In this lecture we will use the “Project1ThirdPage.html” web in the “Code Examples” folder for “Lecture 7.” This file is a copy of the web page we finished in the “Lecture 6,” class. You will find all of the “console.log()” statements commented out to reduce clutter in the “Console.”

In the “CodeExamples” folder you will also find:

* BasicHTMLPage.html – our HTML web page template.
* ChoiceData.js – the JavaScript object containing the array used to create our “select lists,” this has a new “key / value” pair added to each JavaScript object in the arrays – we will discuss.
* IterateSelects.html – used for some miscellaneous operations on NodeLists.
* Project1ThirdPage.html – html file we will use for this lecture’s code.
* Project1ThridPageFinal.html – this is where we left off in “Lecture 6.” “Project1ThirdPage.html” uses this file as its underpinnings but trashes it a lot to do the coding for this lecture.
* SandBox2.html – a web page I used to create “model” code for our in-class work for “Lecture 7.” This code will get used in “Project1ThirdPage.html.”
* ***LetsBuildSelectLists.html – this is solid starting code for “Project 1.” We have discussed most of it, but you will still have to do the hardest part yourself. We will talk about what exactly that hardest part is.***

Note on our script tag for “ChoiceData.js”  
In our code we have been using the following script tag to include the “ChoiceData.js” file in our web page:



It occurred to me that I usually do not use the “type” attribute in my own work while at the same time I see examples on the Internet where the “type” attribute is used. I decided to find out if the “type” attribute is really required, TLDR – it is not. More specifically, for HTML5 compliant browsers this indicates that the script is JavaScript. The HTML5 specification urges authors to omit the attribute rather than provide a redundant MIME type. In other words, when the browser sees the <script> tag it assumes that what is being referenced is JavaScript, so its not needed.

From now on we will just use:



In this lecture we will explore more methods on “child nodes” and how to use some of those methods for our “Project 1” work.

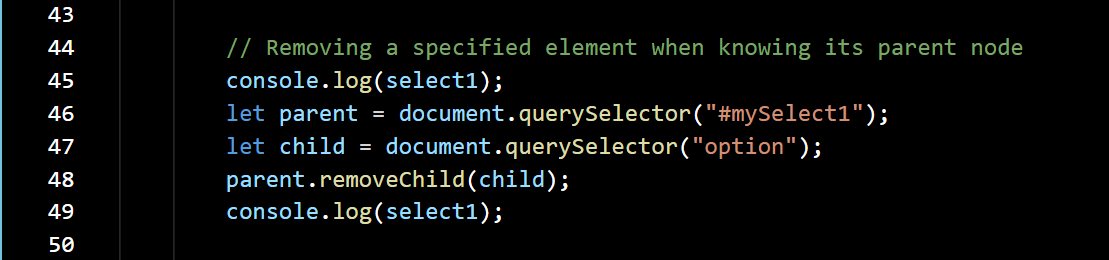
In “Lecture 6,” we used the “*element*.before([*object element*)]” to insert an <h2> heading before our “select list.” This gave us a look at “parent nodes” and “child nodes.” The “*element*.before()” and “*element*.after()” methods will cover most of the scenarios you will encounter where you need to create and place a new HTML element “within” the already existing node structure of an element of the “DOM” as opposed to just “appending” it to an already existing element in the “DOM” in which case it is placed following the last node of the element you are appending to.

On the other hand, there are a several methods of removing node elements and which one you use will depend upon your use case. We will explore the following scenarios:

* Removing a specified element when knowing its parent node.
* Removing a specified element without having to specify its parent node.
* Removing all children from an element
* Removing an element with the “*element*.remove()” method.

From the “CodeExamples” folder open “Project1ThirdPage.html” in VS Code. Let us start working from line 43, which is the point at which we have constructed of “select list” and appended it to div1. If you look through the file you will see that I have commented out all the console.log() statements to keep the “Console” clean. Additionally, I have commented out the code to add a fourth option to the “select list” which we used to examine “live” and “static” NodeLists.

