**Programming with JavaScript Notes**

**Week 3**

**JavaScript in the Browser**

JavaScript Interactivity

JavaScript’s initial purpose was to provide a simple explanation of web page manipulation and some examples of it.

Its initial purpose was to provide interactivity in the browser.

In other words, it was the “set of controls” that would allow web developers to control the behavior of the webpages and even the browsers that these webpages worked on.

This is still the case today.

Initially in the late 1990s, there was plain JS that had to be tweaked to suit individual browsers.

By the mid-2000s, the jQuery library came out, with the idea of writing less code, but doing more with it. It “leveled the playing field” as it allowed developers to use a single codebase for various browsers.

The trend continued and many other frameworks such as React, Vue, Angular, D3, and more came along.

With npm and Node.js, the JS ecosystem is not slowing down.

JS is still the king when it comes to making websites interactive.

Although CSS has developed significantly over the years, it is still JS that allows users to:

* Get their geolocation
* Interact with maps
* Play games in the browser
* Handle all kinds of user-triggered events, regardless of the device
* Verify form input before sending it to the backend of a webapp
* And more!

There are so so many ways JS allows you to build rich, interactive experiences on the web.

Exercise: Web page content update

We’re going to demonstrate how to manipulate information displayed based on user input.

To capture the user input, you can use the built-in user prompt() method:

let answer = prompt('What is your name?');

Once you have the user-provided input inside the answer variable, you can manipulate it any way you need.

You can output the typed-in information on the screen, as an <h11> HTML element:

let answer = prompt('What is your name?');

if (typeof(answer) === 'string') {

var h1 = document.createElement('h1')

h1.innerText = answer;

document.body.innerText = '';

document.body.appendChild(h1);

}

You’re essentially doing the same thing you did before, only this time you’re also dynamically adding the input element, and you’re setting its HTML type attribute to text. That way, when you start typing into it, the letters will be showing in the h1 element above.

When you run the above code into a live website, it will add a h1 element with the text “Type into the input to make this text change”, and an empty input form field under it.

Go to example.com to try it.

Another opiniated thing we did in the code above is: setting my variables using the var keyword

Although it’s better to use either let or const, you’re just running a quick experiment on a live website, and you want to use the most lenient variable keyword, the one which will not complain about you having already set the h1 and input variables.

All in all, working on a complete project with a modern JS tooling setup, use let or const, for trivial code or tests, use var.

The next thing we need to do is, set up an event listener. The event you’re listening for is th change event. In this case, the change event will fire after you’ve typed into the input and pressed the ENTER key.

Here’s the updated code:

var h1 = document.createElement('h1')

h1.innerText = "Type into the input to make this text change"

var input = document.createElement('input')

input.setAttribute('type', 'text')

document.body.innerText = '';

document.body.appendChild(h1);

document.body.appendChild(input);

input.addEventListener('change', function() {

console.log(input.value)

})

This time, when you run the above code on the said example.com website, subsequently typing some text into the input field and pressing the enter key, you’ll get the value of the typed-in text logged to the console.

Now, you need to update the text content of the h1 element with the value you got from the input field.

Here’s the complete and updated code:

var h1 = document.createElement('h1')

h1.innerText = "Type into the input to make this text change"

var input = document.createElement('input')

input.setAttribute('type', 'text')

document.body.innerText = '';

document.body.appendChild(h1);

document.body.appendChild(input);

input.addEventListener('change', function() {

h1.innerText = input.value

})

After this update, whatever you type into the input, after. Pressing the ENTER key, will also be shown as the text inside the h1 element.

Note that combining DOM manipulation and event handling allows for some remarkable interactive websites.

Exercise: Capture Data

Description

The aim of this exercise it to access the content of an element, specifically to use a button click to replace text.

Task 1: the example.com website

Open the example.com website. Open the Developer Tools and focus on the Console Tab.

Task 2: Get h1 into a variable.

Use the document.querySelector() method to query the h1 element on the page and assign it to the variable named h1.

Task 3: Code an array.

Declare a new variable, name it arr, and save the following array into it:

[

    'Example Domain',

    'First Click',

    'Second Click',

    'Third Click'

]

Task 4: Write a click-handling function.

Write a new function declaration named handleClicks. It should not accept any parameters.

Inside of it, code a switch statement, and pass a single parameter to it, h1.innerText.

The body of the switch element should have a total of four cases (fourth being default)

The first case should start with case arr[0]:. It should set the h1.innerText to arr[1]. In other words, it should assign the value of arr[1] to the h1.innerText property. The next line should have only the break keyword.

The second case should start with the case arr[1]:. It should set the h1.innerText property to arr[2]. In other words, it should assign the value of arr[2] to the h1.innerText property. The next line should have only the break keyword.

The third case should start with the case arr[2]:. It should set the h1.innerText to arr[3]. In other words, it should assign the value of arr[3] to the h1.innerText property. The next line should have only the break keyword.

The default case should set the h1.innerText property to arr[0].

