jQuery and AJAX for Responsive Web Sites

Lecture 4
JavaScript, Part 2

Objectives

- Course recap
- Homework
- Arrays
- this
- Closures

jQuery as a Library

- Hides cross-browser differences
- Slick, consistent API for *DOM manipulation* and *AJAX*
- Useful sister projects and plugins

jQuery Selectors, Wrapped Sets

- Selectors based on CSS syntax
- Pick DOM elements to manipulate
- W.S. as collections of DOM elements
- Chaining manipulators on wrapped sets
- W.S. looks like an array + jQuery methods

jQuery $

- Just a variable name
- Overloaded
- noConflict()

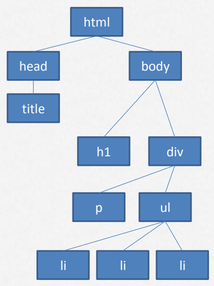
```
// pick dom elements
$('#myList li:last')

// create dom elements
$('<li>my cool list item</li>')

// ready handler
$( function() {
    // dom is ready!
} );
```

DOM Event Model

- HTML and tree representation
- Event bubbling
- on() and off()
- Older methods
  bind, unbind, live, die



JavaScript Scoping

- Use **var** for local variables
- Otherwise things go to global scope
  - Really a special global object: window
- **"use strict"** to catch coding mistakes

JavaScript Functions as 1st Class

- Pass to functions as arguments
- Return from functions as results
- Creatable on the fly
- Can be left anonymous

JavaScript Objects

- Collection of key/value data plus prototype
- Object literal notation

```
var myObj = { 'propA': 1000,
              'propB': true,
              'propC': [1,2,3,4],
              'propD': {'a': 1},
              'propE': "hi there!" };
```

- **new** and Constructor functions

JavaScript Object Manipulation (1)

Setting and getting object properties

```
var myObj = new Object;

myObj.propA = "hi there!";
myObj["crazy prop name"] = true;

console.log( myObj.propA );
```

JavaScript Object Manipulation (2)

Checking for props, removing them

```
var myObj = {'propA': 12};

// check if property is set
console.log( 'propA' in myObj );

// delete a property
delete myObj.propA
```

JavaScript Object Manipulation (3)

Iterating over properties

```
var myObj = {'propA': 12, 'propB': "meowmix"};

for ( var prop in myObj ) {
    console.log( myObj[prop] );
}
```

Prototype based inheritance

- Assign prototype via Ctor function
- Prototype chain for resolving properties

Homework

Assignment 1: Monkey Puzzle

```
// EXERCISE 1: return DOM element with monkeyPuzzle id
return $('#monkeyPuzzle');

// EXERCISE 2: get block at given row and col.
return $( '#monkeyPuzzle .puzzleRow:eq('+row+') .puzzleBlock:eq('+col+')' );

return $( '#monkeyPuzzle .puzzleRow' ).eq( row ).children().eq( col );

// EXERCISE 3: return blocks in odd numbered puzzle rows
return $('#monkeyPuzzle .puzzleRow:odd .puzzleBlock');

// EXERCISE 4: return blocks in even numbered puzzle rows
return $('#monkeyPuzzle .puzzleRow:even .puzzleBlock');

// EXERCISE 5: return all blocks along a column
return $('#monkeyPuzzle .puzzleBlock:nth-child('+col+1+')');
```

Assignment 2, Part 1: Event Bubbling

< Expected Result Demo >

JavaScript Language Wrap Up

Arrays

```
// Explicit construction
var foo = new Array;
foo[0] = “hi”;
foo[1] = “there”;


// Array literal notation
var foo = [“hi”, “there”];
```

Elements and length as properties.

Arrays

```
// mixed with object literals!
var foo = {'myArr': [1,2,3]};

Var bar = [{'a': true}, {'b': false}];
```

Array Internals (1)

- Inherits Array.prototype
- Methods
  - Push, pop
  - Sort
  - Reverse
  - Splice
  - Slice
  - ES5: map, reduce, every
- https://developer.mozilla.org/en-US/docs/JavaScript/Reference/Global_Objects/Array

Array Internals (2)

Nomenclature

- **indices** are non-negative integer prop names

yes:	0, 1, "2", 3.0, 532
no:	"meowmix"
no:	-123

- Each index is prop, but not other way
- Indexed items are array **values**.

