Lecture 9: Frontend JavaScript and Client-Side Form Validation

CS-546 – WEB PROGRAMMING

Client Side JavaScript

Differences between Node.js and Browsers

There is a global scope; variable collisions are a very large issue.

You include JavaScript files through HTML.

There are no native modules; rather, currently, including required code is done with module libraries or by including script files.

You do not have access to any of the file system aspects of node.

The syntax is often not up to date in most browsers

- https://kangax.github.io/compat-table/es6/

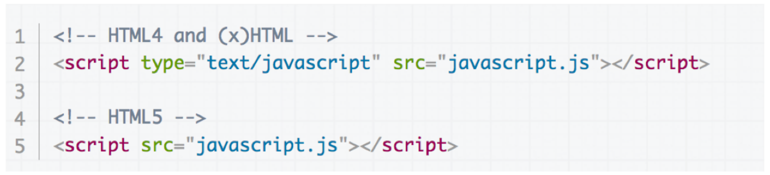
Different browsers behave differently.

Running JavaScript in the browser

In the browser, there are 2 forms of running JavaScript on a web page:

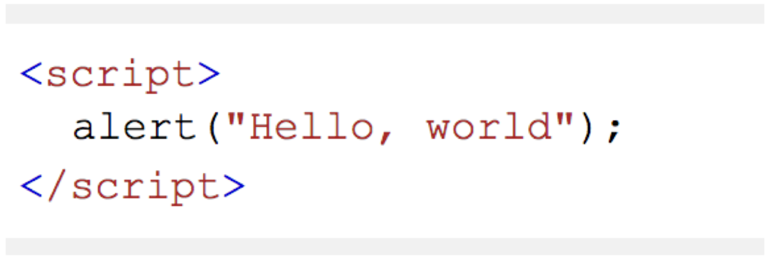
- Including a script element, with an attribute of *src* specifying a link to the JavaScript file, and no content inside the element (top).
- Including a script element, with the content of the JavaScript you wish to execute.

You can have as many script elements as you need, and use as many script files.



```html
<!-- HTML4 and (x)HTML -->
<script type="text/javascript" src="javascript.js"></script>

<!-- HTML5 -->
<script src="javascript.js"></script>
```



```html
<script>
  alert("Hello, world");
</script>
```

JavaScript “libraries”

Browsers, unfortunately, do not have the ability to just use scripts directly from NPM and include external code. Instead of having access to packages and modules, you must instead download “libraries” and include them on your page.

- A library is a pre-written JavaScript file that is released to make developing your own application easier.

Some common libraries are:

- jQuery
- React
- Bootstrap
- AngularJS
- D3
- Underscore
- Vue

How JavaScript is run in the browser

While a web page is loaded, whenever it sees a script tag, it will pause execution and interpret the contents of the script element.

- If the script contains a reference to a file, it will start downloading the file (and download other script files at once) and interpret the contents
- These files will be interpreted in the order of their script tags placement, even if they finish their downloads out of order.

After each script is interpreted, it is executed

- Interpretation is the process of the JavaScript language being parsed so the browser can execute it
- Execution is the part where the script is *run* and the commands are performed

Manipulating the Page

One of the primary reasons for JavaScript to execute in the browser is to interact with the web page that users see. We do this, through manipulating the DOM.

