An Introduction of Javascript for Java Programmers

Sam Houston Association of Computer Scientists

Ed de Luna, Oct 2014

Intended Audience

The following webinar assumes you have sound knowledge on the following:

- What is declaration, initialization and use
- Control Flow Structures (if/else, while, for...)
- Classical Inheritance (Class-Object)

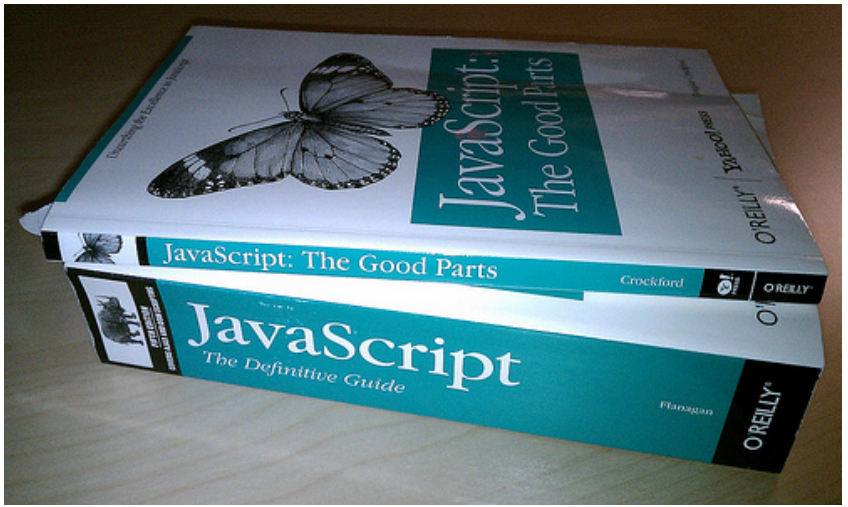
In short, this assumes you already to know how to code, preferably in Java or another C-family language.

Objectives

- To familiarize students with the history, uses, and syntax of the Javascript Programming Language
- To convey to students, familiar with the use of Java and/or Python, the differences between it and Javascript, with particular emphasis on prototypical inheritance
- To introduce students to advanced techniques and practical applications of the Javascript Programming Language
- To incite interest on the potential of Javascript as a capable programming language for future endeavors

What we will be covering

- A brief History of JS
- How to Use and Code in JS
- JS Essentials
- Prototypal Inheritance: The Heart of JS
- Advanced Uses of JS
- External Libraries
- Parting Thoughts



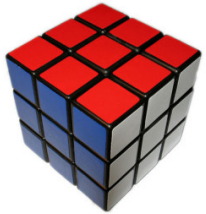
What we will not be covering

- Specific APIs or Libraries (jQuery, mootools, etc.)
- Design Patterns
- Software Engineering
- How to Website

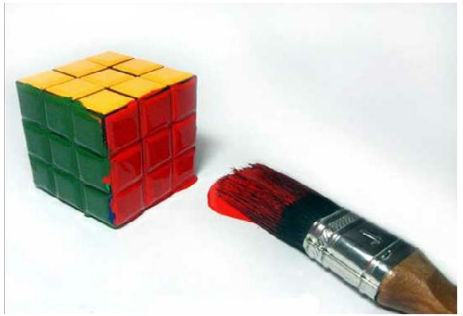
Our Methodology

1. Open up a tab on your browser
2. Put this video alongside the tab
3. Go to samhouston.github.io
4. Click on Code Examples
5. Follow Along!
6. Learn... maybe.

Coding while learning it at College



Coding for a Project in a real Job



How to Use JS

AKA You’re sitting on it right now

The Console as an IDLE

1. Open a New Tab or Window of your web browser.
2. Right click anywhere on the screen.
3. On the sub menu, please click Inspect Element. In a moment, either at the bottom or the right-hand side a window will open, containing the markup of your new tab page. You will see menus labeled Web Inspector, Elements, Network and others.
4. Click on the Button that says "Console". This is the debug console to which the JS compiler prints.
5. Click directly on the console's white space and
6. Type console.log("Hello World");
7. Click enter.

Executing JS files in HTML

The script tag refers to an external file that contains behavior written in JS, or contains the script itself.
Putting the tag at different places changes the information accompanying execution.
(refer to example1.html)

Executing JS files in HTML

It’s also possible to have multiple <script> tags referencing separate files and they can reference each other.

