FP and AOP in JavaScript

Or: “The Little Functional Language That Could”

You Probably Hate JavaScript

Most hackers do

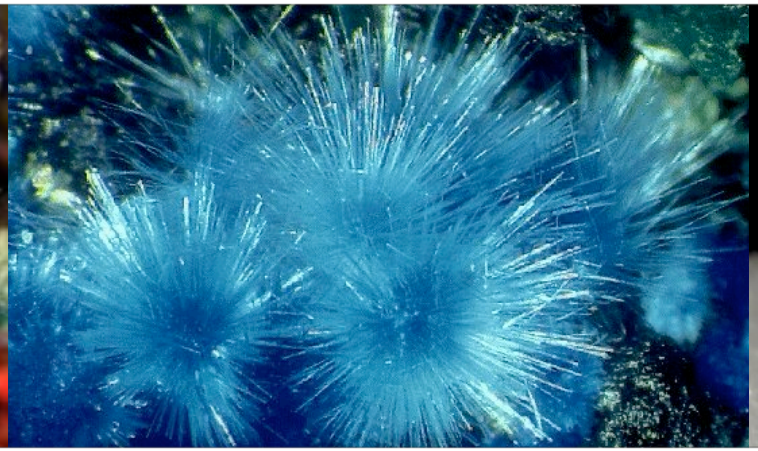
Browsers Suck

But buried down in them is a pretty cool little language

“We use it because we have to”

Never What You Expect

Animal? Vegetable? Mineral?


Whatever You Want It To Be

- Imperative to imperative programmers
- OO (with quirks) to OO programmers
- Functional to FP hackers

Most succesful scripting language *ever*?

Ubiquity

- Server-side
  - Rhino and JDK 1.6 (Java)
  - Spidermonkey and KJS (C/C++)
  - ASP classic (JScript.NET is not JavaScript)
- Client-side
  - *Every* modern browser
  - OSes and desktops
    - WSH, KJS, Konfabulator, Dashboard
  - Others: Adobe Acrobat, Flash 8.5 (ActionScript 3), etc.

Language Background

- First version in 1995 for Netscape 2.0
  - Developed by Brendan Eich as “LiveScript”
  - Renamed “JavaScript” as a marketing exercise
- Standardized via ECMA in 1997, ISO in 1998
- Not well understood
  - Heaviest users have lacked programming backgrounds
  - Easy to pigeon-hole as one “kind” of language
- Ajax/DHTML/JS community now on 3rd generation libraries

Language Features

- Lexical scope
- Functions are objects
- Functions are closures
- Anonymous function support
- Delegation via the prototype chain
  - OOP via constructor closures and prototype inheritance
- Some interpreters support continuations
- C-style syntax

Modern Programming In JS

The whirlwind tour

JavaScript Does OO!

...but it sure looks weird

Prototype-Based Inheritance

```
function FooClass(ctorArg1, ctorArg2){
}

FooClass.prototype.foo = “foo”;
FooClass.prototype.bar = “bar”;

var baz = new FooClass();
baz.foo = “my own foo”;

var thud = new FooClass();
thud.foo == “foo”;
```

Singletons

```
namespace.singleton = new function(){
  var privateVar = 0;
  this.increment = function(){
     return privateVar++;
  }

  function privateMethod(){
  }
  this.publicWrapper = function(){ return privateMethod(); }
}
```

No Interfaces...Mixins!

```
// “interfaces that actually do something”
namespace.mixinClass = function(){
  this.foo = “foo”;
  this.bar = function(){
    alert(this.foo);
  }
};
function thingerClass(){
  namespace.mixinClass.call(this);
}
(new thingerClass()).foo == “foo”;
```

Functional Programming in JS

Functional Building Blocks

- JavaScript may or not be “functional”
  - Depends on who you ask and what time of day it is
- JavaScript 1.6 adds stronger primitives
  - Array.map(), Array.filter(), Array.some(), Array.every()
- Until 1.6 arrives, we can roll our own
  - “Arrives” usually means “95+% of browsers support it”
  - Upgrade trajectories suggest half a decade
  - Downside to heterogeneous ubiquity

JavaScript Closures

```
function foo(){
  var bar = 1;
  return function(){
    return bar++;
  };
}
var baz = foo();
baz() == 1; baz() == 2;
var thud = foo();
thud() == 1; thud() == 2;
```

JavaScript Monads

(with aplogies to Shannon Behrens)

```
function MonadClass(value){
  this.value = value || undefined;
}
MonadClass.prototype.pass = function(value, cb, scope){
  if(typeof value["value"] == "undefined"){
    return new this.constructor();
  }
  // side effects go here!
  if(scope){ return cb.call(scope, value); }
  return cb(value);
}
```

Curried JavaScript

```
// load("dojo.js");
dojo.require("dojo.lang.*");

function foo(bar, baz, thud){
  alert("got at least 3: "+arguments.length);
}
var tmp = dojo.lang.curry(foo, "one");
tmp("two", "three");          // alerts
tmp("two")("three");          // also alerts
tmp("two", "three", "four"); // passes 4 args to foo
```

Curried JavaScript (contd.)

```
// this time with objects and methods
var foo = {
  bar: function(one, two){
    alert(”arguments: “+arguments.length);
  }
};
var tmp = dojo.lang.curry(foo, “bar”, “one”);
tmp(”two”, “three”, “four”); // alerts 4
tmp(”huzzah!”);          // calls foo.bar(), alerts 2
```

Dojo: A Standard Library for JS

- Functional wrappers
  - dojo.lang.map
  - dojo.lang.filter
  - dojo.lang.every (like Python itertools.all)
  - dojo.lang.some (like Python itertools.any)
  - dojo.lang.unnest
  - dojo.lang.curry
  - no “zip” or “reduce” (no user request for them)

Aspect Oriented JavaScript

AOP in JavaScript

- Pre-processor is unworkable
  - “Refresh” is “make all” for the web
- Runtime AOP is possible
  - Function objects can be “moved”
  - Wrapper functions (joinpoints) inserted in their place
  - Closures allow joinpoints to maintain state
- dojo.event.connect() implements joinpoints and advice
- Pointcuts and queries not currently implemented

dojo.event.connect()

- Provides following advice types
  - before
  - before-around
  - around
  - after
  - after-around
- Normalized interface for browser DOM and JavaScript calls
- Fixes memory leaks on IE

Syntactic Workarounds

- Can’t retrieve parent Object from Function Object
  - Reference to “foo.bar” will not allow you to determine foo
  - JavaScript hash/dot duality to the rescue!
    - foo.bar == foo[”bar”]
  - Therefore:
    - connect(foo, “bar”) == connect(foo, foo.bar)

Events for Everyone!

```
var tmpNode = dojo.byId(”someNode”);
var foo = { bar: function(){} };


dojo.event.connect(tmpNode,“onclick”, foo, “bar”);
dojo.event.connect(foo, “bar”, function(){
  alert(”foo.bar was called”);
});
// clicking on the node now will throw an alert
```

AOP Events (contd.)

```
var foo = { bar: function(){ alert(”foo.bar”); } };
var baz = { qux: function(){ alert(”baz.qux”);} };

dojo.event.connect(”before”,
    foo, “bar”,
    baz, “qux”);
foo.bar(); // alerts “baz.qux” and then “foo.bar”
```

Questions?

These slides available at:

http://alex.dojotoolkit.org/

Thanks For Supporting Open Source!