Week 5 Lesson 9 Readings

***Chapter 14: HTML5 APIs***

HTML5: The HTML5 specification is separated into modules that allow different features to be developed at different paces then implemented without having to wait for other features to be completed. It also means that when a previously unforeseen development occurs, a new module can be created to cater for it. Modules can be at different stages of maturity, from ideas to full implementation.

Data = attribute is a way of embedding data in a we page using custom attributes that are ignored by the browser. They are private which means they are no meant to be used by an external service. They are dedicated to use by a JavaScript.

data-powers = 'flight superSpeed'

data-rating = '5'

data-dropdown

data-user = 'DAZ'

data-max-length = '32'

APIs: The HTML5 specification contains several APIs that help to gain access to hardware, such as cameras, batteries, geolocation, and the graphics card. Hardware evolves quickly, and APIs are frequently introduced to give developers access, and control new features that appear in the latest devices.

WEB STORAGE APIs:

* Information stored is *not* shared with the server on every request.
* Information is available in multiple windows of the browser (but only if the domain is the same).
* Storage capacity limit is much larger than the 4KB limit for cookies ( There is no actual limit in the specification, but most browsers have a limit set at 5GB per domain.).
* Any data stored does not automatically expire as it does with cookies. This potentially makes cookies a better choice for something like showing a popup once a day.

localStorage.setItem('name', 'Walter White');

localStorage.getItem('name');

<< "Walter White"

localStorage.name = 'Heisenberg';

console.log(localStorage.name);

<< "Heisenberg";

localStorage.removeItem('name');

delete localStorage.name;

localStorage.clear();

* key tells us the key of the item that changed
* newValue tells us the new value to which it has been changed
* oldValue tells us the previous value before it was changed
* storageArea tells us if it is stored in local or session storage.

localStorage.setItem('superman', JSON.stringify(hero);

superman = JSON.parse(localStorage.getItem('superman'));

GEOLOCATION:

The Geolocation API is used to obtain the geographical position of the device. This means it can be used to find the user’s exact location, then link to nearby places or measure the speed at which the user is moving. This information can then be used to filter data based on the user's location or speed and direction of travel. An example of this might be a search function that returns results based on your location. Because of privacy concerns, permission to use this has to be granted by the user first.

navigator.geolocation.getCurrentPosition(youAreHere);

function youAreHere(position) {

console.log(`Latitude: ${position.coords.latitude}, Longitude: ${position.coords.longitude}`);

}

The position object has several other properties that can be used to find out information about the location and movement of the device:

* position.speed property returns the ground speed of the device in meters per second.
* position.altitude property returns an estimate of the device’s altitude in meters above the [WGS84](http://en.wikipedia.org/wiki/World_Geodetic_System) ellipsoid, which is a standard measurement for the center of the Earth.
* position.heading property returns the direction the device is moving in. This is measured as a bearing in degrees, clockwise from North.
* position.timestamp property returns the time that the position information was recorded.

The position object also has properties that calculate the accuracy of the measurements. These can be useful, as sometimes you only need to know the town or city users are in, while at other times you may need their exact position. position.accuracy property returns the accuracy of the latitude and longitude properties in meters. The lower the returned value, the more accurate the measurements are, as is the case for the position.altitudeAccuracy property, which returns the accuracy of the altitude property in meters.

const id = navigator.geolocation.watchPosition(youAreHere);

navigator.geolocation.clearWatch(id);

WEB WORKERS:

We saw in earlier chapters that JavaScript is a single-threaded language, meaning that only one process can run at one time. Web workers allow processes to be run in the background, adding support for concurrency in JavaScript. The idea is that any processes that could take a long time are carried out in the background, so a website will continue to function without fear of the dreaded 'script has become unresponsive' message that occurs when a script runs for too long

***const worker = new Worker('task.js');***

***worker.postMessage('Hello');***

***self.postMessage('Finished');***

***worker.addEventListener('message', (event) => {***

***console.log(event.data);***

***}, false);***

***worker.terminate();***

***self.close();***

***Chapter 12: Canvas, SVG, and Drag and Drop***

The Canvas API is supported in:

* Chrome 4+
* Firefox 2+
* Opera 9.6+
* Safari 3.1+
* iOS 3.2+
* Internet Explorer 9.0+
* Android 3.0+

CREATING A CANVAS ELEMENT

<canvas>

Sorry! Your browser doesn’t support Canvas.

</canvas>

The text in between the canvas tags will only be shown if the canvas element is not supported by the visitor’s browser.

<canvas id="myCanvas" class="myCanvas">

Sorry! Your browser doesn’t support Canvas.

</canvas>

Since drawing on the canvas is done using JavaScript, we’ll need a way to grab the element from the DOM.

<canvas id="myCanvas" class="myCanvas">

Sorry! Your browser doesn’t support Canvas.

</canvas>

You may be asking yourself, why not set the width and height via CSS? It’s because the width and height attributes determine how large the canvas’s coordinate system is. If we don’t specify width and height, the canvas element will default to a width of 300 and a height of 150. If we set the width and height for a canvas only in CSS, the canvas element will be 300 by 150, and the CSS properties will simply determine how large the box is that displays the image.

<canvas id="myCanvas" class="myCanvas" width="200" height="200">

Sorry! Your browser doesn’t support Canvas.

</canvas>

.myCanvas {

border: dotted 2px black;

}