(refer to example2.html)

Browser-Based Tools

Firefox® Aurora

Scratchpad

Web-Based Tools

CODEPEN

runnable

Text Editors and IDEs

WebStorm

Sublime Text 3

Javascript Engines

Mozilla Spidermonkey

Google V8 Engine

Javascript Engines

Mozilla Spidermonkey

- Implemented in C++
- Whole-Method JIT Compiler
- Powers Mozilla Firefox and Aurora

Javascript Engines

- Complies JS code into machine code
- Code twice-optimized
- Available as a standalone
- Powers Chrome

Google V8 Engine

JS Syntax Essentials

AKA the Lightning Round

Javascript != Java

NO!

Operators

- Same Unary, Binary, Aritmetic, Comparison, Logical and Assignment Operators
- “+” overloaded for String concatenation
- Single and Multi-lined comments identical to Java
- One additional operator “===”, checks for same value and type

Check example3.js

Punctuation

- Statements end with semicolon
- Semicolon optional, however it clears confusion and unexpected errors
- Best practice: Place the semicolon.

```
var a = “Hello”;
var a = “Hello”
//both valid
```

check example3.js

Dynamic (Duck) Typing

- Any identifier is able to store any kind of variable.
- All variables are declared equally
- *typeof* operator checks for data type

```
var a = “Hello”;
a = 9090;
a = false;
//valid
```

check example4.js

Everything is an Object!

Variable Scope

- Variables are function scoped, instead of block scope.
- No access modifiers
- Variable Hoisting: Accessing global variables before declaration
- The Virtual Machine checks from the most local scope to the global scope searching for a variable to reference

check example5.js

Numbers

- All numbers are 64-bit floating point numbers.
- Invalid operations result in Not-A-Number: NaN.
- Nan != Nan
- Parsing strings into numbers is done through global method *parseInt()*

check example6.js

Boolean: Truthy and Falsey

Interpreted as false:

- 0
- “”
- “0”
- null

Interpreted as true:

- all numbers
- all non-empty strings
- all objects
- all arrays

check example7.js

Strings

- Declared through string literals or String() constructor
- Comaprison done through the == operator
- Contains .length property
- Contain similar Methods to Java Strings
- Concatenation achieved through + operator or *concat()*

check example8.js

Arrays

- Dynamically sized
- Direct Access via Square Bracket notation *(array[9])*
- Zero-index scheme
- Declared with square bracket literal or constructor method Array()
- Can hold any type of data
- *push()* method places elements at the end of the array and increases array length

check example9.js

Control Structures

- *if/else*
- *switch*
- *try/catch*
- *do/while*
- *while*
- *for*

Identical to their Java Counterparts!

Functions

- Declaration in the global scope requires a name.
- By default void
- Do not specify return type
- Storable in a variable, variable name is function name.

```
function hello(){
alert(“Hello World”);
}
var world =function(){
alert(“Hello World”);
}
```

check example10.js

Functions - Arguments

- Argument list is an array-like object that can be referenced.
- Possible to place arguments in a function that doesn't declare them and use them.
- Argument list lacks array methods, has .length

```
function aloha(){
alert(arguments[0]);
}
aloha(“Konnichiwa!”);
// valid
```