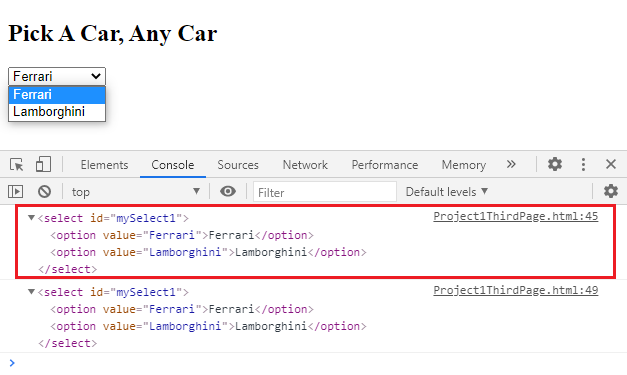
At line 44, insert the following code:



On line 45, we are writing out the “select1” select list for reference, this is our “starting point.” We are then setting the variable “parent” to the “Object Element” for “select1” using “querySelector(#mySelect1).” Similarly, we are setting the variable “child” to the first <option> element in the “DOM” using “querySelector(“option”);” we already know that the first <option> element will be a child of “select1.”

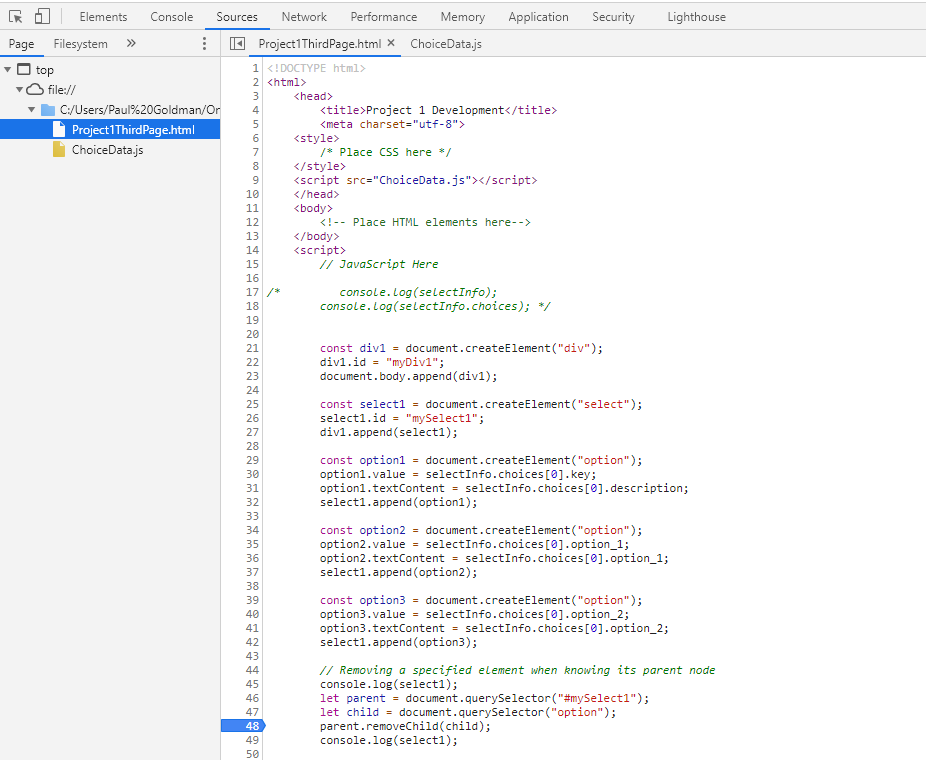
On line 43, we use the “removeChild()” method to remove the first “option” element from “select1” and finish by once more writing out “select1” to the “Console.”

Save the changes and then run the file in your browser, make sure you have “Developer Tools” open and the “Console” tab selected. You will see the following:



If you examine the “console.log()” output on line 45, something looks “not quite right.” On line 45, of our program we executed the “console.log(select1)” statement **before** the *“parent.removeChild(child)”* statement on line 48, yet what we’re seeing is “select1” **after** the execution of the *“parent.removeChild(child)”* statement on line 48? The “console.log()” output on line 49, is what we would expect to see.

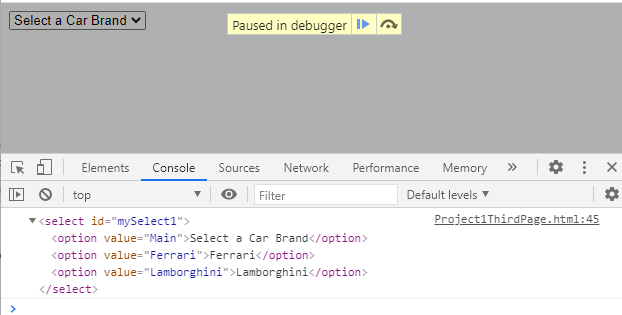
Let us go to the “Sources” tab in the “Developers Tools” and if not already selected click on the “Project1ThirdPage.html” folder in the left-hand column. Place a “breakpoint” on line 48, the *“parent.removeChild(child);”* statement by clicking your mouse on it. Your “Sources” tab should look as follows; this is a partial screen shot.



