From Basic to Advanced JavaScript

for Developers

PRESENTATION BY:
PRATIK PATEL | CTO | TRIPLINGO | @PRPATEL PRATIK@MYPATELSPACE.COM

Twitter: @prpatel
Email: pratik@mypatelspace.com

TOPICS

- INTERMEDIATE JAVASCRIPT
- DESIGN PATTERNS
- FUNCTIONAL JAVASCRIPT (TIME PERMITTING)

JavaScript: the red-headed stepchild of programming langs

(with apologies to those who have red-headed stepchildren)

JavaScript Core

- INTERPRETED
- DYNAMIC TYPING
- JAVASCRIPT OBJECTS ARE ASSOCIATIVE ARRAYS + PROTOTYPES
- FIRST CLASS FUNCTIONS (FUNCTIONS ARE OBJS)
- CLOSURES!

JS OO

- LITERALS ARE OBJECTS
- FUNCTIONS ARE OBJECTS
- NEW KEYWORD

OBJECT LITERAL

```
var myObjectLiteral = {
  variableKey: variableValue,
    functionKey: function () { // ...
    };
};
```

Prototype

- JAVASCRIPT IS A PROTOTYPE LANGUAGE
- OBJECT ORIENTED SIMILAR TO PROTOTYPICAL
- OTHER PROTOTYPE LANGUAGES: IO, LUA, SELF
- EACH OBJECT INHERITS FROM A PROTOTYPE OBJECT

7 LANGS IN 7 WEEKS

PROTOTYPE OBJECTS

```
function Gadget(name, color) {
  this.name = name;
  this.color = color;
  this.describe = function(){
   return 'color: ' + this.color;
  }
}
var gadget = new Gadget(‘ipod’, ‘blue’);
// add more fields/functions to existing
// object instance
gadget.price = 100;
gadget.describe = function() {
  return 'price: ' + this.price;
};
```

PROTOTYPE OBJECTS

```
function Gadget(name, color) {
  this.name = name;
  this.color = color;
  this.describe = function(){
   return 'color: ' + this.color;
  }
}

Gadget.prototype.price = 100;
Gadget.prototype.describe = function() {
  return 'price: ' + this.price;
};
```

SIDEBAR: JS EXECUTION

- I'M RUNNING JS DIRECTLY USING NODE.JS
- THESE EXAMPLES WILL WORK IN THE BROWSER - JUST NEED TO IMPORT/LOAD UNDERSCORE.JS

what happens if you call gadget.describe()?

JavaScript Proto's

- OBJECTS ARE PASSED BY REFERENCE (NOT COPY)
- PROTOTYPE LANGS ARE "LIVE"
- MEANS THAT MODIFICATION ON PROTOTYPES CAN BE DONE AT ANY TIME
- EVEN OBJECTS CREATED _BEFORE_ PROTOTYPE CHANGE INHERIT THESE CHANGES
- OWN PROPERTY THEN PROTOTYPE PROPERTY

JavaScript objects

- "OBJECT" IS THE ROOT OF ALL OBJECTS
- ALL NON-PRIMITIVES ARE OBJECTS
- ARE ENUMERABLE EXCEPT FOR BUILTINS:

  CONSTRUCTOR, LENGTH
- `hasOwnProperty('name')`
- `propertyIsEnumerable('name')`

ENUMERABLE

```
function Gadget(name, color) {
  this.name = name;
  this.color = color;
  this.describe = function(){
   return 'color: ' + this.color;
  }
}
Gadget.prototype.price = 100;
var gadget = new Gadget(‘ipod’, ‘blue);

// what will this return?
gadget.propertyIsEnumerable(‘price’);
```

JAVASCRIPT BASICS

PRATIK PATEL | CTO

TripLingo

JavaScript Functions

- USED FOR OBJECT CONSTRUCTION
- USED FOR "METHOD" INVOCATION
- VARIADIC: ARBITRARY NUMBER OF PARAMS TO FUNCTION CALLS
- ANONYMOUS FUNCTIONS

VARIADIC

```
function sum(x,y) { return x+ y }
function sum() {
    var i,
        x = 0;
    for (i = 0; i < arguments.length; ++i) {
        x += arguments[i];
    }
    return x;
}
sum(1, 2, 3); // returns 6
```

ANONYMOUS FUNCS

```
// this is an anonymous function
append.addEventListener('click', function ()
{
  console.log(‘click called on append’);
});
```

```
// this is not
var printAppend = function () {
 console.log(‘click called on append’);
});

append.addEventListener('click', printAppend)
```

WHICH ARE OBJECTS?

```
var speakers = [‘summers’, ‘pratik’];
var topics = {name: ‘pratik’, title:
 ‘advanced javascript’};
var foo = new Object();
```

JAVASCRIPT SCOPE QUIZ

WHAT DOES THIS DO?

```
var foo = 1;
function bar() {
  if (!foo) {
    var foo = 10;
  }
  alert(foo);
}
bar();
```

WHAT ABOUT THIS?

```
var a = 1;
function b() {
  a = 10;
  return;
  function a() {}
}
b();
alert(a);
```

WHAT ABOUT THIS?

```
var a = 1;
function b() {
  a = 10;
  return;
  function a() {}
}
b();
alert(a);
```

HOW DO WE FIX THIS?

```
function foo() {
  var x = 1;
  if (x) {
    (function () {
      var x = 2;
      // some other code
    }());
  }
  // x is still 1.
}
```

HOISTING

```
function foo() {
   if (false) {
      var x = 1;
   }
   return;
   var y = 1;
}
```

```
function foo() {
   var x, y;
   if (false) {
      x = 1;
   }
   return;
   y = 1;
}
```

Scoping Rules

- ONLY FUNCTION LEVEL SCOPE
- LANGUAGE DEFINED: THIS AND ARGUMENTS
- FORMAL PARAMS: NAMED PARAMS
- FUNCTION DECLARATIONS: FUNCTION FOO() {}
- VAR DECLARATIONS: VAR FOO;

COMMON JAVASCRIPT TECHNIQUES

PRATIK PATEL | CTO

TripLingo

Techniques

- PARASITIC INHERITANCE
- OBJECT AUGMENTATION (NO NEED TO CREATE A NEW CLASS, JUST ADD IT!)
- FUNCTION PASSING
- CLOSURES & CALLBACKS
- TEMPORARY SCOPES
- COMPARISON

