> No class/object duality Flexibility <ul style="list-style-type: none"> Can emulate classes But don't have to E.g., can borrow (copy) method instead of inheriting 	<p>Weaknesses</p> <ul style="list-style-type: none"> Lack of familiarity Lack of static guarantees <ul style="list-style-type: none"> Less error checking Harder to optimize (but, Self pioneered many optimizations later used for Java)
---	---

© Martin Hirzel G22.3033-002 NYU 6/26/2008 22

JavaScript

More Operators on Objects

<code>o["p"]</code>	<ul style="list-style-type: none"> If object <i>o</i> has property <i>p</i>, return it Otherwise, look in <code>o.prototype</code>
<code>o["p"] = expr</code>	<ul style="list-style-type: none"> If object <i>o</i> has property <i>p</i>, assign it Otherwise, create it and assign it
<code>"p" in o</code>	<ul style="list-style-type: none"> If object <i>o</i> has property <i>p</i>, return <code>true</code> Otherwise, look in <code>o.prototype</code>
<code>typeof o</code>	<ul style="list-style-type: none"> String describing JavaScript type, e.g., <code>typeof new Fruit == "object"</code>
<code>delete o.p</code>	<ul style="list-style-type: none"> If object <i>o</i> has property <i>p</i>, set its value to <code>undefined</code> <code>for (x in o)</code> loop omits <code>undefined</code> <i>p</i>

© Martin Hirzel G22.3033-002 NYU 6/26/2008 23

JavaScript

Arrays

- Creation: `a=[4,5,6]`; `b=new Array(size)`; `c=new Array(7,8,9)`;
- Indexing: e.g., `a[2]`; `a[-1]="s"`; `a["k"]=99`
 - Both arrays and non-array objects can be indexed by both numbers and strings
 - Arrays are special: `length` property is 1 + last used integer index
 - Write to non-existent index inserts
- Setting element to undefined: `delete a[i]`
- Resizing array: `a.length = newLength`

© Martin Hirzel G22.3033-002 NYU 6/26/2008 24

JavaScript

Native ECMAScript Objects

Array	Literal notation: [...] length property behaves specially
RegExp	Literal notation: <i>/.../flags</i>
Function	Callable, can serve as constructor
String	Wrappers for auto-boxing
Boolean	(implicit conversion upon <i>x.p</i> access);
Number	Auto-unboxing for primitive operators

- Also: **Global**, **Object**, **Math**, **Date**, **Error**
- See ECMA-262 v.3 specification for properties + methods

© Martin Hirzel G22.3033-002 NYU 6/26/2008 25

Outline

- JavaScript Basics
- Prototypes
- Scopes
- At 7:30pm: Quiz 2

© Martin Hirzel G22.3033-002 NYU 6/26/2008 26

JavaScript

Scope Chain

```

var a = 'ga';
function x(b) {
  c = 'gc';
  var d = 'xd';
  function y(a,d) {
    var e = 'ye';
  }
  y('ya', 'yd');
}
x('xb');
    
```

© Martin Hirzel G22.3033-002 NYU 6/26/2008 27

JavaScript

Variables as Properties

- Globals = properties of "global object"
 - Client-side JavaScript: window is global object; separate windows for each frame
 - Top-level functions are also properties of global object, e.g., **x**, **Apple**
- Locals = properties of "call object"
 - Call objects are chained in scope chain
 - Call object has property arguments, and **arguments.callee** is current function

© Martin Hirzel G22.3033-002 NYU 6/26/2008 28

JavaScript

Receiver Object

<code>y = o.f(x)</code>	During f , this == o
<code>y = f.call(o, x)</code>	
<code>y = f.apply(o, [x])</code>	
<code>y = new f(x)</code>	During f , this == new object
Top-level code (outside any function)	this == global object
<code>y = f(x)</code>	During f , this == global object

© Martin Hirzel G22.3033-002 NYU 6/26/2008 29

JavaScript

Abbreviated Member Access

with(o) stmt

- Prepend object *o* to scope chain during *stmt*
- Declaring variables in *stmt* may cause surprises

```

var o = { p: 'op' };
function f() {
  var p = 'fp';
  with(o) {
    document.write(p);
  }
}
f();
    
```

