

## Developing Back-End Apps with Node.js and Express

### Module 2 Cheat Sheet: Asynchronous I/O with Callback Program

Package/Method	Description	Code Example
<b>Async-await</b>	We can await promises as long as they are being called inside asynchronous functions.	<pre>const axios = require('axios').default; let url = "some remote url" async function asyncCall() {   console.log('calling');   const result = await axios.get(url);   console.log(result.data); } asyncCall();</pre>
<b>Callback</b>	Callbacks are methods that are passed as parameters. They are invoked within the method to which they are passed as a parameter, conditionally or unconditionally. We use callbacks with a promise to process the response or errors.	<pre>//function(res) and function(err) are the anonymous callback functions axios.get(url).then(function(res) {   console.log(res); }).catch(function(err) {   console.log(err) })</pre>
<b>Promise</b>	An object that is returned by some methods, representing eventual completion or failure. The code continues to run without getting blocked until the promise is fulfilled or an exception is thrown.	<pre>axios.get(url).then(   //do something ).catch(   //do something )</pre>
<b>Promise use case</b>	Promises are used when the processing time of the function we invoke takes time like remote URL access, I/O operations file reading, etc.	<pre>let prompt = require('prompt-sync')(); let fs = require('fs'); const methCall = new Promise((resolve,reject)=&gt;{   let filename = prompt('What is the name of the file ?');   try {     const data = fs.readFileSync(filename, {encoding:'utf8', flag:'r'});     resolve(data);   } catch(err) {     reject(err)   } }); console.log(methCall); methCall.then(   (data) =&gt; console.log(data),   (err) =&gt; console.log("Error reading file") );</pre>
<b>object.on()</b>	It defines an event handler that the framework calls when an event occurs	<pre>http.request( options, function(response) {   let buffer = '';   ...   response.on('data', function(chunk) {     buffer += chunk;   });   response.on('end', function() {     console.log(buffer);   }); }).end();</pre>
<b>Callback Hell/The Pyramid of Doom</b>	Nested callbacks stacked below one another and waiting for the previous callback. This creates a pyramid structure that affects the readability and maintainability of the code.	<pre>const makeCake = nextStep =&gt; {   buyIngredients(function(shoppingList) {     combineIngredients(bowl, mixer, function(ingredients){       bakeCake(oven, pan, function(batter) {         decorate(icing, function(cake) {           nextStep(cake);         });       });     });   }); };</pre>
<b>Axios Request</b>	The axios package handles HTTP requests and returns a promise object.	<pre>const axios = require('axios').default; const connectToURL=(url)=&gt;{   const req=axios.get(url);   console.log(req);   req.then(resp=&gt;{     console.log("Fulfilled");     console.log(resp.data);   })   .catch(err=&gt;{     console.log("Rejected");   }); } connectToURL('valid-url') connectToURL('invalid-url')</pre>



# Skills Network