PARASITIC INHERITANCE

```
Shape = {name: 'Shape'};
Shape.prototype.toString = function()
  {return this.name;};

function Rectangle(width, height) {
  var rect;
  var P = function() {};
  P.prototype = Shape;
  rect = new P();
  rect.width = width;
  rect.height = height;
   rect.name = ‘Rectangle’;
  return rect;
}
```

CALLBACKS

```
function exec_random(arg1, arg2, callback) {
  var my_number = Math.ceil(Math.random() *
    (arg1 - arg2) + arg2);
    callback(my_number);
}
// call the function
exec_random(5, 15, function(num) {
  // this anonymous function will run when
  // the callback is called
  console.log("callback called! " + num);
});
```

Closures

- A STACK FRAME WHICH IS NOT DEALLOCATED WHEN THE FUNCTION RETURNS
- THE LOCAL VARIABLES FOR A FUNCTION - KEPT ALIVE AFTER THE FUNCTION HAS RETURNED
- IF YOU USE THE FUNCTION KEYWORD INSIDE ANOTHER FUNCTION, YOU ARE CREATING A CLOSURE

CLOSURES

```
function sayHello(name) {
  var text = 'Hello ' + name;
  var sayAlert = function() { alert(text); }
  return sayAlert;
}

var hello = sayHello('pratik');
hello();
// returns: Hello Pratik
```

TEMP SCOPES

```
function foo() {
  var x = 1;
  if (x) {
    (function () {
      var x = 2;
      // temp scope for var x
    }());
  }
  // x is still 1.
}
```

COMPARISON

```
'' == '0'           // false
0 == ''             // true
0 == '0'            // true
0 === '0'          // false. WTF?

false == 'false'    // false
false == '0'        // true

false == undefined  // false
false == null       // false
null == undefined   // true
```

Comparison

- == DOES TYPE COERCION
- === DOES NOT DO TYPE COERCION
- == EQUALITY
- === IDENTITY

WHAT DOES THIS DO?

```
var c = { x: 1, y: 2 }; // or [1,2,3]
var d = { x: 1, y: 2 }; // or [1,2,3]
```

```
c == d
c === d
```

Call and Apply

- FUNC.CALL(SOMEOBJ, ARG1)
- SOMEOBJ BECOMES THE "THIS" INSIDE THE FUNC
- BECOMES THE 'DELEGATE'
- FUNC.APPLY (SOMEOBJECT, [ARG1, ARG2])
- ONE PARAM FOR THE ARGS

COFFEE BREAK :)

JAVASCRIPT MODULES

PRATIK PATEL | CTO

TripLingo

Modular JavaScript

- LARGE JAVASCRIPT PROJECTS CAN BE A PAIN
- COMMONJS
- NO SUPPORT IN BROWSERS :(
- SERVER-SIDE (NODE)
- DESKTOP/MOBILE (TITANIUM)
- PROVIDES MODULAR ENCAPSULATION AND REUSE

COMMONJS EXAMPLE

```
network = require('services/network')

// network.js
exports.login = function(_creds, _callback) {
  var creds = _creds || JSON.parse(saved);
...}

exports.createAccount = function(_creds,
_callback) {

** Browsers do NOT support CommonJS
```

BASIC MODULE

```
var UserService = (function() {
   var name = 'John Smith';
   var age = 40;
   function updatePerson() {
      name = 'John Smith Updated';
   }
   function setPerson() {
      name = 'John Smith Set';
   }
   function getPerson() {
      return name;
   }
   return {
      set: setPerson,
      get: getPerson
   };
}());
console.log(UserService.getPerson());
```

JavaScript Utils

- JAVASCRIPT HAS POOR SUPPORT FOR COLLECTIONS
- JAVASCRIPT HAS POOR SUPPORT FOR DATES
- UNDERSCORE.JS / LO-DASH
- DATE.JS / MOMENT.JS

UNDERSCORE.JS

```
_.each([1, 2, 3], function(num){ alert(num); });
_.map([1, 2, 3], function(num){ return num * 3; });
_.union([1, 2, 3], [101, 2, 1, 10], [2, 1]);
_.difference([1, 2, 3, 4, 5], [5, 2, 10]);
   var log = _.bind(console.log, console);
_.delay(log, 1000, 'logged later');
   var initialize = _.once(createApplication);
initialize();
   var renderNotes = _.after(notes.length, render);
_.each(notes, function(note) {
  note.asyncSave({success: renderNotes});
});
   var compiled = _.template("hello: <%= name %>");
compiled({name : 'moe'});
```

DATEJS

```
// What date is next thrusday?
Date.today().next().thursday();
// Add 3 days to Today
Date.today().add(3).days();
// Is today Friday?
Date.today().is().friday();
// Number fun
(3).days().ago();
// 6 months from now
var n = 6;
n.months().fromNow();
// Set to 8:30 AM on the 15th day of the month
Date.today().set({ day: 15, hour: 8, minute: 30 });
// Convert text into Date
Date.parse('today');
Date.parse('t + 5 d'); // today + 5 days
Date.parse('next thursday');
Date.parse('Thu, 1 July 2004 22:30:00');
```

JAVASCRIPT DESIGN PATTERNS

PRATIK PATEL | CTO

TripLingo

Creational Patterns

- CONSTRUCTOR
- FACTORY
- ABSTRACT
- PROTOTYPE
- SINGLETON
- BUILDER

Structural Patterns

- DECORATOR
- FACADE
- FLYWEIGHT
- ADAPTER
- PROXY

Behavioral Patterns

- ITERATOR
- MEDIATOR
- OBSERVER
- VISITOR

NAMESPACES

- NO BUILT-IN NAMESPACE ABILITY
- ESSENTIAL FOR LARGE CODEBASES
- ESSENTIAL FOR REUSABILITY

NAMESPACES

```
var app = {};
app.services = {};
app.services.UserService = (function () {
    .....
    return {
        login: login,
        logout: logoutInternal,
        user: username
    }

}());

app.service.UserService.login('pratik', 'mypassword');
app.service.UserService.logout();
```

NAMESPACES II

- USING JAVASCRIPT OBJECT LITERALS
- SINGLE "GLOBAL" OBJECT

Pattern: Module

- MODULE
- REVEALING MODULE
- ENCAPSULATION!

