Lecture 9: Revision

Building Modern Web Applications
Vancouver Summer Program 2018 (Package E)

Julien Gascon-Samson, Karthik Pattabiraman

The Univerity of British Columbia
Department of Electrical and Computer Engineering
Vancouver, Canada

Electrical and Computer Engineering

Tuesday, August 7, 2018

L1: HTML and CSS

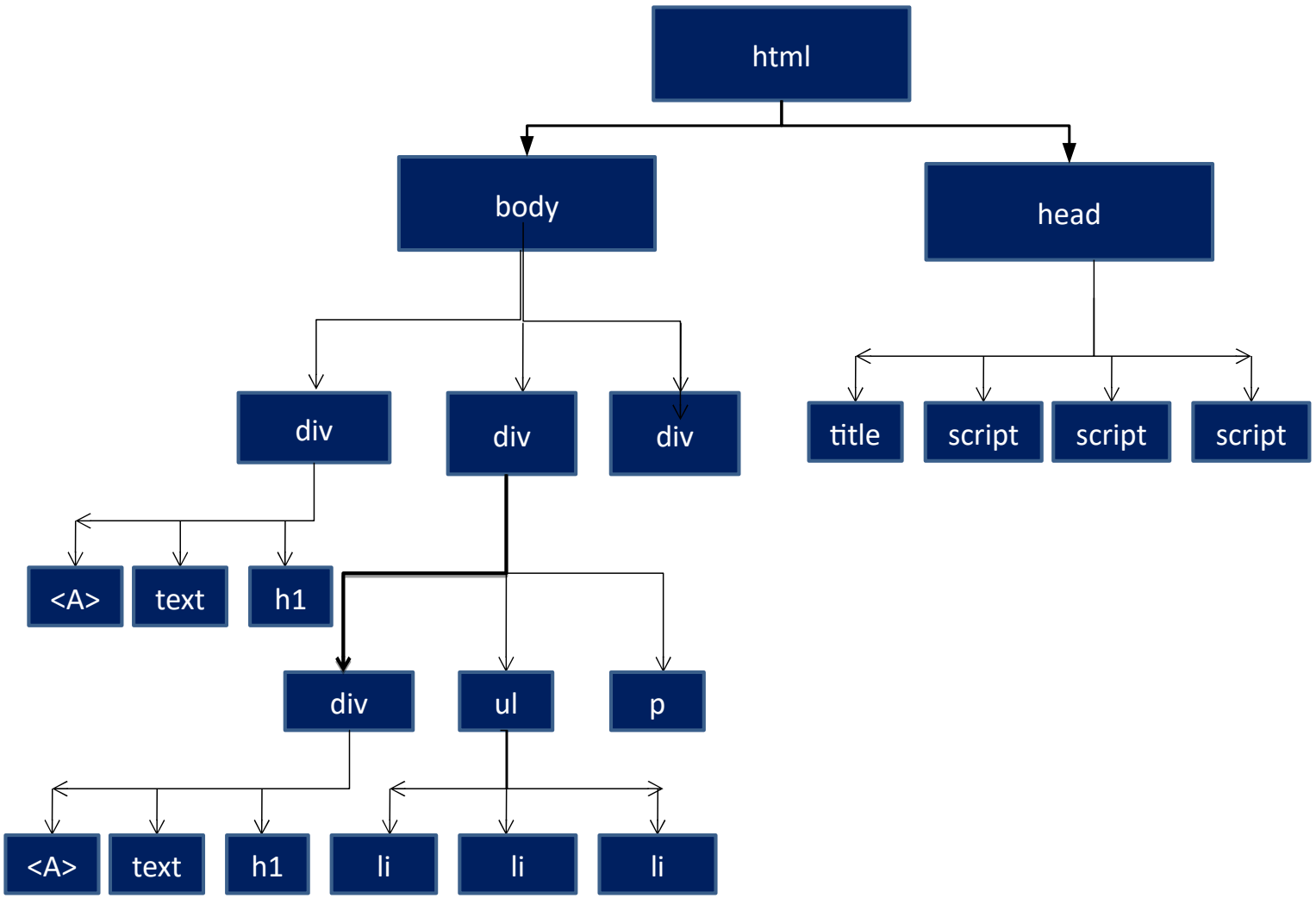
HTML Page

- Base of any web application
- Defines the *structure* and the *data*
- Hierarchical layout: DOM
- Should not containing styling information! $\Rightarrow$ better to link externally!
- Should not contain scripts!$\Rightarrow$ better to link externally!

