JavaScript IV

HTTP Statelessness

- HTTP is a stateless protocol
    - After a request to a webserver is made, and the page returned, no state is tracked
- No way to know which users
    - Have visited your site before
    - Are in the middle of some multi-page process

Cookies

- A Cookie is a single key-value pair that is stored locally on the users computer
  - Exact location dependent on browser
- Two types
  - Session: Are deleted when the browser is closed
  - Persistent: Are deleted on the defined expiration date

Cookies

- Cookies can be used and modified by
  - JavaScript
  - Server Side Languages (PHP, Python, Java, Perl, etc.)
- Cookies were meant to hold small pieces of information
  - Cookies are part of HTTP itself, and are sent to a website **everytime** you make a request as part of the headers

Cookies in JavaScript

- Cookies in JavaScript are set and read using the `cookie` property of the `document` object
- From the JavaScript perspective, cookies can only hold one string of text
    - You can set the `cookie` property multiple times, it won't be overwritten
        - Instead it is appended to
    - When the `cookie` property is read, all cokies for a site are returned as a string
        - Seperated by ';'

In [ ]:
```
%%html
<!DOCTYPE>
<html>
    <head>
    </head>
    <body>
        <p id="cookieValue"></p>
        <script>
            document.cookie="course=433";
            document.cookie="department=CMSC";
            document.getElementById("cookieValue").innerHTML=
            document.cookie
        </script>
    </body>
</html>
```

Cookie Attributes

- There are numerous attributes of cookies that control how long they persist, or when they can be used
  - All are set when setting the cookie, separated by ';'

```
document.cookie = "name=value; attribute1=att_value1; attribute2
=att_value2"
```

- Common attributes
  - domain : What domain the cookie is valid for
  - expires: When the cookie should be deleted
  - max-age: How long the cookie should persist in seconds
  - secure: Prohibits cookies from being sent without HTTPS

In [ ]:

```
%%html
<!DOCTYPE>
<html>
    <head>
    </head>
    <body>
        <p id="cookieValue2"></p>
        <script>
            var expire = new Date('Wed, 31 Dec 2017 23:59:59 EST');
            document.cookie="course=433; expires=" + expire.toUTCString();
            document.cookie="department=CMSC";
            document.getElementById("cookieValue2").innerHTML=document.cookie
        </script>
    </body>
</html>
```

Cookies and Privacy

- Cookies have long be overused and abused
    - Don't store too much data in them, it slows down the connection
    - Don't store sensitive information in them
- Tracking of users may be considered in infringement on their privacy
    - Many browsers support a Do Not Track header, its up to the servers to respect this
    - EU members must display a disclaimer that they are being used

Modern Storage APIs

- All modern browsers support a newer, simpler API to store things locally, known as the Web Storage API
    - Makes reading and writing values much easier
    - Doesn't send on every HTTP request
        - More secure
    - Provides storage events that all tabs/pages can react to

