

Cheatsheet: JavaScript Async

JavaScript Promises, Callback, Fetch and Axios Terminologies	Description	Code Example
JSON	It is a text-based format used for structuring data in a way that is both human-readable and machine-readable.	<pre>1 { 2 "name": "John Doe", 3 "age": 30, 4 "city": "New York", 5 "email": "johndoe@gmail.com", 6 "hobbies": ["Reading", "Hiking", "Cooking"] 7 }</pre>
Callback	A callback in JavaScript is a function passed as an argument to another function, which is then executed at a later time or under certain conditions.	<pre>1 function greet(name, callback) { 2 console.log('Hello, \${name}!'); 3 callback(); // Executes the callback function 4 } 5 6 function sayGoodbye() { 7 console.log('How are you!'); 8 } 9 10 greet('John Doe', sayGoodbye); // Passing sayGoodbye function as a callback</pre>
XMLHttpRequest Object	It is used to create an instance of the XMLHttpRequest object to initiate an HTTP request.	<pre>1 var xhr = new XMLHttpRequest();</pre>
XMLHttpRequest Open Methods	The open() method sets up the request, specifying the HTTP method (GET, POST, and so on) and the URL.	<pre>1 xhr.open('GET', 'https://api.example.com/data', true);</pre>
send() Method	The send() method is invoked to send the request to the specified URL.	<pre>1 xhr.send();</pre>
Load Data Using XMLHttpRequest	This code describes that data can be loaded using Ajax methods.	<pre>1 <!DOCTYPE html> 2 <html> 3 <head> 4 <title>AJAX Example</title> 5 </head> 6 <body> 7 <button id="loadUsersBtn">Load Users</button> 8 <div id="userList"></div> 9 10 <script> 11 // JavaScript for AJAX functionality 12 document.getElementById('loadUsersBtn').addEventListener('click', function() { 13 // Creating an XMLHttpRequest object 14 var xhr = new XMLHttpRequest(); 15 16 // Define the request 17 xhr.open('GET', 'https://jsonplaceholder.typicode.com/users', true); 18 19 // Handle the response 20 xhr.onload = function() { 21 if (xhr.status >= 200 && xhr.status < 400) { 22 var users = JSON.parse(xhr.responseText); 23 displayUsers(users); 24 } else { 25 console.error('Error fetching data'); 26 } 27 }; 28 29 // Handle network errors 30 xhr.onerror = function() { 31 console.error('Network error'); 32 }; 33 34 // Send the request 35 xhr.send(); 36 }); 37 38 // Function to display users on the page 39 function displayUsers(users) { 40 var userListDiv = document.getElementById('userList'); 41 userListDiv.innerHTML = '<h2>User List</h2>'; 42 var ul = document.createElement('ul'); 43 44 users.forEach(function(user) { 45 var li = document.createElement('li'); 46 li.textContent = user.name; 47 ul.appendChild(li); 48 }); 49 50 userListDiv.appendChild(ul); 51 } 52 </script> 53 </body> 54 </html></pre>
Promise Syntax	Promises are used for tasks like fetching data from a server, reading files, or performing other operations that may take some time to complete.	<pre>1 const myPromise = new Promise((resolve, reject) => { 2 // Asynchronous operation goes here 3 // If successful, call resolve with the result 4 // If an error occurs, call reject with an error 5 });</pre>
Promise with .then and .catch	Promises are used for tasks like fetching data from a server, reading files, or performing using '.then()' method and caught error using '.catch()' method.	<pre>1 const myPromise = new Promise((resolve, reject) => { 2 // Simulated asynchronous operation (e.g., making an API request) 3 setTimeout(() => { 4 const success = true; // Simulating a successful operation 5 if (success) { 6 resolve('Data successfully fetched'); 7 } else { 8 reject('Error: Failed to fetch data'); 9 } 10 }, 1000); 11 }); 12 13 myPromise.then(14 (result) => { 15 // Handle the successful result (e.g., update UI with the data) 16 console.log(result); 17 }, 18 (error) => { 19 // Handle the error (e.g., log the error or show an error message) 20 console.error(error); 21 } 22);</pre>
