JS Talk

04 Oct 2016

Vamsi Sai Turlapati

Been fiddling with JavaScript for at least 3 years.

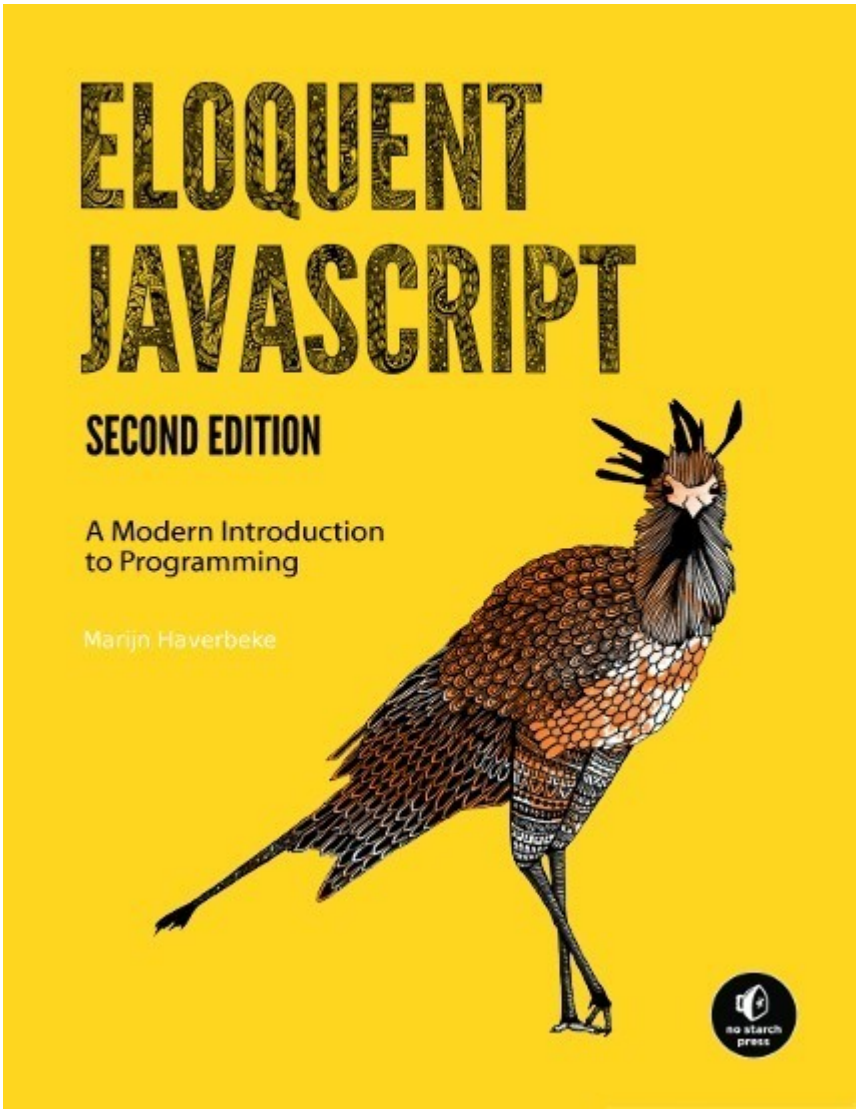
Have worked on frontend on my previous industry engagements. All 2 off 2 internships.

