Introduction to AJAX

Our Goals

- Understand JavaScript callbacks
- What is AJAX?
- What can it do?

First, callbacks

Why do we need them?

```
var setUpRequest = prepareData();

var response = sendRequestSynchronously();

display( response );
```

Synchronous vs. Asynchronous

- Synchronous
  - Runs line by line
- Asynchronous
  - Can run in almost any order

How to we deal with asynchronicity?

Lots of ways!

The most common approach is using ***callbacks***

So, what are callbacks?

- Any function that is called as a response to something else
  - e.g. Event handlers

This should work...

```
var doFirst = function () {
    console.log( "Do first" );
};

var doSecond = function () {
    console.log( "Do second" );
};

doFirst();
doSecond();
```

But this doesn't

```
var doFirst = function () {
    setTimeout( function(){
        console.log("The do_first function was called"
    }, 1000 );
};

var doSecond = function () {
    console.log("The do_second function was called");
};

doFirst();
doSecond();
```

Functions can be passed around

```
var doFirst = function ( cb ) {
    console.log( "This runs" );
    cb();
};

doFirst(function () {
    console.log( "Then this does" );
});
```

Functions can be passed around

```
var doFirstAndSecond = function ( cb ) {
    console.log( "Do First." );
    cb();
}

var doSecond = function () {
    console.log( "Do Second." );
}


doFirstAndSecond( doSecond );
```

To work with timing

```
var doFirstAndSecond = function ( cb ) {
    setTimeout(function () {
        console.log( "Do First." );
        cb();
    }, 1000);
}

var doSecond = function () {
    console.log( "Do Second." );
}

doFirstAndSecond( doSecond );
```

This is how things work

- Sometimes we call callbacks
- Other times, we let the browser call them for us

XMLHttpRequests

XMLHttpRequests

e**X**tensible **M**arkup **L**anguage **H**yper **T**ext **T**ransfer **P**rotocol **R**equests

- Created by Microsoft
- Standardised by the W3C

It can make pages ***live***

Where is it used?

- In feeds (such as twitter)
- Chat rooms and messaging apps
- For voting and rating
- Autocompletion
- Form submission and validation

Why is it good?

- It makes your pages live
- It is much faster
- It tends to give a greater user experience
- It is fancy
- It is popular in the workplace
- It is essential for using frontend frameworks: loading data via APIs

The Approach

- Create an instance of the XMLHttpRequest object
- Open a URL with a particular HTTP method
- Send the request
- Wait to the request comes back
- Deal with the response

The browser tells us this stuff!

Using states

- **0** - Request has been initialised but not sent
- **1** - Server connection established
- **2** - Request has been received by the server
- **3** - Processing response in the browser
- **4** - Response is ready to be interacted with

How does that look?

```
var request = new XMLHttpRequest();

request.open( "GET", "http://www.somewebsite.com" );

request.send();

request.onreadystatechange = function () {
    console.log( "Ready State: ", request.readystate );
}
```

How does that look?

```
request.onreadystatechange = function () {
    if ( request.readystate !== 4 ) {
        return;
    }


    var received_data = request.responseText;
}
```

For more information

- Using XMLHttpRequests
- Synchronous and Asynchronous Requests
- XMLHttpRequest on MDN

Worth a read

- JavaScript is Sexy: Understand JavaScript Callback Functions and Use Them
- SitePoint: Demystifying JavaScript Closures, Callbacks and IIFEs
- Amy Simmons wrote this