OBJECT LITERAL

```
var myObjectLiteral = {
    variableKey: variableValue,
    functionKey: function () { // ...
    };
};
```

MODULE PATTERN

```
var testModule = (function () {
     var counter = 0;
     return {
     incrementCounter: function () { return counter++;
     },
     resetCounter: function () {
     console.log('counter value prior to reset:' + counter);
     counter = 0; }
   };
})();
testModule.incrementCounter(); testModule.resetCounter();
```

REVEALING MODULE

```
var UserService = (function() {
   var name = 'John Smith';
   var age = 40;
   function updatePerson() {
      name = 'John Smith Updated';
   }
   function setPerson() {
      name = 'John Smith Set';
   }
   function getPerson() {
      return name;
   }
   return {
      set: setPerson,
      get: getPerson
   };
}());
console.log(UserService.get());
```

IMPORT MIXIN MOD

```
var _ = require('./underscore.js')

var UserService = (function(underscore) {
   function updatePerson(newName) {
      if (underscore.isString(newName) ){
         name = newName;
         console.log('updatePerson success')
      } else {
         console.log('updatePerson failed, value is not a string')
      }

   }

}(_));
```

COFFEE BREAK :)

Pattern: Pub Sub -

- CLASSIC PUBLISH SUBSCRIBE PATTERN
- USED FOR DYNAMIC UPDATING OF UI ELEMENTS AND DATA SYNC
- PUB/SUB USES AN EVENT ‘CHANNEL’ TO DECOUPLE THE OBSERVER AND SENDER

PUBSUB

example code from backbone:

```
window.app.Todos.trigger('reset',
{eventType: ‘refreshAll’});

window.app.Todos.on( 'reset', this.addAll,
this );
```

PUBSUB - JQUERY

```
// jQuery: $(obj).trigger("channel", [arg1,
arg2, arg3]);

$( el ).trigger( "/login", [{username:"test",
userData:"test"}] );
```

```
// jQuery: $(obj).on( "channel", [data],
fn );

$( el ).on( "/login", function( event )
{...} );
```

Pattern: Observer

- SIMILAR TO THE PUB-SUB
- OBSERVER NEEDS TO SUBSCRIBE
- COUPLING B/N OBSERVER AND SUBJECT
- PUB-SUB USES A MEDIATOR - HENCE LOOSE COUPLING

Pattern: Command

- DELEGATES INVOKING OF FUNCS/METHODS
- EXACTLY LIKE THE CLASSIC COMMAND IN JS

Pattern: Command

- INDIRECTION
- INVOCATION CHAINING

```
var CarManager = {
   requestInfo: function( model, id ){
      return "The information for " + model + " with ID " + id
      + " is foobar"; },
      buyVehicle: function( model, id ){
         return "You have successfully purchased Item " + id + ", a "
         + model;
      },

      arrangeViewing: function( model, id ){
         return "You have successfully booked a viewing of " + model
         + " ( " + id + " ) ";
      }
   };

CarManager.execute = function ( name ) {
     return CarManager[name] && CarManager[name].apply(
        CarManager, [].slice.call(arguments, 1) );
};
var result = CarManager.execute( "buyVehicle", "Ford Escort",
"453543" );
console.log(result)
```

Pattern: Facade

- SAME AS CLASSIC FACADE
- PRESENTS A FACADE TO SOMETHING (USUALLY) MORE COMPLEX UNDERNEATH

FACADE

```
app.services.UserService = (function () {
   var username;
   var hiddenVarOne;
   function login(user, password) {
      // ... do stuff to login user
      username = user;
   }

   function logoutInternal() {
      // .. do some stuff to logout user
      username = null;
   }

   return {
      login: login,
      logout: logoutInternal,
      user: username
   }

}());
```

Pattern: Factory

- SAME AS CLASSIC FACTORY
- USED FOR CREATING OBJECTS IN A SPECIFIC WAY
- MOST FRAMEWORKS USE THIS TO PROVIDE THEIR FEATURES

FACTORY

```
var Item = Backbone.Model.extend({
  idAttribute: “Id”,
  urlRoot: “/Items”
});
```

Pattern: Mixin

- INHERITING PROPERTIES FROM A BASE OBJ
- EXTENDS TYPE OF BEHAVIOUR
- AUGMENTATION

MIXINS

```
var myMixins = {
  moveUp: function(){ console.log( "move
  up" );
},
  moveDown: function(){ console.log( "move
  down" );
},
 stop: function(){
  console.log( "stop! " ); }
};
```

MIXINS II

```
function carAnimator(){
  this.moveLeft = function(){
   console.log( "move left" );
  };
}

_.extend( carAnimator.prototype, myMixins );

var myAnimator = new carAnimator();
myAnimator.moveLeft();
```

Pattern: Prototype

- EXTENDING BASE OBJECT WITH OTHERS

PROTOTYPE OBJECTS

```
var vehiclePrototype = {
  init: function ( carModel ) { this.model =
  carModel;
  },
  getModel: function () {
  console.log( "The model of this vehicle
  is.." + this.model);
} };
```

PARASITIC INHERITANCE

```
function vehicle( model ) {
  function F() {};
  F.prototype = vehiclePrototype;
  var f = new F();
  f.init( model );
  return f;
}
var car = vehicle( "Ford Escort" );
car.getModel();
```