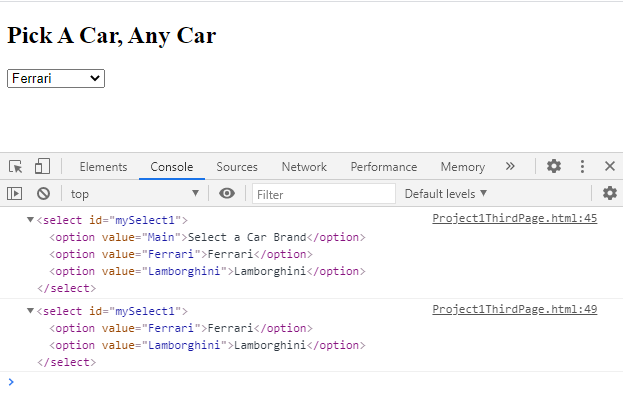
Now refresh the page to rerun the code.

The first thing that you will see is that line 48, is highlighted in blue. This is telling us that our “breakpoint” has been hit.

Now click on the “Console” tab and click on the right facing arrow to the left of the <select> element. You will see that “console.log()” has written out “select1” as it should appear before deleting the first child element.



Now click on the “run” button in the “Paused in debugger” window at the top of the page. When the page completes you will also have the output from the “console.log()” statement on line 49, of the web page. Expand that as well. The “Console” will now show:



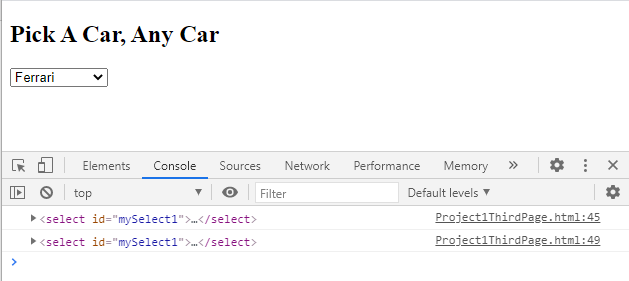
The second “console.log()” shows “select1” as we expect it to appear, the first <option> element has been removed. Additionally, if you look at the “select list” on the page, you will notice that the first choice is no longer “Select a Car Brand” but “Ferrari.”

The point of this exercise is to demonstrate something I have mentioned in class about the “reliability” of using “console.log()” for debugging. *Regardless of when you execute “console.log()” in your code the output you will see is dependent on the state of the element you are “logging” at the time you inspect the “console.log() output in the “Console,” e.g. expand a “select list” by clicking on the right facing arrow to its left.*

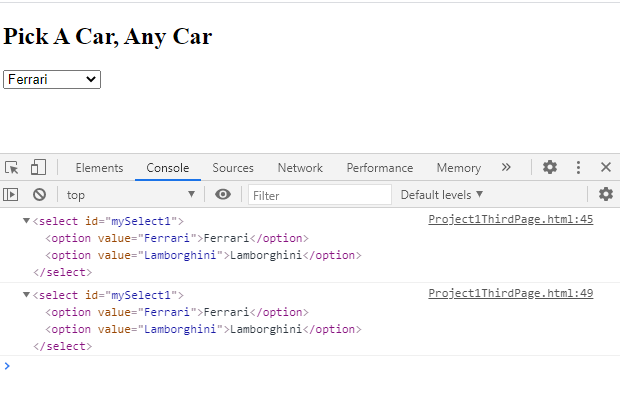
In the first “console.log” for the web page, shown on page 2, of this document, the result of the first “console.log()” reflected the sate of ‘select1’ at the time that we “expanded” it.

When we put in the “breakpoint” and it was “hit” by the code, expanding the “mySelect1” when the code was paused at the “breakpoint” showed the state of the “select list” at that time.