check example10.js

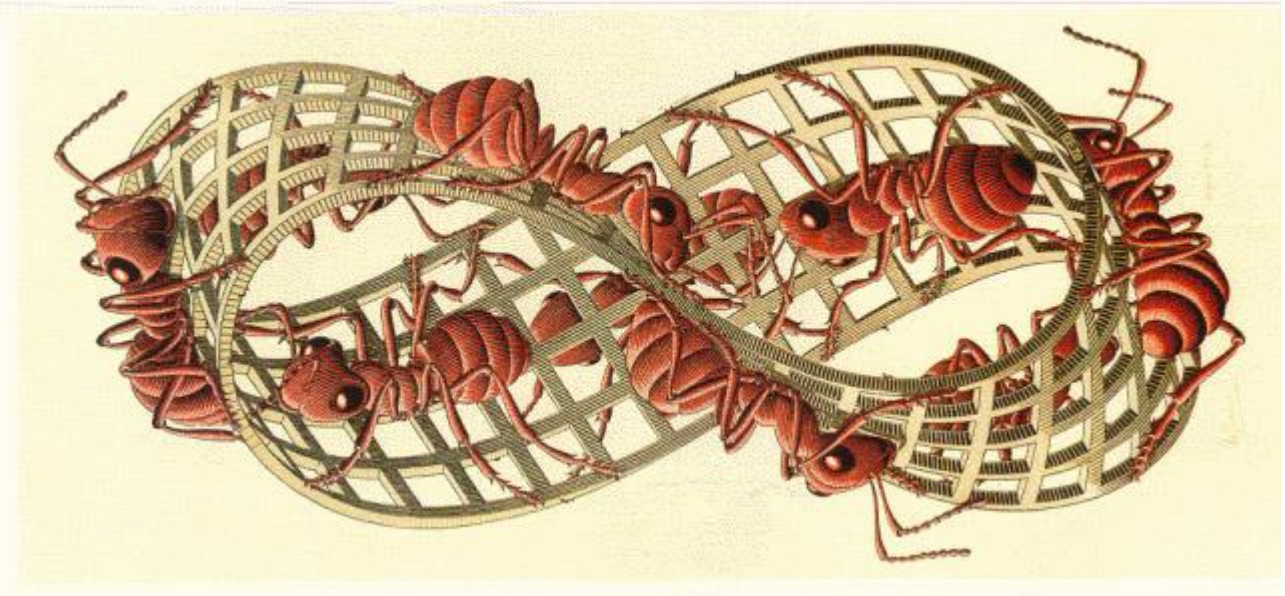
Functions as Arguments

- It’s possible to pass functions as arguments and execute them within the global context.
- Functions ARE objects, have their own methods:
  - *.bind()*
  - *.apply(),* etc.

```
function bonjour(func){
   func(“Hallo”);
}
bonjour(function(arg) {
alert(“We say ”+arg);
});
```

check example10.js

Objects are Everything



Object Declaration

- Declared through curly bracket literal or constructor new Object();
- Fields and Methods accessed through dot notation
- Also accessed through square bracket notation.

```
var a = { italia: “Ciao”};
a.esperanto = “Saluton”;
alert(a[“italia”]);
alert(b.esperanto);
//both valid
```

check example11.js

The “this” object

- The object “this” refers to changes with the scope of where its called.
- Serves to reconcile global scope with local scope.
- This contains the value of the object where it was called.

```
var msg = “Hello”;
var a = { msg: “Hola”,
print: function(){alert(this.
msg)},
printy: function(){alert(msg)}
}
a.print() // Hola
a.printy() // Hello
```

check example12.js

The Global Object

- Regular Javascript Object where all user-made functions, variables, and objects are pooled by Engine
- Has properties depending on implementation
- Also named *window*
- Variables scoped to Global are accessible by all scopes
- May create unencapsulated nightmare
- Creates the need for namespaces

check example13.js

Namespaces

- Regular Javascript Objects that contain safely-scoped variables, away from Global Scope
- Created using Objects, can be used in multiple scripts.
- Limits memory leaks and error propagation.
- Constitutes best practice in production code.
- Namespace declarable in multiple scripts like this:

```
var myNamespace =
myNameSpace || {};
```