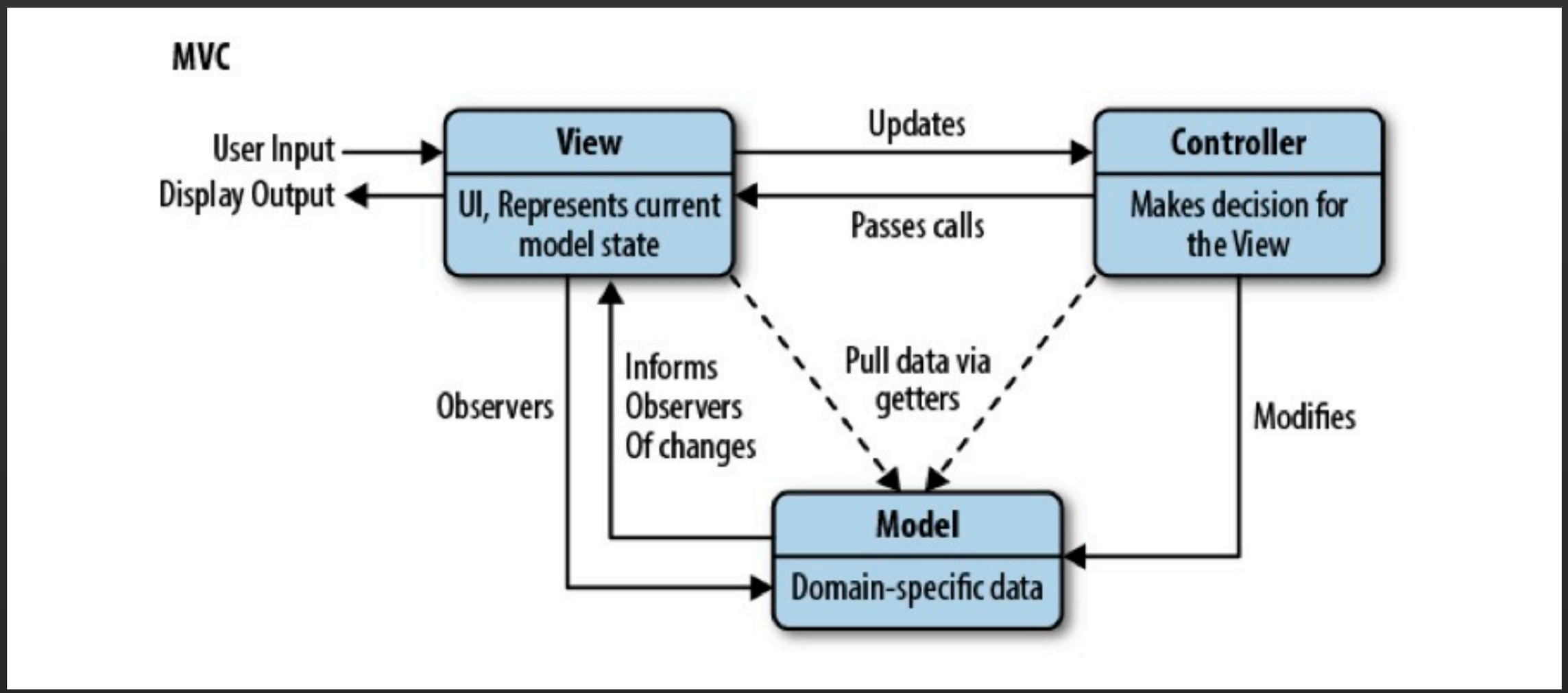
Pattern: Decorator

- SIMILAR TO MIXIN
- ANOTHER ALTERNATIVE TO SUBCLASSING
- ADD BEHAVIOUR TO EXISTING CLASS/OBJECT
- NOT DISCUSSING; COMPLEX AND DIFFICULT TO MAINTAIN

