CSCI-GA.3033.003
Scripting Languages

10/09/2013

Client-Side Scripting (JavaScript)

Outline

- Document Object Model
- Scopes

Native ECMAScript Objects

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

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

The Global Object

Value properties

- `NaN`
- `Infinity`
- `undefined`

Function properties

- `eval(x)`
- `parseInt(`
  `string, radix)`
- `parseFloat(string)`
- `isNaN(string)`
- `isFinite(number)`
- …

Constructor properties

- `Object(…)`
- `Function(…)`
- `Array(…)`
- `String(…)`
- `Boolean(…)`
- `Number(…)`
- `RegExp(…)`
- `Error(…)`
- …

Other properties

- `Math`

The Array Constructor

Called as constructor

- **`new Array(len)`**
- **`new Array(i0,i1,…)`**

Called as function

- **`x=Array(…)`** is same as **`x=new Array(…)`**

Properties of instances

- **`length`**

Properties of prototype

- **`toString()`**
- **`toLocaleString()`**
- **`concat(a0,a1,…)`**

Properties of prototype (continued)

- **`join(separator)`**
- **`pop()`**
- **`push(i0,…)`**
- **`reverse()`**
- **`shift()`**
- **`slice(start, end)`**
- **`sort(compareFun)`**
- **`splice(start, deleteCount,[i0,…])`**
- **`unshift([i0,…])`**

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*

The Math Object

Value properties

- `E`
- `PI`
- …

Function properties

- `abs(x)`
- `acos(x)`
- `asin(x)`
- `ceil(x)`
- `cos(x)`
- `exp(x)`
- `floor(x)`

Function properties (continued)

- `log(x)`
- `max(v1,…)`
- `min(v1,…)`
- `pow(x,y)`
- `random()`
- `round(x)`
- `sin(x)`
- `sqrt(x)`
- `tan(x)`
- …

The RegExp Constructor

Called as constructor

- **`new RegExp( pattern,flags)`**

Called as function

- **`x=RegExp(…)`** is same as **`x=new RegExp(…)`**

Properties of instances

- **`source`**
- **`global`**
- **`ignoreCase`**
- **`multiline`**
- **`lastIndex`**