HTML - DOM Example


```
<html>
<head>
        <title> …. </title>
        <script> …. </script>
        <script> … </script>
     <script> … </script>
</head>
<body>
   <div> <A> …. </A> <text> … </text> <hl> … </hl> </div>
   <div>
   <div> <A> …. </A> <text> … </text> <hl> … </hl>  </div>
   <ul> <li>…</li> <li>…</li><li> … </li> </ul>
   <p> … </p>
   </div>
  <div> … </div>
</body>
</html>
```

HTML - DOM Example - Tree



Some special tags

- head: nodes not pertaining to the visual representation of the page (title, stylesheets, scripts, meta-information, etc.).
- div: logically regroup elements (i.e., for styling)
- span: logically regroup *inline* elements
- script: includes JS code (inline or through an external JS file)
- style: define CSS styling information (inline)
- link: include an external resource (usually to include an external CSS stylesheet)

CSS - Principles

- Declarative language used to apply styles (mostly visual) to the DOM tree of the web page
- Made of a set of rules
  - Selector: query to match some of the nodes
  - Styling rule definitions: apply to matching nodes AND all their descendants

Common selectors:

- By ID (#id)
- By Class (.class)
- By Tag (tag)

Conflicting rules (i.e., same rule defined for the same node):

- Most specific selector (i.e., ID > Class > Tag)
- Most specific node (i.e., the node itself > parent > etc.)

CSS - Example


```html
<!doctype html>
<html>
  <head>
  <meta charset="UTF-8">
  <title>Sample document</title>
  <link rel="stylesheet" href="style1.css">
  </head>
  <body>
    <p id="first">
      <strong class="carrot">C</strong>ascading
      <strong class="spinach">S</strong>tyle
      <strong class="spinach">S</strong>heets
    </p>
    <p id="second">
      <strong>C</strong>ascading
      <strong>S</strong>tyle
      <strong>S</strong>heets
    </p>
  </body>
</html>
```

```css
strong { color: red; }
.carrot { color: orange; }
.spinach { color: green; }
#first { font-style: italic; }
```

Cascading Style Sheets

Cascading Style Sheets

L2: Basic Javascript

Variables, Comments and Functions


```
1  // This is a comment
2  /*
3     This is also a comment
4  */
5
6  // This is a function
7  function areaOfCircle(radius) {
8     var PI = 3.1416;
9     return PI * square(radius);
10 }
11
12 function square(x) {
13    return x*x;
14 }
15
16 // This is a variable
17 var A = areaOfCircle(2);
```

Functions - Nesting


```
1  function areaOfCircle(radius) {
2      var PI = 3.1416;
3
4      function square(x) {
5          // Only invokable inside areaOfCircle
6          return x*x;
7      }
8
9      return PI * square(x);
10 }
11
12 var A = areaOfCircle(2);
```

Scope

- Global scope: all JS code in the web page context (A)
- Local scope: within a function and sub-functions (PI, sq)
- Parameters: within a function and sub-functions (same as local scope) (radius, x)

Simple Arrays


```
1  var vspResults = [99, 96, 93, 91, /* ... */, 41];
2  /* Printing the grade of the top 3 students -- be careful,
3     in most programming languages, the first index is 0! */
4  console.log("Grade of the 1st student: " + vspResults[0]);
5  console.log("Grade of the 2nd student: " + vspResults[1]);
6  console.log("Grade of the 3rd student: " + vspResults[2]);
```

- Adding an item to the end of an array:

```
1  var vspResults = [99, 96, 93, 91, /* ... */, 41];
2  vspResults.push(39);
```

- Removing an item at the end of an array:

```
1  var vspResults = [99, 96, 93, 91, /* ... */, 41, 39];
2  vspResults.pop(); // Removes 39
```

- Getting length of an array:

```
1  var vspResults = [99, 96, 93, 91, /* ... */, 41];
2  console.log( vspResults.length );
```

Boolean Expressions and Operators

- Condition that evaluates to true or false
- Operators:
  - Equals: ==
  - Different than: !=
  - Greater than: >
  - Greater than or equal to: >=
  - Smaller than: <
  - Smaller than or equal to: <=
- In addition to:
  - Equals and same type: ===
  - Different than or different type: !==
- x && y: true if both x and y are true
- x || y: true if at least x or y is true
- !x: true if x is false!