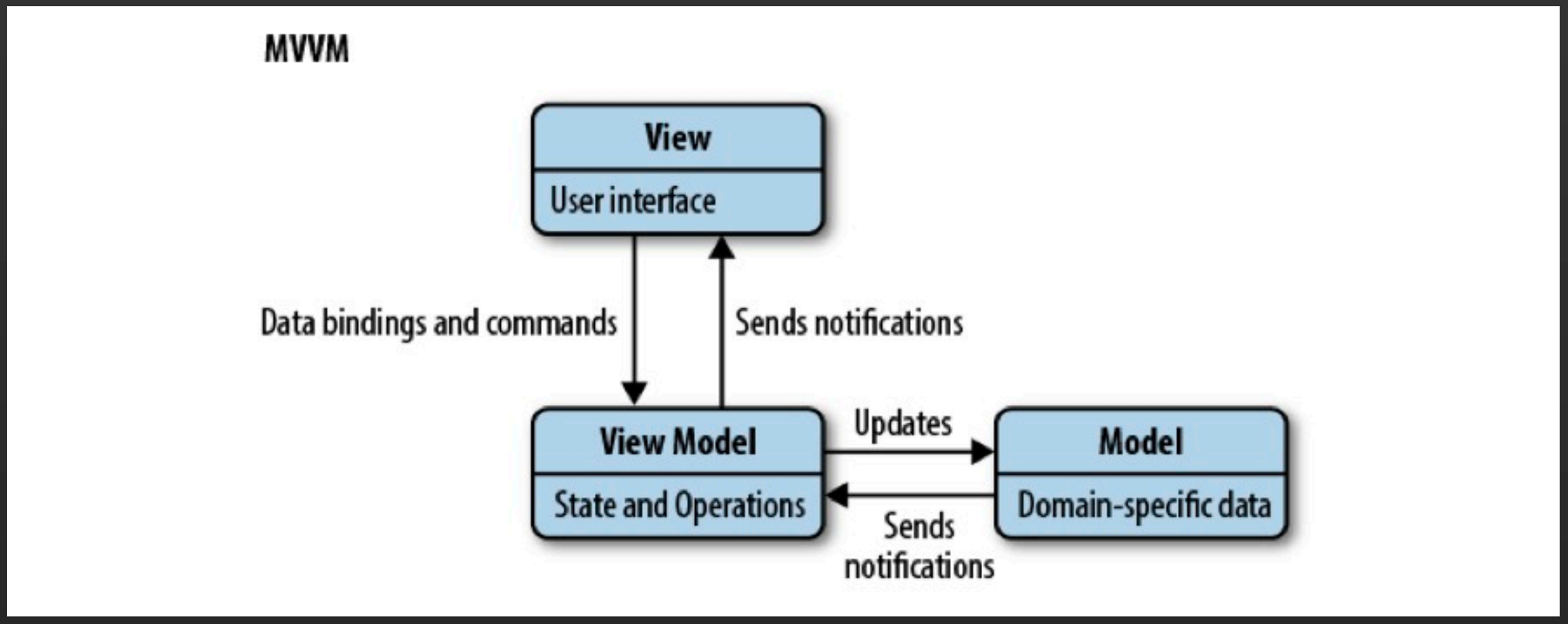
Pattern: MV*

- MVC
- MVVM
- MVP

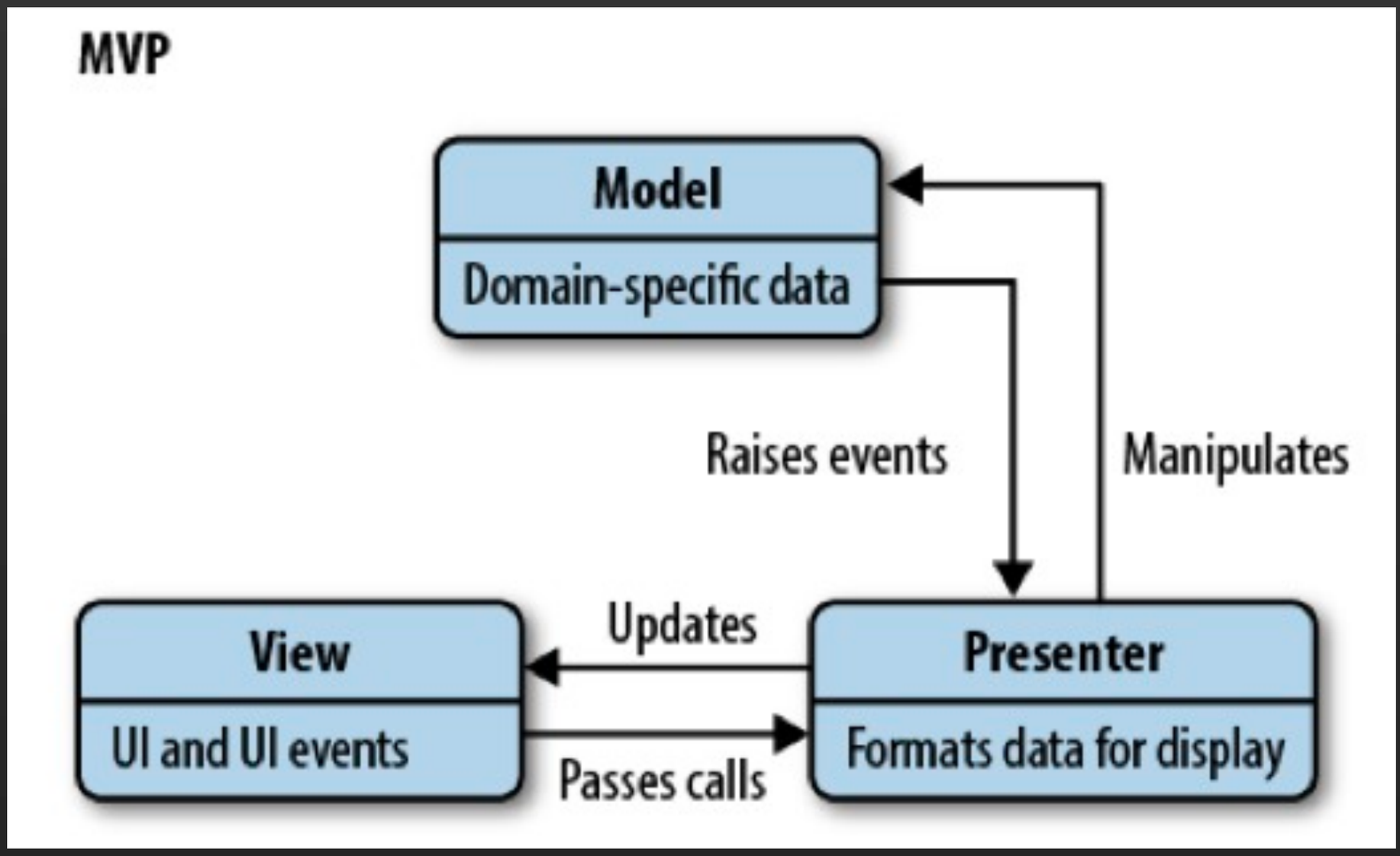
MVC


MVC
User Input
Display Output
View
UI, Represents current model state
Updates
Passes calls
Controller
Makes decision for the View
Observers
Informs Observers Of changes
Pull data via getters
Modifies
Model
Domain-specific data

MVVM

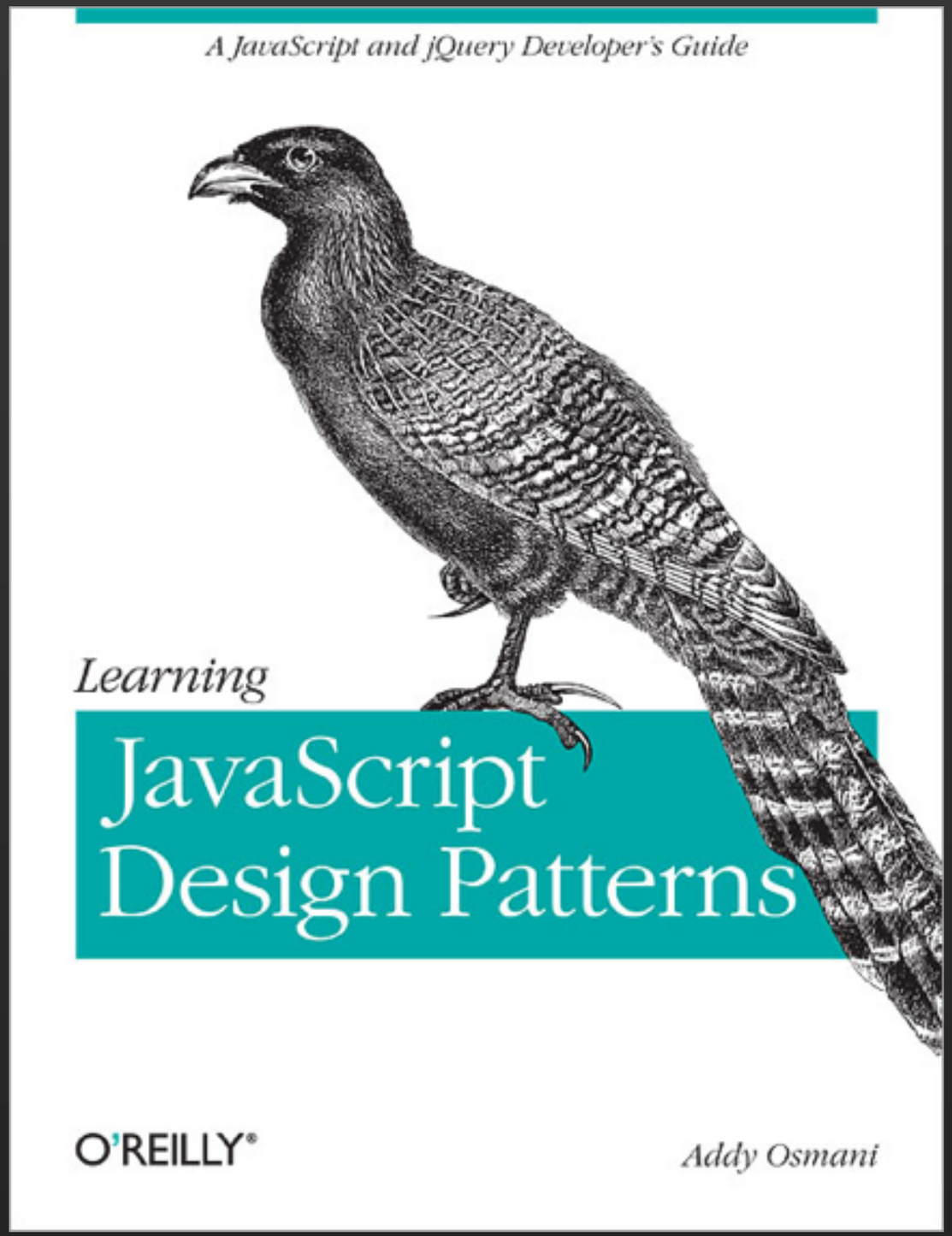

MVVM
View
User interface
Data bindings and commands
Sends notifications
View Model
State and Operations
Updates
Sends
notifications
Model
Domain-specific data

