JavaScript Beyond jQuery

WORDCAMP DAYTON 2016

Ohio Small Business Development Centers

RAJ SOIN College of Business

WRIGHT STATE UNIVERSITY

bluehost

JETPACK

DAYTON most metro.com The Dayton Region's Online Magazine

WOO COMMERCE

GoDaddy Pro

PANTHEON Website Management Platform

yoast

BuckeyeInteractive.

DreamHost

plesk

wiredtree

WPML.ORG The WP Multilingual Plugin

easyname

bashfoo

Celek Media Consulting

LearnDash

OhioGhostWriter

Release Dates of Cars

XANDRAESIGN

OREGON

JavaScript **Beyond** jQuery

- ★ Traversing The DOM
- ★ Element Objects
- ★ Functions
- ★ Native Objects
- ★ Closures
- ★ Manipulating Data
- ★ Prototypal Inheritance
- ★ Revealing Modules
- ★ Function Composition

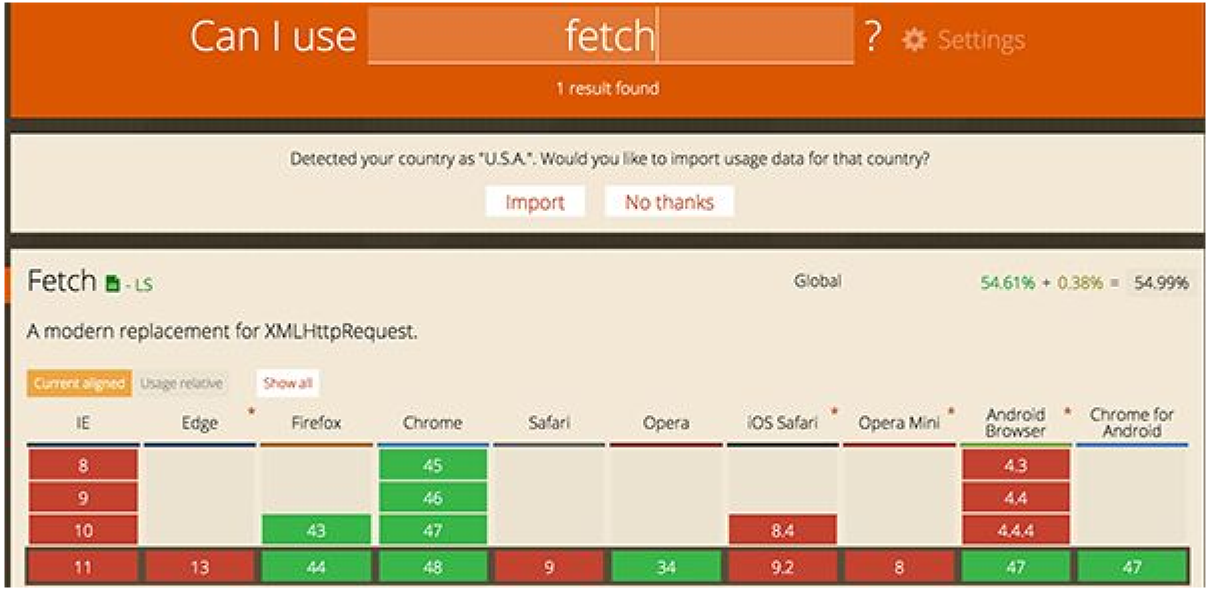
1.

TRAVERSING THE DOM

First Let's Start at the Beginning?

Why Would you Use jQuery?

- Browser Support



Can I use fetch ? Settings

1 result found

Detected your country as "U.S.A.". Would you like to import usage data for that country?

Import No thanks

Fetch - LS

Global 54.61% + 0.38% = 54.99%

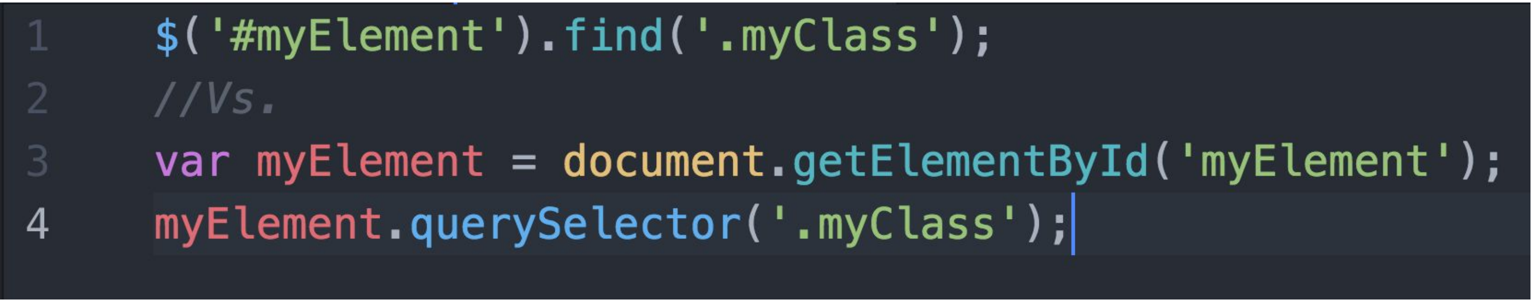
A modern replacement for XMLHttpRequest.

Current aligned Usage relative Show all

IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Chrome for Android
8			45					4.3	
9			46					4.4	
10		43	47			8.4		4.4.4	
11	13	44	48	9	34	9.2	8	47	47

Why Would you Use jQuery?

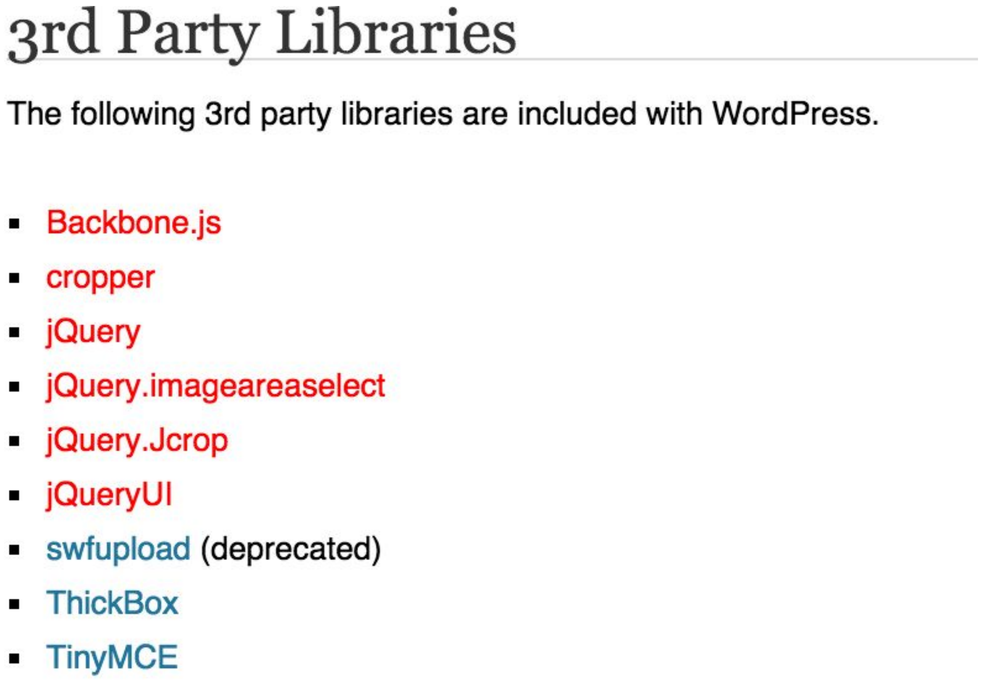
- Less Typing.
- Easier to Learn



```
$('#myElement').find('.myClass');
//Vs.
var myElement = document.getElementById('myElement');
myElement.querySelector('.myClass');
```

Why Would you Use jQuery?

- It already comes packages in WordPress



3rd Party Libraries

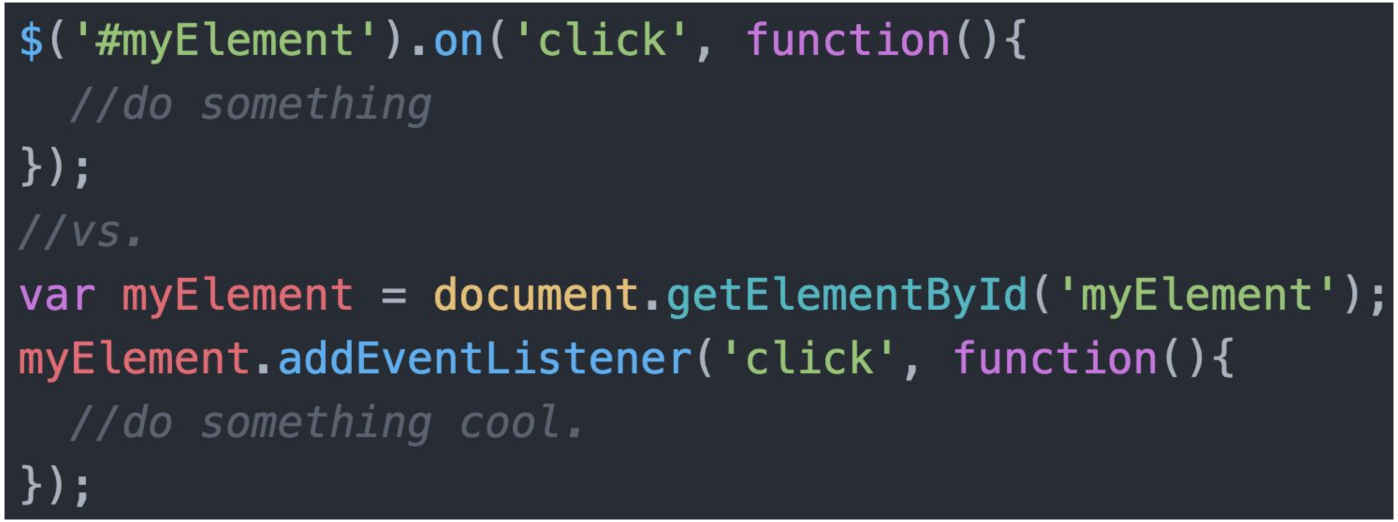
The following 3rd party libraries are included with WordPress.

- Backbone.js
- cropper
- jQuery
- jQuery.imageareaselect
- jQuery.Jcrop
- jQueryUI
- swfupload (deprecated)
- ThickBox
- TinyMCE

