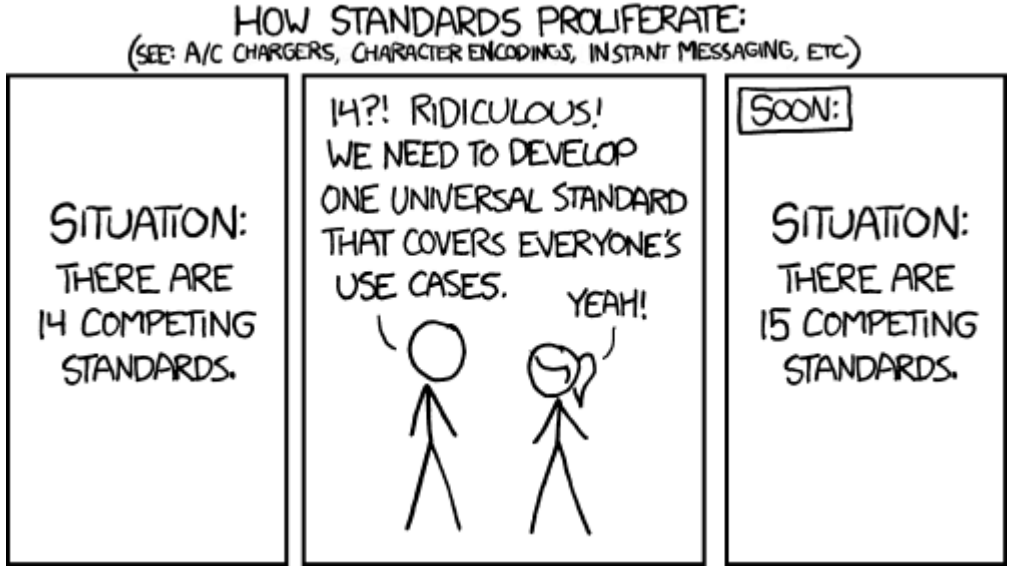
CSc 337



LECTURE 14: JSON AND WEB SERVICES

Weather - exercise

```
{"city": "London",
 "country": "CA",
 "weather": [
              {"temperature": 80.35,"icon": "02d"},
              {"temperature": 83.59,"icon": "01d"},
              {"temperature": 72.14,"icon": "10d"},
              {"temperature": 56.50,"icon": "01d"},
              {"temperature": 61.84,"icon": "01d"},
              {"temperature": 67.93,"icon": "01d"},
              {"temperature": 76.06,"icon": "01d"}
            ]
}
```

Write code to add an `h1` to a page containing the city name and a list with each element containing the temperature

reminder: bad style - the eval function

```
// var data = JSON.parse(this.responseText);
var data = eval(this.responseText);   // don't do this!
...
```
JS

- JavaScript includes an `eval` keyword that takes a string and runs it as code
- this is essentially the same as what `JSON.parse` does,
- but `JSON.parse` filters out potentially dangerous code; `eval` doesn't
- `eval` is evil and should not be used!

Writing server code

URLs and web servers

http://server/path/file

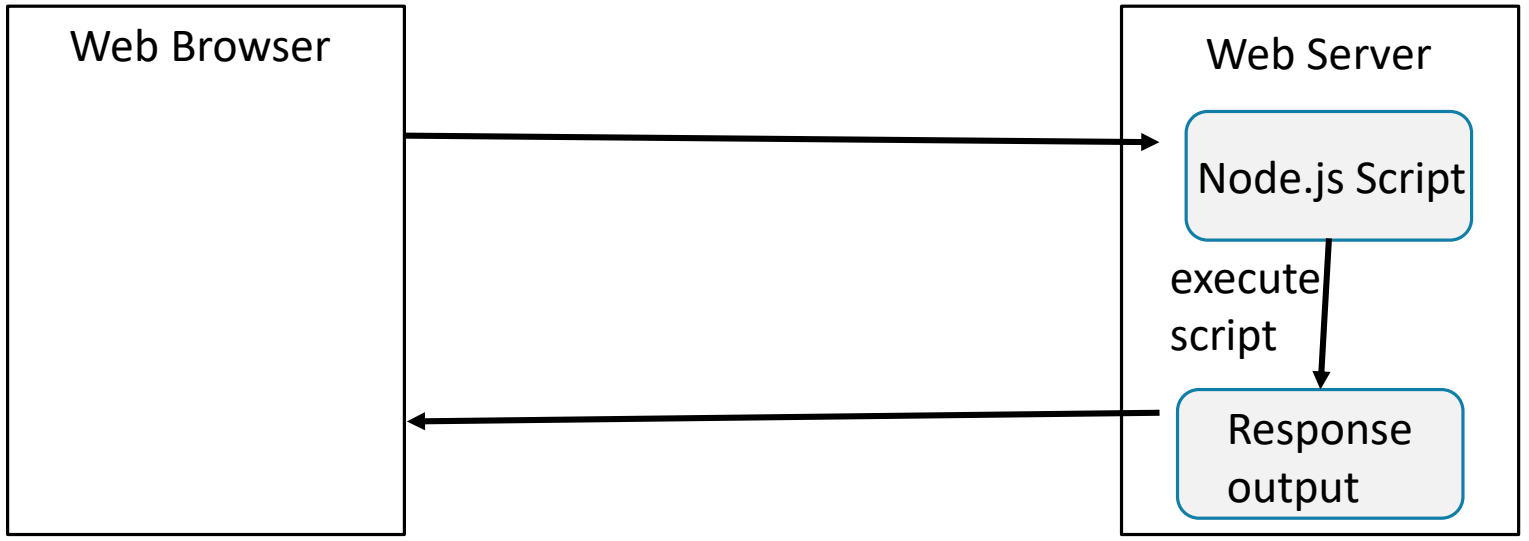
- usually when you type a URL in your browser:
  - your computer looks up the server's IP address using DNS
  - your browser connects to that IP address and requests the given file
  - the web server software (e.g. Apache) grabs that file from the server's local file system, and sends back its contents to you