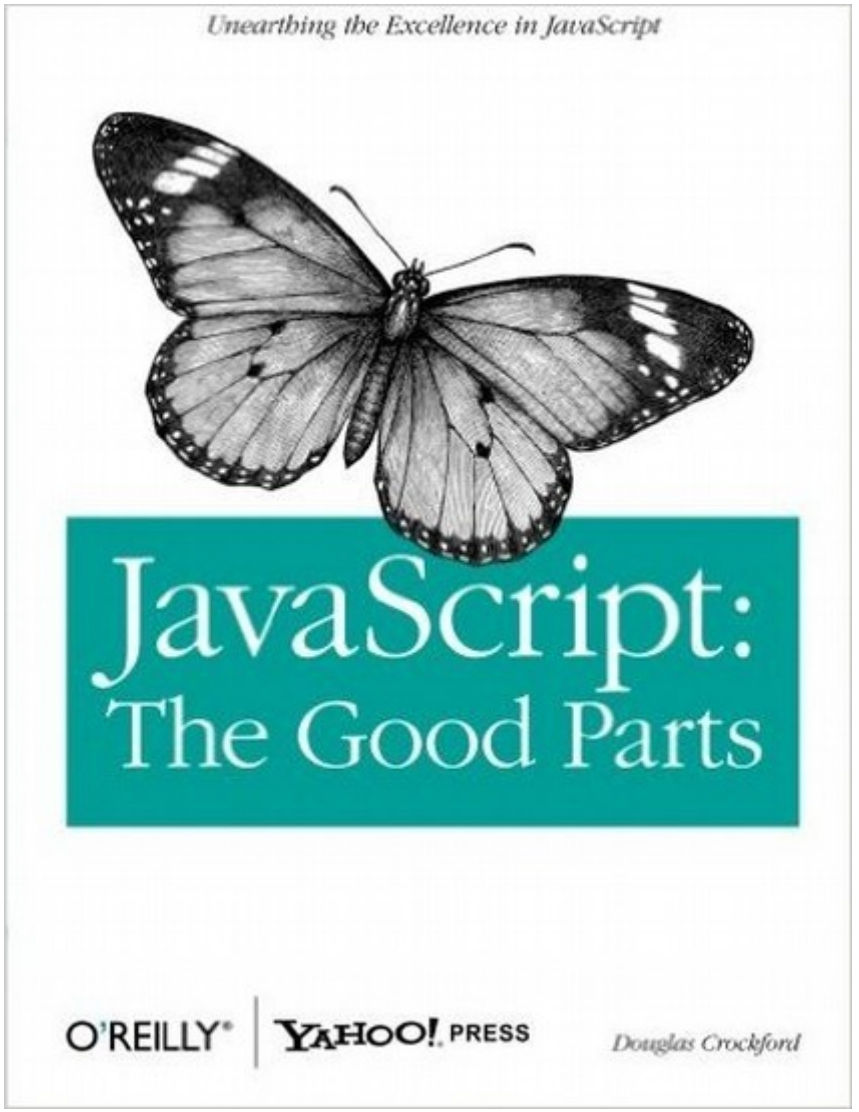
Full Stack Development, Driven by Ideas

Discussions & Resources

http://jstalk.tvamsisai.com

Books for Reference





Introduction

History

Syntax & Semantics

Objects

this keyword

Prototypes

Introduction

History

Syntax & Semantics

Objects

this keyword

Prototypes

History

Mosiac was a university project, a browser

Mozilla was created inspired by it

Marc Andreessen lead the team

Brendan Eich built the spec in 10 days

Codenamed Mocha

Shipped in beta release as LiveScript

Finally named JavaScript, a ploy some say

Why JavaScript?

Why JavaScript?