Array Internals (3)

"Magic"

- Setting an index keeps length prop updated

- Dense arrays length = # elements
- Sparse arrays length >= # elements

< Length Demo >

Array Internals (3)

"More Magic"

What happens if you change length?

< Length Demo #2 >

jQuery Wrapped Set

- Selector results set in object indexes
- Stash the length property
- Steals Array.prototype methods
  - Almost all work on array-like objects

< jQuery Source Demo >

JavaScript Array Iteration (1)

Dense Array Loop

```
var myArr = [1,2,3,4];

for ( var i = 0; i < myArr.length; ++i ) {
    console.log( myArr[i] );
}
```

JavaScript Array Iteration (2)

Sparse Array Loop

```
var myArr = [1,2,3,4];

for ( var i = 0; i < myArr.length; ++i ) {

    // skip unset elements
    if ( !( i in myArr ) ) {
        continue;
    }

    console.log( myArr[i] );
}
```

Can also use property iteration, but careful with array like objects

Array Closing Thoughts

- Associative array: use Object
- Array values may be mixed type

  [“hi”, 123, true, {‘a’:1}]

this

this

- Reserved keyword,
  - no variable name hanky-panky
- Value of this depends on how function was called
  - very different from C++, Java, PHP

Understand **this** or else…..

Image from Wikipedia

this depends on context

Functions	Methods
Constructors	Function.call() Function.apply()

Functions

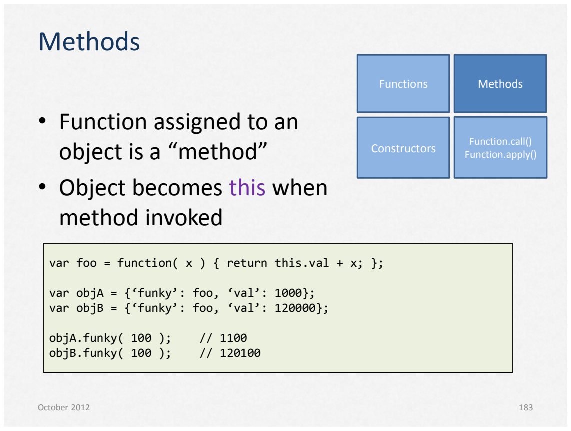
Functions	Methods
Constructors	Function.call() Function.apply()

- Global object window
  - ES3, ES5 non-strict
- undefined
  - ES5 strict mode

```
function foo() { console.log( this.niekVal ); }

// prints 1000
window.niekVal = 1000;
foo();

// fails if ES5 strict mode
“use strict”;
foo();
```



Methods

Functions	Methods
Constructors	Function.call() Function.apply()

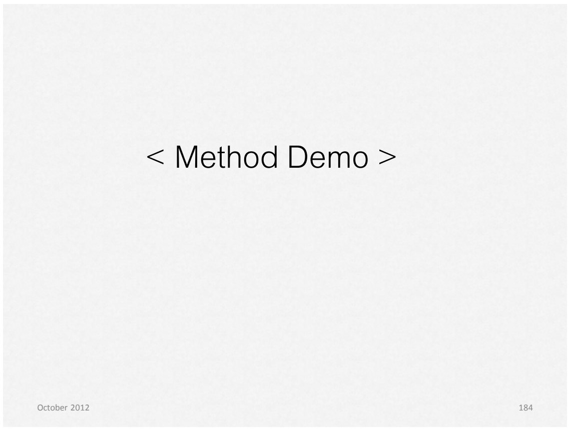
- Function assigned to an object is a “method”
- Object becomes this when method invoked

```
var foo = function( x ) { return this.val + x; };

var objA = {‘funky’: foo, ‘val’: 1000};
var objB = {‘funky’: foo, ‘val’: 120000};

objA.funky( 100 );    // 1100
objB.funky( 100 );    // 120100
```