As a final experiment reload the page and let it hit the “breakpoint.” When it hits the breakpoint open the “Console” tab again but do not expand “mySelect1,” just click on the “run” button in the “Paused in debugger” window at the top of the page and let the page run to completion. The page will appear as follows:



Now expand both “mySelect1” console outputs. You will be presented with the following:

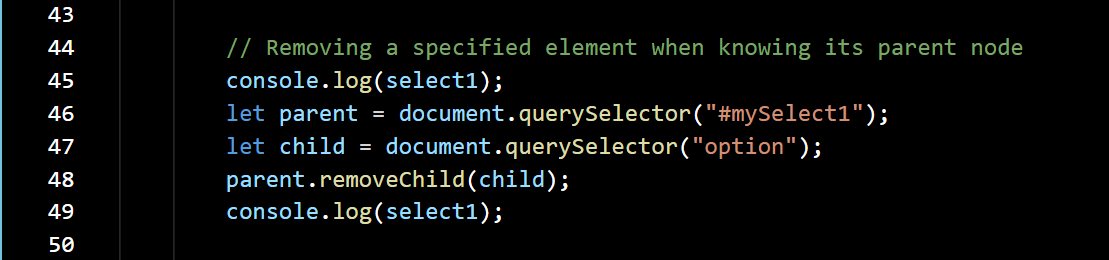


Now, although the first “mySelect1” output was produced before the breakpoint was hit, and thus before the *“parent.removeChild(child)”* statement ran, it still is showing “mySelect1” after the *“.removeChild().”*

This is final confirmation of the fact that what you will see in the output of a “console.log()” is the state of the element you are interested in at the time you actually inspect, not the time that the “console.log()” ran. Keep this in mind when using “console.log()” for debugging. If the “console.log()” output is not what you were expecting, try inserting a breakpoint right after it and inspect the output at that point.

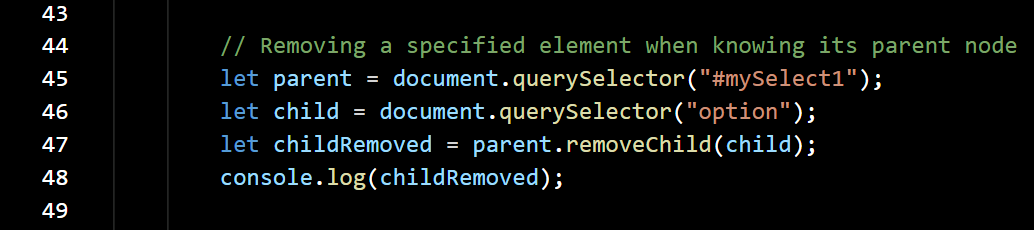
Now let us get back to removing nodes.

Let us take another look at the code we have been running.

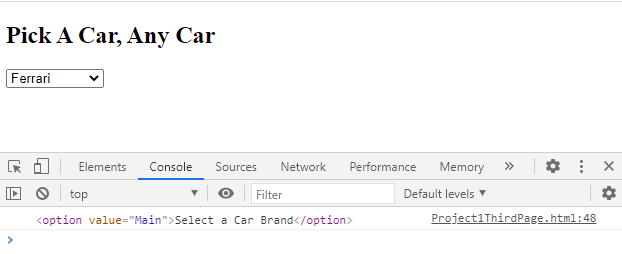


The *“parent.removeChild(child)”* method on line 43, has some interesting properties. While the “child” that is removed is no longer in the “DOM” it is still maintained in memory. In fact, if we wish to “save” the removed “child” for later use, we can create a variable to “capture” it.

Modify the above code as shown below.



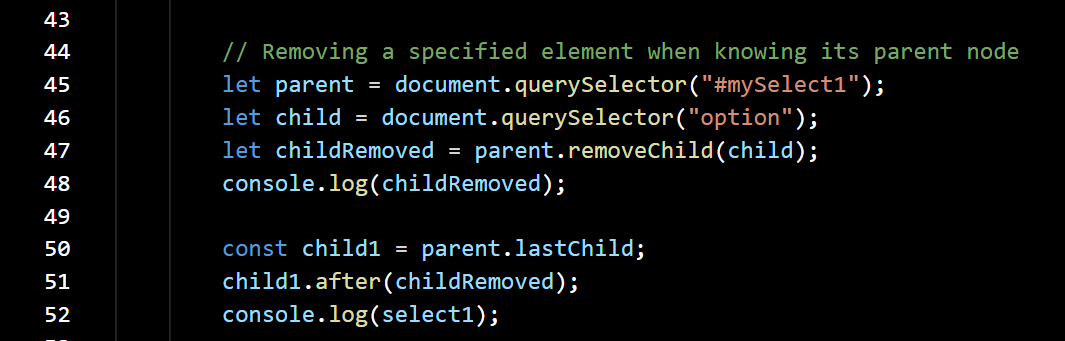
I have removed the two “console.log(select1)” statements since we now know that unless we use a breakpoint and expand things at the correct time their outputs will be the same. On line 48, we have set the variable “childRemoved” to the “Object Element” for “child,” which is defined on line 47. We have then added a “console.log(childRemoved)” write out the removed element. Save the code and run it in the browser. We will see the following:

You can see that “childRemoved” now contains the <option> element removed from the “DOM.” We could reuse it later if we so chose.