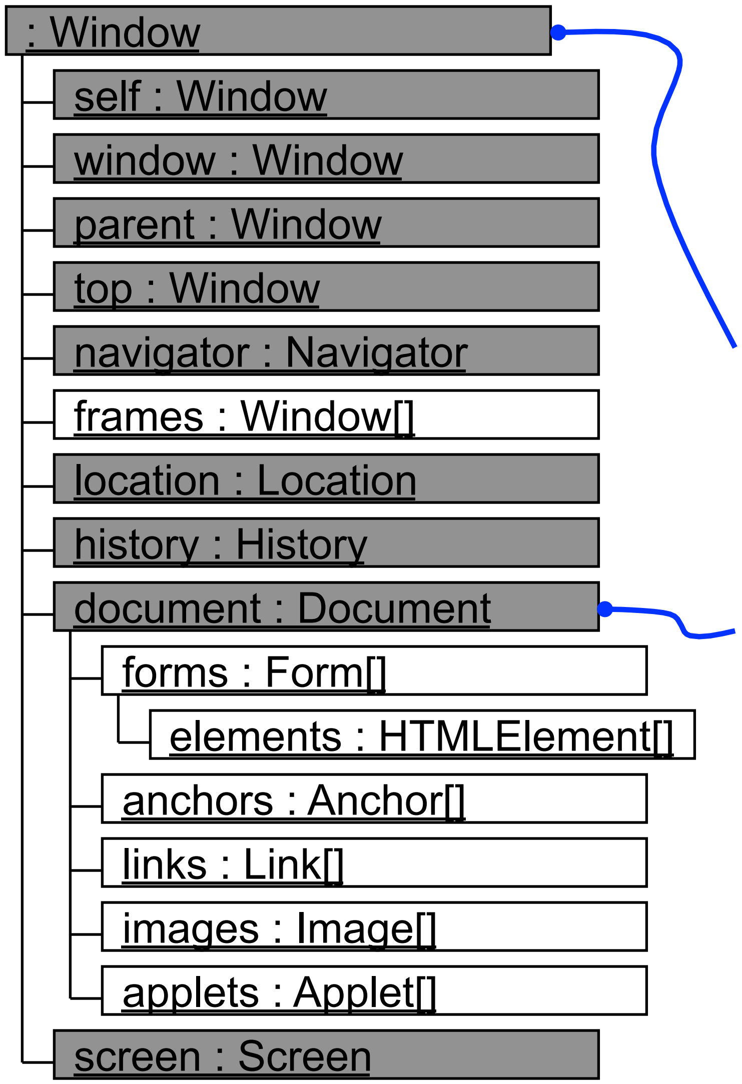
Properties of **`RegExp`** prototype

- **`exec(string)`**
- **`test(string)`**
- **`toString()`**

Properties of **`String`** prototype

- **`match(regexp)`**
- **`search(regexp)`**
- …

Client-Side JavaScript Object Model



- Object Model =
  API for embedded scripts
  (remember VBA)
- In web browser, the Global object is also a Window object
- DOM =
  Document Object Model =
  API for HTML tree
- Browsers implement incompatible DOMs

The Window Object

- All properties shown on slide “The Global Object”
- All properties shown under “Window” on slide “Client-side JavaScript Object Model”
- Feature testing example

```
function getSelectedText() {
  if (window.getSelection)
    return window.getSelection.toString();
  else if(document.getSelection)
    return document.getSelection();
  else if(document.selection)
    return document.selection.createRange().text;
}
```

Document Objects

Value properties

- `bgColor`
- `cookie`
- `domain`
- `lastModified`
- `location`
- `referrer`
- `images[]`
- `forms[]`
- …
- One property for each named form
  - Each form has a property for each named element

Function properties

- `open()`
- `write(s0,...)`
- `writeln(s0,...)`
- `close()`
- `createAttribute(name)`
- `createComment(text)`
- `createDocumentFragment()`
- `createElement(tagName)`
- `createTextNode(text)`
- `getElementId(id)`
- `getElementsByName(name)`
- `getElementsByTagName(tag)`
- `importNode(node, deep)`

Node Objects

Value properties

- **`innerHTML`**
- **`nodeName`**
- **`nodeType`**
- **`nodeValue`**
- **`childNodes[]`**
- **`firstChild`**
- **`lastChild`**
- **`ownerDocument`**
- **`parentNode`**
- **`previousSibling`**
- **`nextSibling`**
- One property for each HTML attribute

Function properties

- **`hasChildNodes()`**
- **`appendChild(newChild)`**
- **`insertBefore(newChild,refChild)`**
- **`removeChild(oldChild)`**
- **`replaceChild(newChild,oldChild)`**
- **`normalize()`**

Constants for **`nodeType`**

1	**`ELEMENT_NODE`**
2	**`ATTRIBUTE_NODE`**
3	**`TEXT_NODE`**
8	**`COMMENT_NODE`**
9	**`DOCUMENT_NODE`**
11	**`DOCUMENT_FRAGMENT`**

