Dashboard Prepare

New Advertise with us →

Quiz

Describe the difference between `<script>`, `<script async>` and `<script defer>`

High Medium HTML JavaScript

Edit on GitHub

TL;DR

All of these ways (`<script>`, `<script async>`, and `<script defer>`) are used to load and execute JavaScript files in an HTML document, but they differ in how the browser handles loading and execution of the script:

- `<script>` is the default way of including JavaScript. The browser blocks HTML parsing while the script is being downloaded and executed. The browser will not continue rendering the page until the script has finished executing.
- `<script async>` downloads the script asynchronously, in parallel with parsing the HTML. Executes the script as soon as it is available, potentially interrupting the HTML parsing. `<script async>` do not wait for each other and execute in no particular order.
- `<script defer>` downloads the script asynchronously, in parallel with parsing the HTML. However, the execution of the script is deferred until HTML parsing is complete, in the order they appear in the HTML.

Here's a table summarizing the 3 ways of loading `<script>`s in a HTML document.

Feature	`<script>`	`<script async>`	`<script defer>`
Parsing behavior	Blocks HTML parsing	Runs parallel to parsing	Runs parallel to parsing
Execution order	In order of appearance	Not guaranteed	In order of

Feature	`<script>`	`<script async>`	`<script defer>`
DOM dependency	No	No	Yes (waits for DOM)

What `<script>` tags are for

`<script>` tags are used to include JavaScript on a web page. The `async` and `defer` attributes are used to change how/when the loading and execution of the script happens.

`<script>`

For normal `<script>` tags without any `async` or `defer`, when they are encountered, HTML parsing is blocked, the script is fetched and executed immediately. HTML parsing resumes after the script is executed. This can block rendering of the page if the script is large.

Use `<script>` for critical scripts that the page relies on to render properly.

```html
<!doctype html>
<html>
  <head>
    <title>Regular Script</title>
  </head>
  <body>
    <!-- Content before the script -->
    <h1>Regular Script Example</h1>
    <p>This content will be rendered before the script executes.</p>

    <!-- Regular script -->
    <script src="regular.js"></script>

    <!-- Content after the script -->
    <p>This content will be rendered after the script executes.</p>
  </body>
</html>
```

`<script async>`

In `<script async>`, the browser downloads the script file asynchronously (in parallel with HTML parsing) and executes it as soon as it is available (potentially before HTML parsing completes). The execution will not necessarily be executed in the order in which it appears in the HTML document. This can improve perceived performance because the browser doesn't wait for the script to download before continuing to render the page.

Use `<script async>` when the script is independent of any other scripts on the page, for example, analytics and ads scripts.

```html
<!doctype html>
<html>
  <head>
    <title>Async Script</title>
  </head>
  <body>
    <!-- Content before the script -->
    <h1>Async Script Example</h1>
    <p>This content will be rendered before the async script executes.</p>

    <!-- Async script -->
    <script async src="async.js"></script>

    <!-- Content after the script -->
    <p>
      This content may be rendered before or after the async script executes.
    </p>
  </body>
</html>
```

`<script defer>`

Similar to `<script async>`, `<script defer>` also downloads the script in parallel to HTML parsing but the script is only executed when the document has been fully parsed and before firing `DOMContentLoaded`. If there are multiple of them, each deferred script is executed in the order they appeared in the HTML document.

If a script relies on a fully-parsed DOM, the `defer` attribute will be useful in ensuring that the HTML is fully parsed before executing.

```html
<!doctype html>
<html>
  <head>
    <title>Deferred Script</title>
  </head>
  <body>
    <!-- Content before the script -->
    <h1>Deferred Script Example</h1>
    <p>This content will be rendered before the deferred script executes.</p>

    <!-- Deferred script -->
    <script defer src="deferred.js"></script>

    <!-- Content after the script -->
    <p>This content will be rendered before the deferred script executes.</p>
  </body>
</html>
```

Notes

- The `async` attribute should be used for scripts that are not critical to the initial rendering of the page and do not depend on each other, while the `defer` attribute should be used for scripts that depend on / is depended on by another script.
- The `async` and `defer` attributes are ignored for scripts that have no `src` attribute.
- `<script>`s with `defer` or `async` that contain `document.write()` will be ignored with a message like "A call to `document.write()` from an asynchronously-loaded external script was ignored".
- Even though `async` and `defer` help to make script downloading asynchronous, the scripts are still eventually executed on the main thread. If these scripts are computationally intensive, it can result in laggy/frozen UI. Partytown is a library that helps relocate script executions into a web worker and off the main thread, which is great for third-party scripts where you do not have control over the code