Conditions

- If-Statements:

```
1  var score = 75;
2  var grade = "";
3
4  if ( score >= 80 ) {
5     grade = "A";
6  } else if (score >= 70) {
7     grade = "B";
8  } else if (score >= 60) {
9     grade = "C";
10 } else if (score >= 50) {
11    grade = "D";
12 } else {
13    grade = "F";
14 }
15
16 console.log("Your grade is " + grade);
```

Loops

- For loops:

```
1  var vspResults = [99, 96, 93, 91, /* ... */, 41];
2  for (var i = 0; i < 3; i++) {
3      console.log("Score #" + (i+1) + ": " + vspResults[i]);
4  }
```

- While loops:

```
1  while (condition) {
2      // Do stuff...
3  }
```

- Do-While loops:

```
1  do {
2      // Do stuff...
3  } while (condition);
```

L3: Objects and Functions

Associative Arrays as Objects


```
1  // Initializing an empty object
2  var empty_object = {};
3
4  // Object with two attributes
5  var name = {
6      firstName: "Karthik",
7      lastName: "Pattabiraman";
8  };
```

NOTE: You don't need a quote around firstName and lastName as they're valid JavaScript identifiers

```
1  name["firstName"]
2  // Equivalent to:
3  name.firstName
4
5  name["lastName"]
6  // Equivalent to:
7  name.lastName
8
9  name["firstName"] = "Different firstName";
10 name.lastName = "Different lastName";
```

Objects, Object Instances and Constructor

- Similar in spirit to defining classes in OO languages

```
1  var Person = function(firstName, lastName, gender) {
2      this.firstName= firstName;
3      this.lastName = lastName;
4      this.gender = gender;
5
6      this.printName = function() {
7          console.log(this.firstName + " " + this.lastName);
8      };
9  }
10 var p = new Person("John", "Smith", "Male");
```

Variadic Functions

- Unknown number of arguments

```
1  var addAll = function( ) {
2          var p = new Point(0,0);
3          for (var i=0; i<arguments.length; i++) {
4                  var point = arguments[i];
5                  p.x = p.x + point.x;
6                  p.y = p.y + point.y;
7          }
8          return p;
9  }
```

Exceptions

- Flagging an error condition

```
1  var addAll = function( ) {
2     var p = new Point(0,0);
3     for (var i=0; i<arguments.length; i++) {
4        var  point = arguments[i];
5        if ( p.x==undefined || p.y==undefined )
6           throw { name: TypeError,
7              message: "Object " + point + " is not of type
                   Point"
8        };
9          p.x = p.x + point.x;
10         p.y = p.y + point.y;
11     }
12     return p;
13 }
```

Functions as Objects

- A function can be passed to another function
  - Anonymous functions

```
1  var map = function( array, fn ) {
2      // Applies fn to each element of list, returns a new list
3      var result = [];
4      for (var i = 0; i < array.length; i++) {
5          var element = array[i];
6          result.push( fn(element) );
7      }
8      return result;
9  }
10
11 map( [3, 1, 5, 7, 2], function(num) { return num + 10; } );
```

L4: DOM Manipulation

Selecting nodes in the DOM: Selectors

getElementById

```
1  if (document.getElementById("Section1") == null)
2      throw new Error("No element found: " + name);
```

getElementsByTagName

```
1  var imgs = document.getElementsByTagName("img");
2  for (var i=0; i<images.length; i++)
3      imgs[i].display = "none";
```

getElementsByClassName

```
1  var warnings = document.getElementsByClassName("
       warning");
2  if (warnings.length > 0) { /* do something */ }
```

Alternative: QuerySelector API

- Perform more complex selection queries
- Results are *not* live lists !
- querySelector() returns the first element
- querySelectorAll() returns all elements

```
1  "#nav"              // Any element with id=nav
2
3  "div"             // Any <div> element
4
5  ".warning"          // Any element with "warning" class
6
7  "#log span"           // Any <span> descendant of id="log"
8
9  "#log > span"        // Any span child element of id="log"
10
11 "body>h1:first-child" // first <h1> child of <body>
12
13 "div, #log"         // All div elements, element with id="log"
```

DOM Nodes Properties: Traversal

parentNode

Parent node of this one, or null

childNodes

A read only array-like object containing all the (live) child nodes of this one

firstChild, lastChild

The first and lastChild of a node, or null if it has no children

nextSibling, previousSibling

The next and previous siblings of a node (in the order in which they appear in the document)

DOM Nodes Properties: Extracting Information

nodeType: 'kind of node'