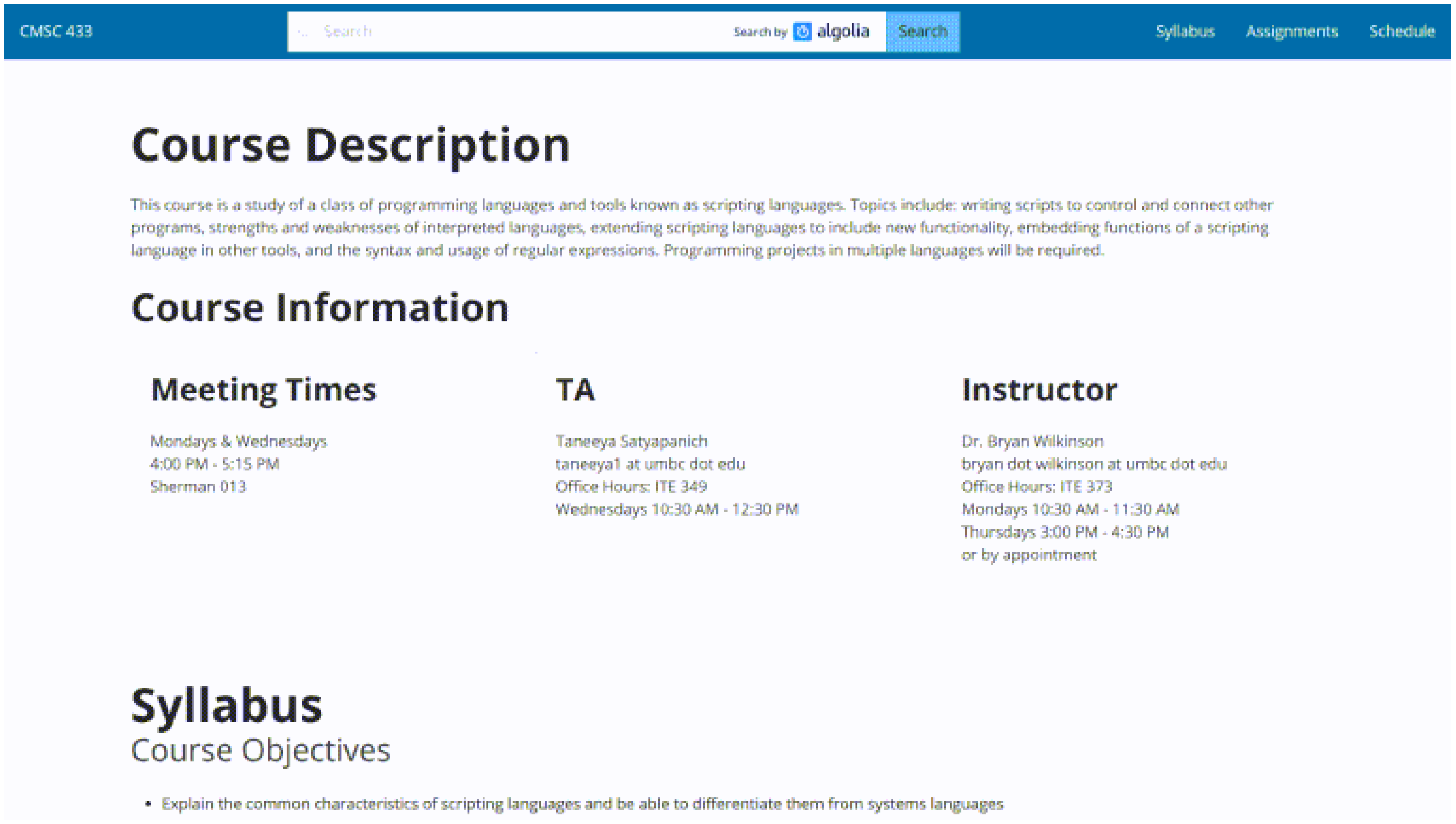
sessionStorage and localStorage

- Both `Storage` objects are members of the `window` object
  - `window` is the default object, so you will often see `sessionStorage` rather than `window.sessionStorage`
- `sessionStorage` is cleared when a new page is navigated to
- `localStorage` has no set expiration date
- Easy API to get and set key/value pairs
  - `setItem(name,key)`
  - `getItem(name,key)`

In [ ]:

```
%%html
<!DOCTYPE>
<html>
    <head>
    </head>
    <body>
        <button id="store">Click Here to Store Things</button>
        <p id="storageValue"></p>
        <script>
            document.getElementById("store").addEventListener('click',
            function(){
                window.sessionStorage.setItem('building',"Sherman");
                window.localStorage.setItem('room','015')
            });
            document.getElementById('storageValue').innerHTML =
                window.sessionStorage.getItem("building") + " " +
                window.localStorage.getItem("room");
        </script>
    </body>
</html>
```

Inspecting Local Storage in a Browser (Chrome)



Course Description

This course is a study of a class of programming languages and tools known as scripting languages. Topics include: writing scripts to control and connect other programs, strengths and weaknesses of interpreted languages, extending scripting languages to include new functionality, embedding functions of a scripting language in other tools, and the syntax and usage of regular expressions. Programming projects in multiple languages will be required.

Course Information

Meeting Times

Mondays & Wednesdays
4:00 PM - 5:15 PM
Sherman 013

TA

Taneeya Satyapanich
taneeya1 at umbc dot edu
Office Hours: ITE 349
Wednesdays 10:30 AM - 12:30 PM

Instructor

Dr. Bryan Wilkinson
bryan dot wilkinson at umbc dot edu
Office Hours: ITE 373
Mondays 10:30 AM - 11:30 AM
Thursdays 3:00 PM - 4:30 PM
or by appointment

Syllabus

Course Objectives

- Explain the common characteristics of scripting languages and be able to differentiate them from systems languages

AJAX

- **AJAX** *was* an acronym for Asynchronous Javascript And XML
  - No one really uses XML anymore
- JavaScript allowed the user to interact with what was on the page
  - What about getting new data after the page loaded
  - Prediction of Text in search
  - Allows to request data from multiple sources on one webpage
    - Google Maps
    - Yelp
    - Twitter

