Common Use Cases of Asynchronous JavaScript Programming

Learn some common use cases for asynchronous code. We'll walk through examples of DOM manipulation, dynamic loading, and file system management.

DOM Manipulation

When we click a button on a web page, we're using asynchronous code. The developer has set up an event listener such that when we click, some piece of code fires. In between the loading of the page and the user clicking that button, the engine sits idle.

If you've ever used jQuery you've seen that it's entirely based on callbacks.

```
$('#submitButton').on('click', function() {
    console.log('You clicked me!');
});
```

The second parameter to `$('#submitButton').on` is an anonymous function. If we had an HTML page with a button with an id of `submitButton`, this code block would set that function as the event handler on it. It'll simply call the function and log `'You clicked me!'` every time we click the button.

Dynamic Loading

Let's say that we're loading content on a user's news feed. There's a button at the bottom of their feed that lets them load more items. To implement this button, we'd have to do a few things.

We need to set an event handler to listen to the button. On click, we need to do something, so we need to attach a callback function to the button.

When the user clicks, we need to make a request to our servers to fetch more content. This fetch operation is asynchronous. On this request, we need to

attach another callback to do something with the data once it's returned. We need this callback to attach the data to the page.

So we have a callback nested inside a callback. Here's what this code might look like.

```
$('#loadMoreItems').on('click', function() {
    $.ajax({
        url: '/more-data',
        success: function(data) {
            $('#content').html(data);
        }
    });
});
```

This is a common use case and one of the simpler ones. Often, callbacks need to be nested much deeper for more complex operations.

File Systems

In Node.js, reading and writing to files upon user request is very common. Reading and writing take time, so it's asynchronous.

Occasionally, the contents of one file need to be copied to another file. For example, if we have an online shopping platform, we might keep track of the items in a customer's cart in a file. When the customer submits an order, we need to copy those items to a separate file where we store their completed orders.

This code would require first setting up a callback that listens for user requests. Inside this callback, we would have to read a file. This read operation would itself take a callback, as we'd need to do something with the data once it's loaded. We'd need to take the data and write it to another file, another async operation. After the file is successfully written to, yet another callback would send a response to the user saying the operation was successful.

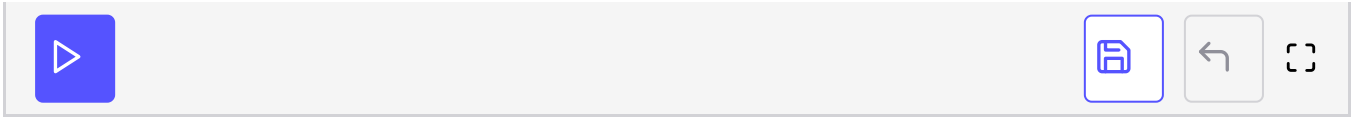
Here's some code as to what that would look like. Assume this is what the `request` parameter passed to `onRequest` looks like:

```
request = {
    fileToRead: 'readFrom.txt',

    fileToWrite: 'writeTo.txt'
};
```

It's also called with a `response` object which will be used to send a response back to the client when everything is completed.

```
onRequest(function(request, response) {
    readFile(request.fileToRead, function(data) {
        writeFile(request.fileToWrite, data, function(status) {
            response.send(status);
        });
    });
});
```



As we can see, the whitespace to the left of the code starts to resemble a triangle or pyramid. This is why nested callbacks are often referred to as the "pyramid of doom". This is also referred to as "callback hell", when callbacks are so nested that it's hard to tell what's going on.

Conclusion

Asynchrony in JavaScript is absolutely essential as we perform tasks that will take some time, or as we need to respond to events that occur. As you build web pages, keep what you've learned here in mind. Your understanding will continue to grow as your user interactions become more complex.