- Document
- Object
- Model

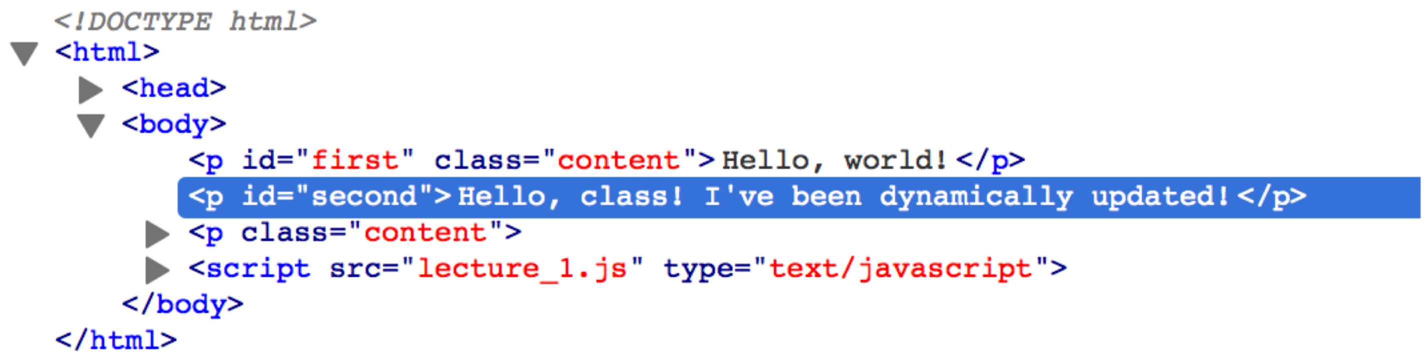
JavaScript, when run in the browser, is able to access the document (web page) and manipulate it in a number of ways through the DOM API.

The DOM and JavaScript

What is the DOM?

The **DOM** (Document Object Model) is how the programmer / browser interacts with the HTML document.

- The DOM has an API to access and manipulate the document. Each element is represented in the DOM.
- You can access the DOM with JavaScript.
- You can think of the DOM as the document-in-memory, and you can manipulate many aspects of it. This leads to programmers being able to create extremely powerful applications.



```
  <!DOCTYPE html>
▼ <html>
   ▶ <head>
   ▼ <body>
        <p id="first" class="content">Hello, world!</p>
        <p id="second">Hello, class! I've been dynamically updated!</p>
      ▶ <p class="content">
      ▶ <script src="lecture_1.js" type="text/javascript">
     </body>
  </html>
```

Where does it fit in?

The DOM is a programming interface for HTML.

- The rendering engine takes in the HTML document
- The rendering engine parse the HTML into the DOM tree
- The rendering engine takes the DOM tree and creates the render tree
- The rendering engine paints the render tree

You can then manipulate the web page through the DOM, which is accessible via JavaScript.

- You will target DOM elements
- You will manipulate them
- The rendering engine will recreate that portion of the rendering tree.
- The rendering engine will repaint.

Why is this important?

Being able to manipulate your web page in real time through the DOM API allows you to do many, many things :

- Enhance the functionality of your web page
- Update data on your web page to reflect the user’s actions
- Turn web pages into robust web applications

Modern web applications constantly mutate the DOM

A practical example of using the DOM

You are creating a page with a comment box and list of comments. You’ve created a very controversial post that will definitely spawn a large amount of comments.

You would manipulate the DOM in three ways:

- Every 5 seconds you would poll the server in order to check for new comments; if there are new comments, you would use the DOM to create and insert new elements with the comment info that would then appear on screen
- When the user submits a comment, you will use the DOM to:
  - Retrieve their new comment information, submit it to the server, and add their new comment to the page
  - Reset the comment box form to its default state

DOM Events

There are many events that you can listen for, representing interaction between the user and the page or the page and other resources. Some common events to listen to are:

- Users hovering over an element
- Images loading or failing to load
- Scrolling to occur
- Forms to be interacted with
- Keys to be pressed
- The DOM to be modified.

Accessing the DOM Via JavaScript

Accessing the DOM

Primarily, we access DOM elements via the *document* global variable, which has many methods to begin searching for DOM elements

- document.getElementById(“content”);
- document.getElementsByTagName(“li”);
- document.getElementsByTagName(“div”);
- document.querySelector("div");
- document.querySelectorAll("div > a");

When we have an initial reference to a DOM node, we can also traverse its children to access other DOM nodes.

You can store these results in a variable! It is often beneficial to do so, to avoid many DOM traversals.

https://developer.mozilla.org/en-US/docs/Web/API/Document

What can we do with a DOM Element?