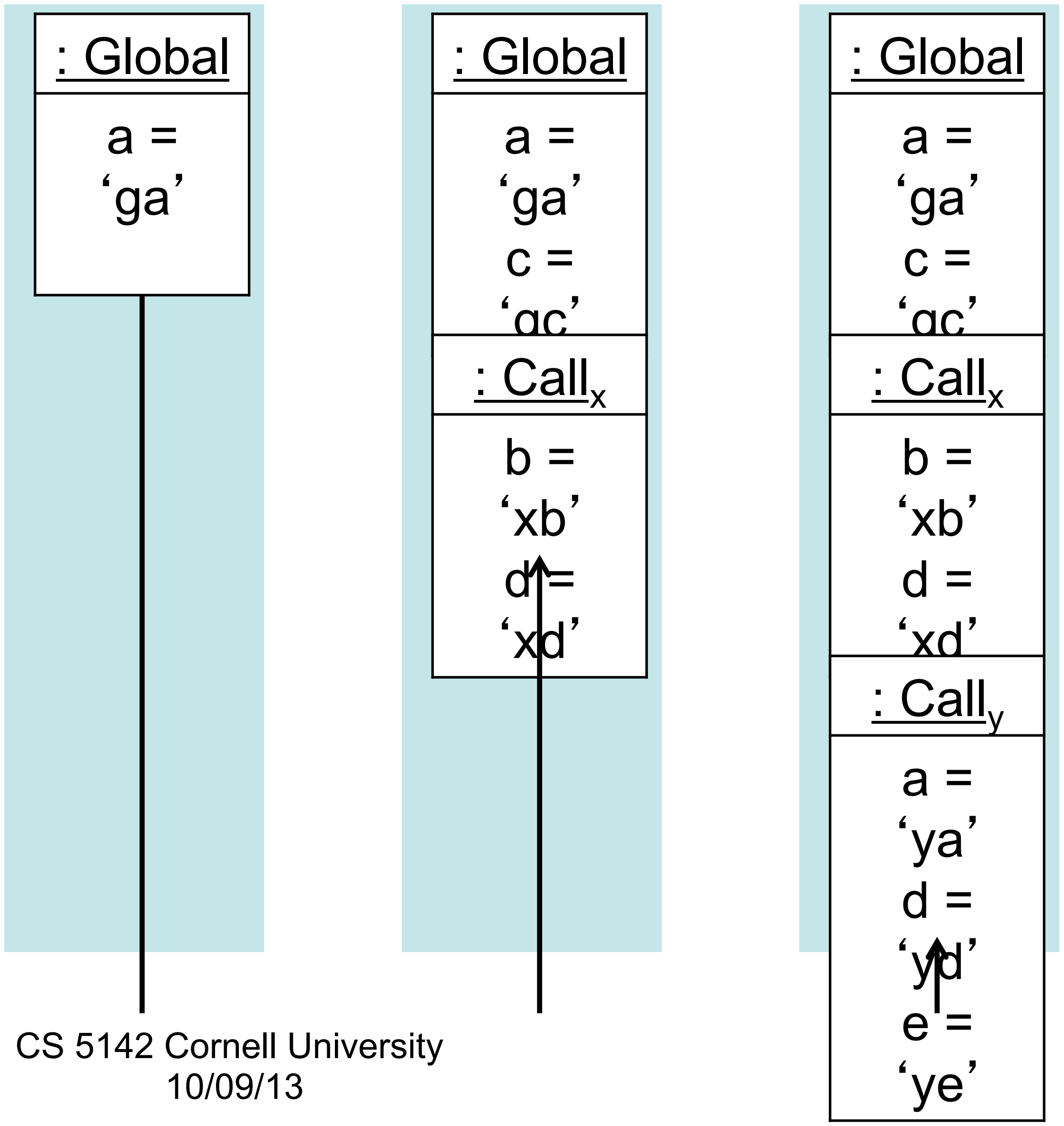
Outline

- Document Object Model
- Scopes

JavaScript

Scope Chain

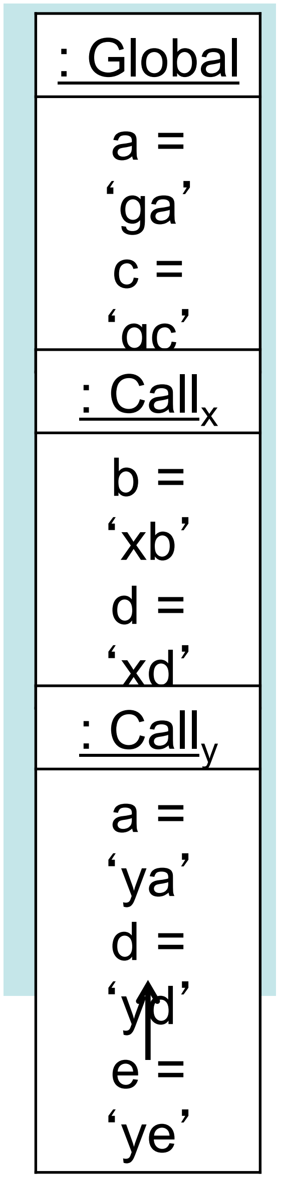
```
var a = 'ga';
function x(b) {
  c = 'gc';
  var d = 'xd';
  function y(a,d){
    var e = 'ye';
  }
  y('ya', 'yd');
}
x('xb');
```



