|  |
| --- |
| [[https://myetudes.org/etudes-melete-tool/images/printer.png](https://myetudes.org/portal/tool/4c4d3792-8b10-40ce-8016-d7a5ac569a1c/print_module.jsf?printModuleId=1436385319) Send to Printer](https://myetudes.org/portal/tool/4c4d3792-8b10-40ce-8016-d7a5ac569a1c/print_module.jsf?printModuleId=1436385319) | [Close Window](https://myetudes.org/portal/tool/4c4d3792-8b10-40ce-8016-d7a5ac569a1c/print_module.jsf?printModuleId=1436385319) |
|  | |
| 8. JavaScript in your browser  8.1. Javascript in HTML  *Copyrighted Material - subject to fair use exception*  The following sections assume that you are familiar with the basics of  HTML. You don’t have to be an expert, but you should be familiar with the structure of HTML documents and how the tags are used to display various kinds of content in your browser.   If HTML is completely new to you or if you need a refresher, you can visit <http://www.codecademy.com/en/tracks/web>  and go through the HTML basics tutorial.  **The  <script> tag:**  We can embed JavaScript code in HTML documents by using the <script> tag.  Although we can write JavaScript code inline between a pair of <script> and </script> tags, **the recommended way is to include the code in an external** **file,**saved with the .js extension, and point to it using the **src attribute** of the <script> tag.  This is part of the **unobtrusive JavaScript** **approach to separate the programming logic from the HTML content.**  < script src ="../scripts/pause.js" > </ script >  **A note about file names and path names:**  The **external file name includes the relative URL if the file is on another server, or the relative path name if the file is in the local directory.**  In the following examples, I’ll be saving html files in the folder CS22A/html/ and the JavaScript files in the folder CS22A/scripts/.  So from my html file location, to get to a script saved in the directory CS22A/scripts/, I’ll have to go up to the parent directory CS22A denoted by ../  and then to scripts/pause.js.  The relative path name is then "../scripts/pause.js".  That is relative to the location of the html file.  **Timeline:**  The browser reads each line of HTML code from top to bottom, and processes and displays it.  Consider the following html file.  hello1.html:  <!DOCTYPE html>  <html>  <head>      <meta charset="utf-8">      <title>JavaScript for Programmers</title>  </head>  <body>      <h1>Hello World!</h1>      <script src="../scripts/pause.js"></script>      <p>Hello again!</p>  </body>  </html>  When a <script> element is encountered, the external file is usually read (downloaded) and the code is executed by the JavaScript interpreter, and then the parsing and rendering of the HTML continues until the end of the document is reached.  This means that the parts of the document that come after the script (Hello again!) will not appear in the browser until the script has been downloaded and executed.  To see how that works, we can create the following script that we'll call pause.js.  We’ll cover the alert() method in the next section but for now, we’ll use it to make the execution of the script pause while waiting on the user.  pause.js contains just the following line:  alert ("Just introducing a pause here");  Create and save the html file hello1.html shown above.  Then create and save the JavaScript file pause.js as shown above.  You may use Komodo Edit or Scratchpad or any other text editor to do that. Now open hello1.html with Firefox.  You can do that from within Komodo Edit as shown in the screencast here:        You can also open hello1.html by right clicking on the file and selecting  Open with Firefox as shown below.  You’ll get the screen shown below.  Hello again does not appear on the page until you click on OK.  We can change the above default behavior by specifying additional attributes for the <script> tag.  **The defer attribute causes the browser to defer execution of the script until after the document has been loaded and parsed.**  hello2.html:    <!DOCTYPE html>  <html>  <head>      <meta charset="utf-8">      <title>JavaScript for Programmers</title>  </head>  <body>      <h1>Hello World!</h1>      <script **defer** src="../scripts/pause.js"></script>      <p>Hello again!</p>  </body>  </html>  Note if you have several deferred scripts, they will run in the order in which they appear in the document.  You can open hello2.html in your browser window now and see the changes in the loading timeline.  **The async attribute causes the browser to run the script as soon as possible but not to block document parsing while the script is being downloaded.** The async attribute is new in HTML5.  hello3.html:  <!DOCTYPE html>  <html>  <head>      <meta charset="utf-8">      <title>JavaScript for Programmers</title>  </head>  <body>      <h1>Hello World!</h1>      <script **async** src="../scripts/pause.js"></script>      <p>Hello again!</p>  </body>  </html>  Note that if you have several asynchronous scripts, they will run as they load, which means that they may execute out of order.  You can open hello3.html in your browser window now and see that for this example, there is no visible difference in the loading timeline between async scripts and deferred scripts.  Using the defer or async tags will usually result in a better user experience since the page is visible and available to the user faster (even if the corresponding script has not loaded or executed yet).  **Placement of the <script> tag:**  Where should the <script> tag be placed in the html source document?  In general, the <script> tag can go either in the <head> or the <body> elements of the source document.  If we place it in the <head> element, **we should make sure we use the defer or async attribute so that the loading of the web page is not delayed.**  <!DOCTYPE html>  <html>  <head>      <meta charset="utf-8">      <title>JavaScript for Programmers</title>      <script **defer** src="../scripts/pause.js"></script>  </head>  <body>      <h1>Hello World!</h1>      <p>Hello again!</p>  </body>  </html>  Another approach is to put the <script> tag at the very end of the <body> element, right before </body> as follows:  <!DOCTYPE html>  <html>  <head>      <meta charset="utf-8">      <title>JavaScript for Programmers</title>  </head>  <body>      <h1>Hello World!</h1>      <p>Hello again!</p>      <script src="../scripts/pause.js"></script>  </body>  </html>  8.2. The Window Object  *Copyrighted Material - subject to fair use exception*  The window object represents a web browser window or frame, and we can refer to it with the identifier window. It is the global object in client-side JavaScript.  This means that it is at the top of the scope chain and that **its properties and methods are global variables and global functions**.   To refer to them in our code, we can just use the property name without prefixing it with window.  So for example, window.alert() is the same as alert().  We’ll cover some of these properties and methods next.  8.3. Dialog Boxes  *Copyrighted Material - subject to fair use exception*  The window object provides some methods that display simple dialog boxes.  The **alert()** method displays a message and waits for the user to dismiss the dialog. It should be used for messages that do not require any response from the user, other than the acknowledgement of the message.  The **confirm()** method displays a message, waits for the user to click an OK or Cancel button and returns true (if the user clicks OK) or false (if the user clicks Cancel).  The **prompt()** method displays a message, waits for the user to enter a string, and returns that string.  **Dialog boxes are modal windows** - they block program execution and prevent the user from accessing the rest of the interface until the dialog box is closed. For this reason, we**should only use them sparingly.**  Example:  var name = prompt("Please enter your name");  var answer = confirm("Would you like to continue?");  if (answer) {      alert("Hello " + name);  } else {      alert ("Bye");  }  8.4. Timers  *Copyrighted Material - subject to fair use exception*  Timers are important global functions of client-side JavaScript, and are therefore defined as methods of the window object even though they don’t really have anything to do with the window.  The **setTimeout**() method calls a function to run after a specified delay (in milliseconds).  It returns a value that can be passed to**clearTimeout()** to cancel the setTimeout().  The **setInterval()** method calls a function repeatedly, with a fixed time delay between each call.  It returns a value that can be passed to **clearInterval()** to cancel any future invocations of the scheduled function.  The following code sets up a 60 seconds timer.  It issues the alert **once**, 60 seconds **later**.  function timeIsUp(){      alert("Time is up!");  }  setTimeout(timeIsUp, 60000);  The following code calls a function, newDay, **repeatedly, every** 24 hours:  function newDay(){      console.log("Good morning!");  }  setInterval(newDay, 86400000);  It is also common to specify an anonymous function expression as the argument to setTimeout or setInterval:  setTimeout(function() {              alert('Time is up!');              },              60000);     setInterval(function() {              console.log("Good morning!");              },              86400000);    8.5. Other Window Properties  *Copyrighted Material - subject to fair use exception*  The **location property** of the window object is itself an object that represents the current URL.  To see what the location property looks like for a given web page, go to www.Foothill.edu, activate Firebug, then at the prompt in the Firebug console, type: location (or window.location which is the same here.)  You’ll get the following:  >>> location  http://www.foothill.edu/index.php { constructor={...},             href="http://www.foothill.edu/index.php", protocol="http:", more...}  The location object has properties that describe the individual parts of the URL, such as protocol, pathname and hostname:  >>>location.protocol  "http:"  >>>location.pathname  "/index.php"  >>>location.hostname  "www.foothill.edu"  Similarly, the **history property** of the window object represents the browsing history.  >>>history  14 history entries  We'll cover the history object in more details in an upcoming module.  The window object has **open()**and **close()** methods that allow you to open a new window and close the current window.  Finally, one of the most important properties of the window object is **document**: it is an object that represents the content displayed in the window. We’ll cover the document object in details in the next module.  8.6. Debugging with Firebug  *Copyrighted Material - subject to fair use exception*  The Firebug debugger allows us to debug JavaScript code that is linked to a web page.  Let’s see how to do that with a simple html source document, helloworld.html.  helloworld.html:  <!DOCTYPE html>  <html>  <head>      <meta charset="utf-8">      <title>JavaScript for Programmers</title>  </head>  <body>      <h1>Hello World!</h1>      <script async src="../scripts/helloworld.js"></script>  </body>  </html>  Our source document points to a script file that has not been defined yet.  Let’s see what happens when we open helloworld.html in Firefox.  The html document is correctly rendered by the browser but how do we know if the associated script has been loaded and executed successfully?  We can open Firebug and click on the HTML panel.  . When we click on all the plus signs (or arrows), we get the following:  Notice the ‘undefined’ under the <script> tag.  That indicates that the script associated with the html file was not loaded successfully.  In this case, it’s because it does not exist in the scripts directory.  Let’s create a simple JavaScript file helloword.js and save it in the scripts directory.  helloworld.js  function hello() {      alert('Hello');  }  setInterval(hello, 120000);  If we refresh our page, we can see that the page looks initially the same.  However the HTML panel now shows the content of helloword.js under the <script> tag.  This way we can make sure we are invoking the right script (or the correct version) from the HTML page.    We can also click on the Script tab, and enable it:    From the Script panel, we can add breakpoints in the script by clicking on a line number, or add a watch expression from the side panel to check on the value of a variable and so on. |  |