check example13.js

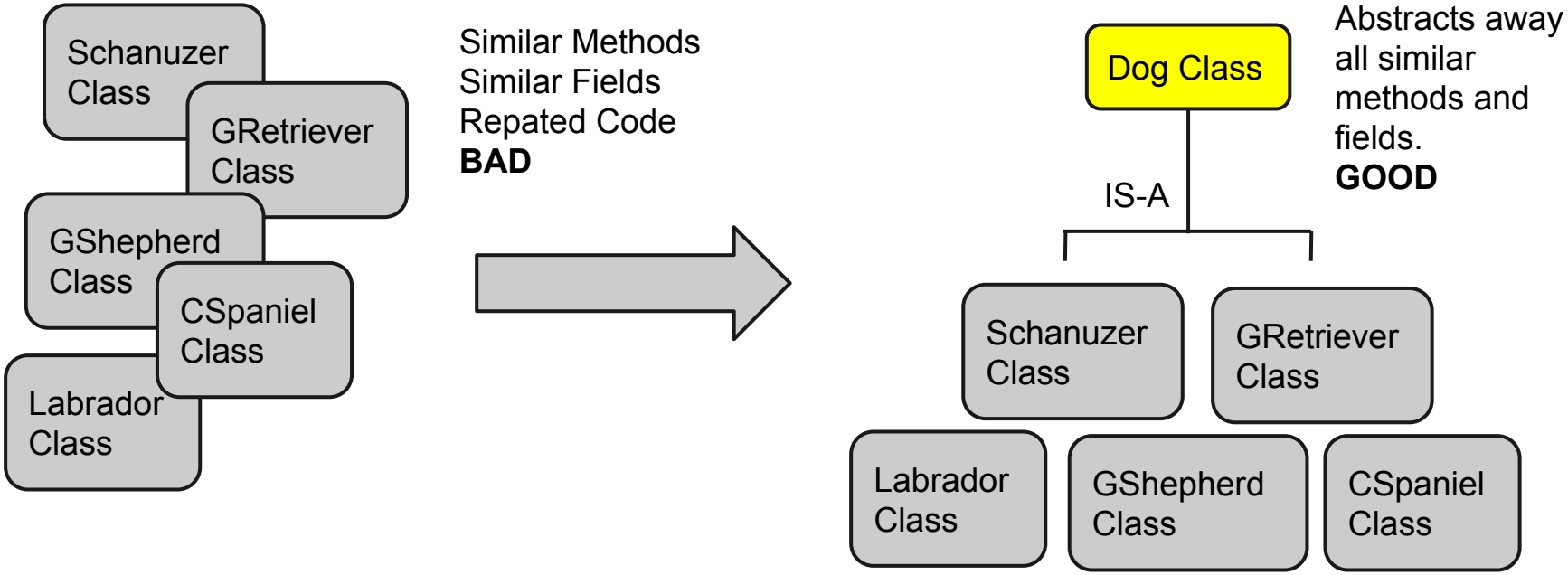
Further Reading

- **Crockford on Javascript** - T*he Javascript Programming Language* (Available on the YUI Theater)

Prototypal Inheritance

AKA the good stuff

Classical Inheritance

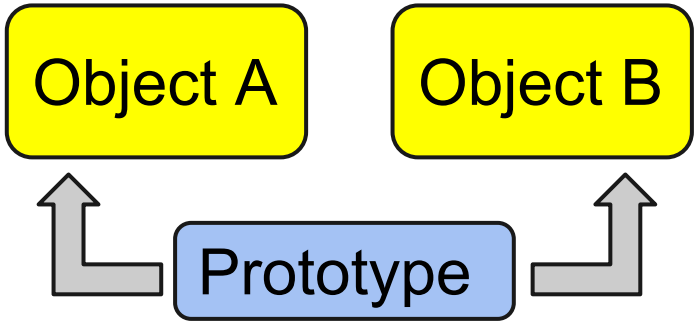


Prototypal Inheritance

Also called

- Classless
- Instance-Based Programming

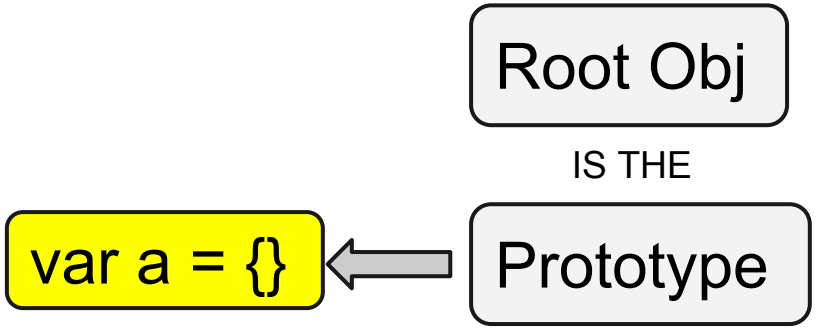
Prototypal Inheritance



Prototype: A data construction that contains initial members and their starting values. Prototypes are accessible through the .__proto__ property

check example14.js

Creating Objects From “Nothing”



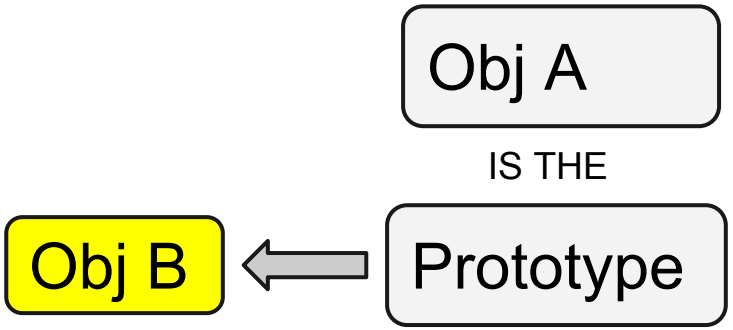
All newly created objects are cloning the prototype of the Root Object, the Root Object is a prototype, a starting point for all objects.

