Asynchronous JavaScript

Cooking Analogy

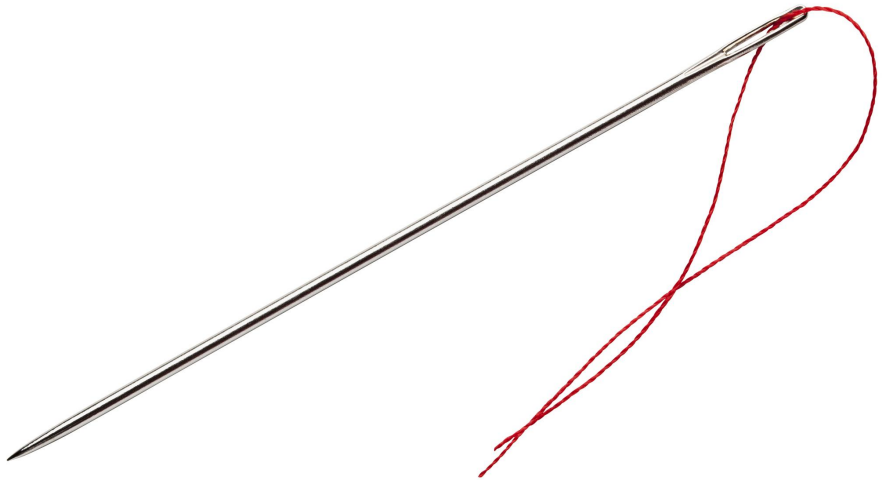
Suppose you are cooking breakfast.

You put a pan on the stove to heat up, and then you put bread in the toaster.

JavaScript is Single Threaded

Back to our cooking analogy, there is only one cook in the kitchen. The cook only has one pair of hands and can only do one thing at a time.

The JavaScript engine is the same way.



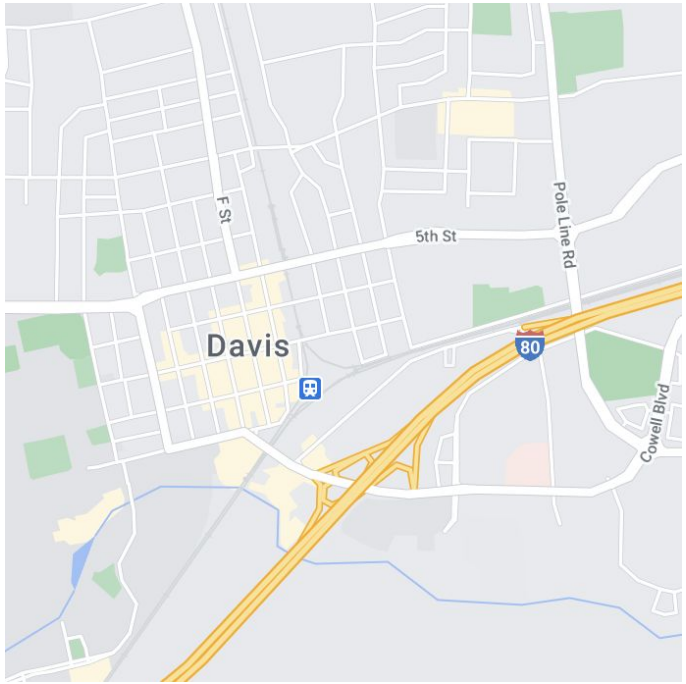
Ajax

The term Ajax was first used in 2005 to describe a group of technologies that come together to make doing tasks asynchronously with JavaScript possible.

My dog, Ajax :-)

Early Applications

Early on, Ajax relied heavily on the XMLHttpRequest web API to asynchronously get data from the server, or send data to a server.



Setting Up Examples

The hallmark of an asynchronous task, is that JavaScript doesn't know how long that task is going to take to complete.