- Document nodes: 9
- Element nodes: 1
- Text nodes: 3
- Comment node: 8

nodeValue

Textual content of Text of comment node

nodeName

Tag name of a node, converted to upper-case

Creating, Copying and Inserting and Removing Nodes

- Creating a new node:

```
1  var newNode = document.createTextNode("hello");
2  var elNode = document.createElement("h1");
```

- *Cloning* a node:

```
1  var existingNode = document.getElementByid("my");
2  var newNode = existingNode.cloneNode( true );
```

- appendChild: adds a new child node at the end
- insertBefore: similar, but inserts it before another child node

```
1  var s = document.getElementByID("my");
2  s.appendChild(newNode);
3  s.insertBefore(newNode, s.firstChild);
```

- removeChild: remove a given node
- replaceChild: replace a given child node with another node

L5: Closures, Window Object and Events

Closures - Creating an array of counters


```
1  var MakeCounters3 = function(n) {
2     var counters = [];
3     for (var i=0; i<n; i++) {
4        counters[i] = function( ) {
5           var initial = i, val = initial;
6           return {
7              increment: function() { val++; },
8              get: function() { return val; },
9              reset: function() { val = initial; }
10          }
11       }();  // Why do we need the parentheses () ?
12    }
13    return counters;
14 }
15 var m = MakeCounters3(10);
16 for (var i=0; i<10; i++) {
17    document.writeln("Counter[ " + i + "] = " + m[i].get());
18 }
```

window object - Timers

- setTimeout: schedule a future event asynchronously **once** after a specified no of milliseconds
  - Clear: clearTimeout method
- setInterval schedule a future event asynchronously **that periodically repeats itself** after a specified interval (milliseconds)
  - Clear: clearInterval method

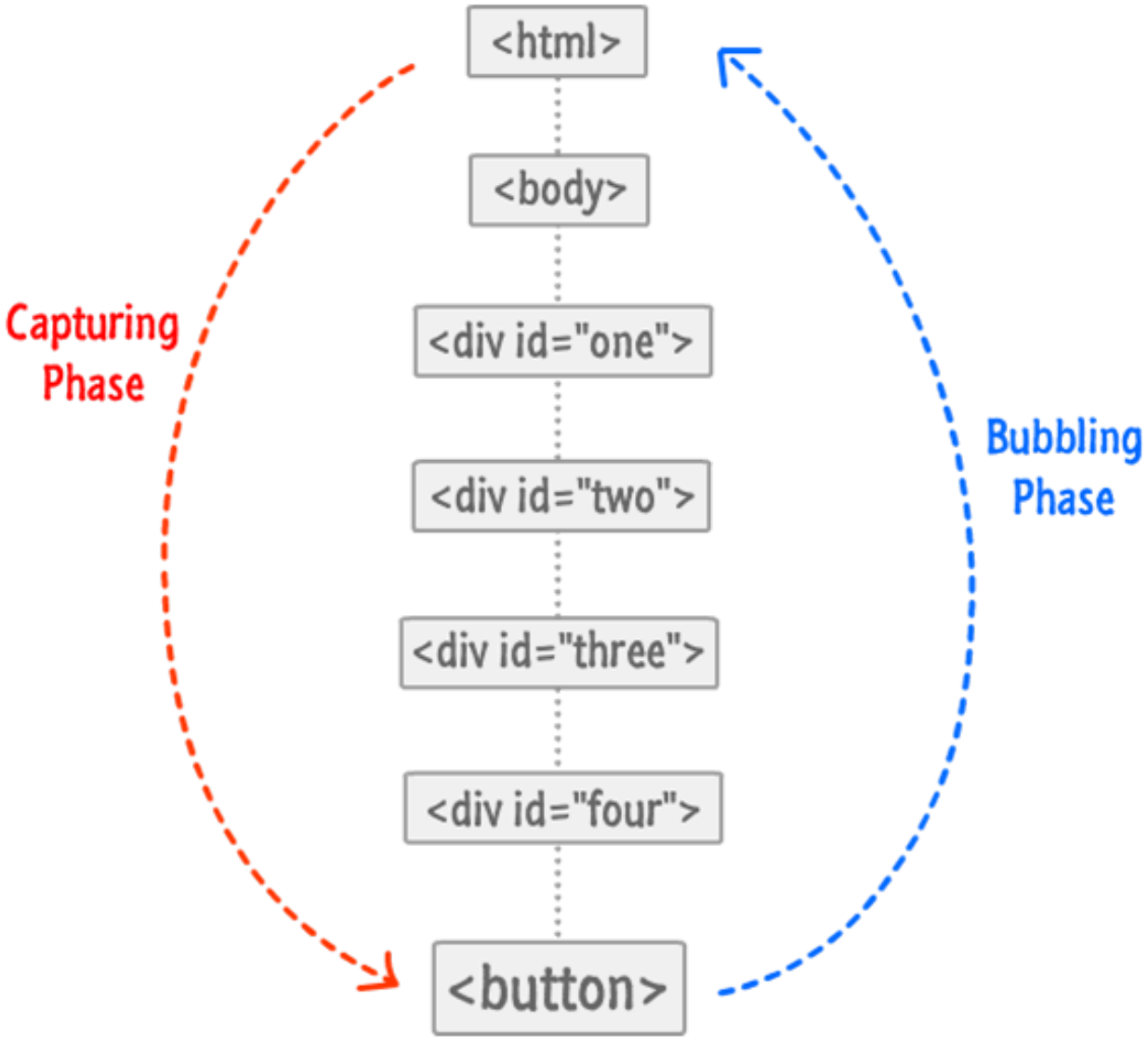
```
1  var intervalHandler = function(message) {
2      var i = 0;
3      return function() {
4          alert(message + '  ' + i);
5          i+= 1;
6      }
7  };
8  var ret = setInterval(intervalHandler("invocation"),1000);
       // [...]
9  if (flag) clearInterval(ret);
```