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

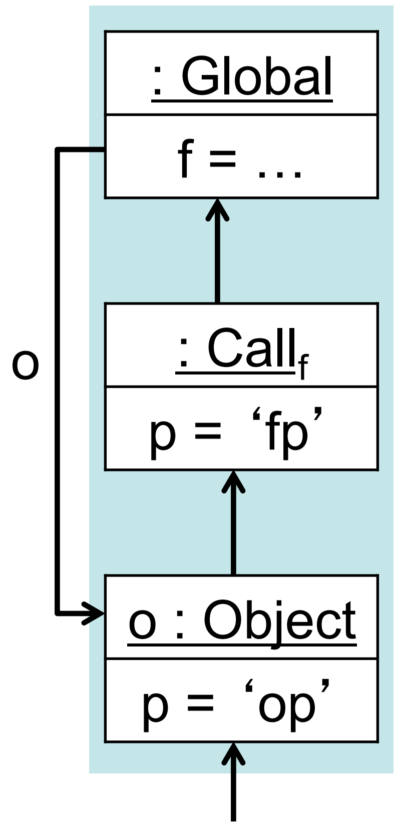


Abbreviated Member Access

`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();
```

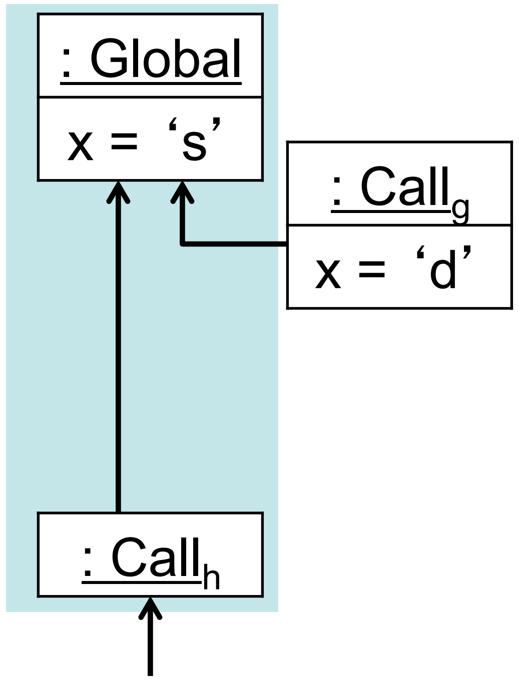


JavaScript

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
```

