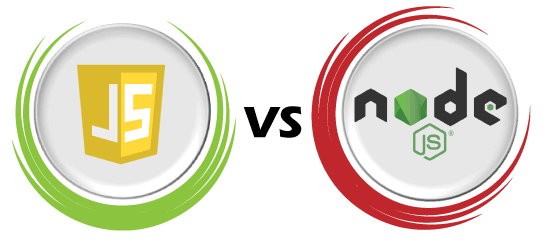
JavaScript vs. Node.js

JavaScript is a **lightweight, object-oriented scripting** language that is used to build **dynamic HTML pages** with interactive effects on a webpage. On the other hand, Node.js usually represents a list of objects and methods accessible to JavaScript code when run in the **V8 engine** or via the node interpreter.

In this article, we are going to discuss the difference between **JavaScript** and **Node.js**. But before discussing the differences, we will know about **JavaScript** and **Node.js**.



**What is JavaScript?**

[Javascript](https://www.javatpoint.com/javascript-tutorial) is a **lightweight, object-oriented scripting** language that is used to build **dynamic HTML pages** with interactive effects on a webpage. JavaScript is also commonly used in **game development** and **mobile app development**. It is an interpreted scripting language, and the code can only be executed and run in a web browser. We can use Node.js to execute and run the code outside of the browser. It's also known as a **browser's language**, and it can be used for both **client-side** and **server-side** development. **Brendan Eich** of **Netscape** created it, and it was first published in **1995**. The language was originally known as **LiveScript** before being renamed JavaScript. JavaScript's syntax is heavily influenced by the [programming language **C**](https://www.javatpoint.com/c-programming-language-tutorial). The extension of JavaScript filename is **.js**.

Advantages and disadvantages of JavaScript

Various advantages and disadvantages of JavaScript are as follows:

65.4M

1.3K

Exception Handling in Java - Javatpoint

**Advantages**

* It is an open-source, flexible, fast, light-weighted framework.
* It allows cross-compilation.
* It supports interfaces, modules, and classes.
* It is used for both frontend and backend development so that it may run on different devices.
* It allows us to build interfaces that react whenever the user hovers the mouse over them.
* It is friendly with other languages; therefore, many other applications may use JavaScript.
* We may extend the JavaScript for writing large applications.

**Disadvantages**

* It makes use of limited libraries.
* Client-side JavaScript doesn't support writing or reading files. It has been only keeping for security purposes.
* A single error may destroy the entire website.
* It is a dynamic language so that developers may make big mistakes easily.
* It only supports the single inheritance, not multiple inheritances. Some programs might be required this object-oriented language feature.

What is Node.js?

[Node.js](https://www.javatpoint.com/nodejs-tutorial) is a **cross-platform, open-source** JavaScript runtime environment that enables JavaScript to be run on the server. Node.js enables JavaScript code to run outside of the browser. Node.js comes with a large number of modules and is mostly used in web creation. It may run on various platforms including, **Windows, Linux, Mac OS**, etc. It provides a cross-platform runtime environment with **event-driven, non-blocking (asynchronous) I/O** for creating highly scalable **server-side** JavaScript applications.

Node.js was **developed** and **introduced** by **Ryan Dahl** in **2009**. Node.js can be used to create a variety of applications, including **web apps, real-time chat apps, command-line apps**, and REST **API servers**, among others. However, it is mainly used to build network programs such as web servers. The standard filename extension of Node.js is **.js**.

Advantages and Disadvantages of Node.js

Various types of advantages and disadvantages of Node.js are as follows:

**Advantages**

* js is an MIT-licensed open-source framework.
* It is a lightweight framework that includes minimum modules. Other modules may be included as per the requirement of an application.
* It's a cross-platform framework that works on Windows, MAC, and Linux.
* It is used the JavaScript language to develop server-side applications.
* js applications never buffered any type of data in applications. Node.js applications simply output data in chunks.
* The Node.js library's APIs are all asynchronous or non-blocking. It simply means that a Node.js based server never waits to return data from an API. After calling an API, the server passes on to the next one, and a Node.js notification mechanism assists the server in receiving a response from the previous API call.
* js is a quick programming execution library built on the V8 JavaScript Engine in Google Chrome.