- some URLs actually specify *programs* that the web server should run, and then send their output back to you as the result:
  `http://localhost:3000/service1.js`
  - the above URL tells the server `localhost:3000` to run the program `service1.js` and send back its output

Server-Side web programming

- server-side pages are programs written using one of many web programming languages/frameworks
  - examples: Node.js PHP, Java/JSP, Ruby on Rails, ASP.NET, Python, Perl
- the web server contains software that allows it to run those programs and send back their output
- each language/framework has its pros and cons
  - we will use Node for server-side programming

Lifecycle of a web request



- browser requests a `.html` file with no Ajax requests in the JavaScript
  (**static content**): server just sends that file
- browser requests a `.html` file with an Ajax request in the JavaScript
  (**dynamic content**): server reads it, runs any script code inside it

Basic web service

```
// CSC 337 hello world server

const express = require("express");
const app = express();

app.use(express.static('public'));

app.get('/', function (req, res) {
     res.header("Access-Control-Allow-Origin", "*");
    res.send('Hello World!');
})

app.listen(3000);
```

ExpressJS

- We're going to use a library called ExpressJS on top of NodeJS
- It is a lightweight framework that will help us organize our code

Basic web service

```
// CSC 337 hello world server

const express = require("express");
const app = express();

app.use(express.static('public'));

app.get('/', function (req, res) {
     res.header("Access-Control-Allow-Origin", "*");
    res.send('Hello World!');
})

app.listen(3000);
```

require()

```
const express = require("express");
```

The NodeJS `require()` statement loads a module, similar to `import` in Java or Python.

> We can `require()` modules included with NodeJS, or modules we've written ourselves.

`listen()`

```
app.listen(3000);
```

The `listen()` function will start accepting connections on the given port number.

Ports and binding

port: In the context of networking, a "logical" (as opposed to a physical) connection place

- A number from 0 to 65535 (16-bit unsigned integer)
- Used to distinguish a message for one program from another

When you start running a server process, you tell the operating system what port number to associate with it. This is called **binding**.

Port defaults

There are many well-known ports, i.e. the ports that will be used by default for particular protocols:

21: File Transfer Protocol (FTP)
22: Secure Shell (SSH)
23: Telnet remote login service
25: Simple Mail Transfer Protocol (SMTP)
53: Domain Name System (DNS) service
80: Hypertext Transfer Protocol (HTTP) used in the World Wide Web
110: Post Office Protocol (POP3)
119: Network News Transfer Protocol (NNTP)
123: Network Time Protocol (NTP)
143: Internet Message Access Protocol (IMAP)
161: Simple Network Management Protocol (SNMP)
194: Internet Relay Chat (IRC) 443: HTTP Secure (HTTPS)

Development Server

- We have been using 3000 in examples but you can use whatever number you want

```
app.listen(3000);
```

Avoiding CORS Errors

Allows us to access our code on localhost.

- otherwise NodeJS thinks we are on different machines

```
app.use(express.static('public'));
```

Making a Request

The type of request we are making right now is GET

`req`: an object representing the request

`res`: an object representing the response

```
app.get('/', function (req, res) {
    res.header("Access-Control-Allow-Origin", "*");
    res.send('Hello World!');
})
```

Get Query Parameters in Express

Query parameters are saved in **`req.query`**

```
app.get('/', function (req, res) {
    res.header("Access-Control-Allow-Origin", "*");
  ◦const queryParams = req.query;
  ◦console.log(queryParams);
  ◦const name = req.query.name;
    res.send('Hello' + name);
})
```

Exercise

Write a web service that takes an exponent and base as parameters and outputs the based raised to the exponent