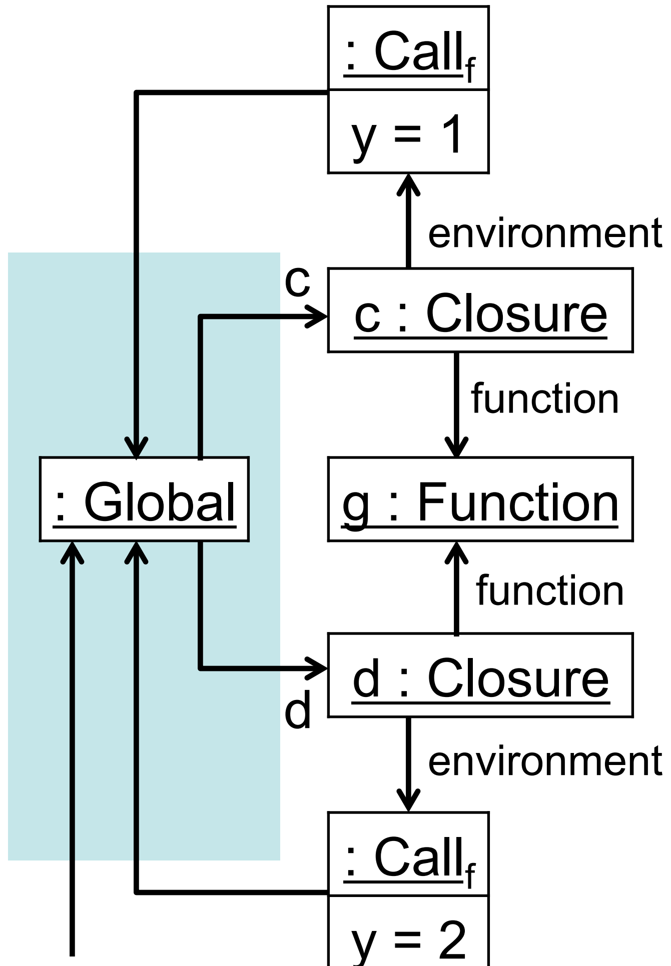


JavaScript

Closure = Function + Environment

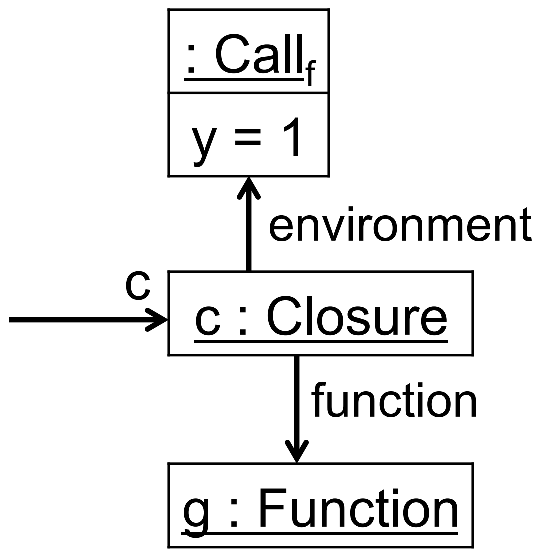
Environment =
Old scope chain

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



Concepts

Closures



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

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

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.cornell.cs.soule`**
- Convention: initialize module in anonymous function that gets called immediately
  - **`(function() {`** /* initialization code */ **`})();`**

JavaScript

Package Example

```
var edu; //put this code in file "edu/cornell/cs/Counter.js"
if (!edu) edu = {}; //only one global symbol, don't pollute namespace
if (!edu.cornell) edu.cornell = {};
if (!edu.cornell.cs) edu.cornell.cs = {};
if (edu.cornell.cs.Counter) throw new Error("Counter already exists");
(function() {
  edu.cornell.cs.Counter = {};
  var private_counter = 0; //after return only visible from closures
  edu.cornell.cs.Counter.increment = function() { private_counter++; }
  edu.cornell.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/cornell/cs/Counter.js"> </script>
</head><body>
  <script>
    var Counter = edu.cornell.cs.Counter; //short name
    document.write(Counter.read() + "<br/>\n");
    Counter.increment(); //private_counter not directly visible here
    document.write(Counter.read() + "<br/>\n");
  </script>
</body></html>
```

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

Last Slide

- Today's lecture
  - JavaScript
  - Closures
- Next lecture
  - JavaScript continued