Web Frontend

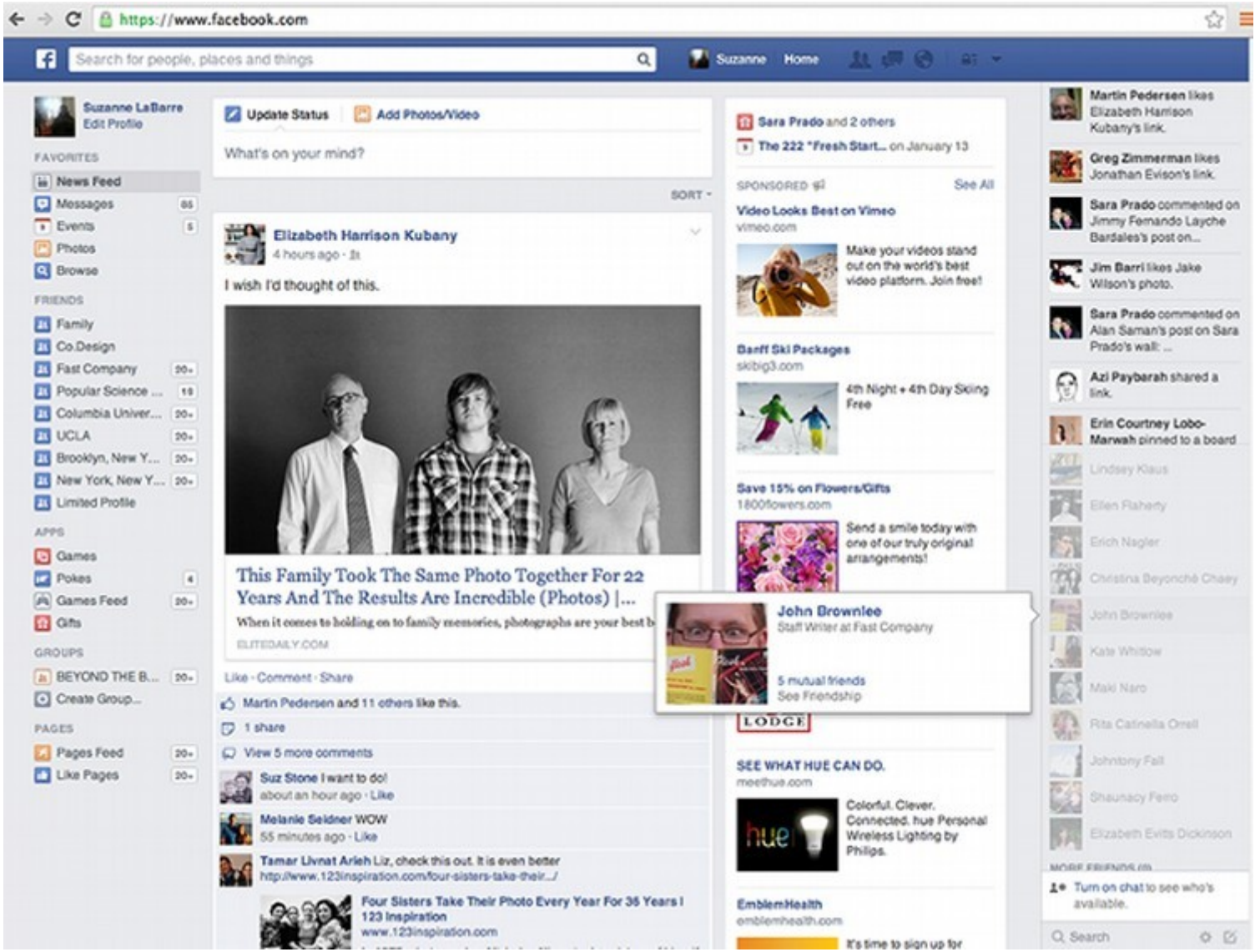
Server-side Software

Desktop Applications

Realtime Analytics & Graphics

Why JavaScript?