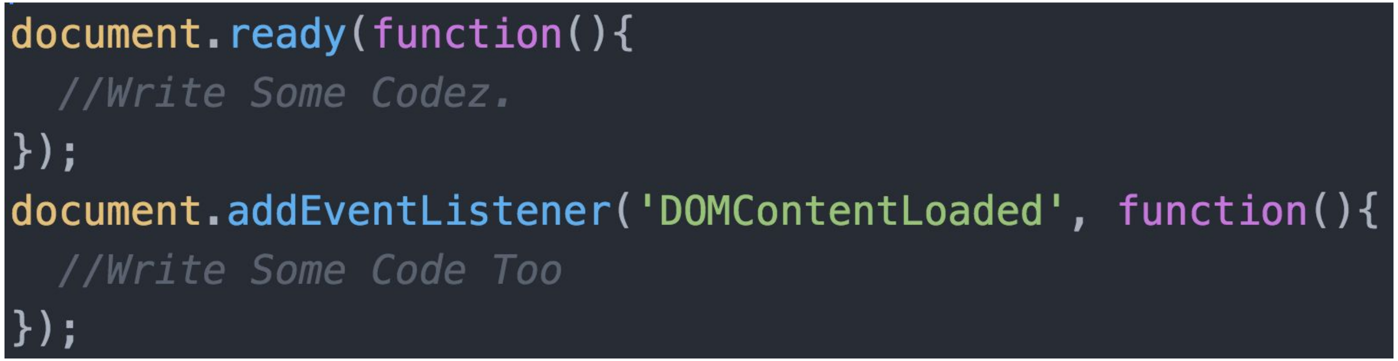
USING VANILLA JS

Is Generally a lot Easier Than You Think

Events

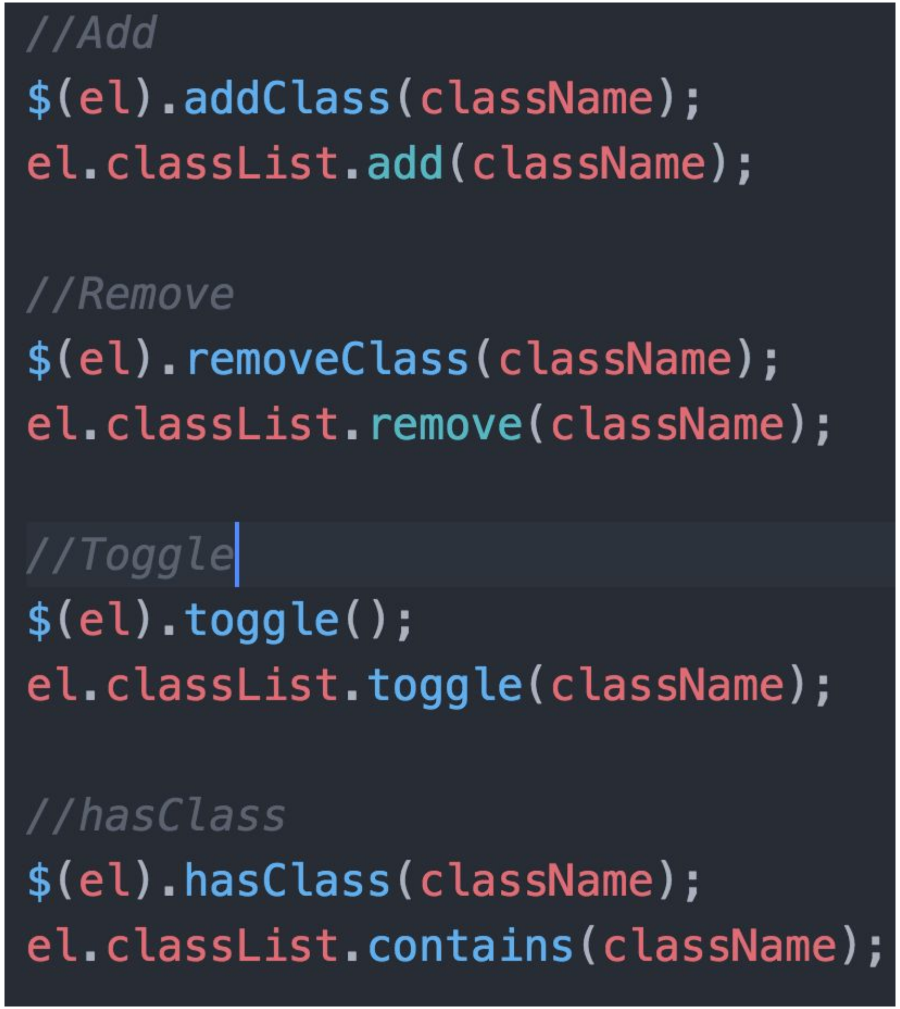


```
$('#myElement').on('click', function(){
  //do something
});
//vs.
var myElement = document.getElementById('myElement');
myElement.addEventListener('click', function(){
  //do something cool.
});
```



```
document.ready(function(){
  //Write Some Codez.
});
document.addEventListener('DOMContentLoaded', function(){
  //Write Some Code Too
});
```

Classes



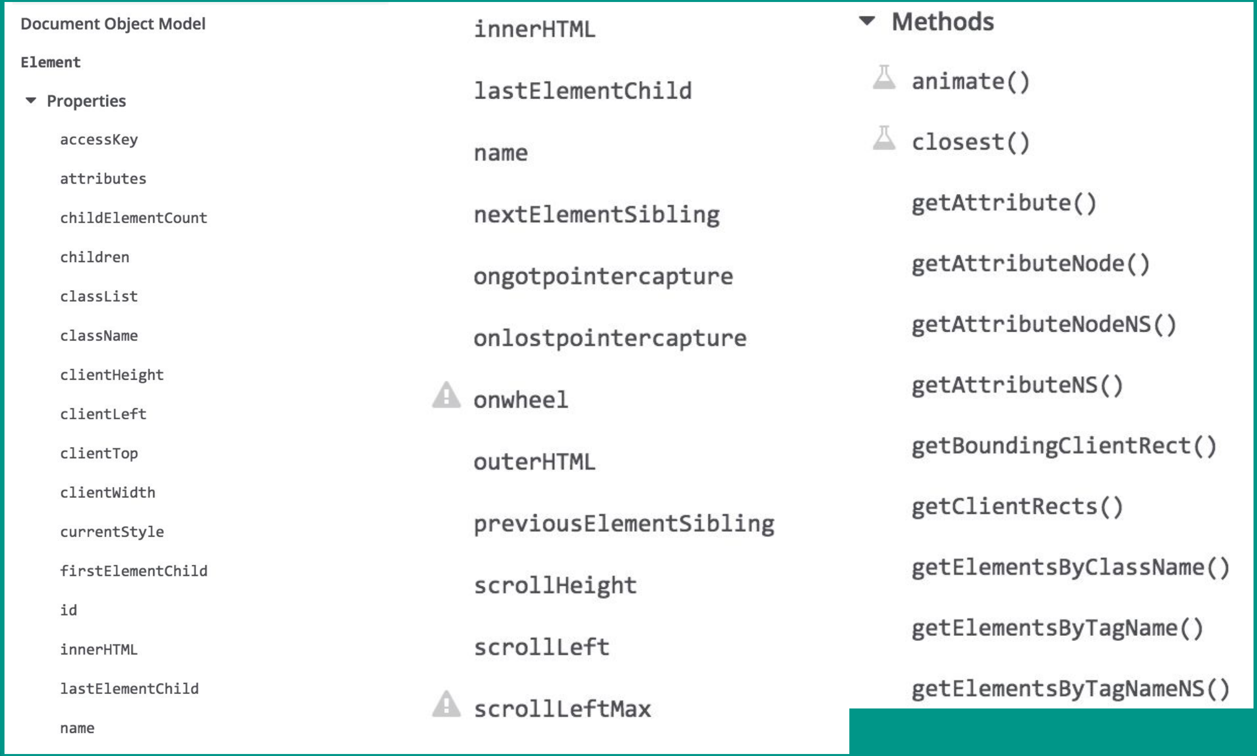
```
//Add
$(el).addClass(className);
el.classList.add(className);

//Remove
$(el).removeClass(className);
el.classList.remove(className);

//Toggle
$(el).toggle();
el.classList.toggle(className);

//hasClass
$(el).hasClass(className);
el.classList.contains(className);
```

DOCUMENT OBJECT MODEL

Let’s Take a Closer Look

THE ELEMENT INTERFACE

The Element Interface represents an Object of a Document.



Document Object Model

Element

▼ Properties

- accessKey
- attributes
- childElementCount
- children
- classList
- className
- clientHeight
- clientLeft
- clientTop
- clientWidth
- currentStyle
- firstElementChild
- id
- innerHTML
- lastElementChild
- name

- innerHTML
- lastElementChild
- name
- nextElementSibling
- ongotpointercapture
- onlostpointercapture
- ⚠ onwheel
- outerHTML
- previousElementSibling
- scrollHeight
- scrollLeft
- ⚠ scrollLeftMax

▼ Methods

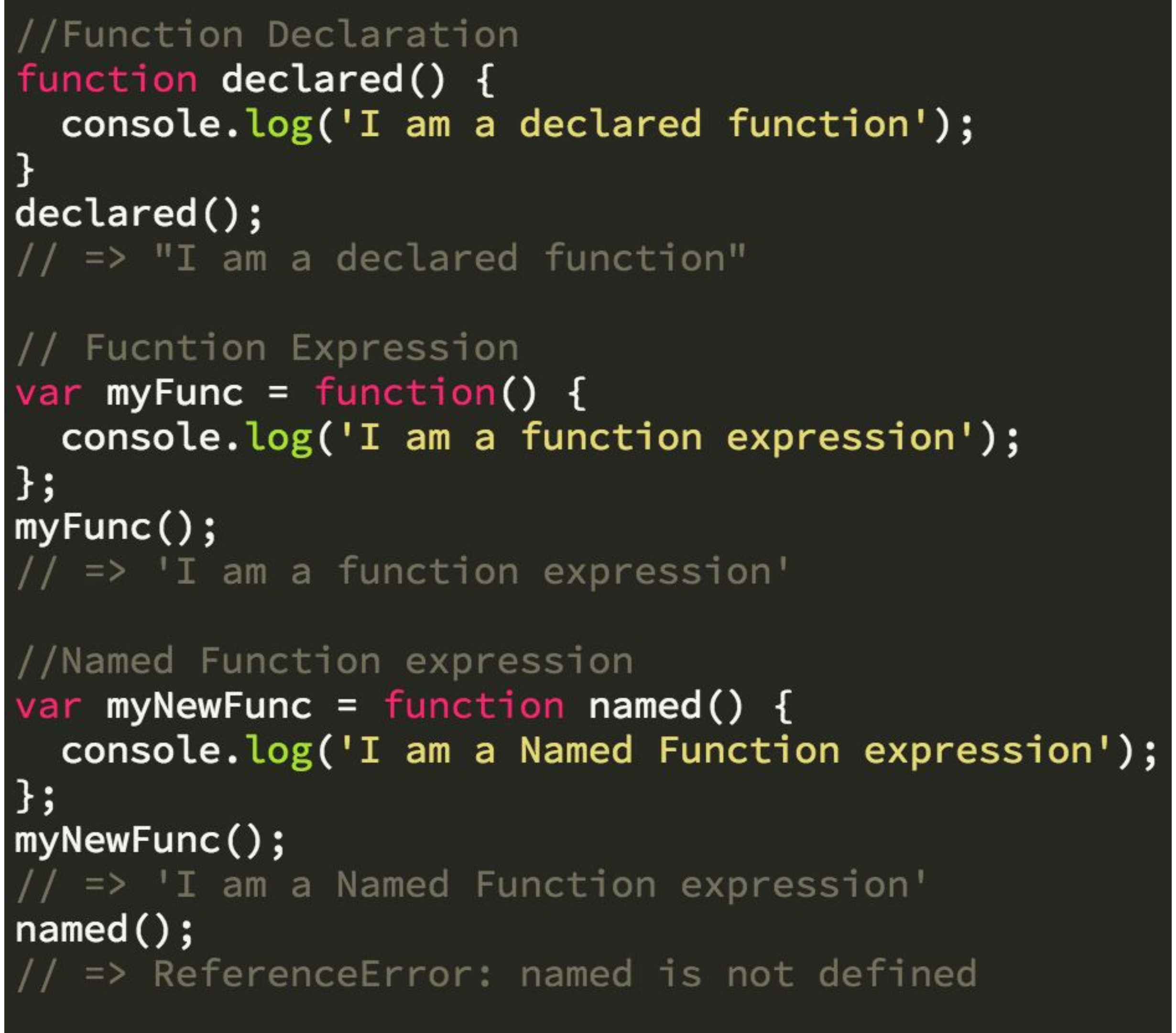
- animate()
- closest()
- getAttribute()
- getAttributeNode()
- getAttributeNodeNS()
- getAttributeNS()
- getBoundingClientRect()
- getClientRects()
- getElementsByClassName()
- getElementsByTagName()
- getElementsByTagNameNS()

2.

THE JAVASCRIPT FUNCTION

It’s Where Lots of the Magic Happens

Function Definitions



```
//Function Declaration
function declared() {
  console.log('I am a declared function');
}
declared();
// => "I am a declared function"

// Fucntion Expression
var myFunc = function() {
  console.log('I am a function expression');
};
myFunc();
// => 'I am a function expression'

//Named Function expression
var myNewFunc = function named() {
  console.log('I am a Named Function expression');
};
myNewFunc();
// => 'I am a Named Function expression'
named();
// => ReferenceError: named is not defined
```

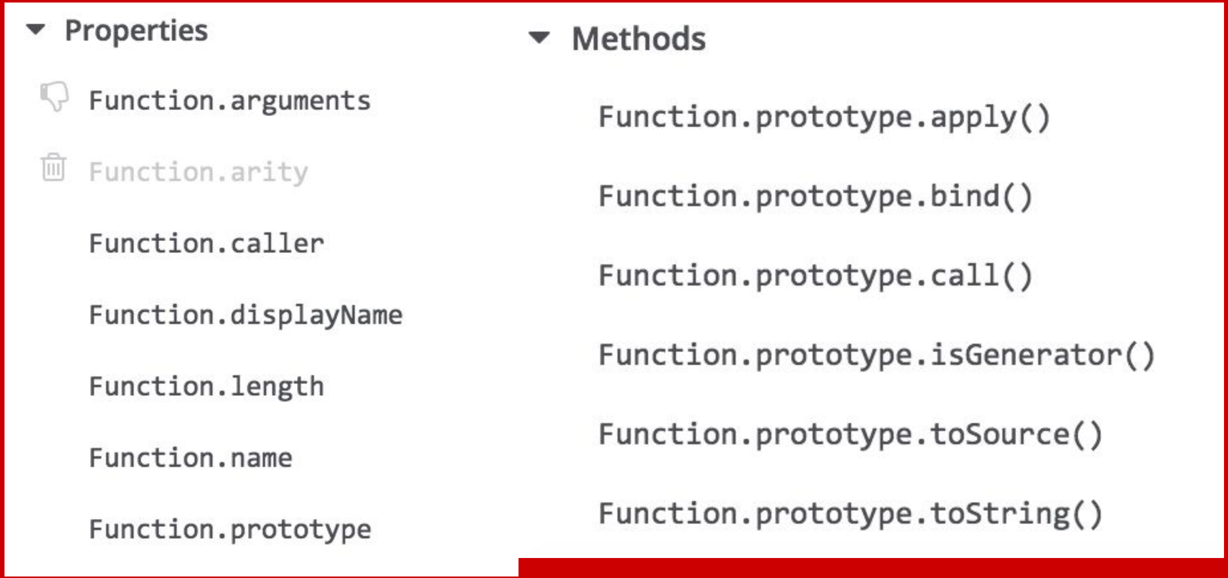
JavaScript Functions are First Class Citizens

★ Functions can be assigned to variables and Passed Around.
★ Functions can accept other Functions as arguments.
★ Functions can return other functions.