MVP


MVP
Model
Domain-specific data
Raises events
Manipulates
View
UI and UI events
Updates
Passes calls
Presenter
Formats data for display

Remember:
(Client MVC) !== (Server MVC)

GET THIS BOOK



JavaScript for Hipsters: Functional JavaScript

PRATIK PATEL | CTO

TripLingo

ONE WHO POSSESSES TASTES, SOCIAL ATTITUDES, AND OPINIONS DEEMED COOL BY THE COOL.

HTTP://WICCIMM.DEVIANTART.COM/ART/HIPSTER-SCOOBY-DOO-321867073

HTTP://WWW.HIPSTERHANDBOOK.COM/

Topics

- MOTIVATION
- BUILDING BLOCKS
- TECHNIQUES

CONVENTION OVERLOAD

PRATIK PATEL | CTO

TripLingo

CONVENTIONS, UGH

- FRAMEWORKS ADD A LAYER OF CONVENTIONS
- FUZZY LINE BETWEEN API AND CONVENTIONS

HIPSTERS ESCHEW COMMON CONVENTIONS

PROTOTYPE OBJECT HELL

PRATIK PATEL | CTO

TripLingo

PROTOTYPE OBJECTS

```
var vehiclePrototype = {
  init: function ( carModel ) { this.model = carModel },
  getModel: function () {
    console.log( "The model is.." + this.model);
  }
};

function vehicle( model ) {
  function F() {};
  F.prototype = vehiclePrototype;
  var f = new F();
  f.init( model );
  return f;
}
var car = vehicle( "Ford Escort" );
car.getModel();
```

Prototypes

- OK, SO IT'S NOT REALLY "HELL"
- DIFFERENT STYLES LEAD TO CONFUSION: STRAIGHT, PARASITIC, REVEALING

HOW ABOUT OO WITH CLASSES?

OO - COFFEESCRIPT

```
class Animal
  constructor: (name) ->
    @name = name

animal = new Animal("Parrot")
alert "Animal is a
#{animal.name}"
```

COMPILE TO JS

- COFFEESCRIPT
- TYPESCRIPT
- BUILDS ON TOP OF JS BY PROVIDING OBJECT ORIENTATION WITH INHERITANCE

OO IN JS???

Functional

- JAVASCRIPT WAS INFLUENCED BY:

- SELF (PROTOTYPE BASED LANG)

- SCHEME (FUNCTIONAL LANG)

RECIPE FOR SIMPLICITY

- HIGHER-ORDER FUNCTIONS
- SIMPLE DATA STRUCTS
- GENERALIZE AND REUSE
- FUNCTIONS OPERATE ON THESE DATA STRUCTS
- FUNCTIONS DON'T MUTATE STATE - TO A DEGREE

NO

```
function printArray(array) {
  for (var i = 0; i < array.length; i++)
    print(array[i]);
}
```

YES

```
forEach(["some", "array", "2"], print);
```

FROM IMPERATIVE TO FUNCTIONAL

WHAT DOES FUNCTIONAL MEAN?

FUNCTIONAL LANGS

- FUNCTIONS ARE FIRST CLASS
- FUNCTIONS CAN BE COMPOSED
- CLOSURES
- NO SIDE-EFFECTS

PURE FUNCTIONS

- ONLY DO 1 THING
- COMPOSE DIFFERENT FUNCTIONS TOGETHER

FIRST-CLASS FUNCS

- FUNCTIONS ARE ALSO VALUES

LOG FUNCTION

- LOG('HELLO', 'MY', 'NAME', 'IS')
- -> HELLO MY NAME IS
- JAVASCRIPT ARGUMENT KEYWORD BUILT-IN:

{'0': 'HELLO', '1': 'MY', '2': 'NAME', '3': 'IS'}

BASIC LOG FUNCTION

```
function logImperative() {
  var x = '';
  var argsLength = arguments.length
     for (var i = 0; i < argsLength; ++i) {
           x += arguments[i] + ' ';
     }
   console.log(x);
}
logImperative('hello', 'my', 'name', 'is',
'pratik')

// hello my name is pratik
```

OUR BASIC FUNC

- IMPERATIVE STYLE

FUNCTIONAL LOG 1

```
function logFunctional1() {
    console.log(
     ["LOGGER:",
     _.toArray(arguments)].join(' '));
}
logFunctional1('hello', 'my', 'name', 'is',
'pratik')

// LOGGER: hello,my,name,is,pratik
```

FIRST FUNCTIONAL

- USING THE UNDERSCORE LIBRARY FUNCTION TOARRAY
- JOINING THE 'LOGGER' STRING WITH THE ENTIRE ARRAY (HENCE THE , IN BETWEEN)

FUNCTIONAL LOG 2

```
function logFunctional2() {
    console.log((_.flatten(["LOGGER:",
    _.toArray(arguments)])).join(' '));
}
logFunctional2('hello', 'my', 'name', 'is',
'pratik')
// LOGGER: hello my name is pratik
```

LOG FUNCTION

- TOARRAY
- FLATTEN

CONGRATS, YOU JUST LEARNED FUNCTIONAL PROGRAMMING BY COMPOSING 2 FUNCTIONS!

UNDERSCORE.JS

PRATIK PATEL | CTO

TripLingo

UNDERSCORE.JS

- LIGHTWEIGHT UTILITY LIB
- CROSS-BROWSER
- FUNCTIONAL IN STYLE

NOT FIRST CLASS

```
function logTag() {
  return "LOGGER:"
}
function logFunctional3() {
   console.log((_.flatten([logTag(),
_.toArray(arguments)])).join(' '));
}

logFunctional3('hello', 'my', 'name', 'is',
'pratik')
// LOGGER: hello my name is pratik
```

FIRST CLASS

```
function logTag() {
  return "LOGGER:"
}

function logFunctional4(tagFunc) {
    console.log((_.flatten([tagFunc(),
    _.toArray(
    _.rest(arguments))])).join(' '));
}
logFunctional4(logTag, 'hello', 'my', 'name',
'is', 'pratik')
```

COMPOSABLE

"A COMBINATOR IS A HIGHER-ORDER FUNCTION THAT USES ONLY FUNCTION APPLICATION AND EARLIER DEFINED COMBINATORS TO DEFINE A RESULT FROM ITS ARGUMENTS."