Web Frontend

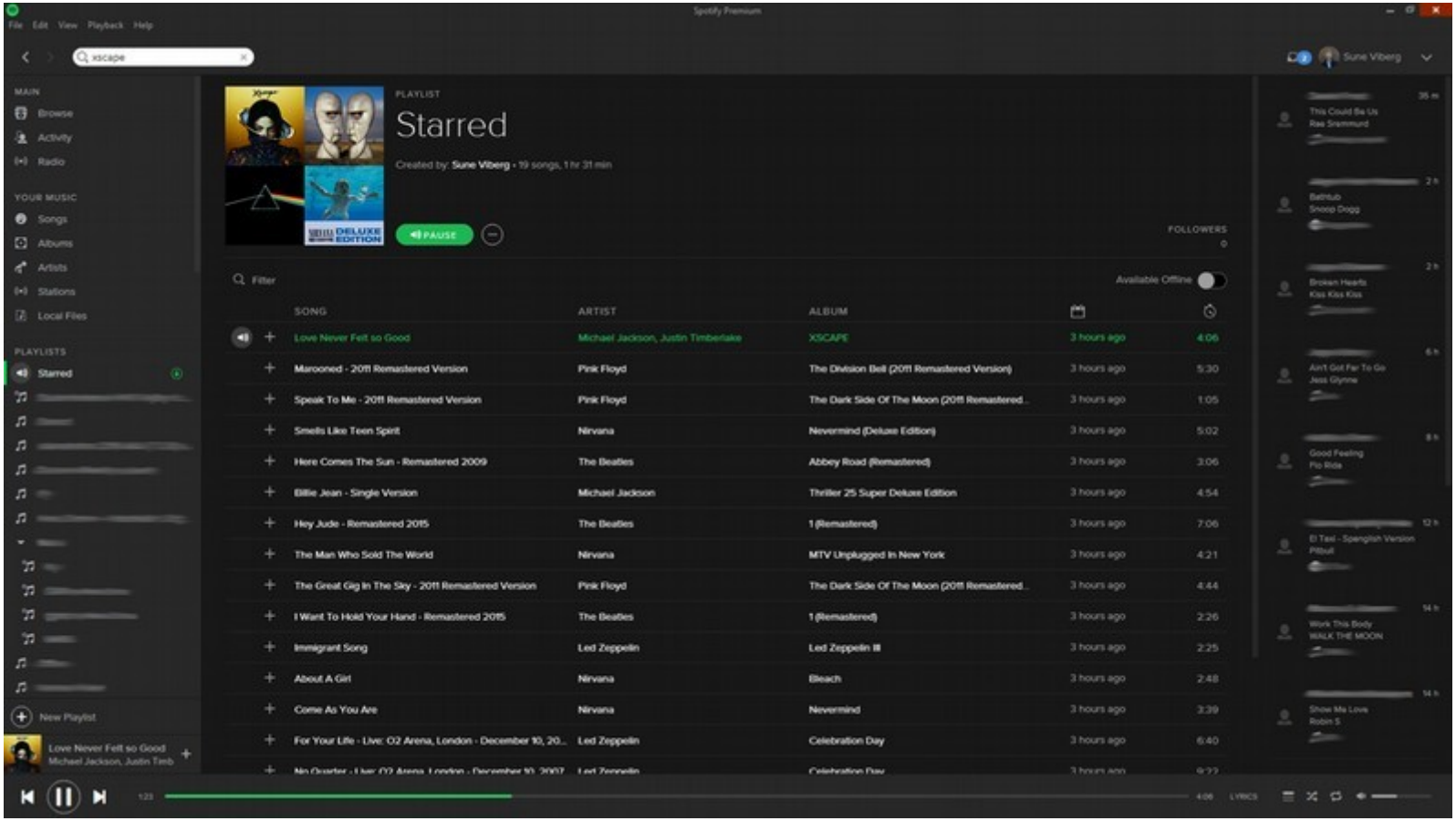


Why JavaScript?

Server-side
Software

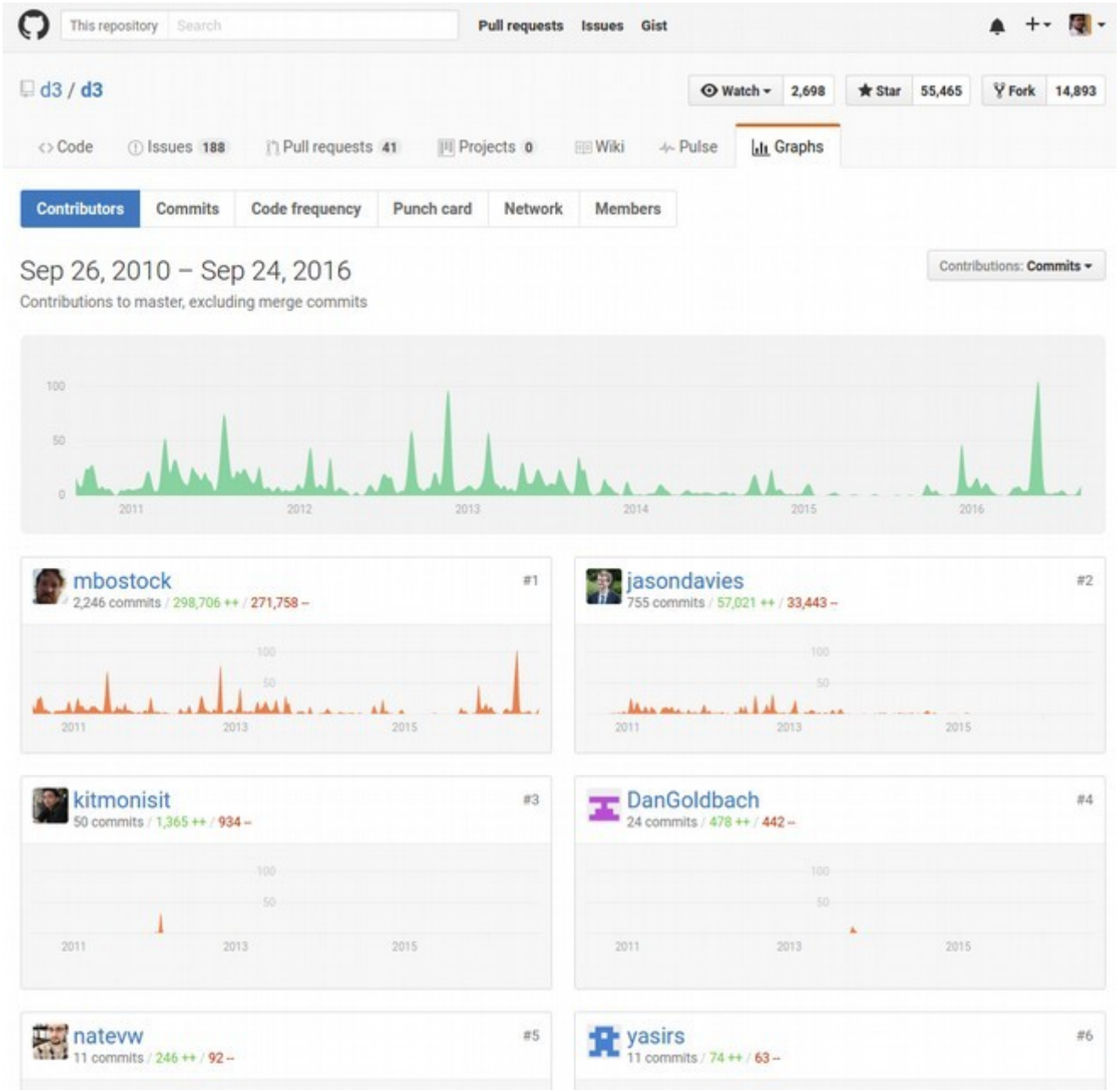
Why JavaScript?