BUT
MOST
OF
ALL

Every Function in Javascript is a Function Object

FUNCTION OBJECT PROPERTIES & METHODS



Properties

- Function.arguments
- Function.arity
- Function.caller
- Function.displayName
- Function.length
- Function.name
- Function.prototype

Methods

- Function.prototype.apply()
- Function.prototype.bind()
- Function.prototype.call()
- Function.prototype.isGenerator()
- Function.prototype.toSource()
- Function.prototype.toString()

3.

JAVASCRIPT OBJECTS

The Building Blocks of the Language.

Primary Javascript Objects

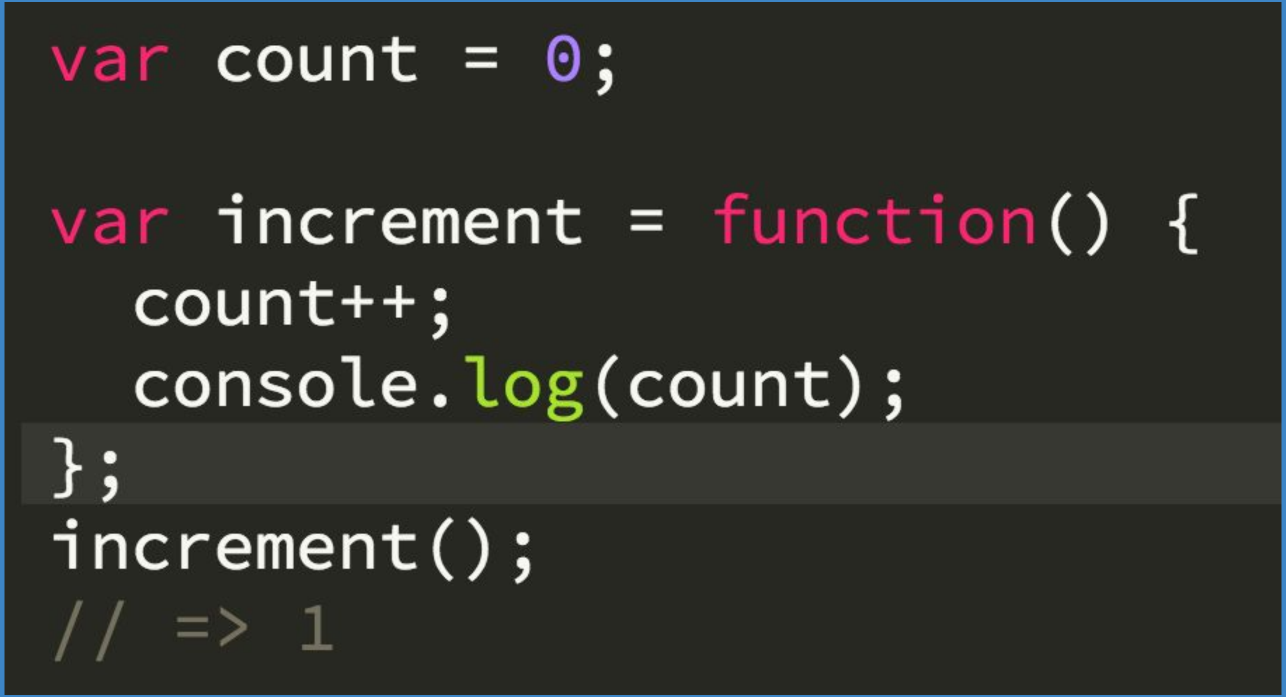
- ★ Function
- ★ String
- ★ Array
- ★ Number
- ★ Boolean
- ★ TypeError

4.

JAVASCRIPT CLOSURES

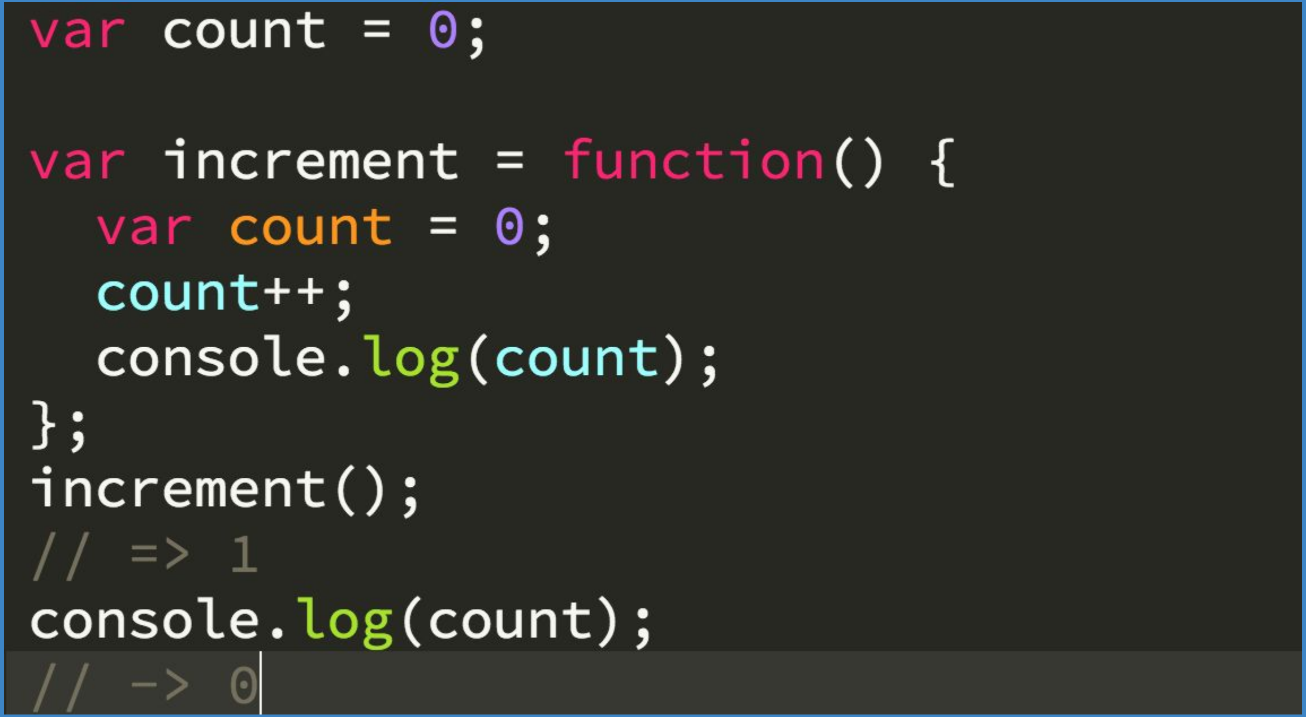
A Function that can be stored as a variable, and that has the special ability to access other variables local to the scope it was created in.

Javascript Function have access to variable defined above their internal scope.



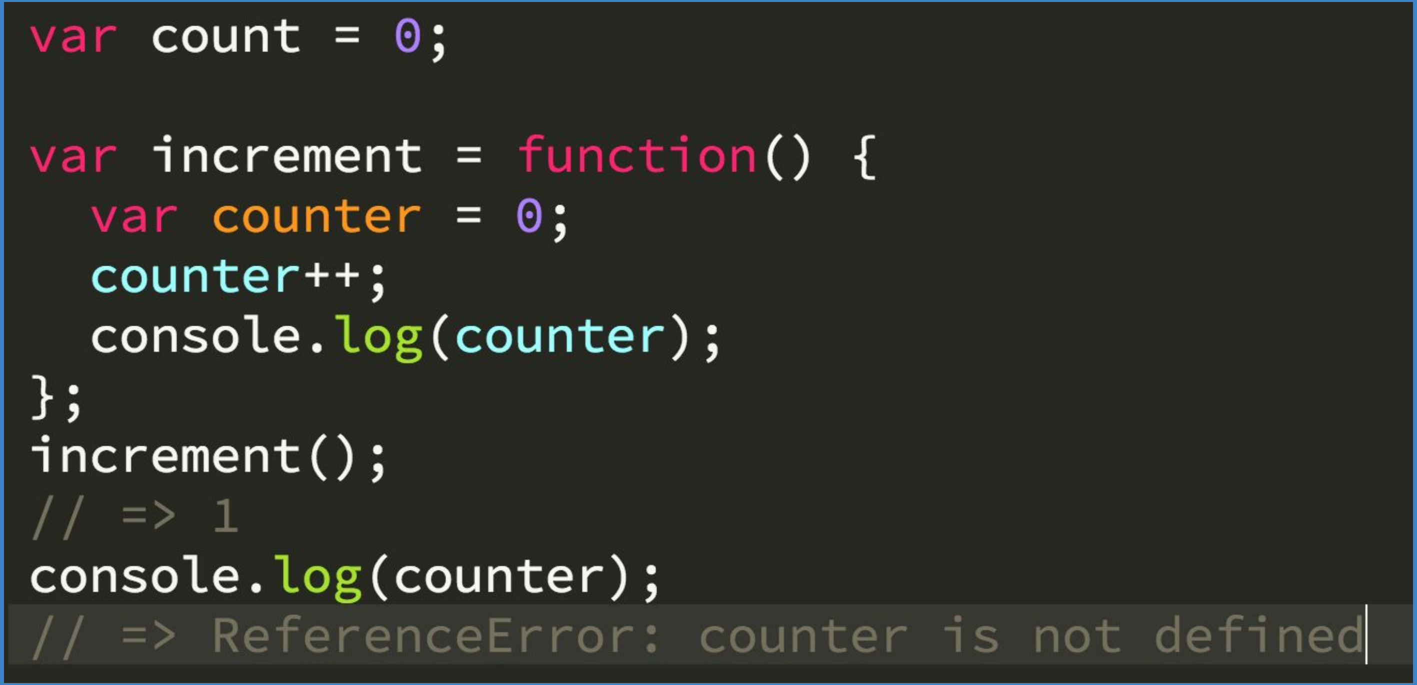
```
var count = 0;

var increment = function() {
  count++;
  console.log(count);
};
increment();
// => 1
```

Javascript Function can also override external variables with their own version with the same name.



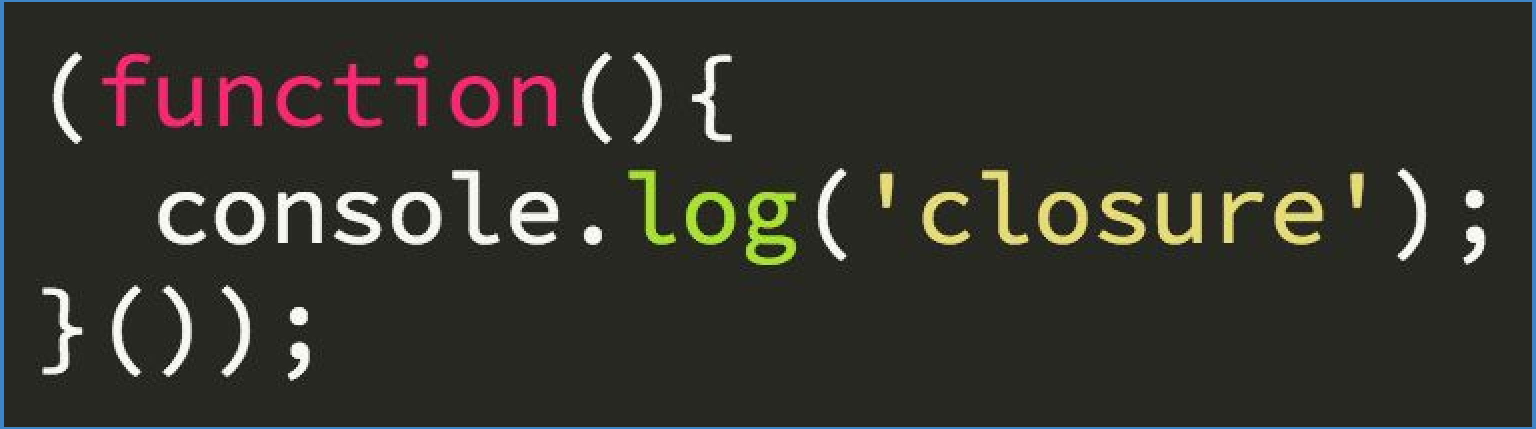
```
var count = 0;

var increment = function() {
  var count = 0;
  count++;
  console.log(count);
};
increment();
// => 1
console.log(count);
// -> 0
```

But Variables defined inside a Javascript Function are not accessible from outside its scope.



