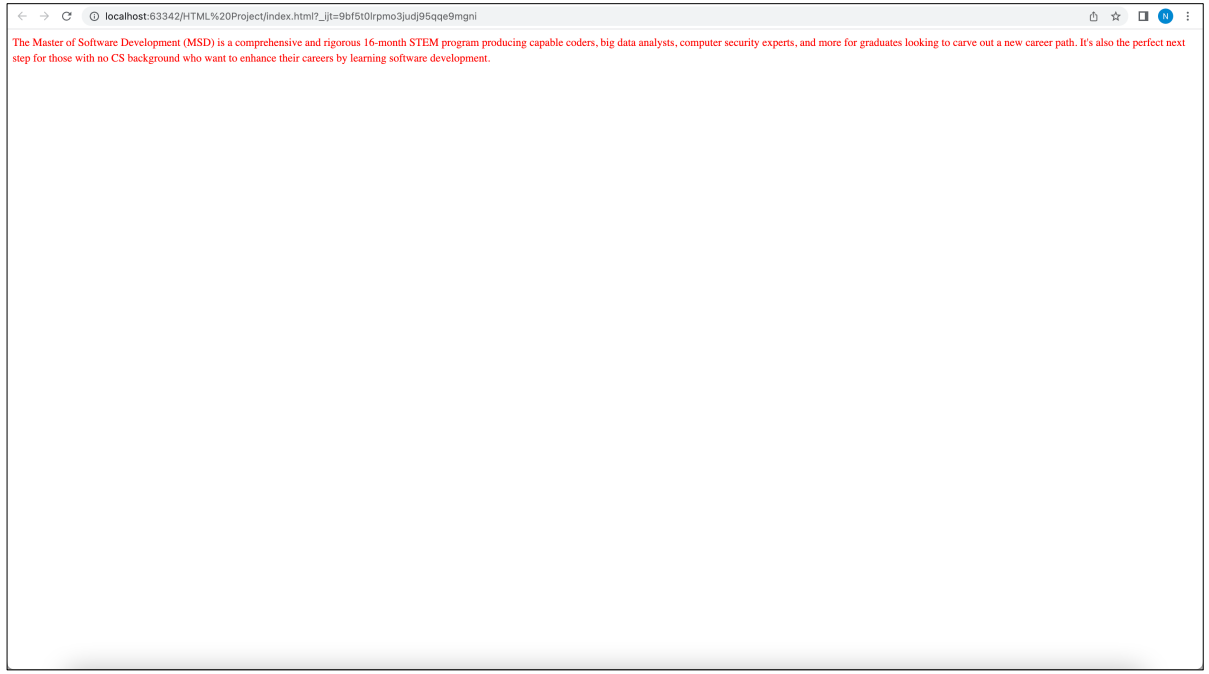
Computer Programming – CS 6011
Lecture 12: Dynamic HTML - AJAX

Fall 2023

Topics

- Event Based Programming (Callbacks)
- **A**synchronous **J**avaScript **A**nd **X**ML (AJAX)

Interacting with a web page



- Small font size in web browser
- Change text color

Interacting with a web page

- Submitting form data with invalid information

Interacting with a web page

- Small font size in web browser
- Changing text color
- Submitting form data with incorrect values
- Approach
  - Using Event based programming / callbacks

Event Based Programming

- Reaction to an “event”
  - Event Based programming
  - Performed at the client side
- In our Synthesizer GUI, what did we use to respond to the user input?
  - Listeners. Also named *callbacks*.
- What events might a webpage want to respond to?
  - Clicks / User Interactions
  - Loading of stuff (images, the page itself, etc)
  - Passage of time

Event Based Programming

- Most elements have properties like *onmousemove()*
- Another approach is using the more modern: addEventListener()
- List of events you can handle:
  - https://developer.mozilla.org/en-US/docs/Web/Events
- JavaScript coding is mainly setting up code to run when an event occurs.

Event Based Programming - Example

- We can perform actions when a user interacts on any element in on our page.
- There are 3 different ways to do this:
  - in HTML:
    <img onclick="myClickFunction();"...
  - in JavaScript:
    myImgElement.onclick = function(event) { ... };
  - in JavaScript:
    myImgElement.addEventListener("click", myHandlerFunction());
- Note, you can only have one onclick function, but you can add multiple event listeners.

Code Practice

- Option 1
  - Save your JavaScript code into a JavaScript file (myCode.js) and link it to the HTML file.