Desktop Applications



Why JavaScript?

Realtime Analytics
& Graphics



Spec Releases

Year	Versions
1996	JS1.0 JS1.1
1997	JS1.2 ES1
1998	JS1.3 ES2
2000	JS1.5 ES3
2005	JS1.6
2006	JS1.7
2008	JS1.8 JS1.8.1 ES4
2009	JS1.8.2 ES5
2010	JS1.8.5
2011	ES5.1
2015	ES6
2016	ES7

As Microsoft (JScript) & Adobe (ActionScript) were planning make their own browser script specs, there was a need to make a standardisation.

Hence, we had ECMAScript as a specification under ECMA.

It followed JS for a while.

Now JS follows ES spec.

That's why we shall be learning about ES6/7 changes rather than as new versions of JavaScript.

Influences

C, Java, Scheme, ML

Influences

C, Java, Scheme, ML

Functional Programming

Object Oriented Programming

Prototypical Inheritance

Influences

C, Java, Scheme, ML

Functional Programming

Functions are first class entities.

Object Oriented Programming

Prototypical Inheritance

Influences

C, Java, Scheme, ML

Functional Programming	Functions are first class entities.
Object Oriented Programming	Abstraction, Encapsulation, Polymorphism, Inheritence
Prototypical Inheritance	

Influences

C, Java, Scheme, ML

Functional Programming	Functions are first class entities.
Object Oriented Programming	Abstraction, Encapsulation, Polymorphism, ~~Inheritence~~
Prototypical Inheritance	

Influences

C, Java, Scheme, ML

Functional Programming	Functions are first class entities.
Object Oriented Programming	Abstraction, Encapsulation, Polymorphism, ~~Inheritence~~
Prototypical Inheritance	Prototypes are references to previous definitions

Recent Changes

Arrow Functions	`(x, y) => x + y`
Classes	`class Student {}`
Generators	`Function* () { yield 0; yield 1; return 0; }`
Object Literals	`{ [(alignInt) alignInt ? “right” : “left”]: 30 }`
Modules	`export { Student, makeStudent }; export default Student;`
Promises	`Promise.all([ getLunch, doHomework, goHome]).resolve/reject`
Destructuring	`(firstArg, ...restArgs) => return 0;`
Let + Const	`let a = 5; const b = 10;`
Iterators	
Proxies	

More on: http://babeljs.io/docs/learn-es2015/

Introduction

History

Syntax & Semantics

Objects

this keyword

Prototypes

Premitive Data Types

`null`

`undefined`

`Boolean`

`Number`

`Object`

`String`

`Function`

Premitive Data Types

```
“number”    === typeof 3
“number”    === typeof 3.4
“string”    === typeof “Vamsi”
“object”    === typeof { name: “Vamsi” }
“object”    === typeof [3, 4, 5]
“boolean”   === typeof true
“function”  === typeof Math.sqrt
“undefined” === typeof asdfjkl
null
```

Operators

Operator	Operation Description
`+`	Addition; Concatenation
`-`	Subraction
`*`	Multiplication
`/`	Division
`%`	Modulus
`++ --`	Increment, Decrement
`|| && !`	Logical OR, Logical AND, Logical NOT
`| & ^`	Bitwise OR, Bitwise AND, Bitwise XOR
`> < <= >=`	Conditional Operators
`== ===`	Equals, Typesafe Equals
`?:`	Ternary Operator

Conditional Structures

- if...else

```
if (Math.random() > 0.5) {
   console.log(“More than half.”);
} else {
   console.log(“Less than or equal to half.”);
}
```

- switch

```
switch(‘a’) {
   case ‘a’:
      console.log(“a”);
      break;
   case ‘b’:
      console.log(“b”);
      break;
   default:
      console.log(“None”);
}
```

Iterative Structures

- for

```
for (var i = 0; i < 10; i++) {
   console.log(i);
}
```

- while

```
var i = 0;
while (i < 10) {
   console.log(i);
}
```

- do...while

```
var i = 10;
do {
   console.log(i);
} while (i < 10);
```

Functions

Definition

```
function myFunction(arg1, arg2) {
   console.log(arg1, “and”, arg2, “passed”);

   return 0;
}
```

Functions

Anonymous

```
function (arg1, arg2) {
   console.log(arg1, “and”, arg2, “passed”);

   return 0;
}(3, “hello”);
```

Functions

Another Method of Definition

```
var myfunction = function (arg1, arg2) {
   console.log(arg1, “and”, arg2, “passed”);

   return 0;
}
```