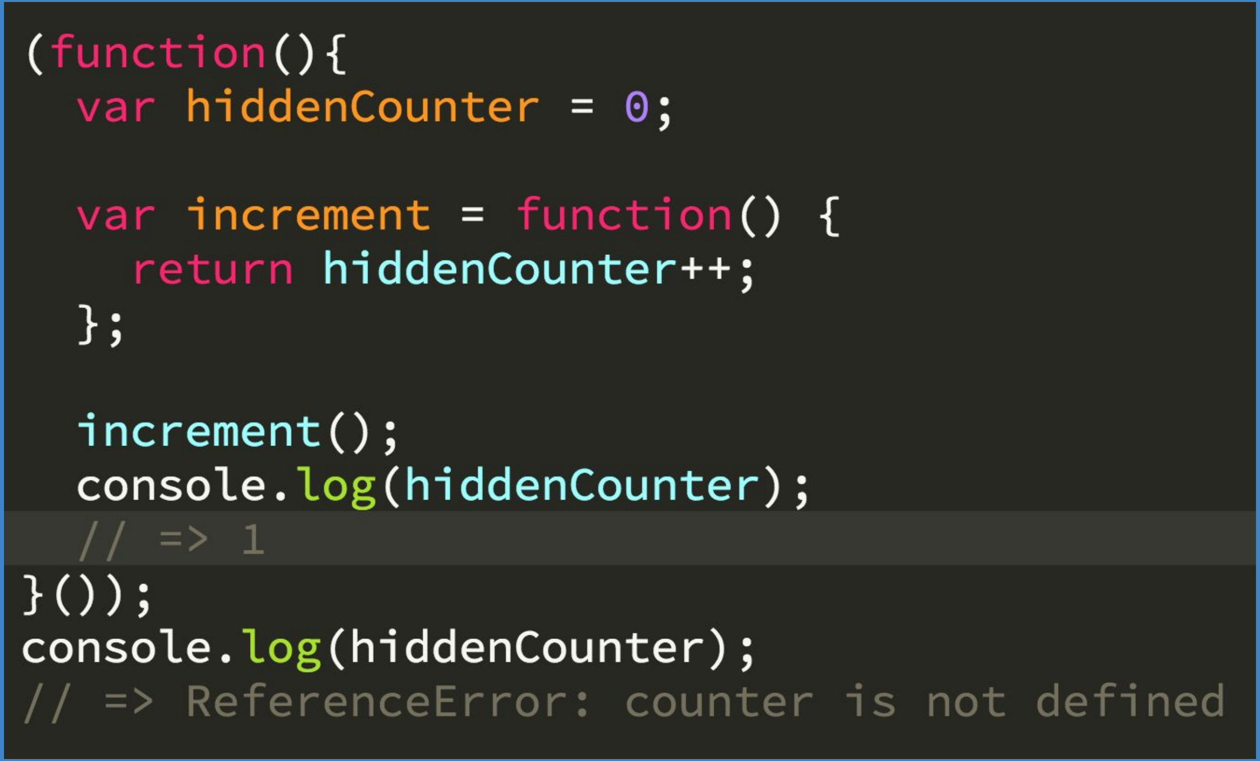
```
var count = 0;

var increment = function() {
  var counter = 0;
  counter++;
  console.log(counter);
};
increment();
// => 1
console.log(counter);
// => ReferenceError: counter is not defined
```

But this highlights one of the top use cases for Closures in Javascript: Immediately-Invoked Function Expression



```
(function(){
  console.log('closure');
}());
```

Practical IFFE Example



```
(function(){
  var hiddenCounter = 0;

  var increment = function() {
    return hiddenCounter++;
  };

  increment();
  console.log(hiddenCounter);
  // => 1
}());
console.log(hiddenCounter);
// => ReferenceError: counter is not defined
```

LIFEFROMDABASSMENT
CAN'T TOUCH THIS!

5.

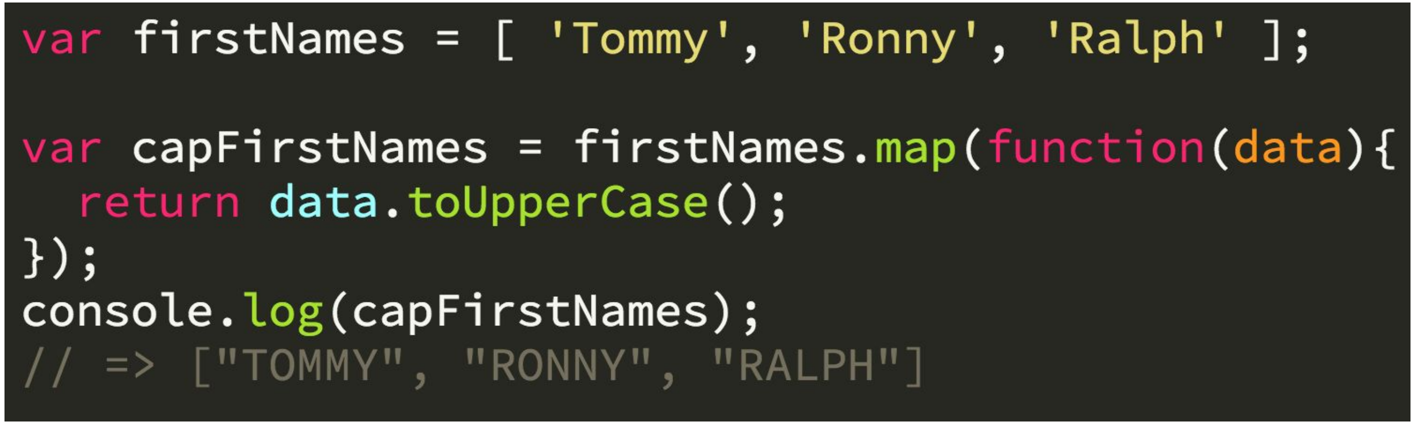
MANIPULATING DATA

The Javascript Array Object Methods

Array.prototype.map( );

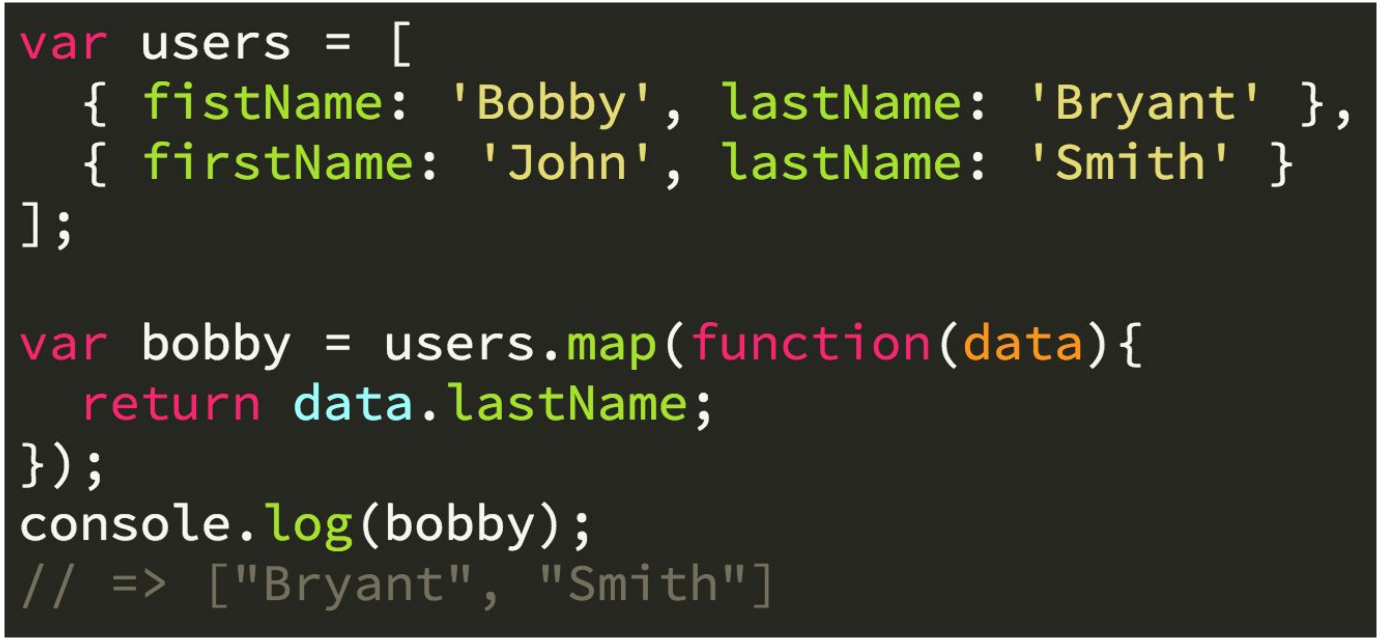
The **`map()`** method creates a new array with the results of calling a provided function on every element in this array.

Array.prototype.map( );



```
var firstNames = [ 'Tommy', 'Ronny', 'Ralph' ];

var capFirstNames = firstNames.map(function(data){
  return data.toUpperCase();
});
console.log(capFirstNames);
// => ["TOMMY", "RONNY", "RALPH"]
```

Array.prototype.map( );

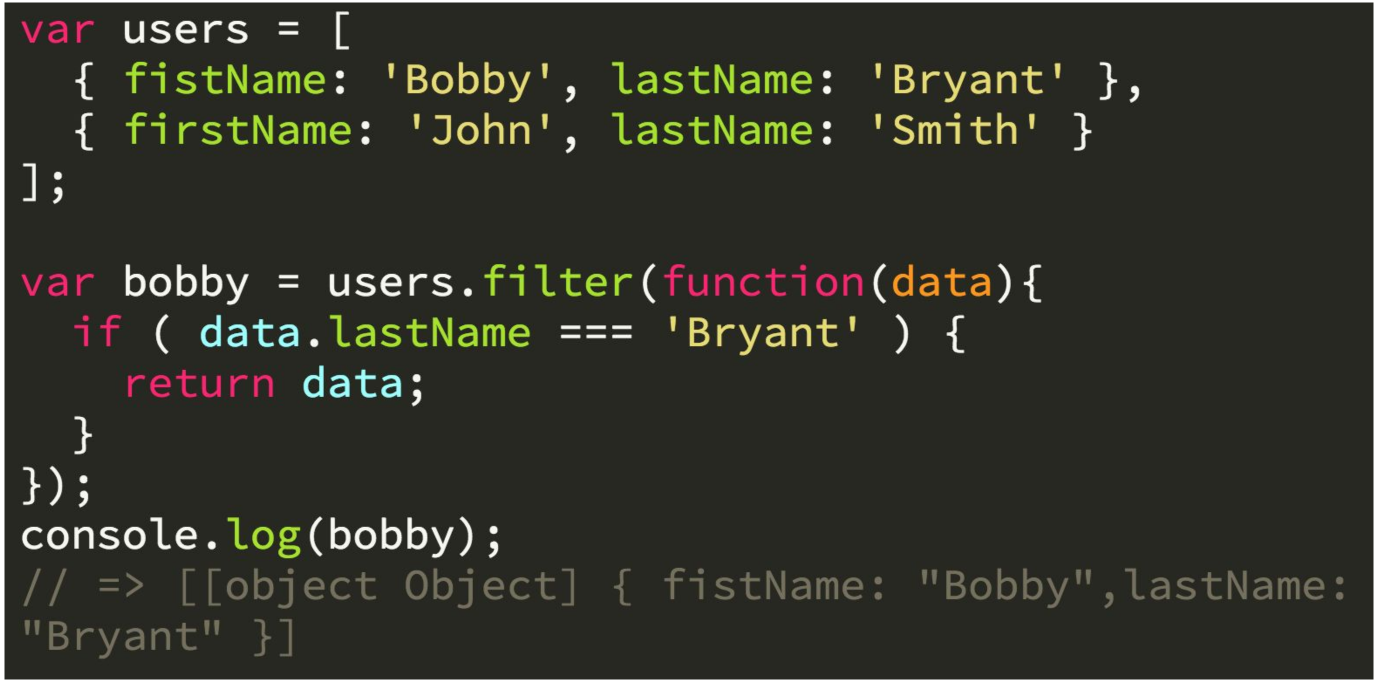


```
var users = [
  { fistName: 'Bobby', lastName: 'Bryant' },
  { firstName: 'John', lastName: 'Smith' }
];

var bobby = users.map(function(data){
  return data.lastName;
});
console.log(bobby);
// => ["Bryant", "Smith"]
```

Array.prototype.filter( );