**Disadvantages**

* One of the big disadvantages of Node.js is its lack of consistency. The API changes regularly, which increases the developers' problems because they'll have to make changes to their current code base to maintain compatibility.
* It doesn't support multi-threading programming, and it is not with the development of heavy computing applications.
* In contrast to other programming languages, JavaScript lacks a well-equipped and functional library framework. As a result, users are forced to need a common library to execute various tasks, including the process of the pictures, **XML parsing, Object-Relational Mapping (ORM)**, and **handling database operations**, etc. It makes it difficult to developers to also implement basic programming tasks with Node.js.

Main differences between JavaScript and Node.js

Here, we are going to discuss the main differences between JavaScript and Node.js:

1. JavaScript is a lightweight, cross-platform, interpreted scripting programming language that is primarily used for client-side scripting. It's built into both Java and HTML. On the other hand, Node.js is a server-side scripting language based on the **Google Chrome V8 engine**. As a result, it's used to build network-centric applications. It is a distributed system that is used for data-intensive real-time applications.
2. JavaScript is a simple programming language that could be run in any browser that supports the JavaScript Engine. On the other hand, Node.js is a running environment or interpreter for the JavaScript programming language. It needs libraries that may be easily accessed from JavaScript programming to make it more useful.
3. JavaScript can run on any engine, including **Firefox's Spider Monkey, Safari's JavaScript Core**, and **V8 (Google Chrome)**. As a result, JavaScript programming is very simple to create, and any running environment is equivalent to a proper browser. On the other hand, Node.js only supports the V8 engine, which is exclusive to Google Chrome. However, written JavaScript code can run in any environment, regardless of whether it supports the V8 engine.
4. A particular non-blocking task is needed to access any operating system. There are some basic objects in JavaScript, but they are all os-specific. **For example, ActiveX Control** is a Windows-only application. However, Node.js is granted the ability to run non-blocking tasks unique to the operating system from any JavaScript programming. There are no os-specific constants in it. Node.js is well-versed in creating a clear binding with the file system, allowing the developer to read and write to disk.
5. For a single web application, JavaScript is typically used for any client-side operation. An operation may be dealing with business validation, dynamic page display at a set time interval, or a simple Ajax call. For any web application, these are used for a maximum amount of time. On the other hand, Node.js is generally used for non-blocking accessibility or execution of any operating system. On an operating system, a non-blocking operation involves generating or executing a shell script, retrieving unique hardware-related information in a single call, obtaining installed certificate details in the system, or performing a large number of specified tasks.
6. JavaScript's main advantages include a wide range of interfaces and interactivity and just the right amount of server contact and direct input to visitors. On the other hand, Node.js has a node package manager with over 500 modules and the ability to handle concurrent requests. It also has the special capability of supporting micro-service architecture as well as IoT.

Head-to-head Comparison between JavaScript and Node JS

Here, we are going to discuss the head-to-head comparison between JavaScript and Node JS:

|  |  |  |
| --- | --- | --- |
| **Features** | **JavaScript** | **Node JS** |
| **Definition** | It is an open-source, cross-platform, interpreted, lightweight scripting programming language that is used to develop dynamic and web applications. | It is a cross-platform, open-source JavaScript runtime environment that allows JavaScript to be run on the server. |
| **Type** | It is a programming language. It works in any browser that has a proper browser engine. | It's a JavaScript interpreter and environment with some valuable libraries that JavaScript programming can use separately. |
| **Dedicated Server** | It is generally used on the client-side server. | It is generally used on the server-side. |
| **Community** | All the JavaScript is not important to the node community. | All node projects represent the JavaScript community. |
| **Running Engines** | JavaScript can be run on any engine, including Spider Monkey, V8, and JavaScript Core. | Node JS is only supported by the V8 engine, which Google Chrome mostly uses. Any JavaScript program written with Node JS will always be run in the V8 engine. |
| **Used for** | It is designed to build network-centric applications. | It's designed for data-intensive real-time applications that run on several platforms. |
| **Languages** | It's a newer version of the ECMA script that runs on Chrome's V8 engine, which is written in C++. | It uses C, C++, and JavaScript. |
| **Modules** | Few JavaScript frameworks are **TypedJS, RamdaJS**, etc. | Lodash, express are examples of Nodejs modules. These all modules are to be imported from **npm**. |
| **Companies Uses** | Various companies use JavaScript like **Google, Shopify, Udacity, Sendgrid, Groupon, Okta, Instacart**, etc. | Various companies use Node Js like **Netflix, Hapi, Walmart, Paypal, Linkedin, Trello, Medium, eBay**, etc. |