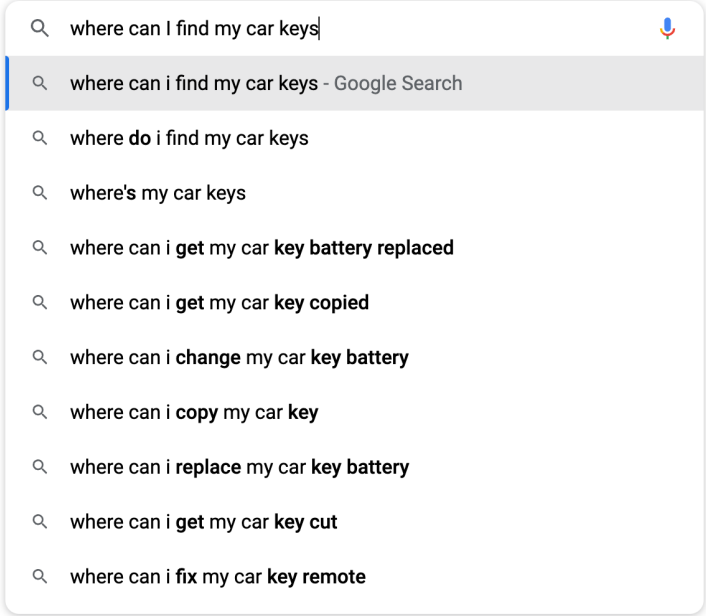
- Option 2
  - Place JavaScript directly in an HTML file (inside a <script> tag), but it is cleaner and easier to maintain when it is in its own file.

Wrapping Up

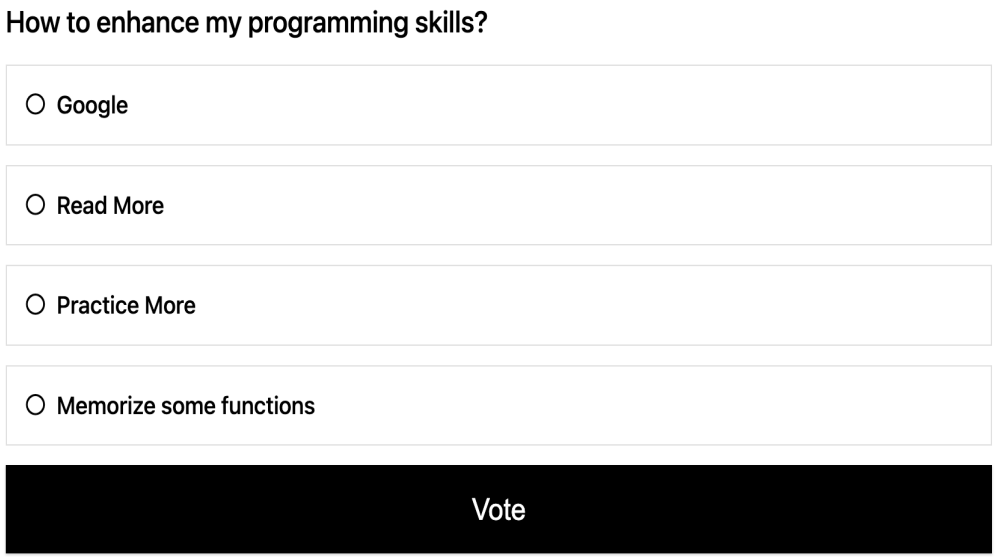
- JS can handle mouse and keyboard events
- JS can handle date and time
  - getTime(), getDate(), ….
    - getTime() returns the number of milliseconds
      - January 1, 1970 00:00:00 (since the ECMA Script)
- JS performs HTML Form Validation.
- No compiler is needed - no JVM is required to run it.
  - Only a browser is needed to JS.