Functions

Nested Function

```
function (arg1, arg2) {
   console.log(arg1, “and”, arg2, “passed”);

   function conc() {
      return arg1 + arg2;
   }

   console.log(“Concatenated:”, conc());

   return 0;
}(3, “hello”);
```

Functions

Recursion

```
function series(n) {
   if (n <= 0)
      return;
   else if (n < 2)
      return 1;
   else
      return series(n-1) + series(n-2);
}

series(5);
```

Functions

Higher Order Functions

```
function series(n, fn) {
   if (n <= 0)
      return;
   else if (n < 2)
      return 1;
   else
      return fn(series(n-1), series(n-2));
}

series(5, function(a, b) { return a + b });
```

Functions

Currying

```
function setNum(n) {
   return function setFunction(fn) {
      return function series() {
         if (n <= 0)
            return;
         else if (n < 2)
            return 1;
         else
            return fn(series(n-1), series(n-2));
      }
   }
}
var addTwoNums   = function(a, b) { return a + b }
var getFive      = setNum(5);
var doAddGetFive = getFive(addTwoNums);
doAddGetFive();
```

Functions

Currying

```
const setNum = (n) =>
   (fn) =>
      () => {
         if (n <= 0)
            return;
         else if (n < 2)
            return 1;
         else
            return fn(series(n-1), series(n-2));
      }
   }
}
const addTwoNums   = (a, b) => return a + b;
const getFive      = setNum(5);
const doAddGetFive = getFive(addTwoNums);
doAddGetFive();
```

Introduction

History

Syntax & Semantics

Objects

this keyword

Prototypes

Objects

Definition

```
var student = {
    name: “Vamsi”,
    year: 4,
    interests: [
        “programming”,
        “cycling”,
        “photography”
    ]
}

student.name     === “Vamsi”;
student[“name”] === “Vamsi”;
```

Objects

Can Include Functions

```
var student = {
    name: “Vamsi”,
    year: 4,
    interests: [
        “programming”,
        “cycling”,
        “photography”
    ]
    code: function(problem) { return solution; },
    peddle: function() { return distanceCovered; },
    clickPhoto: function() { return picture; }
}

student.code();
```

Objects

Can Include Objects and Array of Objects

```
var student = {
    name: “Vamsi”,
    year: 4,
    friends: [
        {
            name: “Ronak”,
            year: 4
        },
        {
            name: “Soumitro”,
            year: 4
        }
    ]
}

student.friend[0].name === “Ronak”;
```

JSON

JavaScript Object Notation

```
{
    name: “Vamsi”,
    year: 4,
    friend_count: 2,
    friends: [
        {
            name: “Ronak”,
            year: 4
        },
        {
            name: “Soumitro”,
            year: 4
        }
    ]
}
```

Introduction

History

Syntax & Semantics

Objects

this keyword

Prototypes

this

- Context

```
var student = {
    name: “Vamsi”,
    friends: [
        { name: “Ronak” },
        { name: “Soumitro” }
    ]
}

Student.showFriends = function() {
    this.friends.forEach(function(friend) {
        console.log(friend.name);
    });
}

student.showFriends();
```

Bind

```
var student = {
    name: “Vamsi”,
    friends: [
        { name: “Ronak” },
        { name: “Soumitro” }
    ]
}

showFriends = function() {
    this.friends.forEach(function(friend) {
        console.log(friend.name);
    }
}

var studentFriends = showFriends().bind(student);
StudentFriends();
```