Prototypal Inheritance

This copies object A (the “parent”) object as the prototype of B. All properties and methods A had, now B has them.

check example14.js

Objects from other Objects



var b = Object.create(A)

or

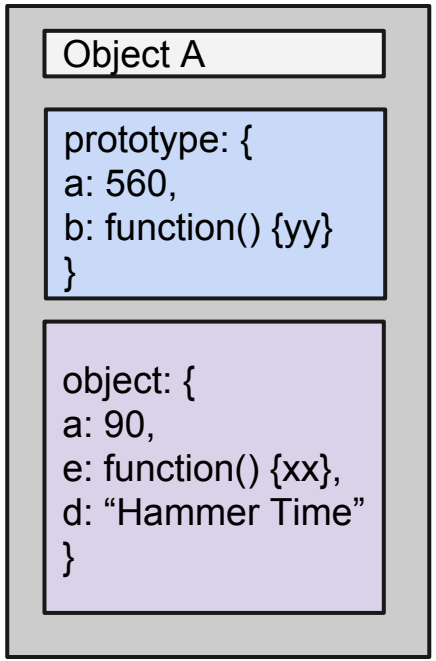
var b = {}; b.__proto__ = A;

Prototypal Inheritance

What happens when we have two variables on the same object, one declared inside the object, the other inside it's prototype? What gets called?

The Most Local Scope is the priority.

check example14.js



what gets called with...

```
A.b()
A.a
A.d
```

Answer:

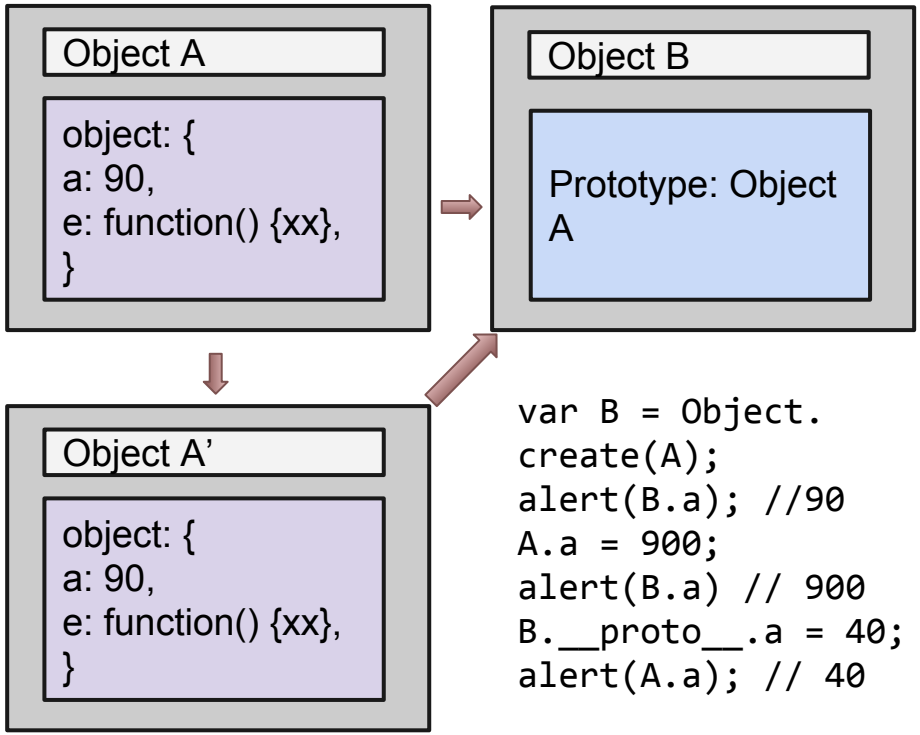
```
// function()
{yy} executes
// 90
// “HammerTime”
```

Prototypal Inheritance

What happens when we change a value in the prototype?