Other forms of interactive webpages

Google



Autofill



Polls / Votes

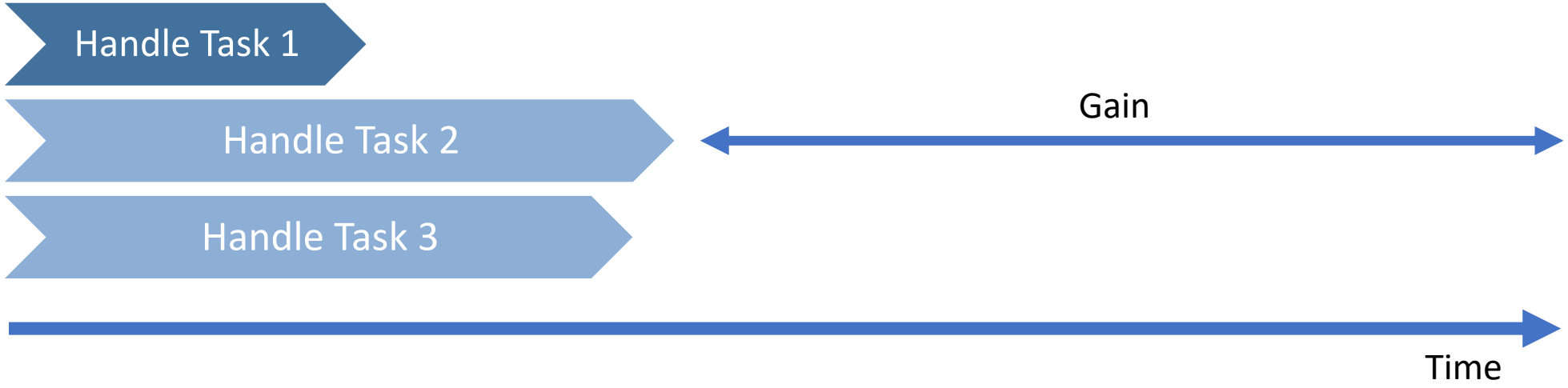
Other forms of interactive webpages

- Example:
  - Google autofill
  - Polls / Votes
  - Loading other chuncks of data blocks after a webpage has finished its initial loading
    - Facebook, Amazon, etc
- Approach
  - **A**synchronous **J**avaScript **A**nd **X**ML (AJAX)

Asynchronous vs Synchronous Programs

- Synchronous

- Asynchronous



Asynchronous vs Synchronous Programs

- Synchronous
  - `step1() // Step 1 executes until it is done...`
  - `step2()`
  - `step3()`
- Asynchronous
  - `start step1() // but don’t wait for it to finish, just continue on to:`
  - `start step2() // but don’t wait for it to finish...`
  - `start step3() // but don’t wait for it to finish...`

How to utilize Asynchronous approach in Web Page loading?

Web Page Loading

- [illegible]?
  - [illegible].
- [illegible] `step1` [illegible]?
  - A[illegible]H[illegible]. [illegible]. D[illegible] [illegible] [illegible].
  - [illegible], [illegible]H[illegible], [illegible]([illegible]) [illegible] [illegible]H[illegible]([illegible], [illegible], [illegible]).
- A[illegible], [illegible]?
- [illegible]?
  - [illegible]
- H[illegible]?
  - C[illegible]/ [illegible]AA[illegible]

AJAX

- Webpages use AJAX to allow then to do this kind of dynamic updating.
- Many different advantages:
  - NO NEED to reload the whole page
  - Faster with fewer full page reloads
  - Reduce server load
    - when loading new content into a page without refreshing the entire browser
  - Javascript based
    - Some parsing happens on the client-side instead of servers having to handle all requests.

AJAX - Implementation

- AJAX uses the XMLHttpRequest object (XHR) to load data (ask for data) from the server without reloading the whole page.
- The XHR lifecycle looks like:

```
let xhr = new XMLHttpRequest();  // Create the object we will use to do the work for use.
```

- What information needs to be sent in an (any) HTTP request?
  - `xhr.open( "GET", url );` // C[illegible]
- What type of events might we expect after we've sent the request?
  - Either the request succeed or it failed.

AJAX - Implementation

- What do we use to let us know when the requested data becomes available?
  - We need a callback to handle the result.

```
xhr.addEventListener( "load", funcToCallWhenLoaded );
xhr.addEventListener( "error", funcToCallWhenTheresAnError );
```

- Note, we have not actually sent the request yet... so once the above things are set up:

```
xhr.send();  // Actually send the request to the server.
```

- In the callback functions (eg: `funcToCallWhenLoaded`) the *`this`* variable refers to the response object which has useful methods / fields like *`responseText`*.

Other Useful Methods

- The *onload* function is run when the entire HTML page has been loaded (and only run once). This will allow you to run your JavaScript code after all HTML/CSS have finished loading.

```
window.onload = function() { ... }
```

- Run *someFunction()* after (at least) a *delay* of milliseconds.

```
window.setTimeout( someFunction, delay );
```

- Run *someFunction()* every delay milliseconds.

```
window.setInterval( someFunction, delay );
```

`window.requestAnimationFrame`(callback); // Browser will decide when to call *callback*.

- Events will "bubble up" through parent objects...

```
event.stopPropagation();  // If your function has handled the event.
```