Introduction

History

Syntax & Semantics

Objects

this keyword

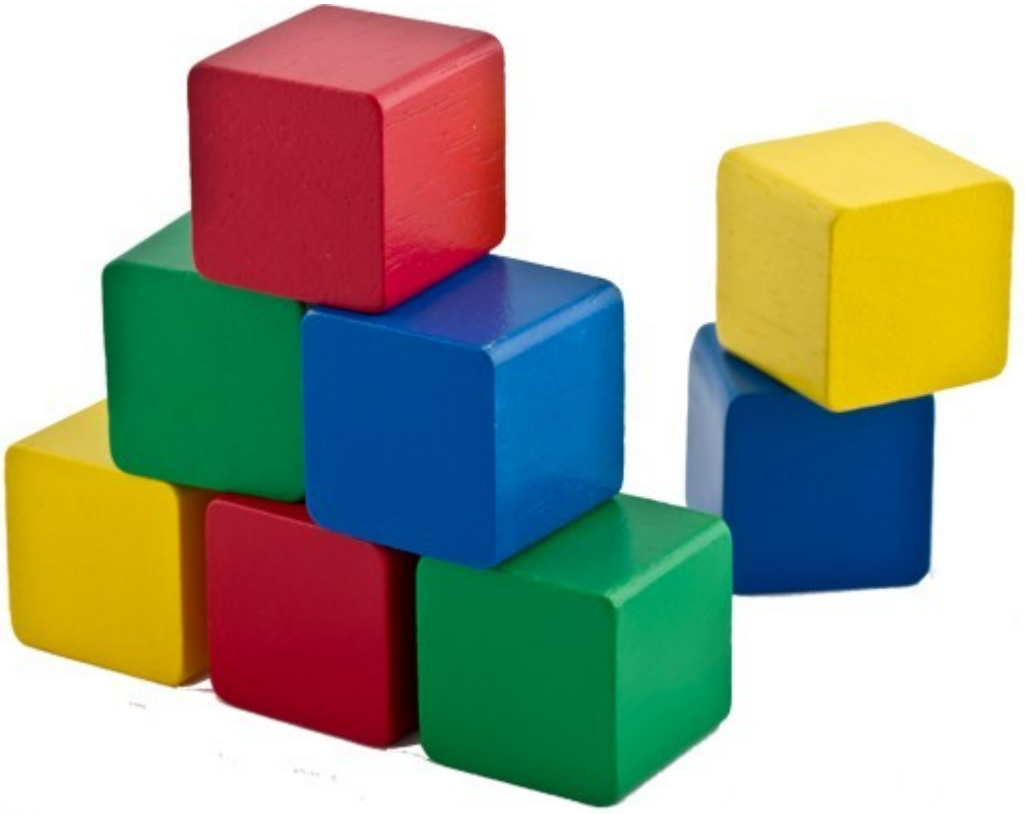
Prototypes

Prototype

- Property of objects
- Allows making ‘*new*’ instances of objects

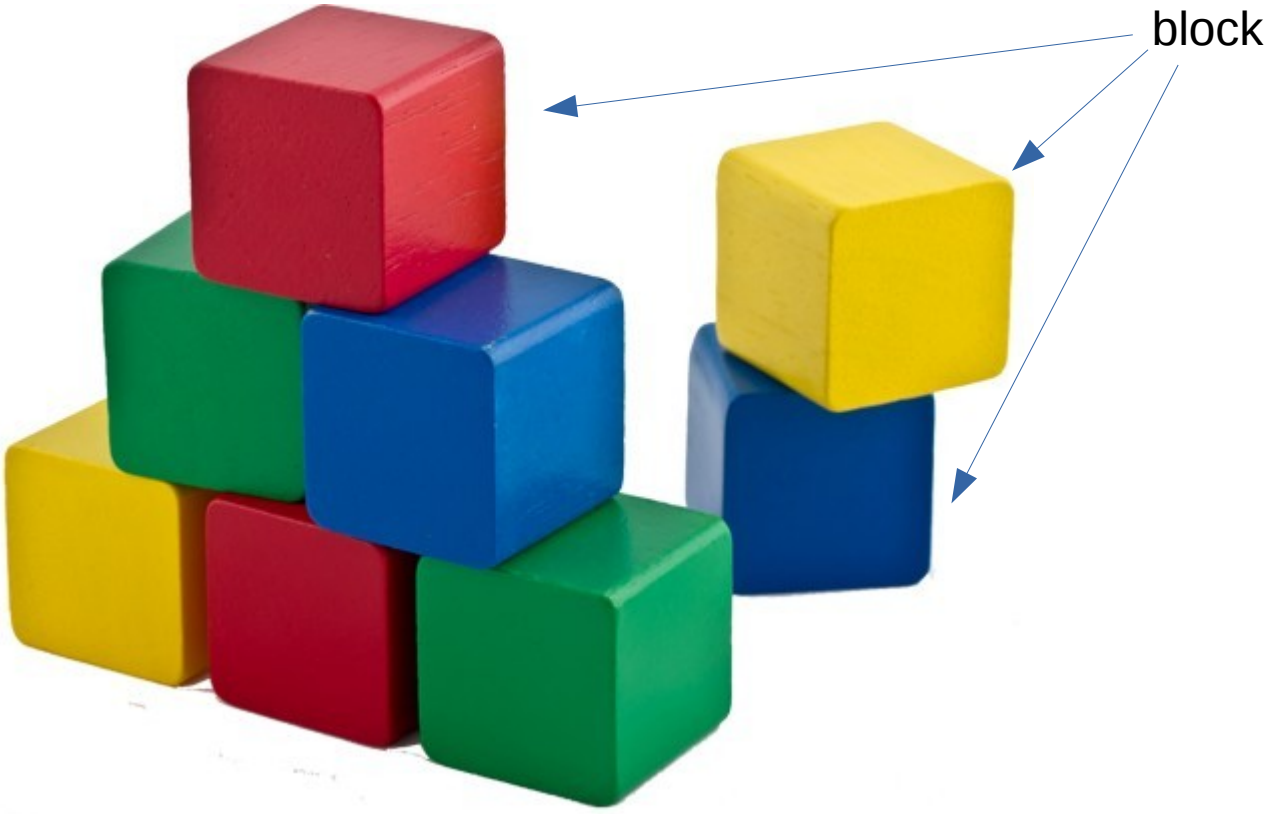
Prototype

- Property of objects
- Allows making '*new*' instances of objects



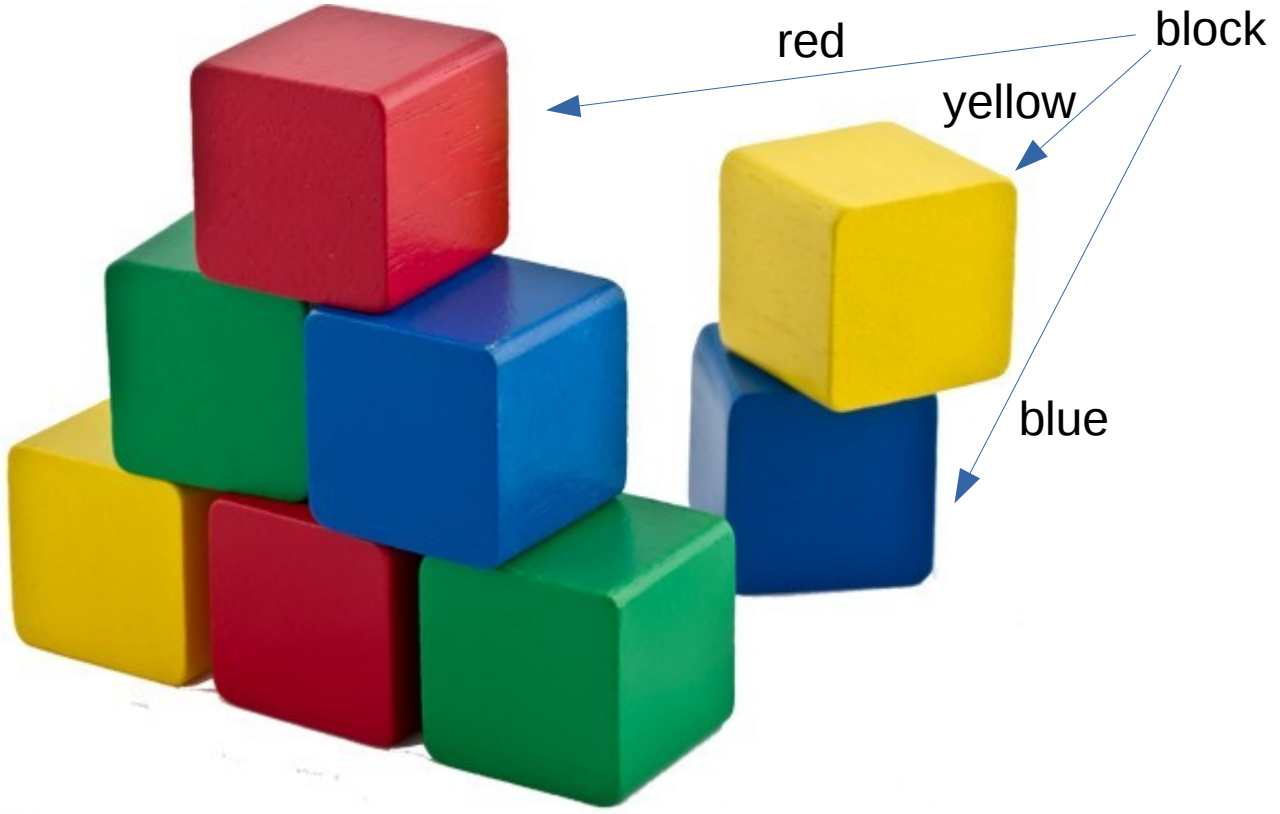
Prototype

- Property of objects
- Allows making '*new*' instances of objects



Prototype

- Property of objects
- Allows making '*new*' instances of objects



Prototype

```
var block = { shape: “cube” };

var redBlock = { color: “red” };
Object.setPrototype(redBlock, block);

Object.getPrototype(block);
```

Prototype

```
var Block = function() {};
Block.prototype.shape = “cube”;

var redBlock = new Block();
redBlock.color = “red”;

Object.getPrototype(redBlock);
RedBlock.shape === “cube”;
```

Questions?