© Martin Hirzel G22.3033-002 NYU 6/26/2008 30

JavaScript

Static Scoping

Static scoping
Bound in closest nesting scope in program text

```

x = 's';
function g() {
  var x = 'd';
  h();
}
function h() {
  document.write(x); //s
}
g();
document.write(x); //s
    
```

© Martin Hirzel G22.3033-002 NYU 6/26/2008 31

JavaScript

Closure = Function + Environment

Environment = Old scope chain

```

function f(y) {
  function g() {
    document.write(y);
  }
  return g;
}
var c = f(1);
var d = f(2);
c();
d();
    
```

© Martin Hirzel G22.3033-002 NYU 6/26/2008 32

Concepts

Closures

```

function f(y) {
  function g() {
    document.write(y);
  }
  return g;
}
var c = f(1);
var d = f(2);
x(); // prints "1"
y(); // prints "2"
    
```

Closure c
= Function g() {document.write(y);}
+ Environment {y:1}

© Martin Hirzel G22.3033-002 NYU 6/26/2008 33

JavaScript

Packages

- HTML tag `<script src="mod.js">` and JavaScript coding conventions
 - No separate JavaScript language feature!
 - Convention: module should never define more than a single global name
 - Usually: object, which provides properties
 - Convention: nested objects named by reverse domain, e.g., `edu.nyu.cs.hirzel`
 - Convention: initialize module in anonymous function that gets called immediately
 - `(function() { /* initialization code */ }) ();`

© Martin Hirzel G22.3033-002 NYU 6/26/2008 34

JavaScript

Structure of a JavaScript Application

parent frames [i]	Other browser windows showing documents from same origin
Global object	Own window object, all top-level variables and functions are properties
Included file	Included with <code><script src=...></code> tag, no separate scope (part of global object)
Object	Contains properties (including functions); can emulate package by convention
Call object	Created at call time, may survive when nested function gets returned as closure
Function	Callable object, may serve as constructor
Statements	In function or directly at top-level

© Martin Hirzel G22.3033-002 NYU 6/26/2008 35

JavaScript

Scopes and Visibility

```

var edu; //put this code in file "edu/nyu/cs/Counter.js"
if (!edu) edu = {}; //only one global symbol, don't pollute namespace
if (!edu.nyu) edu.nyu = {};
if (!edu.nyu.cs) edu.nyu.cs = {};
if (edu.nyu.cs.Counter) throw new Error("Counter already exists");
(function() {
  edu.nyu.cs.Counter = {};
  var private_counter = 0; //after return only visible from closures
  edu.nyu.cs.Counter.increment = function() { private_counter++; }
  edu.nyu.cs.Counter.read = function() { return private_counter; }
})();
<html><head> <!-- put this code in file "CounterTest.html" -->
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<script src="edu/nyu/cs/Counter.js"> </script>
</head><body>
<script>
var Counter = edu.nyu.cs.Counter; //short name (like Java import)
document.write(Counter.read() + "<br/>\n");
Counter.increment(); //private_counter not directly visible here
document.write(Counter.read() + "<br/>\n");
</script>
</body></html>
    
```

© Martin Hirzel G22.3033-002 NYU 6/26/2008 36

JavaScript

JavaScript Documentation

- Core JavaScript specification:
<http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf>
- Book: Programming JavaScript, 5th edition [safari], David Flanagan. O'Reilly, 2006.
- Tutorial: <http://www.w3schools.com>
- Browser support summary:
<http://www.webdevout.net/browser-support>
- W3C DOM: <http://www.w3.org/DOM/DOMTR>
- JavaScript Archive Network:
<http://www.openjsan.org>

© Martin Hirzel G22.3033-002 NYU 6/26/2008 37

JavaScript

Last Slide

- Nothing to announce
- Today's lecture
 - Client-side scripting
 - JavaScript
 - Prototypes
 - Closures
- Next lecture
 - Web applications
 - Databases
 - Session state
 - Form validation
 - AJAX

© Martin Hirzel G22.3033-002 NYU 6/26/2008 38

Outline

- JavaScript Basics
- Prototypes
- Scopes
- At 7:30pm: Quiz 2

© Martin Hirzel G22.3033-002 NYU 6/26/2008 39