Brief History of AJAX

- Was first implemented in Internet Explorer
- Other browers quickly adopted it, but changed the method names
- Was based on XML (eXtensible Markup Language) due to heavy use in businuess at the time
- Today is standardized and XML is hardly used anymore

XMLHTTPRequest

- XMLHTTPRequest is the object used to initiate and interact with the request

```
var theRequest = new XMLHttpRequest();
```

- After we have the object, `open` is used to set where the data is from and how to get it
  - For security reasons, this location needs to be part of the same website
- `send` is used to add parameters and send the request to the URL given in the `open` parameter

```
theRequest.open('METHOD','location',Asynchronous?)
theRequest.send(ParametersObject)
```

Making the Request (GET)

- The method is the string "GET"
- A GET request requires all parameters to be passed as part of the URL
    - Sent in the location parameter of the `open` method
- There are no additional parameters to get, so `sent` is passed null

In [ ]:

```
%%html
<!DOCTYPE>
<html>
    <head>
    </head>
    <body>
        <input type="text" id="zip">
        <button id="lookup">Lookup</button>
        <p></p>
        <script>
            document.getElementById("lookup").addEventListener('click',
            function(){
                var request = new XMLHttpRequest();
                var zip = document.getElementById('zip').value;
                request.open('GET', './lookup.php?zip=' + zip);
                request.send(null);
            });

        </script>
    </body>
</html>
```

Listening for a Response

- `XMLHTTPRequest.send()` doesn't return anything
- To get the response, we must attach an event listener to the `XHR` object
    - Rather than use `addEventListner`, set the property `onreadystatechange`
- Set equal to a function that takes no parameters

The Response Object

- The reponse object holds all the information sent back from the server
  - Is the same as the request object actually
- `onreadystatechange` actually fires multiple times during the request, but we only care about it when its done usually
  - `response.readyState` holds where in the process the request is
    - 4 corrisponds to being done
- `response.status` holds the HTTP status of the request, it should be 200 if successful
- `response.responseText` holds the content returned from the server

In [ ]:

```
%%html
<!DOCTYPE>
<html>
    <head>
    </head>
    <body>
        <h3>Respose Object Stages</h3>
        <p>Response State: <span id="state"></span><p>
        <input type="text" id="zip">

        <button id="open">Open</button>
        <button id="lookup2">Lookup</button>
        <p id="city"></p>
        <script>
            var request = new XMLHttpRequest();
            request.onreadystatechange = function(){
                  document.getElementById("state").innerHTML = document.getElement
ById("state").innerHTML + " " + request.readyState;
                };
            document.getElementById("open").addEventListener('click',function(){
                var zip = document.getElementById('zip').value;
                request.open('GET', 'https://www.csee.umbc.edu/~bwilk1/lookup.php?
zip=' + zip);

            });
            document.getElementById("lookup2").addEventListener('click',
            function(){
                request.send(null);
            });

        </script>
    </body>
</html>
```

In [ ]:

```
%%html
<!DOCTYPE>
<html>
    <head>
    </head>
    <body>
        <input type="text" id="zip3">
        <button id="lookup3">Lookup</button>
        <p id="city3"></p>
        <script>
            document.getElementById("lookup3").addEventListener('click',
            function(){
                var request = new XMLHttpRequest();
                var zip = document.getElementById('zip3').value;
                request.open('GET', 'https://www.csee.umbc.edu/~bwilk1/lookup.php?
zip=' + zip);
                request.onreadystatechange = function(){
                  if(request.readyState == 4){
                      if(request.status == 200){
                          var info = request.responseText.split(":");

                          document.getElementById("city3").innerHTML = info[0] +
 "," + info[1]
                      }
                  }
                };
                request.send(null);
            });

        </script>
    </body>
</html>
```

Get Example

- Use the PokeAPI to allow someone to find out information about a Pokemon by providing the pokedex number
  - http://pokeapi.co/api/v2/pokemon/NUMBER

```
In [ ]: %%html
<!DOCTYPE>
<html>
    <head>
    </head>
    <body>
        <input type="text" id="dex">
        <button id="find">Who's That Pokemon?!</button>
        <p id="results"></p>
        <script>
        </script>
    </body>
</html>
```

Get Practice

- Write a script to get the appropriate lecture given a number below, and display the conte
  the user
  - The format of the lecture URLS are all `https://www.csee.umbc.edu/courses/undergraduate/433/spring18/ lec=NUM`

In [ ]:

```
%%html
<!DOCTYPE>
<html>
    <head>
    </head>
    <body>
        <input type="text" id="number">
        <button id="find">Get Lecture</button>
        <p id="title"></p>
        <script>
        </script>
    </body>
</html>
```

Making the Request (POST)

- A post request is made very similar to a get request
    - The method passed to open should be "POST"
- The data must be sent as the paramter to `send`
    - Should be formatted like it was being sent with "GET"
        - name1=val1&name2=val2...