# JavaScript Async/Await

JavaScript is always synchronous and single-threaded that provides the event loops. The event loops enable us to queue up an activity. This activity will not happen until the loops become available after the program that queued the action has completed the execution. However, our program contains a large number of functionalities, which causes our code to be asynchronous. The **Async/Await** functionality is one of them. **Async/Await** is an extension of **promises** that we get as language support.

In this article, we are going to discuss the [JavaScript](https://www.javatpoint.com/javascript-tutorial) Async/Await with some examples.

## JavaScript Async

An async function is a function that is declared with the async keyword and allows the await keyword inside it. The async and await keywords allow asynchronous, **promise-based** behavior to be written more easily and avoid configured promise chains. The async keyword may be used with any of the methods for creating a function.

### Syntax:

The syntax of JavaScript may be defined as:

Play Videox

1. Async function myfirstfunction() {
2. return "Hello World"
3. }

It is the same as:

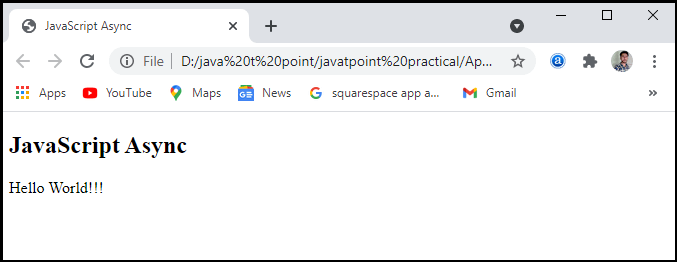
1. async function myfirstfunction() {
2. return Promise.resolve("Hello World");
3. }

### Example:

Let's take an example to understand how we can use a JavaScript Async in the program.

1. <!DOCTYPE html**>**
2. **<html>**
3. **<head>**
4. **<meta** charset="utf-8"**>**
5. **<title>**JavaScript Async**</title>**
6. **</head>**
7. **<body>**
8. **<h2>**JavaScript Async**</h2>**
9. **<p** id="main"**></p>**
10. **<script>**
11. function myDisplayer(some) {
12. document.getElementById("main").innerHTML = some;
13. }
14. async function myfirstFunction() {
15. return "Hello World!!!";
16. }
17. myfirstFunction().then(
18. function(value) {myDisplayer(value);},
19. function(error) {myDisplayer(error);}
20. );
21. **</script>**
22. **</body>**
23. **</html>**

**Output:** After executing the above code, we will get the output shown below in the screenshot.



## JavaScript Await

JavaScript Await function is used to wait for the promise. It could only be used inside the async block. It instructs the code to wait until the promise returns a response. It only delays the async block. Await is a simple command that instructs JavaScript to wait for an asynchronous action to complete before continuing with the feature. It's similar to a **"pause until done"** keyword. The await keyword is used to retrieve a value from a function where we will usually be used the **then()** function. Instead of calling after the asynchronous function, we'd use await to allocate a variable to the result and then use the result in the code as we will in the synchronous code.