After querying a DOM element, you will be able to access *tons* of data based on the type of element.

- All elements will allow you to get/set their innerHTML
- All elements will allow you to get/set their children
- All elements will allow you to get/set attributes, classes, width, height, etc.
- Inputs will allow you to get/set their values

You will also be able to hook into events related to some element types

- Inputs have events for when they change
- Images will allow you to watch for the image loading / failing
- Forms have events for submissions

https://developer.mozilla.org/en-US/docs/Web/API/Element

https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement

Creating a new element (input)

Using JavaScript, you can create a new element with ease.

The *document.createElement("input")* method will create and return a new DOM element, but it will not yet be attached to the DOM tree (and therefore, the render tree; it will not show up on the screen).

By storing that result, you can then manipulate it using methods such as setting the input type or giving it an initial value.

What can we do to manipulate the DOM?

It is easy to add elements to the DOM by targeting parent elements and appending the newly created elements into those parents.

You can move the elements from parent to parent by using the *parent.appendChild(newNode)* method, or the *parentNode.insertBefore(newNode, oldNode)* methods.

You may remove elements using the *parent.removeChild(childNode)* method.

In this way, you can manipulate the content of elements. You can also directly set the *innerHTML* property of a method; this, however, forces a complete rebuild of the entire node and is an expensive repainting operation.

DOM Events

Using the DOM API, we can wait for events to occur and execute callbacks after the event has triggered.

In order to listen for an event, first you must:

- Target a DOM element
- Call the *addEventListener* method on that, and supply the the following as parameters:
  - Event name
  - Callback function

The callback function will receive an object representing the event as its first parameter.

Event Bubbling

Some events, such as click, will bubble up to parent elements. For example, if you had two *div* elements, each with an event listener listening for the click event, and clicked the inner *div*, which event should the DOM trigger?

By default, it will trigger the inner div first, then the outer div.

You can prevent the event from bubbling past the first *div* by using the *stopPropogation* method on the event object, which will be passed into the event listener via the first parameter.

Other Useful document properties

document.title allows you to get/set the Document’s Title; this is originally set to the content of the *title* element inside your document’s *head* element.

document.cookies allows you to get/set cookies that are currently being shared between your browser and the entire server at the current domain.

https://developer.mozilla.org/en-US/docs/Web/API/Document

Client-Side Form Validation

What is Client-Side Form Validation?

Client side form validation is the process of checking the user's input through the browser so that they can adjust their input accordingly before it is submitted to the server.

This allows your users to adjust their input while its still fresh in their minds and not have to re-input it each time.

The algorithm is simple:

- Target your element in JavaScript
- Capture the form submission event
- Prevent the default form submission from continuing
- Check if all inputs are correct (correct range, required, etc)
  - If yes, allow the form to submit
- If there's a bad input, then show an error message describing to the user how to correct it.

Targeting your element and attaching an event listener

jQuery was built with the purpose of making DOM tasks easier; therefore, you can use jQuery to target your particular form and attach an event listener to the ’submit’ event.

The event listener is a callback function that runs each time the form is submitted. You will use it to check through each input to make sure they are all valid.

You’ll want to store references to the form and each of it’s inputs so that you do not re-query each input each time the form is submitted; this is a relatively slow and expensive operation.

Checking each input

Some common things to check for:

- Check if this input is required and if so, check if it has a value
- Check if the input is within an acceptable range
- Check if the input is within some other criteria (ie: is a password that has certain properties)

If an input is bad, you will prevent the form from submitting and add some sort of error message.

Showing error messages

If the form should not be submitted, you will prevent the default event and show the user some form of error message. You can also do other helpful things such as:

- Highlight the inputs that need to be corrected
- Focus the user’s cursor into a bad input
- Offer a suggested correction

Example

You can find an example at:

- http://localhost:3000/calculator/static

Things to note:

- If you were submitting to the server, if the input was successful you would simply allow the form to complete (ie: never prevent the form default event from occurring).
- You can and should store references to events when possible