Javascript

Daniel Zappala

CS 360 Internet Programming
Brigham Young University

Introduction

What You Should Read

- ▸ Eloquent JavaScript
  - introductory
  - interactive coding
- ▸ 'Why Prototypical Inheritance Matters'
  - advanced
  - prototypical inheritance

Why Javascript?

- client-side computation
- more responsive web user interface
  - asynchronous communication with server
  - dynamically change HTML being displayed by browser
- built into most browsers

Javascript vs Java

- JavaScript has no relationship to Java
- Javascript is becoming what Java was meant to be
  - lightweight, downloadable program that runs in browser and is compatible across many platforms
  - does much of what Java applets do, with a fraction of the resources

Features

- interpreted
- dynamic typing (delays binding of types until they are used)
- first-class functions (can take functions as arguments and return functions)
- prototypes (objects based on prototypes instead of inheritance)

Hello World

```
1  alert("Hello world!");
```

Functions

```
1  var factorial = function(n) {
2      if (n === 0) {
3          return 1;
4      }
5      return n * factorial(n - 1);
6  }
7  factorial(5);
```

DOM Parsing and Manipulation

```
1  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.
       w3.org/TR/html4/strict.dtd">
2  <html>
3    <meta charset="utf-8">
4    <head>
5      <title>simple page</title>
6    </head>
7    <body>
8      <h1 id="header">This is JavaScript</h1>
9      <script type="text/javascript">
10       document.write('Hello World!');
11       // get element with CSS ID "header"
12       var h1 = document.getElementById("header");
13       console.log(h1.innerHTML);
14       // get first h1 on the page
15       h1 = document.getElementsByTagName("h1")[0];
16       console.log(h1.innerHTML);
17       // change element
18       h1.innerHTML = "This is dope!";
19     </script>
20   </body>
21 </html>
```

Basics

Variables

```
1  // local variable
2  var x = 12;
3  // global variable
4  y = 12;
```

Operators

- comparison

```
1   >
2   <
3   >=
4   <=
5   !=
6   ==
7   !
8   ||
9   &&
10  === // no automatic type conversion
11  !== // no automatic type conversion
```

Operators

- expressions and assignment

```
1   +
2   -
3   *
4   /
5   %
6   =
7   +=
8   -=
9   ++
10  --
```

Control

```
if ( boolean statement ) {
   ...
} else {
   ...
}
```

```
switch (variable) {
   case 1:
      ...
      break;
   case 2:
      ...
      break;
   case default:
      ...
      break;
}
```

Control

```
while (boolean condition)
{
  ...
}
```

```
do
{
  ...
}
while (boolean condition);
```

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

Functions

Functions

```
1  var add = function(a,b) {
2      return a + b;
3  }
4
5  add(5,3);
```

Scope

```
var landscape = function() {
  var result = "";
  var flat = function(size) {
    for (var count = 0; count < size; count++)
      result += "_";
  };
  var mountain = function(size) {
    result += "/";
    for (var count = 0; count < size; count++)
      result += "'";
    result += "\\";
  };

  flat(3);
  mountain(4);
  flat(6);
  mountain(1);
  flat(1);
  return result;
};

console.log(landscape());
// ___/''''\______/'\_
```

Optional Parameters

```
1  function power(base, exponent) {
2    if (exponent == undefined)
3      exponent = 2;
4    var result = 1;
5    for (var count = 0; count < exponent; count++)
6      result *= base;
7    return result;
8  }
9
10 console.log(power(4));
11 // 16
12 console.log(power(4, 3));
13 // 64
```

Closure

```
1  var increment = function() {
2      var count = 0;
3      return function () {
4          return ++count;
5      };
6  }
7  console.log(increment());
8  // 1
9  console.log(increment());
10 // 2
```

Data Structures

Lists

```
1  var list = [2, 3, 5, 7, 11];
2  console.log(list[1]);
3  // 3
4  console.log(list.length);
5  // 5
6  list.push(13,17);
7  console.log(list);
8  // [ 2, 3, 5, 7, 11, 13, 17 ]
```

Objects

```
var user = {
    loggedIn: false,
    items: ["homework", "read The Martian", "play frisbee golf"],
    whatShouldIDo: function() {
        return "You should be doing " + this.items[0]
    }
};
console.log(user.loggedIn);
// false
console.log(user.name);
// undefined
user.name = "Emma";
console.log(user.name);
// Emma
console.log(user.whatShouldIDo());
// You should be doing homework
```

Higher-Order Functions

forEach

- executes a function for each element in a list

```
1  var pets = ["dog","cat","lizard"]
2  pets.forEach(function(pet) {
3    console.log(pet.toUpperCase());
4  });
5  // DOG
6  // CAT
7  // LIZARD
```

filter

- returns a new list by applying a function that checks for membership

```
1  var housepets = pets.filter(function(pet) {
2      return (pet === "dog" || pet == "cat");
3  });
4  console.log(housepets);
5  // [ 'dog', 'cat' ]
```

map

- creates a new list by applying a function to transform all elements

```
1  var uppers = pets.map(function(pet) {
2      return pet.toUpperCase();
3  });
4  console.log(uppers);
5  // [ 'DOG', 'CAT', 'LIZARD' ]
```

reduce

- returns a value by applying a function to all elements

```
1  var total = pets.map(function(pet) {
2        return pet.length;
3  }).reduce(function(a,b) {
4        return a + b;
5  });
6  console.log(total);
7  // 12
```

Prototypes

Prototypes

- Object.create() clones an object, specifying its prototype

```
1  var rectangle = {
2      area: function () {
3          return this.width * this.height;
4      }
5  };
6  var rect = Object.create(rectangle);
7  rect.width = 5;
8  rect.height = 10;
9  console.log(rect.area());
```

see ▸ Why Prototypical Inheritance Matters

Simplifying Object Creation

- wrap create into a function

```
1  var rectangle = {
2    create: function (width, height) {
3      var self = Object.create(this);
4      self.height = height;
5      self.width = width;
6      return self;
7    },
8    area: function () {
9      return this.width * this.height;
10   }
11 };
12 var rect = rectangle.create(5, 10);
13 console.log(rect.area());
```

Overriding Functions

- square calls its own create(), which calls create() of rectangle
- square calls area() on rectangle

```
1  var square = Object.create(rectangle);
2
3  square.create = function (side) {
4     return rectangle.create.call(this, side, side);
5  };
6  var sq = square.create(5);
7  console.log(sq.area());
```

Extending Objects

- extend() creates a new object that inherits properties from "this" (delegation) and copies properties from "extension" (concatenation)

```
1  Object.prototype.extend = function (extension) {
2      var hasOwnProperty = Object.hasOwnProperty;
3      var object = Object.create(this);
4
5      for (var property in extension)
6          if (hasOwnProperty.call(extension, property) ||
7              typeof object[property] === "undefined")
8                  object[property] = extension[property];
9
10     return object;
11 };
```

Extending The Square

- extends rectangle with create

```
1  var square = rectangle.extend({
2      create: function (side) {
3          return rectangle.create.call(this, side, side);
4      }
5  });
6
7  var sq = square.create(5);
8
9  console.log(sq.area());
```

Extending The Rectangle

- extends Object with height, width

```
1  var rectangle = {
2    create: function (width, height) {
3      return this.extend({
4        height: height,
5        width: width
6      });
7    },
8    area: function () {
9      return this.width * this.height;
10   }
11 };
12
13 var rect = rectangle.create(5, 10);
14 console.log(rect.area());
```