In [ ]:
```
%%html
<!DOCTYPE>
<html>
    <head>
    </head>
    <body>
        <input type="text" id="zip">
        <button id="lookup">Lookup</button>
        <p id="city"></p>
        <script>
            document.getElementById("lookup").addEventListener('click',
            function(){
                var request = new XMLHttpRequest();
                var zip = document.getElementById('zip').value;
                request.open('POST', './lookup.php");
                request.onreadystatechange = function(){
                  if(request.readyState == 4){
                      if(request.status == 200){
                          var info = request.responseText.split(":");
                          document.getElementById("city").innerHTML = info[0] +
 "," + info[1]
                      }
                  }
                };
                request.send("zip=" + zip);
            });

        </script>
    </body>
</html>
```

POST Example

- Send a POST request to `https://geocode.xyz` to perform geoparsing
  - Set the values of the `scantext` parameter

In [ ]:

```
%%html
<!DOCTYPE>
<html>
    <head>
    </head>
    <body>
        <input type="text" id="geo">
        <button id="geo_lookup">Lookup</button>
        <p id="coded"></p>
        <script>
            document.getElementById("geo_lookup").addEventListener('click',
            function(){
                var request = new XMLHttpRequest();
                request.setRequestHeader("Content-type", "application/x-www-form-u
rlencoded");
                request.onreadystatechange = function(){

                };

            });

        </script>
    </body>
</html>
```

POST Practice

- Use the geocode.xyz service to locate an IP
    - The needed fields are `locate` which should hold the IP

In [ ]:
```
%%html
<!DOCTYPE>
<html>
    <head>
    </head>
    <body>
        <input type="text" id="geo">
        <button id="geo_lookup">Lookup</button>
        <p id="coded"></p>
        <script>

        </script>
    </body>
</html>
```

JSON

- Sending one piece of text back and forth doesn't require much parsing
  - Larger data needs to be sent as a parsable string
- Originally, XML was used for this purpose, but that is annoying
- JSON stands for JavaScriptObjectNotation
  - Uses {} for objects, and [] for arrays
  - The major difference between this and actual JavaScript code is that keys must be quoted

```
{
"my_key": 10,
"an_array":[1,2,3,4]
}
```

JSON Example

- Write the JSON that would be generated from an object declared as:

```
let apple = new Object();
apple.color = 'red';
apple['name'] = 'gala';
apple.sizes = [1 , 2, 1, .5];
```

JSON Practice

- Write the JSON that would be generated from an object declared as:

```
let orange = new Array();
orange.push(1)
orange.push('2')
orange.push({pi: 3.14, e: 2.71});
```

Converting To and From JSON

- When JSON was first introdcued, parsing was done by hand, or by running the code through `eval`
  - Running the code through `eval` is a very bad idea and a major security risk
- Eventually some standard libriaries started to pop up to handle this task for us
- Now it is part of the JavaScript language, using the JSON object
  - `JSON.parse` takes a JSON string, and returns the corrisponding JS object
  - `JSON.stringify` takes a JS object and returns the corresponding JSON string

```
In [ ]: %%script node
        var today = new Date();
        console.log(JSON.stringify(today))
        console.log(JSON.parse(JSON.stringify(today)))
```

AJAX + JSON

- By combining AJAX and JSON we can make very large complex web applications
- Most standard APIs return JSON, or at least have it as an option

AJAX Saftey

- To prevent malicious code execution, most AJAX calls can only be made to pages on the same server
  - This is known as the same-origin policy
- This can be overridden, but is a bit complex for the purposes of this course
- Never use `eval`, this can execute code from anywhere
  - Parse using `JSON.parse`

A note about the future

- Two new capabilities are beginning to be implemented, but aren't widely supported
- The Fetch API is essentially a replacement for XHR objects
    - XHR was creating a bit organically, the fetch API aims to rebuild it from the ground up with better design
    - Has seperate `Request` and `Response` objects
    - Built around a paradigm known as promises
- Server Sent Events
    - Rather than constanly polling a server, let the server initiate sending events
    - Need to tell the server the page is willing to receive events, after that server initiates