—WIKIPEDIA

COMPOSING 1

```
function addLogTag(str) {
  return "LOGGER: " + str;
}
function argsToString() {
   return
_.flatten( _.toArray(arguments)).join(' ');
}
function logToConsole() {
console.log( addLogTag(
argsToString(_.toArray(arguments))));
}
logToConsole('hello', 'my', 'name', 'is',
'pratik');
```

COMPOSING 2

```
function compose (a, b) {
  return function (c) {
    // return a(b(c);
    return a(b(_.toArray(arguments)))
  }
}
function addLogTag(str) {
  return "LOGGER: " + str;
}
function argsToString() {
    return _.flatten( _.toArray(arguments)).join('
');
}
var constructLogString = compose(addLogTag,
argsToString);
console.log( constructLogString('hello', 'my',
'name', 'is', 'pratik'));
```

Closures

- A STACK FRAME WHICH IS NOT DEALLOCATED WHEN THE FUNCTION RETURNS
- THE LOCAL VARIABLES FOR A FUNCTION - KEPT ALIVE AFTER THE FUNCTION HAS RETURNED
- IF YOU USE THE FUNCTION KEYWORD INSIDE ANOTHER FUNCTION, YOU ARE CREATING A CLOSURE

CLOSURES

```
function sayHello(name) {
  var text = 'Hello ' + name;
  var sayAlert = function() { alert(text); }
  return sayAlert;
}

var hello = sayHello('pratik');
hello();
// returns: Hello Pratik
```

SIDE-EFFECTS

- PURELY FUNCTIONAL LANGS HAVE NO SIDE-EFFECTS
- BUT CONSOLE.LOG IS A SIDE-EFFECT!
- THE IDEA IS TO MINIMIZE SIDE-EFFECTS

OO vs FUN

- OO -> ACTIONS MUTATE INTERNAL OBJ STATE
- FUN -> ACTIONS MUTATE EXTERNAL DATA

OO vs FUN II

- OO -> NOUNS
- FUN -> VERBS

OO vs FUN III

- OO -> OBJECTS ARE THE ABSTRACTION
- FUN -> FUNCTIONS ARE THE ABSTRACTION

WHERE IS JS?

- OO SUPPORT
- FUNCTIONS ARE FIRST CLASS

PROBLEM WITH JS

- HAS GOOD OO AND FUNCTIONAL CONSTUCTS
- MISSING MID AND HIGH-LEVEL CONSTRUCTS

FUNC JS

- FOCUS ON THE FUNCTIONAL CONSTRUCTS

FUNC CONSTRUCTS

- COMPOSING FUNCTIONS
- CURRYING
- CHAINING
- HIGHER-ORDER FUNCTIONS

CURRY

```
function applyFirst (fn, larg) {
   return function () {
      var args = Array.prototype.slice.call(arguments, 0);
      return fn.apply(this, [larg].concat(args))
   }
}
function logFunctional5(tag) {
   console.log((_.flatten([tag,
_.toArray(_.rest(arguments))])).join(' '));
}
// non-curry
logFunctional5('LOGGER:', 'hello', 'my', 'name', 'is', 'pratik');
// curry
var logFunctional6 = applyFirst(logFunctional5, 'LOGGER CURRY:');
logFunctional6('hello', 'my', 'name', 'is', 'pratik');
```

COMPLEX DATA

- OO ALLOWS ENCAPSULATION OF DATA
- YOU'D THINK YOU _NEED_ OO FOR COMPLEX DATA
- FUNCTIONS ARE GOOD AT WORKING WITH COMPLEX DATA

CHAINING

```
var _ = require('underscore');
var lyrics = [
{line : 1, words : "I'm a lumberjack and I'm okay"},
{line : 2, words : "I sleep all night and I work all day"},
{line : 3, words : "He's a lumberjack and he's okay"},
{line : 4, words : "He sleeps all night and he works all day"}
];

var result = _.chain(lyrics)
.map(function(line) { return line.words.split(' '); })
.flatten()
.reduce(function(counts, word) {
   counts[word] = (counts[word] || 0) + 1;
   return counts;
},
{}).value();

console.log(result);
```