October 2012 183

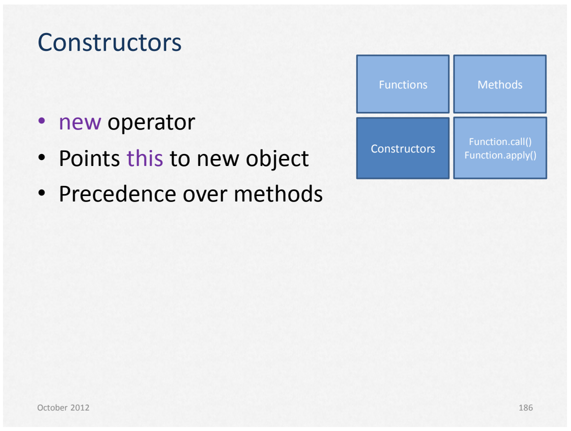


< Method Demo >

October 2012 184

< Nested Object Method Demo >

October 2012 185

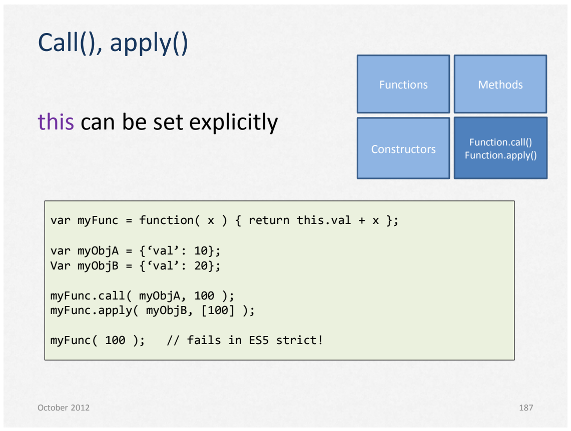


Constructors

Functions	Methods
Constructors	Function.call() Function.apply()

- new operator
- Points this to new object
- Precedence over methods

October 2012 186



Call(), apply()

Functions	Methods
Constructors	Function.call() Function.apply()

this can be set explicitly

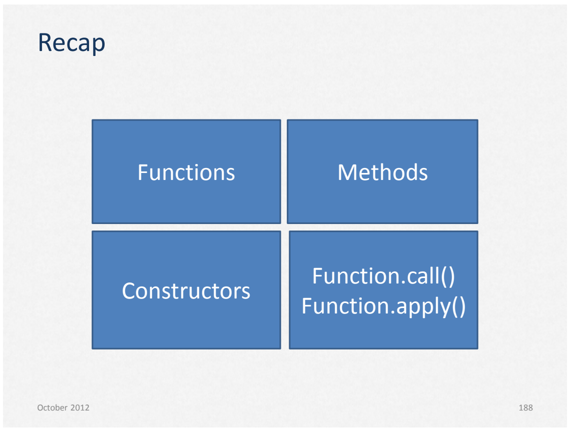
```
var myFunc = function( x ) { return this.val + x };

var myObjA = {‘val’: 10};
Var myObjB = {‘val’: 20};

myFunc.call( myObjA, 100 );
myFunc.apply( myObjB, [100] );

myFunc( 100 );   // fails in ES5 strict!
```

October 2012 187



Recap

Functions	Methods
Constructors	Function.call() Function.apply()

October 2012 188

Event handlers and this

- Event handler functions attached as props in DOM elements
- this is the DOM element!
- $(this)

< Chaining via this Demo >

closures

Function + Scope = Closure

Closures: Big Idea

- Functions hold on to scope from when they were defined, not when they're invoked

```
function outerFunc() {

    var myArr = [100,200,300];

    return function() {
        return myArr[1]++;
    };
}

outerFunc()();
```

Closures and this

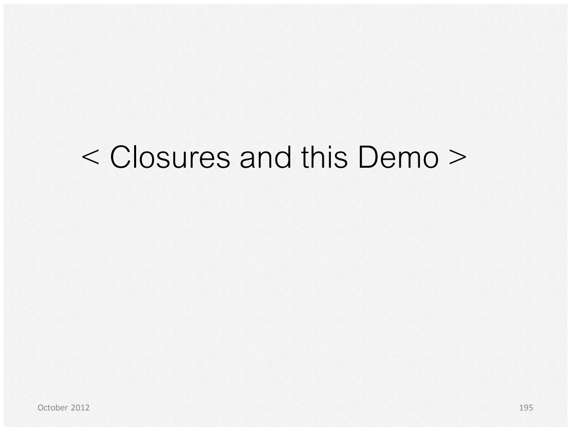
- this does not follow normal scoping!

```
var myObj = {'val': 100};

myObj.func =
    function() {

        var self = this;

        return function() {
            return self.val++;
        }
    };
```



< Closures and this Demo >

October 2012

195