Task 5: Add an event listener

You’ve created an h1 variable in task 2. Now, use that variable to run the addEventListener() method on it. Pass two arguments to the addEventListener() method: ‘click’ and ‘handleClicks’.

var h1 = document.querySelector('h1')

var arr = [

'Example Domain',

'First Click',

'Second Click',

'Third Click'

]

function handleClicks() {

switch(h1.innerText) {

case arr[0]:

h1.innerText = arr[1]

break

case arr[1]:

h1.innerText = arr[2]

break

case arr[2]:

h1.innerText = arr[3]

break

default:

h1.innerText = arr[0]

}

}

h1.addEventListener('click', handleClicks)

Moving Data Around on the Web

Around 2001, Douglas Crockford came up with a data interchange format based on JS objects. The name given to this format was JSON, which is JavaScript Object Notation.

Before JSON, the most common data interchange file was XML (Extensible Markup Language). Due to XML’s syntax, it required more characters to describe the data that was sent. Also, since it was a specified stand-alone language, it wasn’t as easily inter-operable with JS.

Thus, the two reasons JSON format is becoming the most dominant interchange format today:

* It’s lightweight, with syntax very similar to a “a stringified JavaScript object”.
* It’s easier to handle in JS code, since after all, it is just JS.

All JSON code is JavaScript but not all JavaScript code is JSON.

Besides being a data interchange format, JSON is also a file format.

It’s not uncommon to access some third-party data from a third-party website into our own code in the form of a JSON file.

For example:

If you had a website with the data on stock price movements, you might want to get the data of the current stock prices from a data vendor. They might offer their data services by giving you access to the file named, say stockPrices.json, that you could access from their servers.

Once the stringified json data is downloaded and put into your own code, you could then convert that data to plain JS object.

That would mean that you could use your web application’s code to “dig into” third-party-data-converted-to-a-JavaScript-object, so as to obtain specific information based on a given set of criteria.

For example, if the stringified JSON data was converted to an object that had the following structure:

const currencyInfo = {

[

USD: {

// ...

},

GBP: {

// ...

},

EUR: {

// ...

}

]

}

You could then access only the data from the USD property, if that’s what was needed by your app at a given point in time.

Hopefully you can understand why and how you might want to use JSON in your own code.

It’s all about:

* Getting stringified data from a server,
* Converting (parsing) that data into JS objects into your own code,
* Working with the converted data in your own code,
* And stringifying the result into JSON, so that this data is then ready to be sent back to a server after your code has processed it locally.

JSON is just a string – but there are rules that it must follow.

JSON is a string, but it must be a properly formatted string. It must adhere to some rules.

If a JSON string is not properly formatted, JS would not be able to parse it into a JS object.

JSON can work with some primitives and complex data types

Only a subset of values in JS can be properly stringified to JSON and parsed from a JS object into a JSON string.

These values include:

* Primitive values: strings, numbers, boolean, null,
* Complex values: objects and arrays (no functions)
* Objects that have double-quoted strings for all keys
* Properties are comma-delimited both in JSON object and in JSON arrays, just like in regular JS code
* String properties must be surrounded in double quotes, For example:  
  “fruits”,  
  “vegetables”
* Number properties are represented using the regular JS number syntax:  
  5,  
  10,  
  1.2
* Boolean properties are represented using the regular JS Boolean syntax:  
  true  
  false
* Null as a property is the same as in JS:  
  null

You can use object literals, as long as you follow the above rules.

What happens if you try to stringify a data type which is not accepted in JSON syntax?

If you try to stringify something like a function, the operation will silently fail.

If you try to unstringify some other data types, such as BigInt number, say 123n, you’d get the following error: Uncaught TypeError: Do not know how to serialize a BigInt.

Some examples of JSON srtrings

Here’s an example of a stringified JSON object, with a single key-value pair:

`{“color”:”red”}`

Here’s a bit more complex JSON object:

`{“color”:”red”, “nestedObject”: { “color”: “blue” } }`

The above JSON object encodes two properties:

* “color”:”red”
* “nestedObject”: { “color”: “blue” }

It’s also possible to have a JSON string encoding just an array:

`[“one”, “two”, “three”]`

The above string encodes an array holding three items, three values of the string data type.

Obviously, just like objects, arrays can nest other simple or complex data structures.

For example:

`[{ “color”: “blue” }, {“color”: “red”}]`

In the above example the JSON string encodes an array which holds two objects, where each object consists of a single key-value pair, where both values are strings.

You can use these two methods to convert a JS object to and from a JSON string:

* JSON.parse
* JSON.stringify

Additional Resources

Week3test.js

MDN: Modules

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules>

Nodejs.org Official Docs on Common JS

<https://nodejs.org/api/modules.html#modules-commonjs-modules>

MDN: DOM  
<https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model>

MDN: Document.querySelector

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

MDN: Event

<https://developer.mozilla.org/en-US/docs/Web/API/Event>

MDN: EventTarget.addEventListener

<https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/addEventListener>

MDN: Working with JSON

<https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON>