Where should I put <script> tags in HTML markup?

Asked 14 years, 3 months ago Modified 4 months ago Viewed 784k times

1849

When embedding JavaScript in an HTML document, where is the proper place to put the `<script>` tags and included JavaScript? I seem to recall that you are not supposed to place these in the `<head>` section, but placing at the beginning of the `<body>` section is bad, too, since the JavaScript will have to be parsed before the page is rendered completely (or something like that). This seems to leave the *end* of the `<body>` section as a logical place for `<script>` tags.

So, where *is* the right place to put the `<script>` tags?

(This question references this question, in which it was suggested that JavaScript function calls should be moved from `<a>` tags to `<script>` tags. I'm specifically using jQuery, but more general answers are also appropriate.)

javascript html

Share Improve this question Follow

edited Dec 16, 2022 at 14:19

TylerH
20.6k 64 76 97

asked Jan 12, 2009 at 18:15

mipadi
395k 89 522 478

1 in case you're also just looking for a simple solution and you're using some server side generator like Jekyll, i recommend including the script with it instead. so much simpler! – cregox Sep 20, 2019 at 5:17

2 If coming from a search engine looking for this: Many of the answers are not clear ***exactly where the 'script tag' should be at the end***. If the 'script' tag is ***after*** '</body>', HTML validation will result in "*Error: Stray start tag script*" (check option "*source*" and click "*check*" to see the HTML source). It should be ***before*** '</body>'. (The result is similar if the 'script' tag is at the very end, after the ***</html>*** tag.) – Peter Mortensen Nov 21, 2021 at 15:22

1 This is also addressed in *Is it wrong to place the <script> tag after the </body> tag?*. – Peter Mortensen Nov 21, 2021 at 17:54

3 T.L.D.R. put it inside the `<head>` tag with `defer` attribute, or even better make your script `type='module'` . It is 2022 now. – Beki May 28, 2022 at 12:06

21 Answers

Sorted by:

Highest score (default)

Here's what happens when a browser loads a website with a `<script>` tag on it:

2. Begin parsing the HTML
3. The parser encounters a `<script>` tag referencing an external script file.
4. The browser requests the script file. Meanwhile, the parser blocks and stops parsing the other HTML on your page.
5. After some time the script is downloaded and subsequently executed.
6. The parser continues parsing the rest of the HTML document.

Step #4 causes a bad user experience. Your website basically stops loading until you've downloaded all scripts. If there's one thing that users hate it's waiting for a website to load.

Why does this even happen?

Any script can insert its own HTML via `document.write()` or other DOM manipulations. This implies that the parser has to wait until the script has been downloaded and executed before it can safely parse the rest of the document. After all, the script *could* have inserted its own HTML in the document.

However, most JavaScript developers no longer manipulate the DOM *while* the document is loading. Instead, they wait until the document has been loaded before modifying it. For example:

```
<!-- index.html -->
<html>
    <head>
        <title>My Page</title>
        <script src="my-script.js"></script>
    </head>
    <body>
        <div id="user-greeting">Welcome back, user</div>
    </body>
</html>
```

JavaScript:

```
// my-script.js
document.addEventListener("DOMContentLoaded", function() {
    // this function runs when the DOM is ready, i.e. when the document has been
parsed
    document.getElementById("user-greeting").textContent = "Welcome back, Bart";
});
```

Because your browser does not know *my-script.js* isn't going to modify the document until it has been downloaded and executed, the parser stops parsing.

The old approach to solving this problem was to put `<script>` tags at the bottom of your `<body>`, because this ensures the parser isn't blocked until the very end.

This approach has its own problem: the browser cannot start downloading the scripts until the entire document is parsed. For larger websites with large scripts and stylesheets, being able to download the script as soon as possible is very important for performance. If your website doesn't load within 2 seconds, people will go to another website.

In an optimal solution, the browser would start downloading your scripts as soon as possible, while at the same time parsing the rest of your document.

The modern approach

Today, browsers support the `async` and `defer` attributes on scripts. These attributes tell the browser it's safe to continue parsing while the scripts are being downloaded.

async

```
<script src="path/to/script1.js" async></script>
<script src="path/to/script2.js" async></script>
```

Scripts with the async attribute are executed asynchronously. This means the script is executed as soon as it's downloaded, without blocking the browser in the meantime. This implies that it's possible that script 2 is downloaded and executed before script 1.

According to http://caniuse.com/#feat=script-async, 97.78% of all browsers support this.

defer

```
<script src="path/to/script1.js" defer></script>
<script src="path/to/script2.js" defer></script>
```

Scripts with the defer attribute are executed in order (i.e. first script 1, then script 2). This also does not block the browser.

Unlike async scripts, defer scripts are only executed after the entire document has been loaded.

(To learn more and see some really helpful visual representations of the differences between async, defer and normal scripts check the first two links at the references section of this answer)

Conclusion

The good thing is that your website should still load correctly on the 2% of browsers that do not support these attributes while speeding up the other 98%.

References

- async vs defer attributes
- Efficiently load JavaScript with defer and async
- Remove Render-Blocking JavaScript
- Async, Defer, Modules: A Visual Cheatsheet

Share Improve this answer Follow

edited Dec 16, 2022 at 14:23

TylerH
20.6k 64 76 97

answered Jun 5, 2014 at 21:20

Bart
26.1k 1 23 24

1 @Jori Common practice is to load your javascript asap (and non-blocking via async) and postpone any DOM manipulations until the "DOMContentLoaded" event is dispatched. – Bart Jul 24, 2014 at 12:09

8 I'm not clear on what touches the DOM and what doesn't. Can you clarify? Is it safe to do an async load on something like jquery.js? – Doug Sep 6, 2014 at 23:11

7 @Doug For example `document.write` operates on the dom. The question isn't *if* a script manipulates the dom, but *when* it does. As long as all dom manipulation happens after `domready` event has triggered, you're ok. jQuery is a library, and as such doesn't - or shouldn't - manipulate the dom by itself. – Bart Sep 9, 2014 at 15:34

26 This answer is misleading. Modern browsers don't stop parsing when they reach a synchronous script tag that may affect the HTML, they just stop rendering/executing, and continue parsing optimistically to start downloading other resources that will probably be requested subsequently if no HTML is affected. – Fabio Beltramini Oct 16, 2014 at 21:23

47 Why the `async` and `defer` attributes are not used nowhere? I mean, I viewed a lot of HTML sources from internet, and I don't see the `async` and `defer` attributes anywhere. ... ? – john c. j. May 10, 2016 at 14:02

254

Just before the closing body tag, as stated on *Put Scripts at the Bottom*:

> Put Scripts at the Bottom
>
> The problem caused by scripts is that they block parallel downloads. The HTTP/1.1 specification suggests that browsers download no more than two components in parallel per hostname. If you serve your images from multiple hostnames, you can get more than two downloads to occur in parallel. While a script is downloading, however, the browser won't start any other downloads, even on different hostnames.

 Peter Mortensen
31.2k 21 106 129

 Cammel
2,647 2 16 5

7 Agreed with the concept and its explanation. But what happens if the user starts playing with the page. Suppose I've an AJAX dropdown which will start loading after the page has appeared to the user but while it is loading, the user clicks it! And what if a 'really impatient' user submits the form? – Hemant Tank Jan 3, 2012 at 14:29

9 @Hermant Old comment but you may do the trick disabling the fields by default then enabling them using JS when the DOM is fully loaded. That's what Facebook seems to be doing nowadays. – jmic Nov 11, 2012 at 6:20

55 If this is best practice, why does stack overflow include all their script tags in <head>? :-P – Philip Apr 20, 2013 at 5:04

10 In some cases, especially in ajax heavy sites, loading in head can actually result in faster load times. See: encosia.com/dont-let-jquerys-document-ready-slow-you-down (note that the "live()" function is deprecated in jquery, but the article still applies with the "on()" or "delegate" function). Loading in <head> may also be needed to guarantee correct behavior as pointed out by @Hermant. Finally, modernizr.com/docs recommends placing its scripts in the <head> for reasons explained on its site. – Nathan Jun 7, 2013 at 15:54

3 WARNING: No, this WAS a better practice. put it inside the `<head>` tag with `defer` attribute, or even better make your script `type='module'` . It is 2022 now. – Beki May 28, 2022 at 12:09

102

Non-blocking script tags can be placed just about anywhere:

```
<script src="script.js" async></script>
<script src="script.js" defer></script>
<script src="script.js" async defer></script>
```

- `async` script will be executed asynchronously as soon as it is available
- `defer` script is executed when the document has finished parsing
- `async defer` script falls back to the defer behavior if async is not supported

Such scripts will be executed asynchronously/after document ready, which means you cannot do this:

```
<script src="jquery.js" async></script>
<script>jQuery(something);</script>
<!--
  * might throw "jQuery is not defined" error
  * defer will not work either
-->
```

Or this:

```
<script src="document.write(something).js" async></script>
<!--
  * might issue "cannot write into document from an asynchronous script" warning
  * defer will not work either
-->
```

Or this:

```
<script src="jquery.js" async></script>
<script src="jQuery(something).js" async></script>
<!--
  * might throw "jQuery is not defined" error (no guarantee which script runs
first)
  * defer will work in sane browsers
-->
```

Or this:

```
<script src="document.getElementById(header).js" async></script>
<div id="header"></div>
<!--
  * might not locate #header (script could fire before parser looks at the next
line)
  * defer will work in sane browsers
-->
```

Having said that, asynchronous scripts offer these advantages:

- **Parallel download of resources**:
  Browser can download stylesheets, images and other scripts in parallel without waiting for a script to download and execute.
- **Source order independence**:
  You can place the scripts inside head or body without worrying about blocking (useful if you are using a CMS). Execution order still matters though.

It is possible to circumvent the execution order issues by using external scripts that support callbacks. Many third party JavaScript APIs now support non-blocking execution. Here is an example of loading the Google Maps API asynchronously.

Share Improve this answer Follow

edited Feb 6, 2015 at 13:37

answered Feb 6, 2015 at 11:19

Salman A
257k 81 428 518

2 This is the correct answer for today - using this approach means it's easier to keep your widgets self contained, no need to do fancy `<head>` include logic. – Daniel Sokolowski Jul 7, 2015 at 18:17

is still loading]. Thanks! – elbowlobstercowstand Aug 6, 2015 at 8:10

3 @elbow 99% of times `<script src=jquery.js>` is followed by `$(function(){ ... })` blocks somewhere in the page. Asynchronous loading does not guarantee that jQuery will be loaded at the time browser tries to parse those blocks hence it will raise *$ is not defined* error (you may not get the error if jQuery was loaded from cache). I answered a question about loading jQuery asynchronously and preserve `$(function(){ ... })` . I'll see if I could find it, or you can look at this question: stackoverflow.com/q/14811471/87015 – Salman A Aug 6, 2015 at 9:12

1 @SalmanA Thank you! Yes, I fall in that 99%. I first need `jquery` lib to load, *then* my remaining `.js` scripts. When I declare `async` or `defer` on the `jquery` lib script tag, my `.js` scripts don't work. I thought `$(function(){ ... })` protected that—guess not. **Current solution:** I don't add `defer` or `async` on `jquery` lib script, but I do add `async` on my follow up `.js` scripts. Note: the reason I'm doing *any* of this is to make Google Page Speed happy. Thx again for the help! Any other advice is welcome. (Or a link to your previous answer). :) – elbowlobstercowstand Aug 6, 2015 at 21:22

@elbow See stackoverflow.com/a/21013975/87015, it will only give you an idea but not the complete solution. You could instead search for "jquery async loader libraries". – Salman A Aug 7, 2015 at 5:40

42

The standard advice, promoted by the *Yahoo!* Exceptional Performance team, is to put the `<script>` tags at the end of the document's `<body>` element so they don't block rendering of the page.

But there are some newer approaches that offer better performance, as described in this other answer of mine about the load time of the Google Analytics JavaScript file:

> There are some great slides by Steve Souders (client-side performance expert) about:
>
> - Different techniques to load external JavaScript files in parallel
> - their effect on loading time and page rendering
> - what kind of "in progress" indicators the browser displays (e.g. 'loading' in the status bar, hourglass mouse cursor).

Share Improve this answer Follow

edited Dec 16, 2022 at 14:36
TylerH
20.6k 64 76 97

answered Jan 12, 2009 at 23:37

orip
72.5k 21 118 148

39

The modern approach is using ES6 'module' type scripts.

```
<script type="module" src="..."></script>
```

By default, modules are loaded asynchronously and deferred, i.e. you can place them anywhere

Further reading:

- The differences between a script and a module
- The execution of a module being deferred compared to a script(Modules are deferred by default)
- Browser Support for ES6 Modules

Share Improve this answer Follow

edited Dec 16, 2022 at 14:34

TylerH
20.6k 64 76 97

answered Feb 1, 2019 at 15:17

cquezel
3,699 1 29 31

5 Just a note that this will not work if you're just trying stuff out on your local filesystem with no server. At lest on Chrome you get a cross-origin error trying to load the js from the HTML even though they both have the same origin, your filesystem. – hippietrail Sep 10, 2019 at 6:26

25

If you are using jQuery then put the JavaScript code wherever you find it best and use `$(document).ready()` to ensure that things are loaded properly before executing any functions.

On a side note: I like all my script tags in the `<head>` section as that seems to be the cleanest place.

Share Improve this answer Follow

edited Nov 21, 2021 at 13:22

Peter Mortensen
31.2k 21 106 129

answered Jan 12, 2009 at 18:18

Andrew Hare
342k 71 636 633

8 Note that using `$(document).ready()` doesn't mean you can put your JavaScript *anywhere* you like – you still have to put it after the `<script src="../jquery.min.js">` where you include jQuery, so that `$` exists. – Rory O'Kane Jul 10, 2013 at 15:42

2 It is not optimal to put script tags in the <head> section - this will delay display of the visible part of the page until the scripts are loaded. – CyberMonk Oct 14, 2013 at 0:33

20

```
    <script src="myjs.js"></script>
</body>
```

The script tag should always be used before the **body close** or at the **bottom in HTML** file.

The Page will load with HTML and CSS and later JavaScript will load.

Check this if required:

Share Improve this answer Follow

edited Nov 21, 2021 at 14:19

Peter Mortensen
31.2k 21 106 129

answered Jul 16, 2016 at 11:16

AmanKumar
1,293 2 20 33

3 This actually answered the question. I was wondering nearly all the examples posted never gave the proper visual context of "end of the page" – Ken Ingram Dec 16, 2016 at 1:23

2 This answer is very misleading and most probably wrong. The articles in Google and in MDN suggests that synchronous JS (which is the case here) always blocks DOM construction and parsing which will result in delayed first-render. Therefore, you cannot see the content of the page until the JS file is fetched and finishes executing regardless of where you place your JS file in the HTML document as long as it is synchronous – Lingaraju E V Oct 26, 2017 at 6:31

1 It also references points made in 2009 and no longer relevant. – toxaq Oct 30, 2017 at 1:37

1 did you actually mean "otherwise you'll be able to see the content before loading the js file and that's bad?" – ahnbizcad Sep 10, 2020 at 23:27

1 It is good practice not to put script tags at the end of the body or html code. Just like other declarative or meta information this should go in the head section reducing clutter of none content related information all over the place. Get used to using async or even better defer. Word Press can be a bit tricky, though. It puts the JavaScript in the wp_head but without the defer. stackoverflow.com/questions/18944027/... – theking2 Apr 28, 2021 at 15:51

13

The **best** place to put `<script>` tag is before closing `</body>` tag, so the downloading and executing it doesn't block the browser to parse the HTML in document,

Also loading the JavaScript files externally has its own advantages like it will be cached by browsers and can speed up page load times, it separates the HTML and JavaScript code and help to manage the code base better.

But modern browsers also support some other optimal ways, like `async` and `defer` to load external JavaScript files.

Async and Defer

Normally HTML page execution starts line by line. When an external JavaScript `<script>` element is encountered, HTML parsing is stopped until a JavaScript is download and ready for execution. This normal page execution can be changed using the `defer` and `async` attribute.

`Defer`

When a defer attribute is used, JavaScript is downloaded parallelly with HTML parsing, but it will be execute only after full HTML parsing is done.

Async

When the async attribute is used, JavaScript is downloaded as soon as the script is encountered and after the download, it will be executed asynchronously (parallelly) along with HTML parsing.

```
<script src="/local-js-path/myScript.js" async></script>
```

When to use which attributes

- If your script is independent of other scripts and is modular, use `async` .
- If you are loading script1 and script2 with `async` , both will run parallelly along with HTML parsing, as soon as they are downloaded and available.
- If your script depends on another script then use `defer` for both:
- When script1 and script2 are loaded in that order with `defer` , then script1 is guaranteed to execute first,
- Then script2 will execute after script1 is fully executed.
- Must do this if script2 depends on script1.
- If your script is small enough and is depended by another script of type `async` then use your script with no attributes and place it above all the `async` scripts.

Reference: *External JavaScript JS File – Advantages, Disadvantages, Syntax, Attributes*

Share Improve this answer Follow

edited Nov 23, 2021 at 18:24

Peter Mortensen
31.2k 21 106 129

answered Jul 9, 2019 at 10:26

Haritsinh Gohil
5,554 48 46

10

It turns out it can be everywhere.

You can defer the execution with something like jQuery so it doesn't matter where it's placed (except for a small performance hit during parsing).

Share Improve this answer Follow

edited Nov 21, 2021 at 13:31

Peter Mortensen
31.2k 21 106 129

answered Jan 12, 2009 at 18:22

Allain Lalonde
90.9k 70 186 238

1 XHTML will validate with script tags in the body, both strict and transitional. Style tags however may only be in the head. – I.devries Jan 12, 2009 at 18:43

Re "*it doesn't matter where it's placed*": But not so the result is not well-formed HTML? – Peter Mortensen Nov 21, 2021 at 13:33

Fair enough. It's been 13 years since I was that naive. – Allain Lalonde Dec 17, 2022 at 1:01

6

The most conservative (and widely accepted) answer is "at the bottom just before the ending tag", because then the entire DOM will have been loaded before anything can start executing.

There are dissenters, for various reasons, starting with the available practice to intentionally begin execution with a page onload event.

Share Improve this answer Follow

edited Nov 21, 2021 at 21:19

answered Jan 12, 2009 at 18:18

dkretz

37.3k 13 80 138

Re "*at the bottom*"": But *before* the ending body tag (`</body>`)? Can you make it clearer? (But ***without*** "Edit:", "Update:", or similar - the answer should appear as if it was written today). – Peter Mortensen Nov 21, 2021 at 13:46

4

It depends. If you are loading a script that's necessary to style your page / using actions in your page (like click of a button) then you better place it at the top. If your styling is 100% CSS and you have all fallback options for the button actions then you can place it at the bottom.

Or the best thing (if that's not a concern) is you can make a modal loading box, place your JavaScript code at the bottom of your page and make it disappear when the last line of your script gets loaded. This way you can avoid users using actions in your page before the scripts are loaded. And also avoid the improper styling.

Share Improve this answer Follow

edited Nov 21, 2021 at 13:50

Peter Mortensen

31.2k 21 106 129

answered Nov 18, 2013 at 4:41

mze3e

418 2 5 14

Re "*place your JavaScript code at the bottom of your page*": But ***before*** the ending body tag (`</body>`)? Can you make it clearer? (But ***without*** "Edit:", "Update:", or similar - the answer should appear as if it was written today). – Peter Mortensen Nov 21, 2021 at 13:51

3

Including scripts at the end is mainly used where the content/ styles of the web page is to be shown first.

Including the scripts in the head loads the scripts early and can be used before the loading of the whole web page.

If the scripts are entered at last the validation will happen only after the loading of the entire

Share Improve this answer Follow

edited Nov 21, 2021 at 16:53

Peter Mortensen
31.2k 21 106 129

answered Nov 27, 2017 at 6:12

Sanjeev S
1,093 2 14 33

2

You can add JavaScript code in an HTML document by employing the dedicated HTML tag `<script>` that wraps around JavaScript code.

The `<script>` tag can be placed in the `<head>` section of your HTML, in the `<body>` section, or after the `</body>` close tag, depending on when you want the JavaScript to load.

Generally, JavaScript code can go inside of the document `<head>` section in order to keep them contained and out of the main content of your HTML document.

However, if your script needs to run at a certain point within a page's layout — like when using `document.write` to generate content — you should put it at the point where it should be called, usually within the `<body>` section.

Share Improve this answer Follow

answered Jan 22, 2021 at 15:31

Nitesh Singh
385 1 3 15

> Re "*after the </body> close tag*": ***No, no, no!***. If the 'script' tag is ***after*** '</body>', HTML validation will result in "*Error: Stray start tag script*" (check option "*source*" and click "*check*" to see the HTML source). If it is ***before***, it validates. – Peter Mortensen Nov 21, 2021 at 15:26

2

Depending on the script and its usage the best possible (in terms of page load and rendering time) may be to not use a conventional <script>-tag per se, but to dynamically trigger the loading of the script asynchronously.

There are some different techniques, but the most straightforward is to use document.createElement("script") when the window.onload event is triggered. Then the script is loaded first when the page itself has rendered, thus not impacting the time the user has to wait for the page to appear.

This naturally requires that the script itself is not needed for the rendering of the page.

For more information, see the post *Coupling async scripts* by Steve Souders (creator of YSlow, but now at Google).

Share Improve this answer Follow

edited Nov 21, 2021 at 13:44

Peter Mortensen
31.2k 21 106 129

answered Jan 12, 2009 at 21:06

stpe
3,591 3 31 38

2

Script blocks DOM load until it's loaded and executed.

If you place scripts at the end of `<body>`, all of the DOM has a chance to load and render (the page will "display" faster). `<script>` will have access to all of those DOM elements.

On the other hand, placing it after the `<body>` start or above will execute the script (where there still aren't any DOM elements).

You are including jQuery which means you can place it wherever you wish and use .ready().

Share Improve this answer Follow

edited Nov 21, 2021 at 14:08

Peter Mortensen
31.2k 21 106 129

answered Jun 29, 2015 at 12:38

Szymon Toda
4,394 11 41 62

2

You can place most of `<script>` references at the end of `<body>`.

But *if* there are active components on your page which are using external scripts, then their dependency (.js files) should come before that (ideally in the *head* tag).

Share Improve this answer Follow

edited Nov 21, 2021 at 14:23

Peter Mortensen
31.2k 21 106 129

answered Jan 27, 2017 at 15:51

Tech AG
64 3

2

The best place to write your JavaScript code is at the end of the document after or right before the `</body>` tag to load the document first and then execute the JavaScript code.

```
<script> ... your code here ... </script>
</body>
```

And if you write in jQuery, the following can be in the head document and it will execute after the document loads:

```
<script>
    $(document).ready(function(){
        // Your code here...
    });
</script>
```

Share Improve this answer Follow

edited Nov 21, 2021 at 14:26

Peter Mortensen
31.2k 21 106 129

answered Oct 28, 2017 at 16:34

nada diaa
139 1 10

It is not a good idea to put the scripts at the end of your body or html code. It is common practice to put this kind of meta information right where it belongs: in the head. – theking2 Apr 28, 2021 at 15:52

1 Re "*at the end of the document after ... the </body> tag*": No, no, no!. If the 'script' tag is after '</body>', HTML validation will result in "Error: Stray start tag script" (check option "source" and click "check" to see the HTML source). If it is before, it validates. – Peter Mortensen Nov 21, 2021 at 16:50

1

- If you still care a lot about support and performance in Internet Explorer before version 10, it's best to *always* make your script tags the last tags of your HTML body. That way, you're certain that the rest of the DOM has been loaded and you won't block and rendering.
- If you don't care too much any more about in Internet Explorer before version 10, you might want to put your scripts in the head of your document and use `defer` to ensure they only run after your DOM has been loaded (`<script type="text/javascript" src="path/to/script1.js" defer></script>`). If you still want your code to work in Internet Explorer before version 10, don't forget to wrap your code in a `window.onload` even, though!

Share Improve this answer Follow

edited Nov 21, 2021 at 14:15

Peter Mortensen
31.2k 21 106 129

answered Jan 20, 2016 at 19:50
user5803163

1 In the accepted answer, this is refered to as the “antiquated recommendation”. If you still mean it, you probably should produce some reference to back it. – dakab Jan 20, 2016 at 20:10

1

I think it depends on the webpage execution.

If the page that you want to display can not displayed properly without loading JavaScript first then you should include the JavaScript file first.

But if you can display/render a webpage without initially download JavaScript file, then you should put JavaScript code at the bottom of the page. Because it will emulate a speedy page load, and from a user's point of view, it would seems like that the page is loading faster.

Share Improve this answer Follow

edited Nov 21, 2021 at 14:21

Peter Mortensen
31.2k 21 106 129

answered Dec 1, 2016 at 7:09

Amit Mhaske
451 5 10

0

Always, we have to put scripts before the closing body tag expect some specific scenario.

For Example :

```
`<html> <body> <script> document.getElementById("demo").innerHTML = "Hello
JavaScript!"; </script> </body> </html>`
```

Share Improve this answer Follow

edited Mar 8, 2022 at 5:36

answered Mar 8, 2022 at 5:24

Patel Charul
197 1 8

Prefer to put it before the `</body>` closing tag.

0

Why? As per the official doc: https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/JavaScript_basics#a_hello_world!_example

> Note: The reason the instructions (above) place the element near the bottom of the HTML file is that the browser reads code in the order it appears in the file.
>
> If the JavaScript loads first and it is supposed to affect the HTML that hasn't loaded yet, there could be problems. Placing JavaScript near the bottom of an HTML page is one way to accommodate this dependency. To learn more about alternative approaches, see Script loading strategies.

Share Improve this answer Follow

answered Jun 7, 2022 at 10:32

Pedro Soares
21 2