DOM 2.0 Events

- Allowing to be notified of internal events and user actions in all nodes of the DOM
- addEventListener for adding a event handler
- removeEventListener for removing event handlers
- stopPropagation and stopImmediatePropagation for stopping the propagation of an event

```
1  var b = document.getElementById("mybutton");
2  b.addEventListener("click", function() {
3          alert("hello");
4      }, false);
5  b.addEventListener("click", function() {
6          alert("world");
7      }, false);
```

DOM 2.0 Events - Capture and Bubbling Phases



L6: Prototypes and Reflection

Prototype Field


```
1  var p = new Person("John", "Smith", "Male");
2  console.log( Object.getPrototypeOf(p) );
```

- Prototypes of objects created through {} is
  - Object.prototype
- Prototype of objects created using new Object
  - Object.prototype
- Prototype of objects created using new and constructors functions (e.g., Person)
  - Prototype field set according to the constructor function (if object) (e.g., Person)

Prototype - Example


```
1  function Point( x, y) {
2        this.x = x; this.y = y;
3
4  };
5
6
7  var p1 = new Point(2,3);
8  var p2 = new Point(5,7);
9
10 console.log(Object.getPrototypeOf(p1) == Object.
       getPrototypeOf(p2));
11 console.log(Object.getPrototypeOf(p1).constructor);
```

```
1  Point.prototype.toString = function( ) {
2        return "(" + this.x + " , " + this.y + ")";
3  }
```

Prototype Inheritance


```
1  var Employee = function(firstName, lastName, Gender, title)
       {
2      Person.call( this, firstName, lastName, Gender );
3      this.title = title;
4  }
5
6  Employee.prototype = new Person();
7      /* Why should you create a new person object ? */
8
9  Employee.prototype.constructor = Employee;
10
11 var emp = new Employee("ABC", "XYZ", "Male", "Manager");
12 console.log(emp.firstName); // OK - Property defined in
       Person
```

Prototype Inheritance - Assoc. array / *Object.create*


```
1  var Person = {
2      firstName:  "John";
3      lastName: "Smith";
4      gender: "Male";
5      print : function() {
6          console.log( "Person : " + this.firstName
7                  + this.lastName + this.gender;
8      }
9  };
10 var e = Object.create( Person );
11 e.title = "Manager";
```

Reflection and Type-Checking

- In JS, you can query an object for its type, prototype, and properties at runtime
  - To get the Prototype: getPrototypeOf()
  - To get the type of: typeof
  - To check if it's of certain instance: instanceof
  - To check if it has a certain property: in
  - To check if it has a property, and the property was not inherited through the prototype chain: hasOwnProperty()

Iterating over all properties

```
1  var name;
2  for (name in obj) {
3      if ( typeof( obj[name] ) != "function") {
4          document.writeln(name + " : " + obj[name]);
5      }
6  }
```

L7: AJAX and JSON

AJAX - XHR2 Model - Example (1)

- Launching a request and getting the response:

```
1  var xhr = new XMLHttpRequest();
2  xhr.open("GET", "example.html");
3  xhr.onload = function() {
4      if (xhr.status==200) {
5          console.log( xhr.responseText );
6          console.log("Request success");
7      }
8  }
9  xhr.send();
```