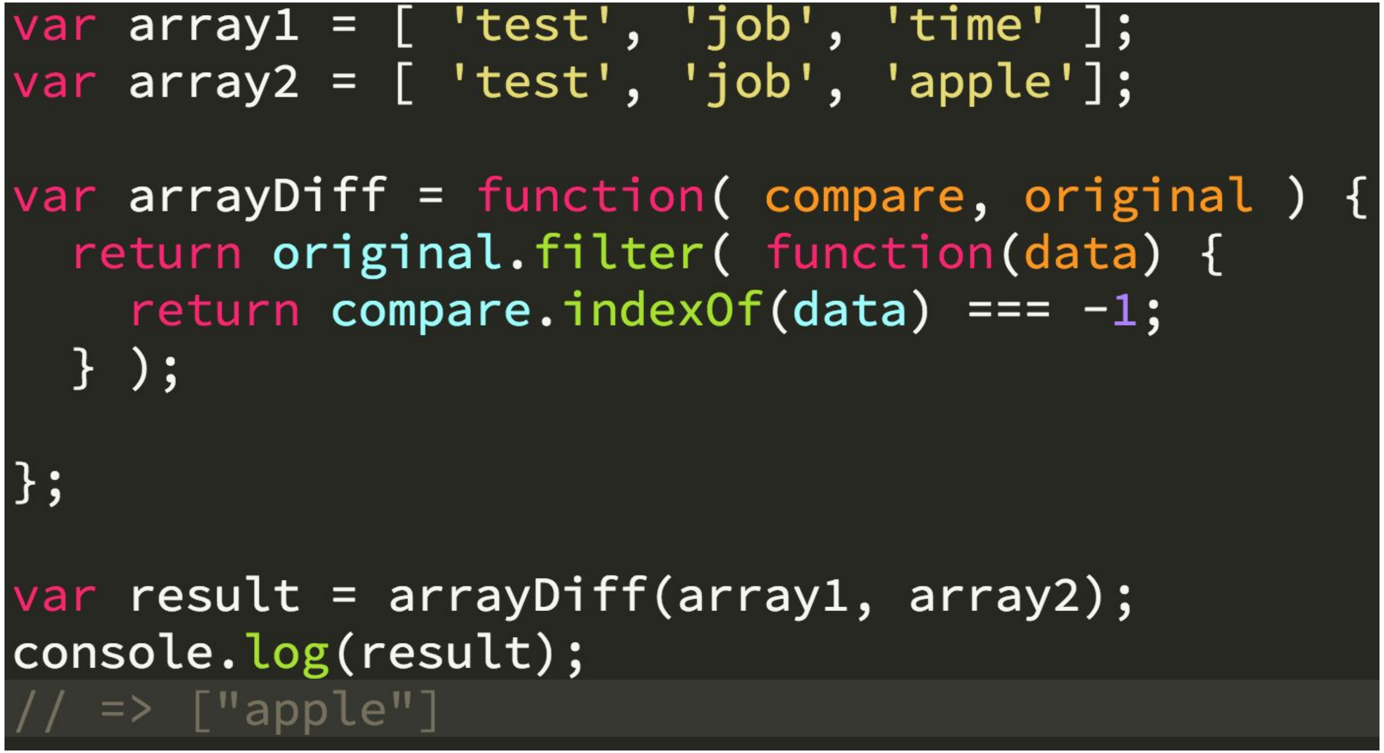
The **`filter()`** method creates a new array with values that pass the test implemented by the function provided.

Array.prototype.filter( );



```
var users = [
  { fistName: 'Bobby', lastName: 'Bryant' },
  { firstName: 'John', lastName: 'Smith' }
];

var bobby = users.filter(function(data){
  if ( data.lastName === 'Bryant' ) {
    return data;
  }
});
console.log(bobby);
// => [[object Object] { fistName: "Bobby",lastName:
"Bryant" }]
```

Array.prototype.filter( );

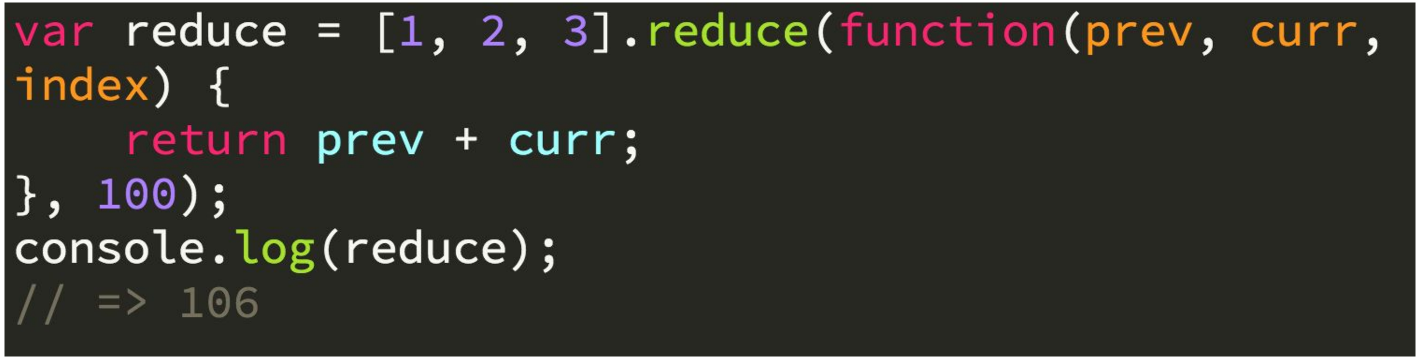


```
var array1 = [ 'test', 'job', 'time' ];
var array2 = [ 'test', 'job', 'apple'];

var arrayDiff = function( compare, original ) {
  return original.filter( function(data) {
    return compare.indexOf(data) === -1;
  } );

};

var result = arrayDiff(array1, array2);
console.log(result);
// => ["apple"]
```

Array.prototype.reduce( );

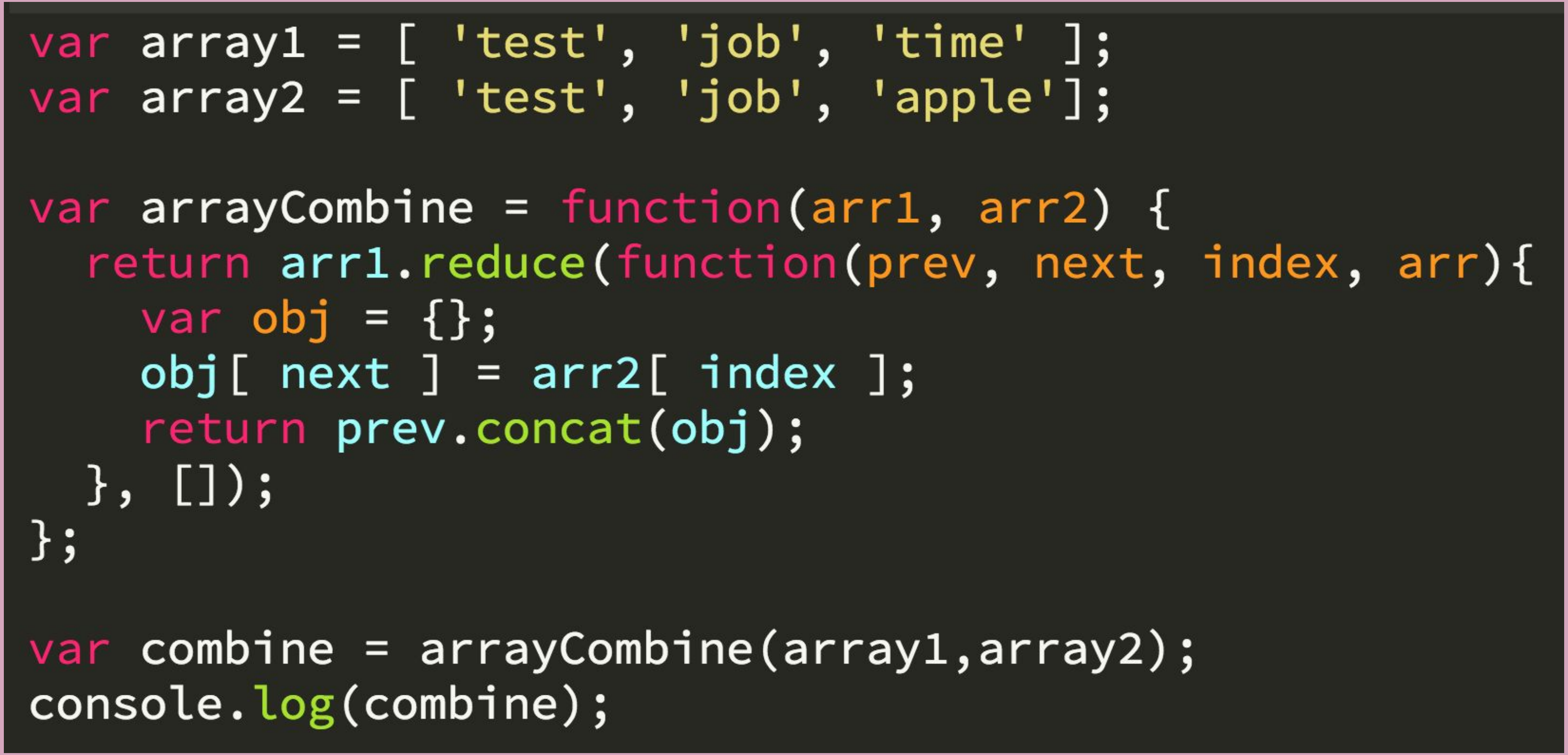
The **`reduce()`** method applies a function against an accumulator and each value of an array (from left to right) to reduce it to a single value.

Array.prototype.reduce( );



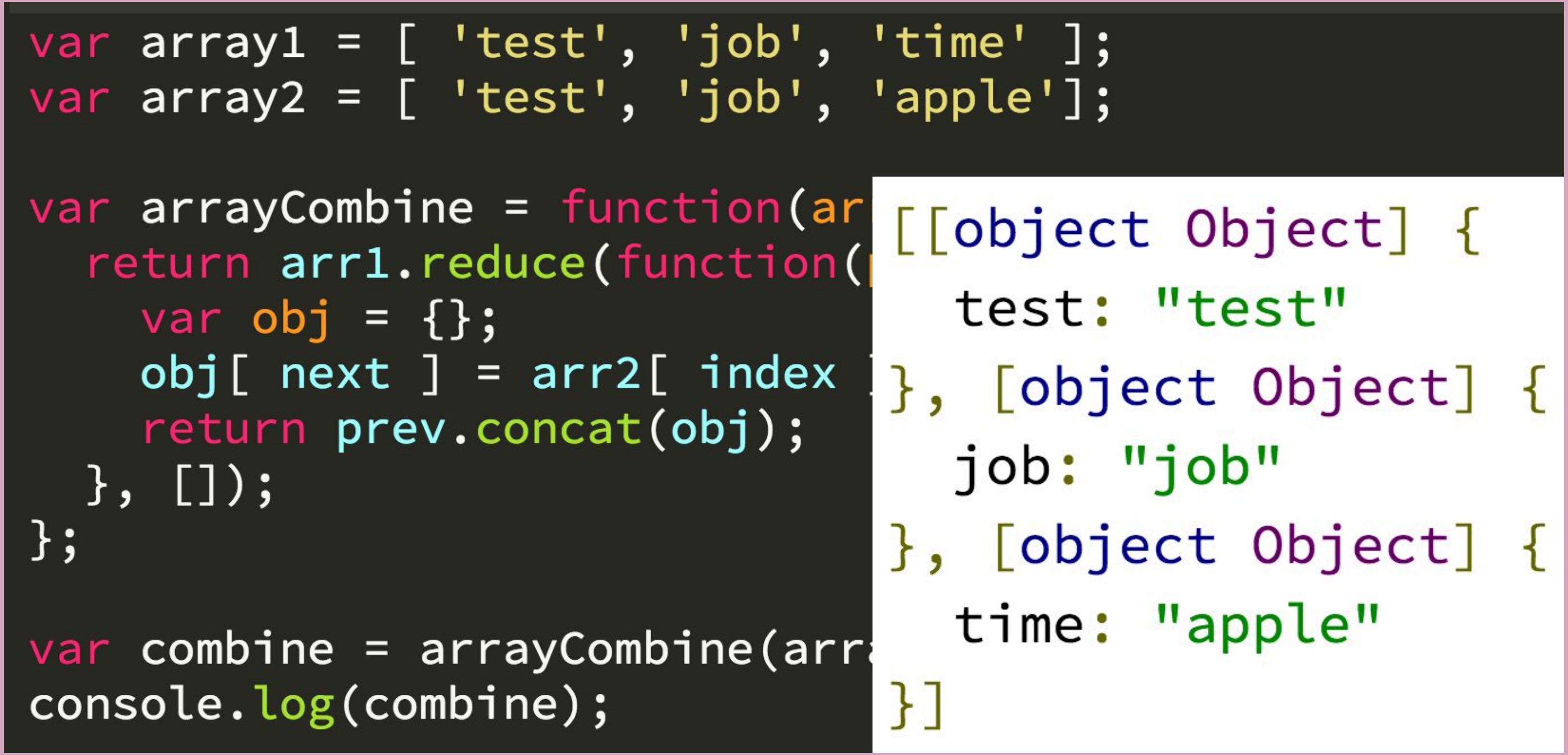
```
var reduce = [1, 2, 3].reduce(function(prev, curr,
index) {
    return prev + curr;
}, 100);
console.log(reduce);
// => 106
```

Array.prototype.reduce( );



```
var array1 = [ 'test', 'job', 'time' ];
var array2 = [ 'test', 'job', 'apple'];

var arrayCombine = function(arr1, arr2) {
  return arr1.reduce(function(prev, next, index, arr){
    var obj = {};
    obj[ next ] = arr2[ index ];
    return prev.concat(obj);
  }, []);
};

var combine = arrayCombine(array1,array2);
console.log(combine);
```

Array.prototype.reduce( );



```
var array1 = [ 'test', 'job', 'time' ];
var array2 = [ 'test', 'job', 'apple'];

var arrayCombine = function(ar
  return arr1.reduce(function(
    var obj = {};
    obj[ next ] = arr2[ index
    return prev.concat(obj);
  }, []);
};

var combine = arrayCombine(arr
console.log(combine);
```

```
[[object Object] {
  test: "test"
}, [object Object] {
  job: "job"
}, [object Object] {
  time: "apple"
}]
```

6.