Remote Resource

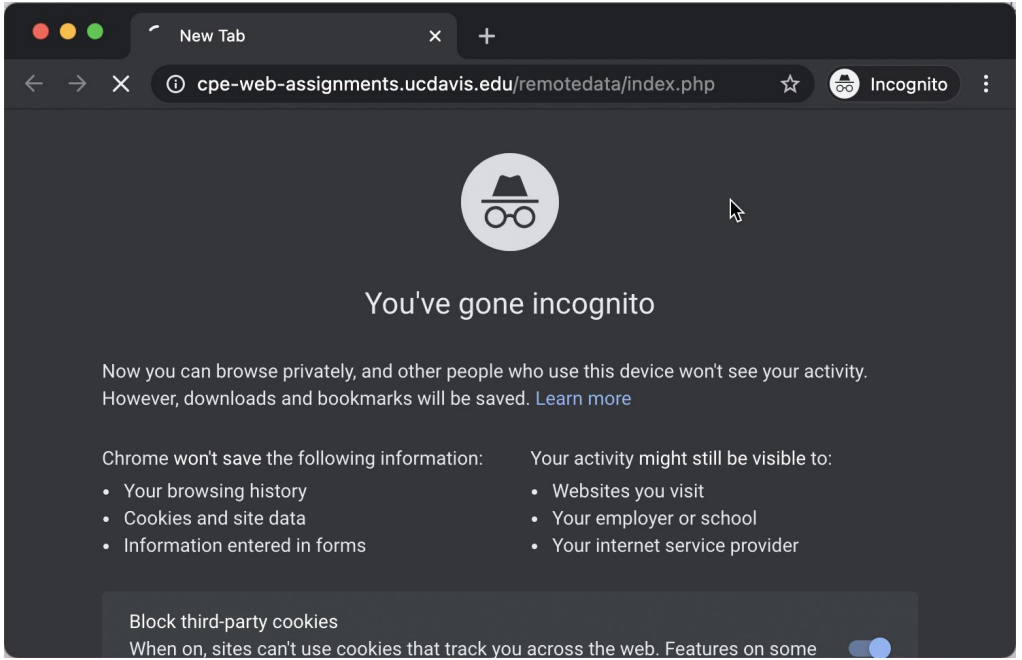
This simple file waits between 5 and 30 seconds before returning some content.

```php
<?php

date_default_timezone_set("America/Los_Angeles");

$time_requested = date('l jS \of F Y h:i:s A');

$sleep_time = rand(5, 30);

sleep($sleep_time);

$time_processed = date('l jS \of F Y h:i:s A');

echo "<div class=\"serverdata\">
<h2>$sleep_time Seconds</h2>
<p>Data requested on $time_requested</p>
<p>Data processed on $time_processed</p>
</div>";

?>
```

Getting the Div

You can go to the page directly by going to this address in your browser:

https://cpe-web-assignments.ucdavis.edu/remotedata/index.php



Example 1: XMLHttpRequest – getData();

```
document.getElementById('loaddata').addEventListener('click', getData);
// AJAX function for getting data from the file...
function getData() {
    const req = new XMLHttpRequest();
    const url = "https://cpe-web-assignments.ucdavis.edu/remotedata/index.php";

    req.onreadystatechange = function () {
        useResponse(req);
    };

    req.open("GET", url, true);
    req.send(null);
}
```

This is a very simple task for XMLHttpRequest. It has two parts, get the data and using the requested data.

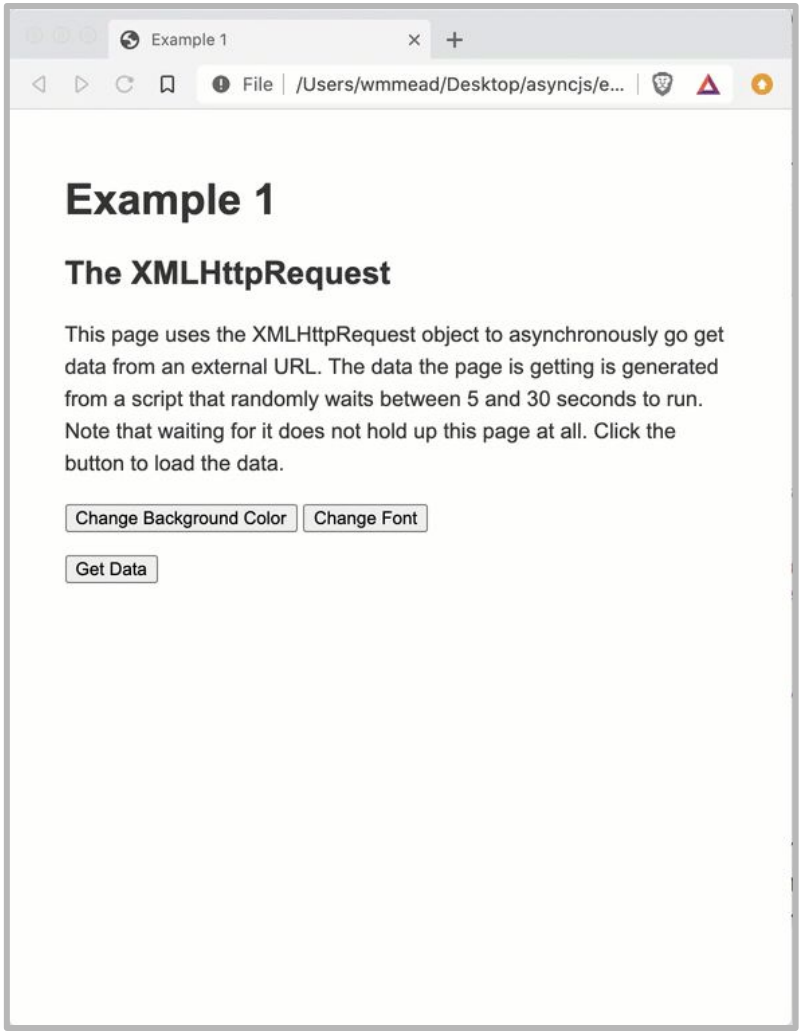
Example 1: XMLHttpRequest – useResponse()

```
function useResponse(req) {
    if (req.readyState == 4) {
        if (req.status == 200) {
            document.getElementById("remotedata").innerHTML = req.responseText;
        }
    }
    else {
        document.getElementById('remotedata').innerHTML = '<img src="images/spinner.gif">';
    }
}
```