Fetch API Syntax	It is used for fetching resources from the web, such as data from a server or an API.	<pre>1 fetch(url, options) 2 .then(response => { 3 // Handle the response 4 }) 5 .catch(error => { 6 // Handle any errors that occurred during the fetch 7 });</pre>
Fetch API Get Methods	The GET method is used to retrieve data from the specified resource.	<pre>1 fetch('https://jsonplaceholder.typicode.com/posts') 2 .then(handleResponse) 3 .then(data => { 4 console.log('GET Request Result:', data); 5 }) 6 .catch(error => { 7 console.error('Error:', error); 8 });</pre>
Fetch API POST Method	The POST method is used to submit data to be processed to a specified resource.	<pre>1 const newPost = { 2 title: 'New Post', 3 body: 'This is a new post.', 4 userId: 1 5 }; 6 7 fetch('https://jsonplaceholder.typicode.com/posts', { 8 method: 'POST', 9 headers: { 10 'Content-Type': 'application/json' 11 }, 12 body: JSON.stringify(newPost) 13 }) 14 .then(handleResponse) 15 .then(data => { 16 console.log('POST Request Result:', data); 17 }) 18 .catch(error => { 19 console.error('Error:', error); 20 });</pre>
Fetch API PUT Method	The PUT method is used to update or replace data at the specified resource. It is typically used to update existing records on the server.	<pre>1 const updatedPost = { 2 id: 1, 3 title: 'Updated Post', 4 body: 'This post has been updated.', 5 userId: 1 6 }; 7 8 fetch('https://jsonplaceholder.typicode.com/posts/1', { 9 method: 'PUT', 10 headers: { 11 'Content-Type': 'application/json' 12 }, 13 body: JSON.stringify(updatedPost) 14 }) 15 .then(handleResponse) 16 .then(data => { 17 console.log('PUT Request Result:', data); 18 }) 19 .catch(error => { 20 console.error('Error:', error); 21 });</pre>
Fetch API PATCH Method	The PATCH method is used to apply partial modifications to a resource. It is typically used to update parts of a resource while leaving the rest of the resource unchanged.	<pre>1 const updatedData = { 2 title: 'Updated Title' 3 }; 4 5 fetch('https://jsonplaceholder.typicode.com/posts/1', { 6 method: 'PATCH', 7 headers: { 8 'Content-Type': 'application/json' 9 }, 10 body: JSON.stringify(updatedData) 11 }) 12 .then(handleResponse) 13 .then(data => { 14 console.log('PATCH Request Result:', data); 15 }) 16 .catch(error => { 17 console.error('Error:', error); 18 });</pre>
Fetch API DELETE Method	The DELETE method is used to request the removal of a resource from the server. It is used to delete records or resources.	<pre>1 fetch('https://jsonplaceholder.typicode.com/posts/1', { 2 method: 'DELETE' 3 }) 4 .then(response => { 5 if (response.ok) { 6 console.log('DELETE Request Successful'); 7 } else { 8 throw new Error('DELETE request failed'); 9 } 10 }) 11 .catch(error => { 12 console.error('Error:', error); 13 });</pre>
Axios Library Syntax	It provides a consistent way for making asynchronous HTTP requests to interact with RESTful APIs or other web services.	<pre>1 axios({ 2 method: 'HTTP_METHOD', 3 url: 'URL', 4 headers: { 5 // Headers (optional) 6 }, 7 data: { 8 // Request data (optional) 9 } 10 }) 11 .then(response => { 12 // Handle the successful response 13 }) 14 .catch(error => { 15 // Handle errors 16 });</pre>
install axios	You can install axios using npm in the terminal after installing node.	<pre>1 npm install axios</pre>
Axios Methods	Axios have HTTP method for the request such as 'GET', 'POST', 'PUT', 'DELETE'.	<pre>1 axios({ 2 method: 'HTTP_METHOD', 3 url: 'URL', 4 headers: { 5 // Headers (optional) 6 }, 7 data: { 8 // Request data (optional) 9 } 10 }) 11 .then(response => { 12 // Handle the successful response 13 }) 14 .catch(error => { 15 // Handle errors 16 });</pre>