### Syntax:

The syntax of JavaScript Await function may be defined as:

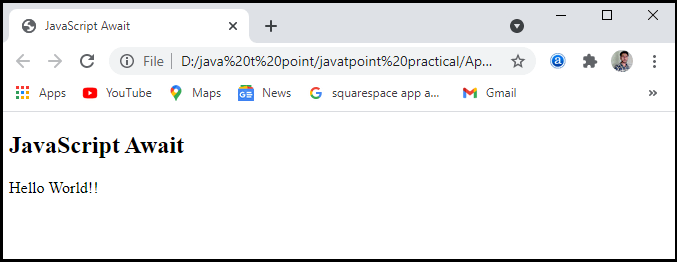
1. // Await function works only inside the async function
2. let value = await promise;

### Example:

Let's take an example to understand how we can use the JavaScript Await function in the program.

1. <!DOCTYPE html**>**
2. **<html>**
3. **<head>**
4. **<meta** charset="utf-8"**>**
5. **<title>**JavaScript Await**</title>**
6. **</head>**
7. **<body>**
8. **<h2>**JavaScript Await**</h2>**
9. **<p** id="main"**></p>**
10. **<script>**
11. async function myDisplay() {
12. let myPromise = new Promise(function(myResolve, myReject) {
13. myResolve("Hello World!!");
14. });
15. document.getElementById("main").innerHTML = await myPromise;
16. }
17. myDisplay();
18. **</script>**
19. **</body>**
20. **</html>**

**Output:** After executing this code, we will get the output as shown below in the screenshot:

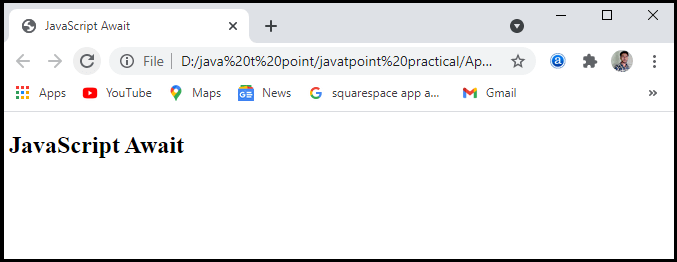


### Example 2: Waiting for a timeout in the program

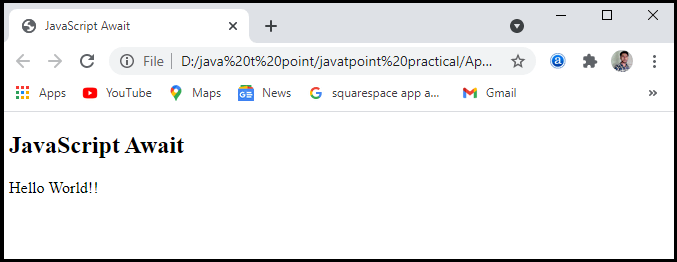
Let's take a program to understand the JavaScript Await using timeout waiting function.

1. <!DOCTYPE html**>**
2. **<html>**
3. **<head>**
4. **<meta** charset="utf-8"**>**
5. **<title>**JavaScript Await**</title>**
6. **</head>**
7. **<body>**
8. **<h2>**JavaScript Await**</h2>**
9. **<p** id="main"**></p>**
10. **<script>**
11. async function myDisplay() {
12. let myPromise = new Promise(function(myResolve, myReject) {
13. setTimeout(function() { myResolve("Hello World!!"); }, 2000);
14. });
15. document.getElementById("main").innerHTML = await myPromise;
16. }
17. myDisplay();
18. **</script>**
19. **</body>**
20. **</html>**

**Output:** After executing this code, we will get the output as shown below in the screenshot.



When we execute the code, it will show the result after **2** seconds. It uses the timeout function.



## Error Handling

It is very easy to handle errors in async functions. Promises have a **catch()** method for dealing with rejected promises, and because the async functions only return a promise, we may call the function and add a method to the end. We should use the promise's capture in the same way as we would any other catch. And all are easy to grasp. Remember that a then callback will fail. It can generate an error (with an explicit throw or by trying to access a property of a null variable). These crashes would also be caught by the grab process. Remind yourself that the promise's capture approach is similar to a standard catch.