That change is reflected on all objects that share the prototype.

check example14.js




```
var B = Object.
create(A);
alert(B.a); //90
A.a = 900;
alert(B.a) // 900
B.__proto__.a = 40;
alert(A.a); // 40
```

Prototypal Inheritance

With prototypal inheritance is possible to:

- Have a “child” to change the parent’s “state”
- Have an object to have multiple “parents”
- Eschew the concept of a top-down inheritance tree
- Modify “parent” objects and have those changes echo in all objects that they are related to through prototype

Constructor Functions

- Holdovers from classical inheritance concepts.
- Are actually normal functions accessed through the `new` keyword.
- Do not have special syntax.
- Underlying process identical to `Object.create( );`

Called like so:

```
var Vehicle = function
Vehicle() {
        // do smth
    }
var mazda = new Vehicle();
```

Constructor Functions

What's happening under the hood:

1. Creates a new, empty object.
2. Sets the new object's prototype to the constructor's prototype (remember functions are objects)
3. Sets the constructor function as the object's constructor.
4. The constructor is executed within the context of the new object

Called like so:

```
var Vehicle = function
Vehicle() {
        // do smth
    }
var mazda = new Vehicle();

//Vehicle() is also in
mazda.prototype.constructor
```

Constructor Functions

For a time this was the only way to have inheritance in Javascript without accessing internal properties. Constructors coexist with `Object.create( )`, but are messier.

For further information, check *Javascript Objects Deconstruction* by David Walsh

Called like so:

```
var Vehicle = function
Vehicle() {
        // do smth
    }
var mazda = new Vehicle()
```

Object Augmentation

- Override prewritten and prototype properties on the fly.
- Other objects can derive their prototypes from it and keep on modifying properties even further.
- Eschews inheritance schemas entirely.

check example15.js

```
myParez = new Parenizor(0);

myParez.toString = function () {
if (this.getValue())
return this.uber('toString’);
return "-0-";
};
myString = myParenizor.
toString();
```

Parasitic Inheritance

- A function takes control of another function, injects its own code, and returns the modified function as if it was its own output.
- Makes it possible to cherry-pick properties and incorporate them into the object or function

check example16.js

```
function ZParenizor2(value) {
var that = new Parenizor(value);
that.toString = function () {
    if (this.getValue())
    return this.uber('toString');
    return "-0-" };
return that;
}
```

Traditional Classes in JS

- ECMAScript 6th Edition (or Harmony) will incorporate classes to emulate classical inhertiance.
- Constitutes Syntactic Sugar for more traditionally-minded developers
- Coming to browsers and big implementations in 2015.

Check example17.js

```
class View {
    constructor(options) {
    this.model = options.model;
this.template =  options.template;
    }
    render() {
    return
    _.template(this.template,
this.model.toObject());
    }
}
```

Try TypeScript

- Created by Andres Hejlsberg
- Curated by Microsoft
- Supports classes, interfaces, header files, static variables, generics
- All JS code is TS code
- Compiler transcodes .ts to .js
- Integrated with VS 2013
- Available for free in npm

Further Information

Prototypes, Prototypical Inheritance done Right

Source Decoded YT Channel

Advanced Uses of JS

AKA the really good stuff

Functional Programming

- A way to reduce complexity by abstracting trivial sections of code
- Uses the functions-as-objects principle to iterate through an array of data and automate tasks usually left to control structures
- Exemplified in the Array Methods
  - .forEach( );
  - .map ( );
  - .filter( );
  - .reduce( );

forEach( );

Instead of traversing elements of an array like so:

```
var names = ["Ben", "Jafar", "Matt",
"Priya", "Brian"],
var counter;
for(counter = 0; counter < names.length;
counter++) {
     console.log(names[counter]);
}
```

Traverse them like so:

```
var names = ["Ben", "Jafar", "Matt",
"Priya", "Brian"];
     names.forEach(function(name) {
          console.log(name);
     });
```

for Each() abstracts array traversal and calls the function in the args for each one

map ( );

Acts exactly the same as forEach(), but in addition of applying the function to each element of the array, it creates a new array with the modified elements.
Enables access to original data and modified data.

```
var names = ["Ben", "Jafar", "Matt",
"Priya", "Brian"];
var NAMES = names.map(function(name) {
     return name.toUpperCase();
});
console.log(names);
Console.log(NAMES);
// Ben, Jafar, Matt, Priya, Brian
// BEN, JAFAR, MATT, PRIYA, BRIAN
```

filter ( );