This function will run, asynchronously, until the readystate is at 4, which means it's done, and the status is 200, which means it got the data.

Asynchronous!

Notice, if you test the example 1 file, it is working asynchronously. You can click the change color and change font buttons and those scripts will run, while JavaScript is waiting for the data to come back.



jQuery to the Rescue

Working with the XMLHttpRequest API directly was always cumbersome. It became popular in the early 2010s to use a library like jQuery to smooth out the experience.

```
$('#loaddata').click(function () {

    $('#remotedata').html('<img src="images/spinner.gif">');

    $.ajax({
        url: "https://cpe-web-assignments.ucdavis.edu/remotedata/index.php",
        cache: false
    }).done(function (data) {
        $("#remotedata").html(data);
    });

});
```

The Fetch API and Promises

The ES6, or ES2015 specification included promises, which work with the newer, more robust Fetch API.

The Fetch API was designed to replace the old cumbersome XMLHttpRequest API. When you fetch a resource, it always returns a promise.

Fetch and Promise Example

```
document.getElementById('loaddata').addEventListener('click', getData);
// Asynchronously getting data from the file...
function getData() {
    document.getElementById('remotedata').innerHTML = '<img src="images/spinner.gif">';
    const fetchPromise = fetch('https://cpe-web-assignments.ucdavis.edu/remotedata/index.php');
    //console.log(fetchPromise);
    fetchPromise.then(function (response) {
        //console.log(response.text());
        response.text().then(function (text) {

            document.getElementById('remotedata').innerHTML = text;
        });
    });
}
```

Here is our simple example, working the exact same way with the fetch API and returning promises.

Chaining the promise .then() statements remains a bit cumbersome, but it was a step up from the old XMLHttpRequest API.

Async and Await

The async and await keywords were added to JavaScript with ES2017 as a way of making it easier to work with the fetch API and promises.

```
async function getData() {
    document.getElementById('remotedata').innerHTML = '<img src="images/spinner.gif">';
    const fetchPromise = await
fetch('https://cpe-web-assignments.ucdavis.edu/remotedata/index.php');
    const content = await fetchPromise.text();
    //console.log(content);
    document.getElementById('remotedata').innerHTML = content;
}
```

Working with JSON Data

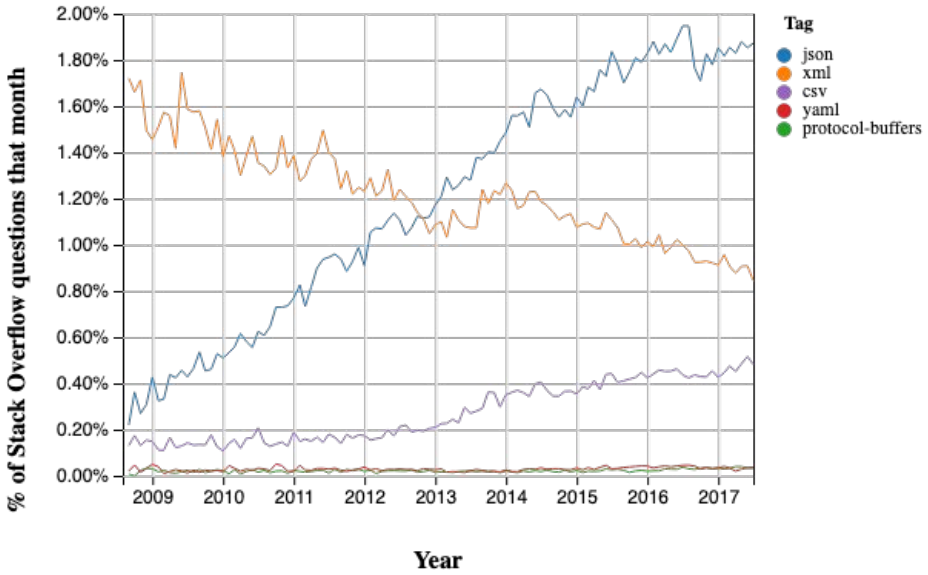
In the first four examples, the PHP file just outputs some text that happens to be in HTML format.