### Syntax:

The syntax of error handling may be defined as:

1. asyncFunc().catch(err =**>**
2. {
3. Console.error(err)
4. // catch error and do something
5. });

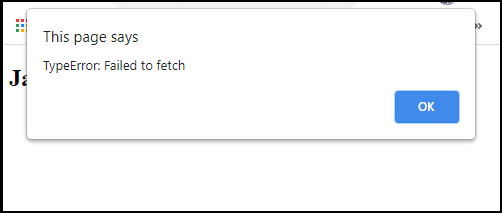
But there is another option: the all-powerful **try/catch block.** If we want to handle errors directly inside the async function, we may use try/catch in the same way we would in synchronous code.

### Example:

Let's take an example to understand the error handling in the JavaScript Async and Await function.

1. <!DOCTYPE html**>**
2. **<html>**
3. **<head>**
4. **<meta** charset="utf-8"**>**
5. **<title>**JavaScript Await**</title>**
6. **</head>**
7. **<body>**
8. **<h2>**JavaScript Await**</h2>**
9. **<script>**
10. async function f() {
11. try {
12. let response = await fetch('http://no-url');
13. } catch(err) {
14. alert(err); // TypeError: failed to fetch
15. }
16. }
17. f();
18. **</script>**
19. **</body>**
20. **</html>**

**Output:** After executing this code, we will get the output as shown below in the screenshot.



It may appear sloppy, but it is a very simple way to handle errors without appending. After the function calls, use the **catch()** method. It is up to us how we manage mistakes, and which approach we use should be dictated by how our code was written. Over time, we'll get a sense of what needs to be achieved.

|  |  |
| --- | --- |
| **Java** | **JavaScript** |
| Java is a complete and strongly typed programming language used for backend coding. In Java, variables must be declared first to use in the program, and the type of a variable is checked at compile-time. | JavaScript is a weakly typed, lightweight programming language (most commonly known as scripting language) and has more relaxed syntax and rules. |
| Java is an object-oriented programming (OOPS) language or structured programming languages such as C, C++, or .Net. | JavaScript is a client-side scripting language, and it doesn't fully support the OOPS concept. It resides inside the HTML documents and is used to make web pages interactive (not achievable with simple HTML). |
| Java creates applications that can run in any virtual machine (JVM) or browser. | JavaScript code can run only in the browser, but it can now run on the server via Node.js. |
| The Java code needs to be compiled. | The JavaScript code doesn't require to be complied. |
| Java Objects are class-based. You can't make any program in Java without creating a class. | JavaScript Objects are prototype-based. |
| Java is a Complete and Standalone language that can be used in backend coding. | JavaScript is assigned within a web page and integrates with its HTML content. |
| Java programs consume more memory. | JavaScript code is used in HTML web pages and requires less memory. |
| The file extension of the Java program is written as ".Java" and it translates source code into bytecodes which are then executed by JVM (Java Virtual Machine). | The JavaScript file extension is written as ".js" and it is interpreted but not compiled. Every browser has a JavaScript interpreter to execute the JS code. |
| Java supports multithreading. | JavaScript doesn't support multithreading. |
| Java uses a thread-based approach to concurrency. | JavaScript uses an event-based approach to concurrency |

| **var** | let |
| --- | --- |
| **1.** | The var is a keyword that is used to declare a variable | The let is also a keyword that is used to declare a variable. |
| **2.** | Syntax -:  **var name = value;** | Syntax -:  **let name = value;** |
| **3.** | The variables that are defined with **var**statement have function scope. | The variables that are defined with **let**statement have block scope. |
| **4.** | We can declare a variable again even if it has been defined previously in the same scope. | We cannot declare a variable more than once if we defined that previously in the same scope. |
| **5.** | Hoisting is allowed with **var**. | Hoisting is not allowed with **let**. |
| **6.** | Example -:  **var websitename = “geeksforgeeks”;** | Example -:  **let x = 69;** |
| **7.** | var is an ECMAScript1 feature. | let is a feature of ES6. |
| **8.** | Its supported browsers are: Chrome, Internet Explorer, Microsoft Edge, Firefox, safari, opera | Its supported browsers are -: Chrome49, Microsoft Edge12, firefox44 , safari11, opera36 |