- Aborting a request: abort method

```
1  xhr.onabort = function() {
2      console.log("Request aborted");
3  }
```

AJAX - XHR2 Model - Example (2)

- Handling timeouts:

```
1  xhr.timeout = 200; // 200 ms timeout
2  xhr.ontimeout = function() {
3      console.log("Request timed out");
4  };
```

- Handling errors (all other errors which are NOT status code-related!)

```
1  xhr.onerror = function() {
2      console.log("error occurred on request");
3  }
```

JSON

- JSON.parse(string): converts string to JavaScript (code/data)
- JSON.stringify(object): converts object to JSON notation
- Header must be set to "Application/JSON"

Example

```
1  var xhr = new XMLHttpRequest();
2  xhr.open("GET", "example.html");
3  xhr.onload = function() {
4      if (xhr.status==200) {
5          if ( xhr.getResponseHeader("Content-type")
               == JSON) {
6          var result = JSON.parse(xhr.responseText);
7          // Do something with the result variable
               here
8      }
9  }
10 xhr.send();
```

L8: Node.js

Node.js - Motivation

- Writing JS programs that can be run independently of a browser, as JS is a powerful programming language on it's own
  - Notable use: web server / web services
- Easy sharing of code and data between a JS web app and a Node.js web service

```
1  console.log("Hello");  // Same as before
2  setTimeout( function() {  // Same as before
3        console.log("World") }, 1000);
4
5  // New stuff - can't do this in client-side JavaScript
6  var fs = require("fs");  // Load file system object
7  var contents = fs.readFilesync( fileName );
8  console.log(contents);
```

Modules

Calculator.js

```
1  function sum(a, b) { return a + b; }
2  module.exports.sum = sum;
```

Shapes.js

```
1  var Point = function(x, y) {
2      this.x = x; this.y = y; }
3  module.exports = Point;
```

```
1  var calculator = require("Calculator.js");
2  calculator.sum(10, 20);
3
4  var Point = require("Shapes.js");
5  var p = new Point(1, 2);
```

Event Streams

- Registering an event: on
- Triggering an event: emit

```
1  var EventEmitter = require('events').EventEmitter;
2  if (! EventEmitter) process.exit(1);
3  var myEmitter = new EventEmitter();
4
5  // Add event handlers
6  myEmitter.on("connection", function(id) { /* ... */ });
7  myEmitter.on("message", function(msg) { /* ... */ });
8
9  // Emit the events
10 myEmitter.emit("connection", 100);
11 myEmitter.emit("message", "hello");
```

File I/O: Reading a File

- Synchronous read: bad!

```
1  var f= fs.readFileSync(fileName);
```

- Asynchronous reads: much better

```
1  var fs = require("fs");     // Filesystem module in node.js
2  var length = 0;
3  var fileName = "sample.txt";
4
5  fs.readFile(fileName, function(err, buf) {
6      if (err) throw err;
7      length = buf.length;
8       console.log("Number of characters read = " + length);
9  } );
```

File I/O: Reading a File using Streams


```
1  var fs = require('fs');
2  var length = 0;
3  var fileName = "sample.txt";
4  var readStream = fs.createReadStream(fileName);;
5
6  readStream.on("data", function(blob) {
7      console.log("Read " + blob.length);
8      length += blob.length;
9  });
10
11 readStream.on("end", function() {
12     console.log("Total number of chars read = " + length);
13 });
14
15 readStream.on("error", function() {
16     console.log("Error occurred when reading from file " +
           fileName);
17 });
```

Piping Streams

- To copy a file, one can pipe the *output* of the read stream to the *input* of the write stream

```
1  var fs = require("fs");
2
3  // Open the read and write streams
4  var readStream = fs.createReadStream("sample.txt");
5  var writeStream = fs.createWriteStream("sample-copy.txt");
6
7  // Copies contents of read stream to write stream
8  readStream.pipe( writeStream );
```

Handling Http Connections using Streams


```
1  var http = require('http');
2
3  // Create a simple function to serve a request
4  var serveRequest = function(request, response) {
5     console.log("Received request " + request);
6     response.writeHeader(200, { "Content-type":"text/htm"});
7     response.write("Received: " + request.url);
8     response.end();
9  };
10
11 // Start the server on the port and setup response
12 var port = 8080;
13 var server = http.createServer();
14 server.on("request", serveRequest);
15 server.listen(port);
```

Table of Contents