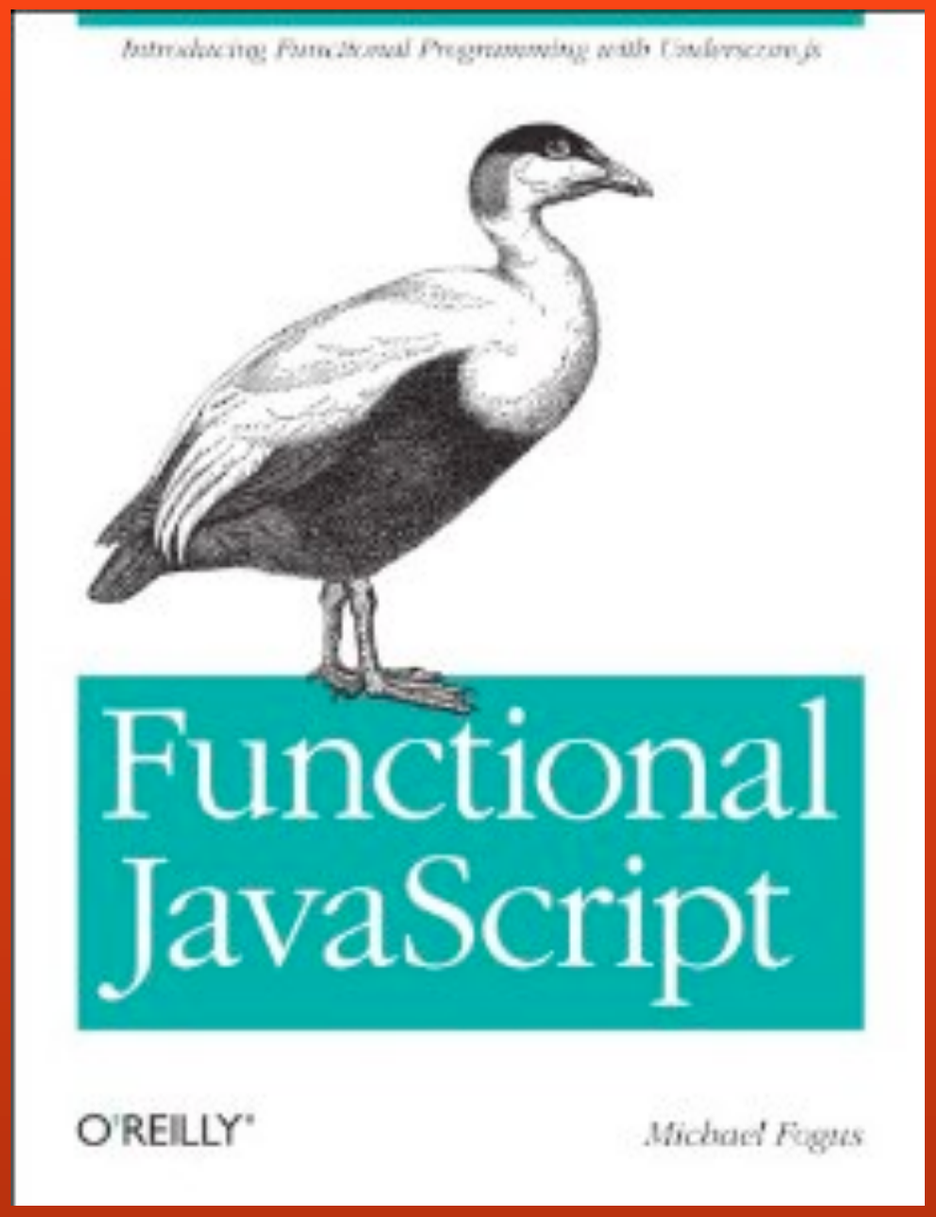
HIGHER ORDER

- TAKE A FUNCTION AS AN ARG
- RETURN A FUNCTION AS THE VALUE
- AND/OR

SIMPLE HIGHER ORDER

```
var _ = require('underscore');
var isntString = _.compose(
function(x) { return !x },
  _.isString);

console.log(isntString(1));
console.log(isntString('qwe'));
```



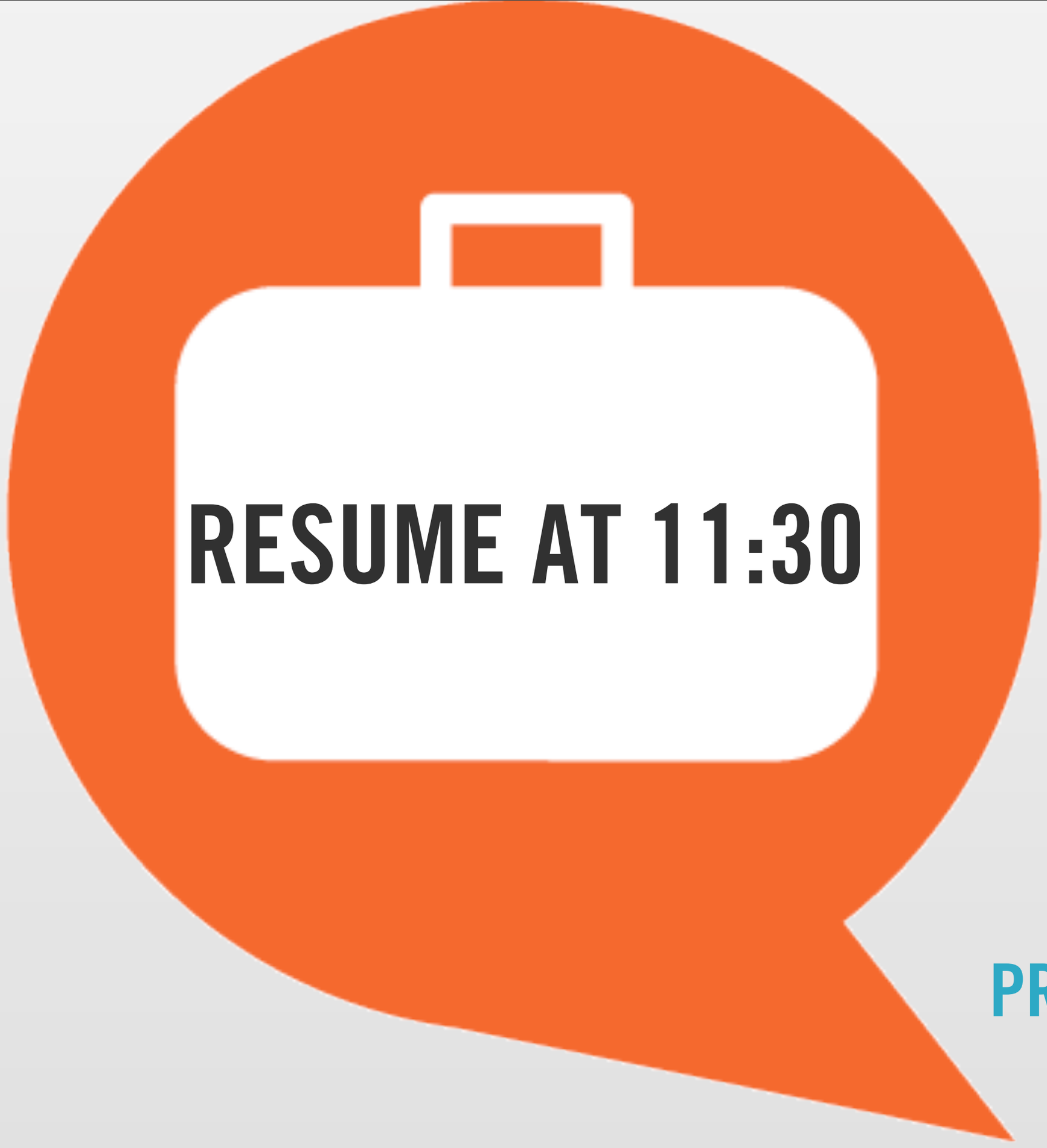
FURTHER READING - THIS SESSION WAS DEVELOPED WITH MATERIAL FROM THESE TITLES

GETTING FUN.JS

- 2 BOOKS FROM PREVIOUS SLIDE
- CLOJURE & CLOJURESCRIPT
- FUNCTIONAL REACTIVE PROGRAMMING
- HTTP://GITHUB.COM/PRPATEL

Hej då

PRATIK PATEL
@PRPATEL
pratik@mypatelspace.com



PRATIK PATEL
@PRPATEL
pratik@mypatelspace.com