PROTOTYPAL INHERITANCE

Javascript Objects inheriting from other Objects

Class Based Inheritance

In a class based system you can define a Class, which will act as a blueprint for each new Object.

Classes can inherit from other classes to create a hierarchy.

When you create a new object from the class it is considered an Instance of that class.

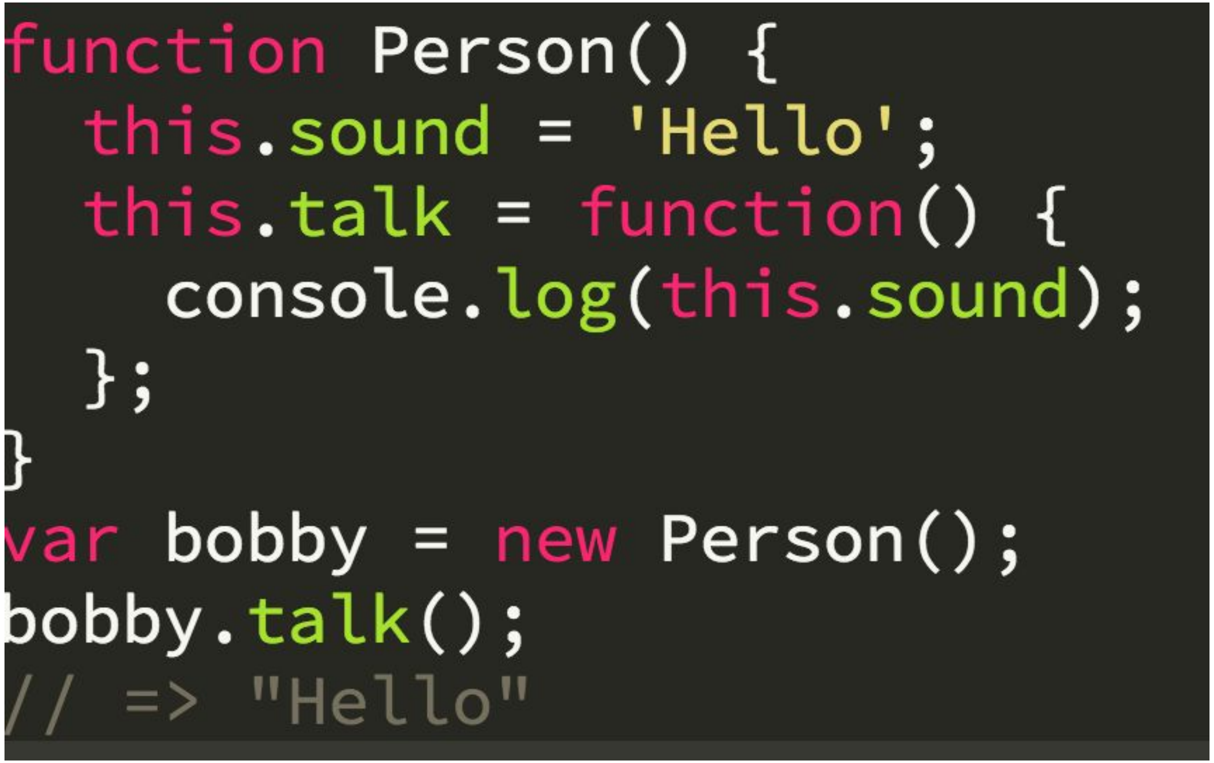
Prototype Based Inheritance

Prototype Languages such as Javascript to not have this distinction.

Languages like Javascript simply have Objects, which can inherit from other objects.

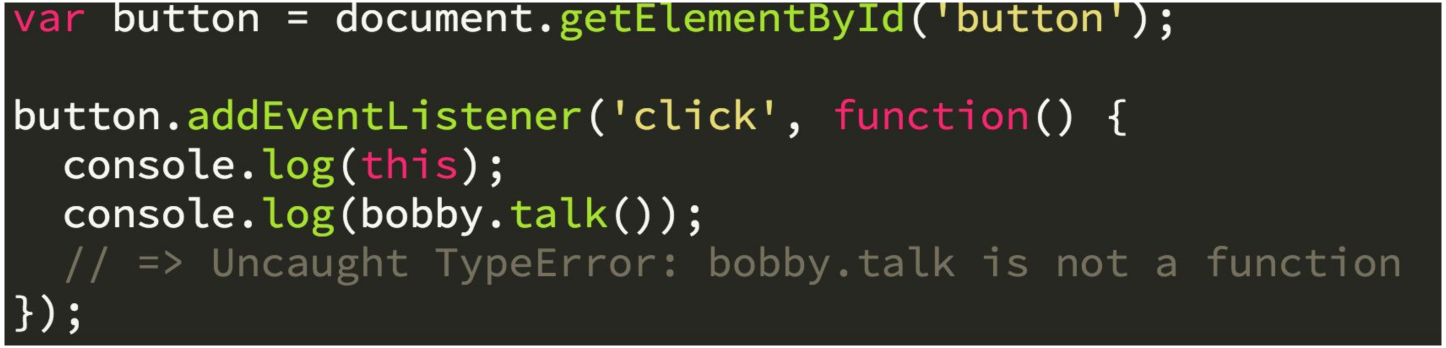
Javascript’s Prototypal Inheritance is so hard to understand, because it gives us this **new** keyword, and tries to mimic class based inheritances.

Classical Inheritance



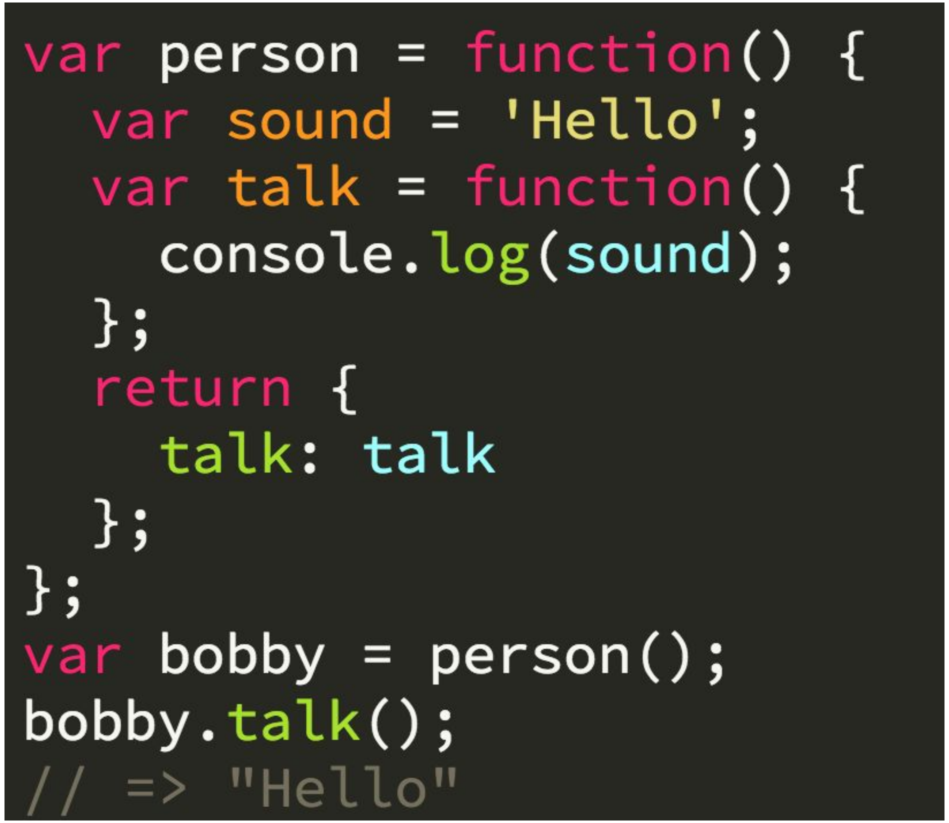
```
function Person() {
  this.sound = 'Hello';
  this.talk = function() {
    console.log(this.sound);
  };
}
var bobby = new Person();
bobby.talk();
// => "Hello"
```

Classical Inheritance (continued)



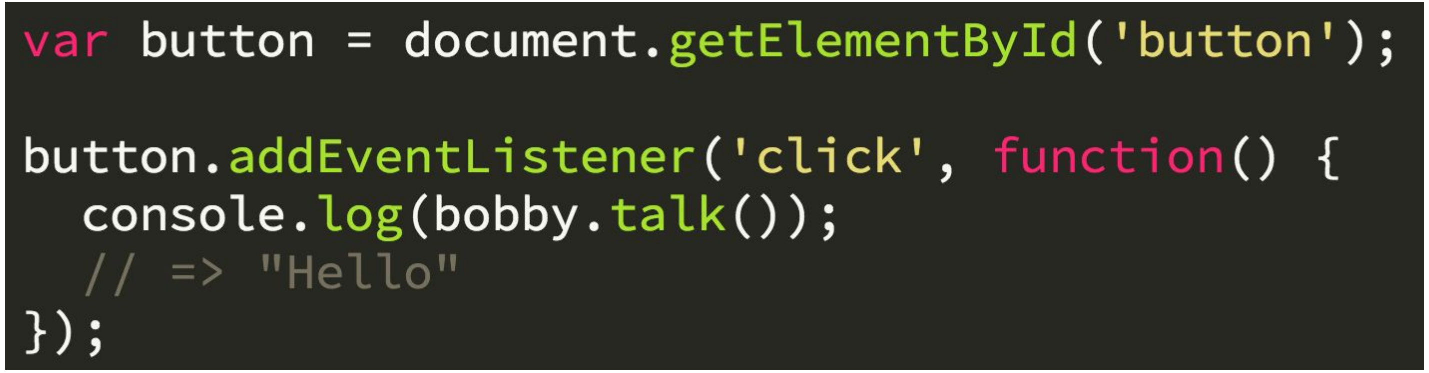
```
var button = document.getElementById('button');

button.addEventListener('click', function() {
  console.log(this);
  console.log(bobby.talk());
  // => Uncaught TypeError: bobby.talk is not a function
});
```

Factory Function



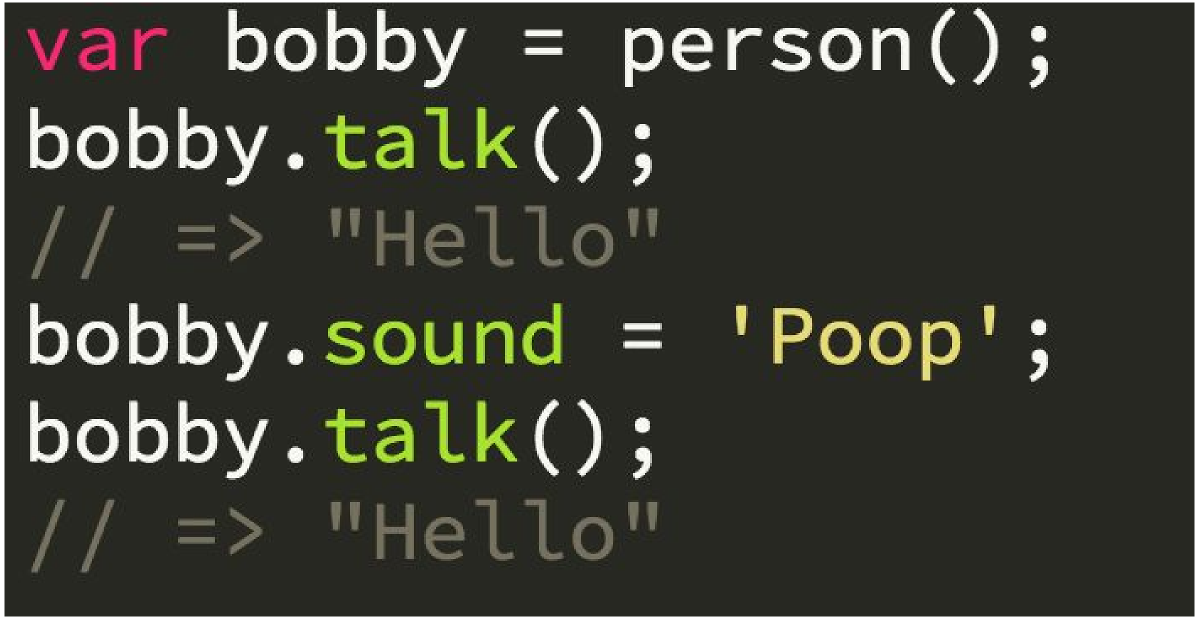
```
var person = function() {
  var sound = 'Hello';
  var talk = function() {
    console.log(sound);
  };
  return {
    talk: talk
  };
};
var bobby = person();
bobby.talk();
// => "Hello"
```