Fills a new array with elements of another one that return true after a comparison has been made in the callback. Passes data through a condition and then returns the results that did comply with the filter.

```
var names = ["Ben", "Jafar", "Matt",
"Priya", "Brian"];
var namesB = names.filter(function
(name) {
     return name.indexOf("B") >= 0;
});
console.log(names);
console.log(namesB);


// Ben, Jafar, Matt, Priya, Brian
// Ben, Brian
```

map( ); + filter ( );

```
var names = ["Ben", "Jafar", "Matt", "Priya", "Brian"];
var namesB = names.filter(function(name) {
          return name.indexOf("B") >= 0;
}).map(function(nam){
     return nam.toUpperCase();
});
console.log(names);
console.log(namesB);

// Ben, Jafar, Matt, Priya, Brian
// BEN, BRIAN
```

reduce( );

Reduce takes the elements of a function, an accumulator value, and lets the user decide what to do with each incoming element. The accumulator gets reset every time the function returns a value, overriding whatever was before it. Traverses an array and continuously updates a variable that would normally be outside of a traditional for loop.

```
var names = ["Ben", "Jafar", "Matt",
"Priya", "Brian"];
var biggest = names.reduce(
function(cumulator, current){
if(cumulator < current)return current;
else return cumulator;
}, names[0]);
console.log(biggest);
//Priya
```

Promises

- Not part yet of ECMA Standard, supported by External Libraries
- Methods that return a dynamically generated callback once execution of a function is done.
- Ensures execution order, one after another

- Promises are resolved (exec. success) or rejected (exec. failure)

```
var promise = new Promise(function
(resolve, reject) {
  // do a thing, possibly async, then…
  if (/*success*/)
    resolve("Stuff worked!");
  else  reject(Error("It broke"));
});
```

Modules

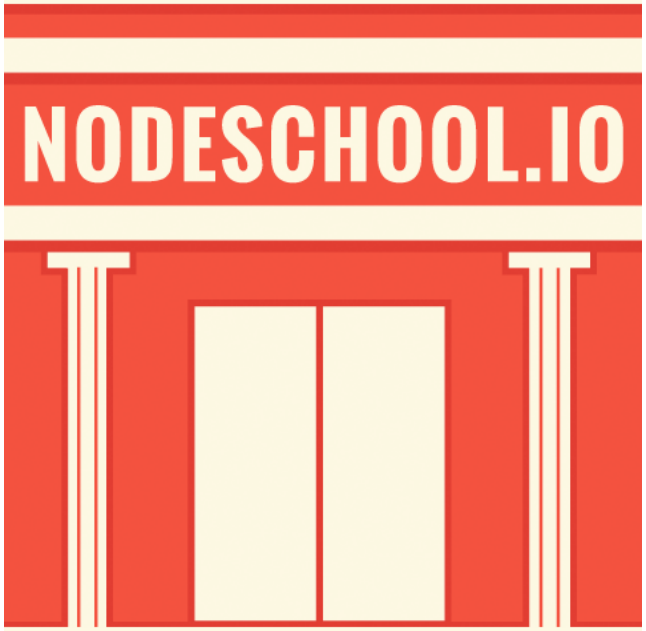
- Discreet, self-stored packages of functionality that only expose their public methods when necessary.
- Modules can bring pre-packaged code that can maximize the potential of a codebase.

- Two standards:
  - AMD (browser-friendly)
  - CommonJS (Node.js)
- ES6 has different schematic.

Check out Addy Osmani's *Writing Modular JavaScript with AMD, CommonJS and ES Harmony.*

Further Information

Nodeschool.io CLI learning tools



External Libraries

AKA the stuff people have done for you

jQuery

- A Library to modify DOM
- Handlers, Animations, Style Changes
- Almost Omnipresent
- Base for many frameworks and libraries

Phaser Game Engine

- Open-Sourced Web Framework that renders to WebGL and Canvas
- Production tested at PBS Kids and BBC
- Supports TypeScript and JavaScript
- Constantly Updated

Three.js

- Animated 3D Computer Graphics
- Abstracts low-level, WebGL code
- Supports materials, GLSL Shaders, JSON data

MV* Frameworks

ANGULARJS

by Google

BACKBONE.JS

Apache Cordova

- Mobile development framework that enables developers to build applications for mobile devices using JavaScript, HTML5 and CSS.
- Supports all major mobile OS

Node.js

- Server programming done in Javascript using the V8 Engine
- Has own package manager: npm
- Explosion of support from corporations such as WalMart and Mozilla

Node.js

- Enforces non-blocking asynchronous I/O.
- Despite being single-threaded process, can take big loads of data.
- Infinitely extensible through npm modules and packages

Notable Node.js Modules

Parting Thoughts...

AKA I’m almost done

Thanks for watching!