It is much more often when getting data from a remote service, it will come in a standard form for transmitting data.

Data Formats - XML

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
     <age>25</age>
     <first_name>Joe</first_name>
     <last_name>Jones</last_name>
     <social_media>
          <network>facebook</network>
          <network>Twitter</network>
          <network>Instagram</network>
     </social media>
     <online>true</online>
     <phone_numbers>
          <home>916-123-4567</home>
          <mobile>530-212-1111</mobile>
          <work>916-321-9876</work>
     </phone_numbers>
     <status>active</status>
</root>
```

XML remains a popular format for data, but has largely been surpassed by JSON.



JSON & Objects

```
var user = '{
     "first_name": "Joe",
     "last_name": "Jones",
     "age": 25,
     "social_media": [
          "facebook",
          "Twitter",
          "Instagram"
     ],
     "online": true,
     "phone_numbers": {
          "home": "916-123-4567",
          "work": "916-321-9876",
          "mobile": "530-212-1111"
     },
     "status": "active"
}';
```

```
var user = {
     first_name: "Joe",
     last_name: "Jones",
     age: 25,
     social_media: [
          "facebook",
          "Twitter",
          "Instagram"
     ],
     online: true,
     phone_numbers: {
          home: "916-123-4567",
          work: "916-321-9876",
          mobile: "530-212-1111"
     },
     status: "active"
};
```

Updated PHP File

This PHP file differs from the previous one in that it does not output a string of text, instead it outputs data in JSON format.

The address for this file is: https://cpe-web-assignments.ucdavis.edu/remotedata/data.php

If you go to that page, after waiting between 5 and 30 seconds, you will get JSON data.

```php
<?php

date_default_timezone_set("America/Los_Angeles");

$time_requested = date('l jS \of F Y h:i:s A');

$sleep_time = rand(5, 30);

sleep($sleep_time);

$time_processed = date('l jS \of F Y h:i:s A');

$myData->time_requested = $time_requested;
$myData->sleep_time = $sleep_time;
$myData->time_processed = $time_processed;

$myJSON = json_encode($myData);

echo $myJSON;

?>
```

Example 5

```
function getData() {
    document.getElementById('remotedata').innerHTML = '<img src="images/spinner.gif">';
    const fetchPromise = fetch('https://cpe-web-assignments.ucdavis.edu/remotedata/data.php');
    fetchPromise.then(function (response) {
        response.json().then(function (data) {
            document.getElementById('remotedata').innerHTML = outputHTML(data);
        });
    });
}
```

The getData() function has been updated to handle the JSON data.

Example 5: OutputHTML Function

You can see that this function simply takes the data object, and extracts the data while constructing the same HTML shown in examples 1-4.

That HTML is returned and put into the div on the page with innerHTML.

```
function outputHTML(data) {
    const html = `<div class="serverdata">
        <h2>${data.sleep_time} Seconds</h2>
        <p>${data.time_requested}</p>
        <p>${data.time_processed}</p>
    </div>`;
    return html;
}
```

Example 6: Async and Await with JSON

```
async function getData() {
    document.getElementById('remotedata').innerHTML = '<img src="images/spinner.gif">';
    const fetchPromise = await fetch('https://cpe-web-assignments.ucdavis.edu/remotedata/data.php');
    const data = await fetchPromise.json();
    document.getElementById('remotedata').innerHTML = outputHTML(data);
}
```

You can see that the version with async and await is not very different.

Summary

That gives you an overview of various ways to asynchronously handle tasks with JavaScript.

Error handling is not included in any of these examples and would important to include in any real world projects.