Factory Function (Continued)



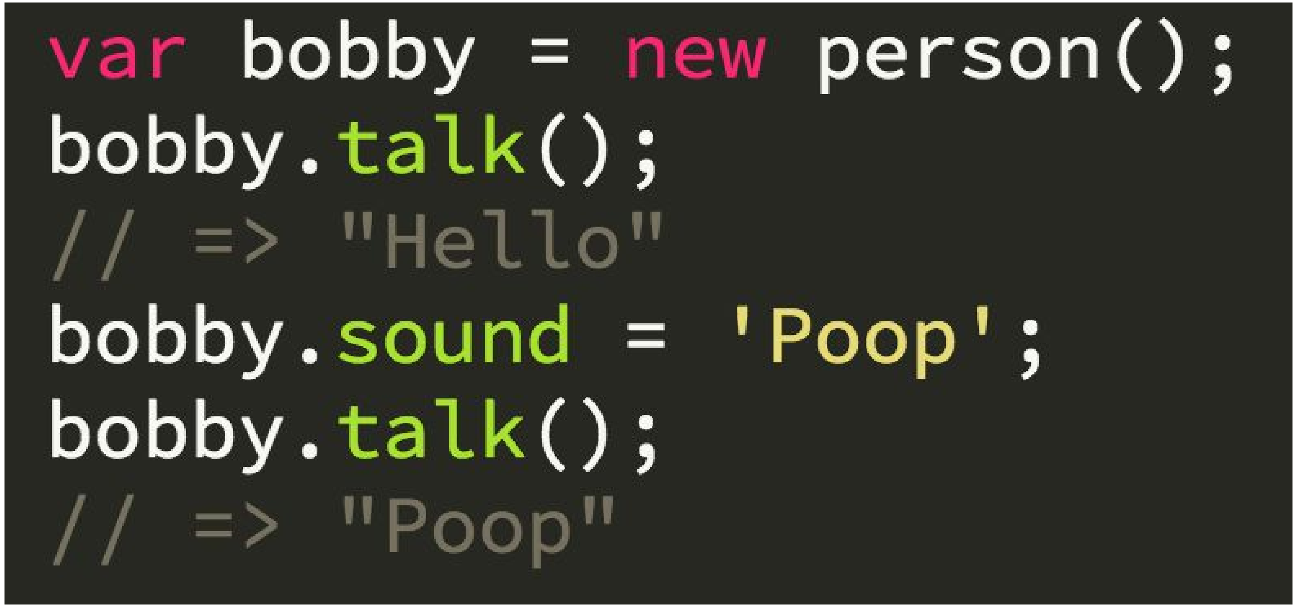
```
var button = document.getElementById('button');

button.addEventListener('click', function() {
  console.log(bobby.talk());
  // => "Hello"
});
```

Factory Function (Continued)



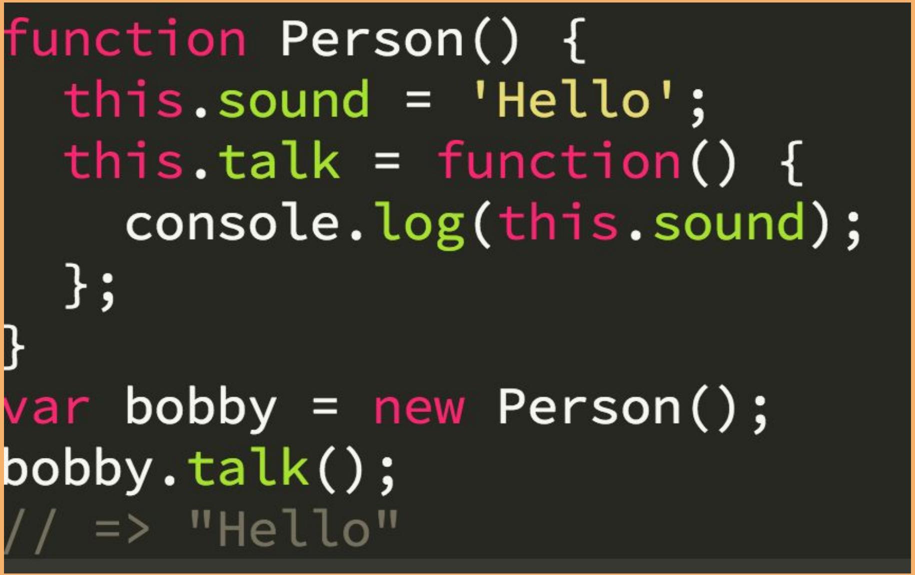
```
var bobby = person();
bobby.talk();
// => "Hello"
bobby.sound = 'Poop';
bobby.talk();
// => "Hello"
```

Classical Inheritance (Continued)

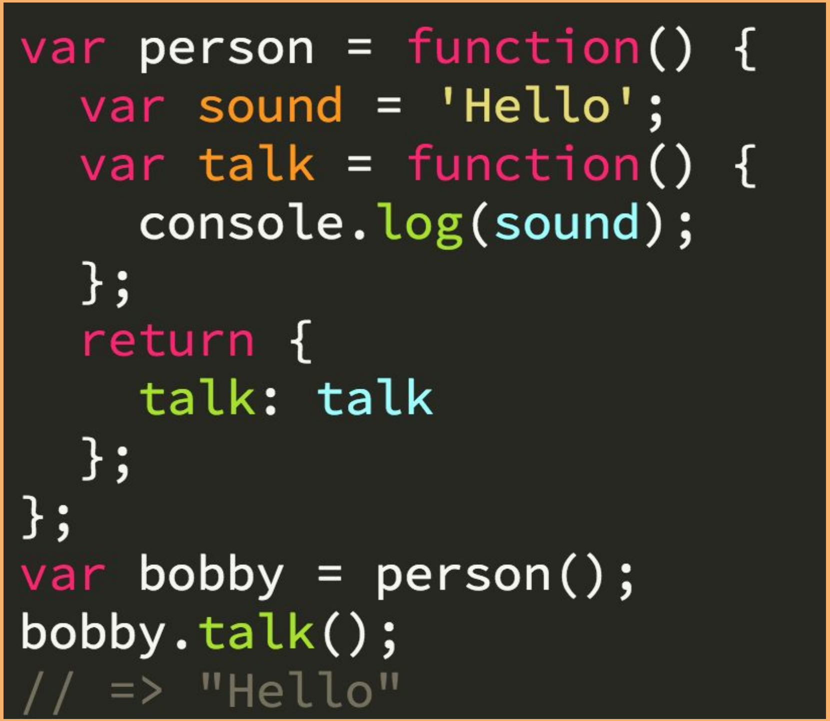


```
var bobby = new person();
bobby.talk();
// => "Hello"
bobby.sound = 'Poop';
bobby.talk();
// => "Poop"
```

Composition vs Inheritance



```
function Person() {
  this.sound = 'Hello';
  this.talk = function() {
    console.log(this.sound);
  };
}
var bobby = new Person();
bobby.talk();
// => "Hello"
```



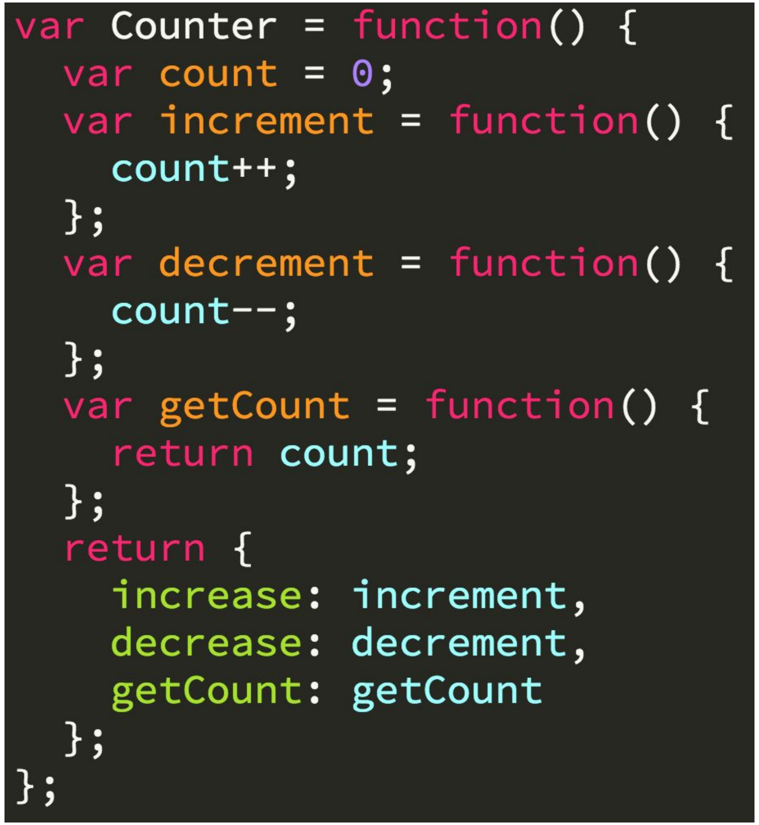
```
var person = function() {
  var sound = 'Hello';
  var talk = function() {
    console.log(sound);
  };
  return {
    talk: talk
  };
};
var bobby = person();
bobby.talk();
// => "Hello"
```

7.

REVEALING MODULE PATTERN

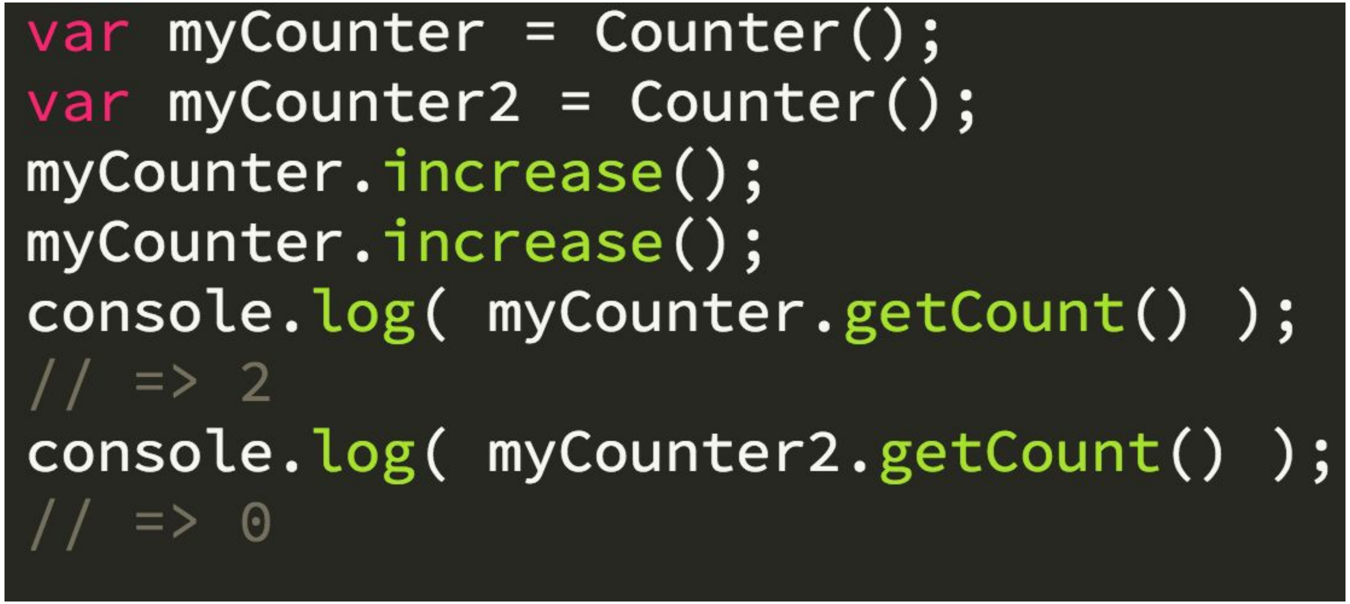
Another Way to Build Objects

Function + Closure = Revealing Module Pattern



```
var Counter = function() {
  var count = 0;
  var increment = function() {
    count++;
  };
  var decrement = function() {
    count--;
  };
  var getCount = function() {
    return count;
  };
  return {
    increase: increment,
    decrease: decrement,
    getCount: getCount
  };
};
```

Revealing Module Pattern

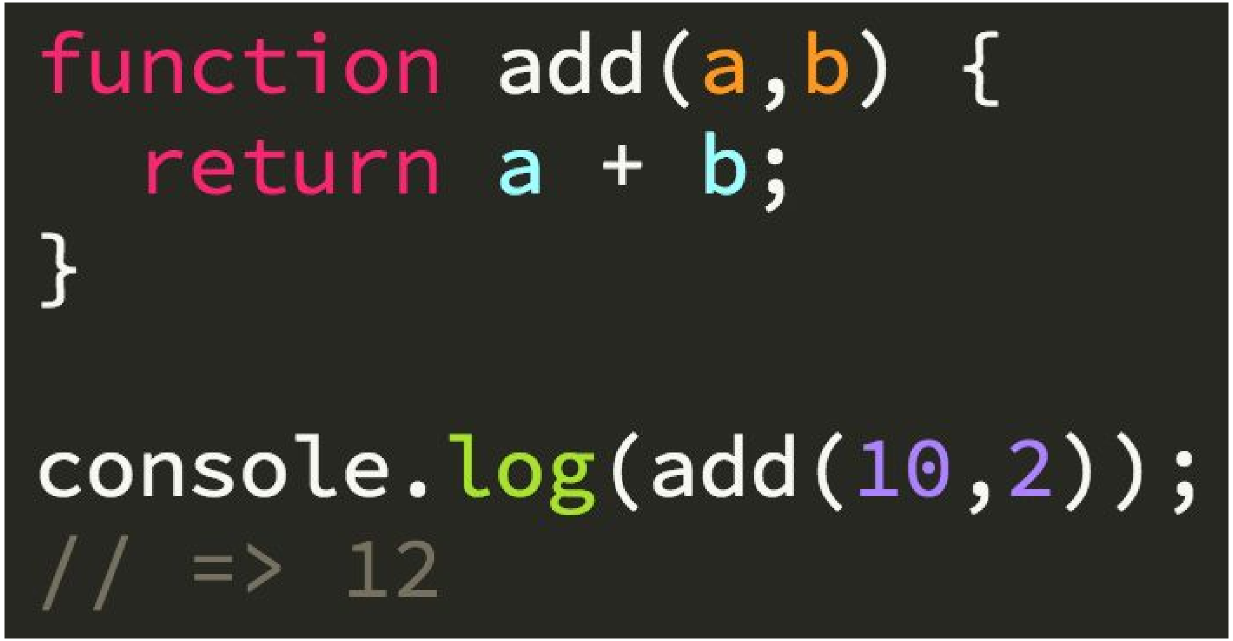


```
var myCounter = Counter();
var myCounter2 = Counter();
myCounter.increase();
myCounter.increase();
console.log( myCounter.getCount() );
// => 2
console.log( myCounter2.getCount() );
// => 0
```

8. FUNCTION COMPOSITION

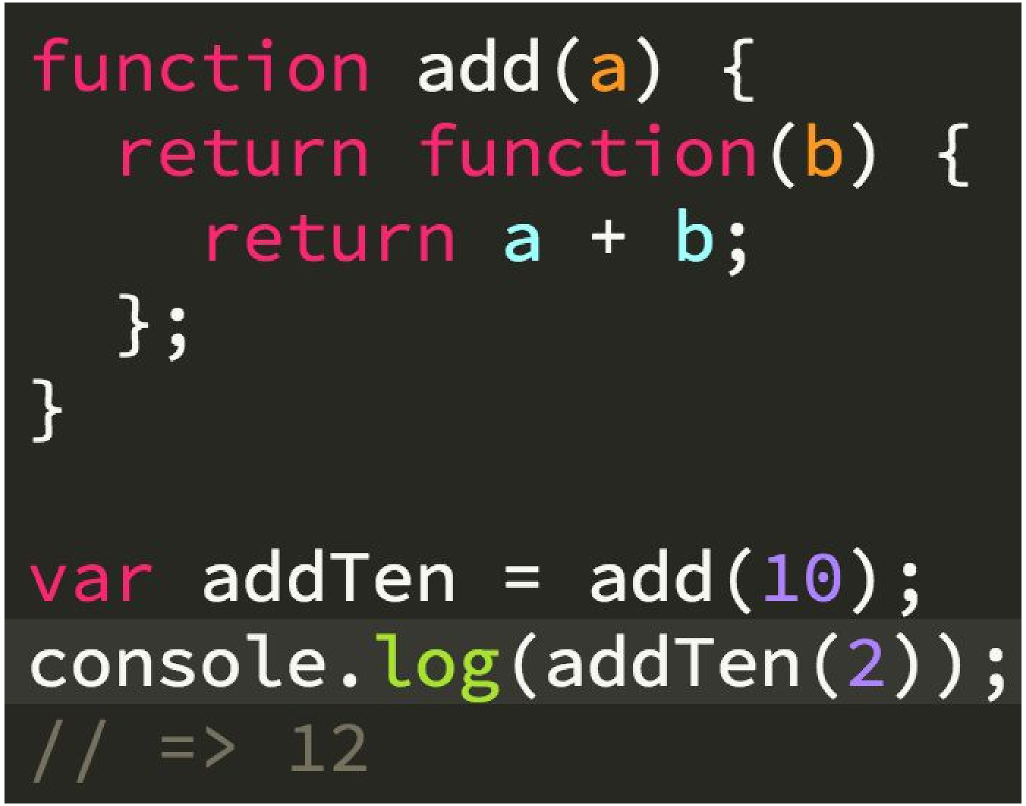
Functions As Ingredients

FUNCTION COMPOSITION



```
function add(a,b) {
  return a + b;
}

console.log(add(10,2));
// => 12
```

FUNCTION COMPOSITION



```
function add(a) {
  return function(b) {
    return a + b;
  };
}

var addTen = add(10);
console.log(addTen(2));
// => 12
```

FUNCTION COMPOSITION

```
function filter(collection) {
  return function(item, value){
    return collection.filter(function(data){
      if (data[item] === value) {
        return data;
      }
    });
  };
}

var posts = [
  {id:1, title: 'Once Upon A Time', author: 'Bobby Bryant'},
  {id:2, title: 'WordCamp Dayton', author: 'Bobby Bryant'},
  {id:3, title: 'Object Composition', author: 'John Smith'}
];
```

FUNCTION COMPOSITION

```
var filterPostsBy = filter(posts);
console.log(filterPostsBy('author', 'Bobby Bryant'));
```

```
[[object Object] {
  author: "Bobby Bryant",
  id: 1,
  title: "Once Upon A Time"
}, [object Object] {
  author: "Bobby Bryant",
  id: 2,
  title: "WordCamp Dayton"
}]
```

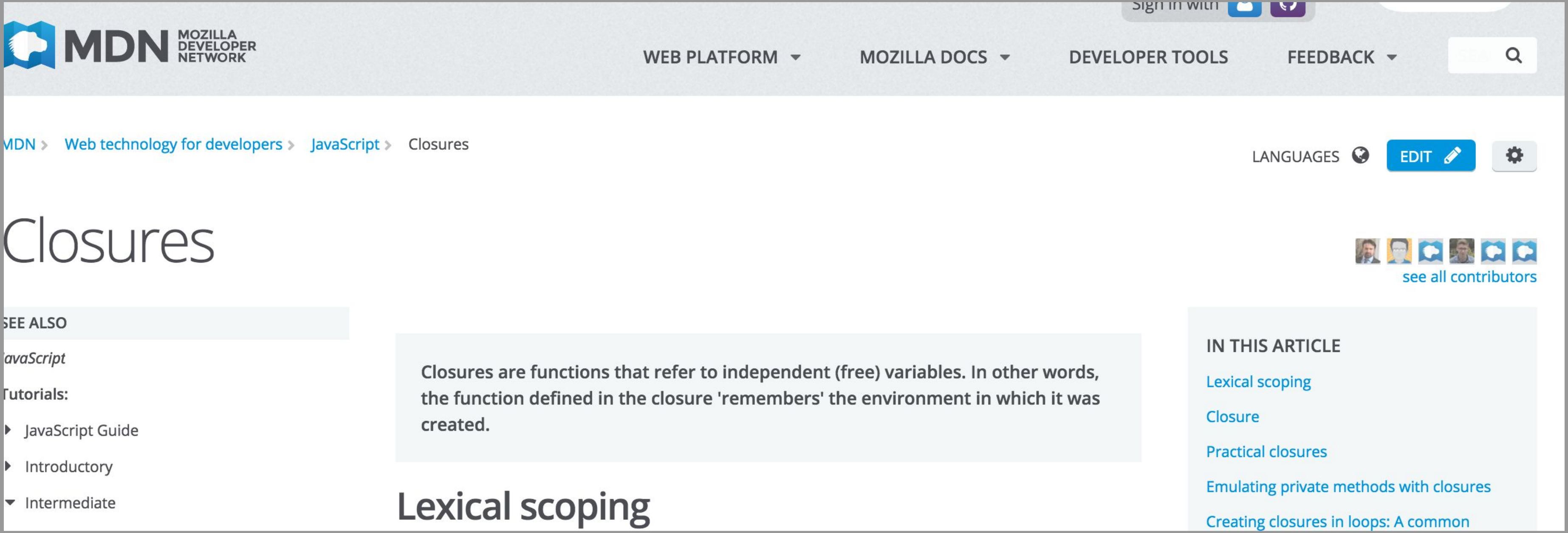
FUNCTION COMPOSITION

```
var filterPostsBy = filter(posts);
console.log(filterPostsBy('author', 'Bobby Bryant'));
var users = [];
var filterUsersBy = filter(users);
```

Resources

FUNFUN
Function
Old wisdom
wild ideas
DESIGUAL
funfunfunction
Subscribed
18,650

Resources


Sign in with

MDN MOZILLA DEVELOPER NETWORK

WEB PLATFORM MOZILLA DOCS DEVELOPER TOOLS FEEDBACK

MDN › Web technology for developers › JavaScript › Closures

LANGUAGES EDIT

Closures

see all contributors

SEE ALSO

JavaScript

Tutorials:

- JavaScript Guide
- Introductory
- Intermediate

Closures are functions that refer to independent (free) variables. In other words, the function defined in the closure 'remembers' the environment in which it was created.

Lexical scoping

IN THIS ARTICLE

- Lexical scoping
- Closure
- Practical closures
- Emulating private methods with closures
- Creating closures in loops: A common

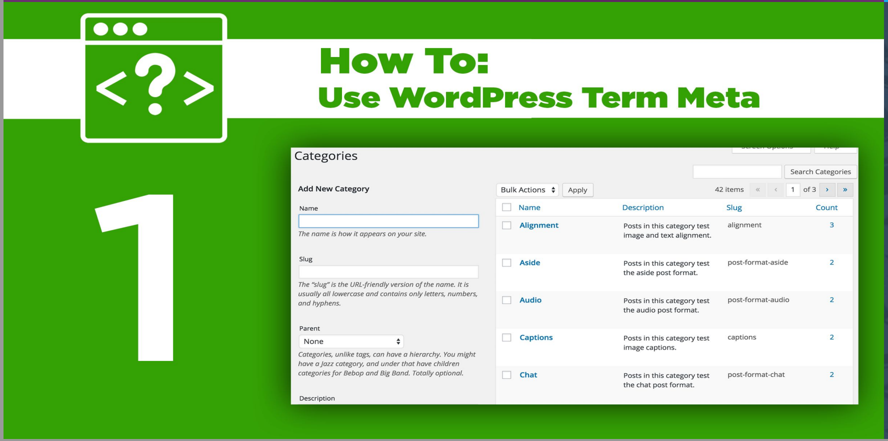
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Closures

Resources

YOU MIGHT NOT NEED JQUERY

http://youmightnotneedjquery.com/

Develop With WP



Categories

Search Categories

Add New Category

Name

The name is how it appears on your site.

Slug

The "slug" is the URL-friendly version of the name. It is usually all lowercase and contains only letters, numbers, and hyphens.

Parent

None

Categories, unlike tags, can have a hierarchy. You might have a Jazz category, and under that have children categories for Bebop and Big Band. Totally optional.

Description

Bulk Actions | Apply | 42 items « ‹ 1 of 3 › »

Name	Description	Slug	Count
Alignment	Posts in this category test image and text alignment.	alignment	3
Aside	Posts in this category test the aside post format.	post-format-aside	2
Audio	Posts in this category test the audio post format.	post-format-audio	2
Captions	Posts in this category test image captions.	captions	2
Chat	Posts in this category test the chat post format.	post-format-chat	2

Hello my Name is Bobby Bryant. I am A web developer at 10up

twitter: /mrbobbybryant
youtube: /developwithwp
github: /mrbobbybryant