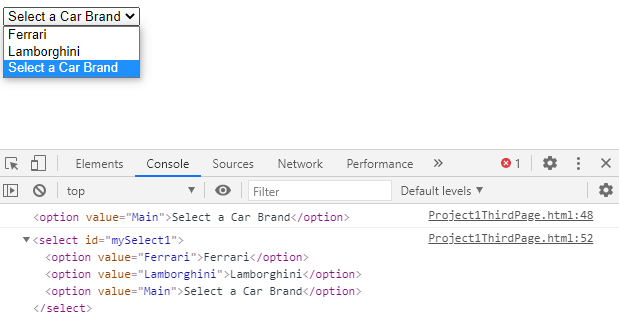
If we had not saved the removed <option> node in a variable it would have remained in memory until “garbage collection.”

A quick experiment with the code we just completed.

Add lines 50, through 52, to the code as shown below.

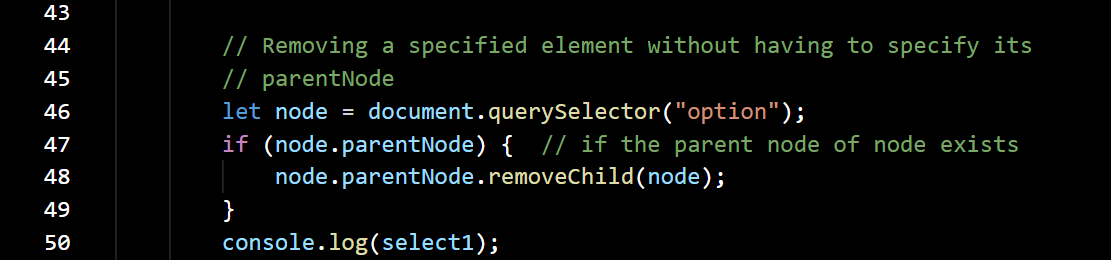


Save and run it in the browser. The output will appear as follows.



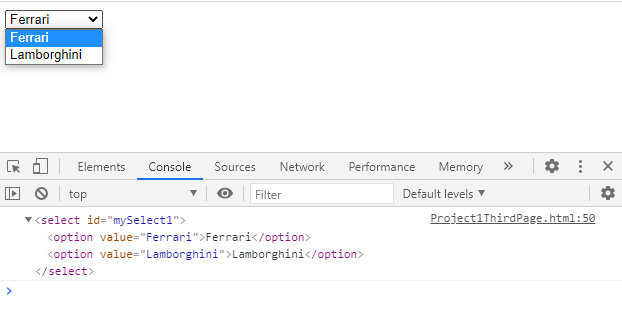
All we have done here is taken the <option> element removed on line 47 and placed back in “select1” as the last <option> element.

Now replace the code on lines 44, through 52, with the following.



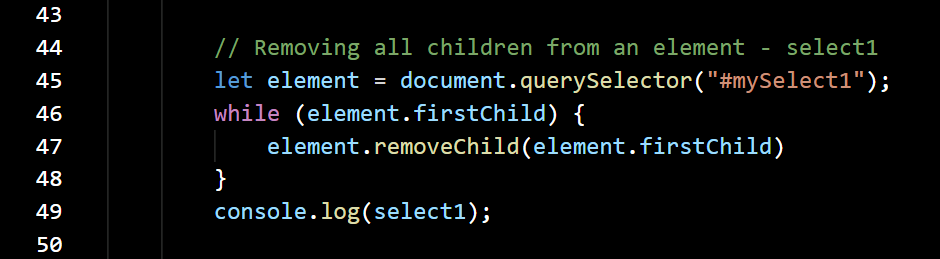
Here we are going to remove the first <option> element from the “DOM” without specifying the parent node. On line 46, we find the first <option> element using “querySelector()” and set the variable “node” to its “Object Element.” On line 47, we check to see if it has a “parentNode” and if it does, we remove it from the “parentNode.” Knowing that it has a “parentNode” allows us to identify the “parentNode” by the expression “node.ParentNode.”

Save the changes and run it in the browser. We see the results in the “Console.”



The first <option> element has been removed.

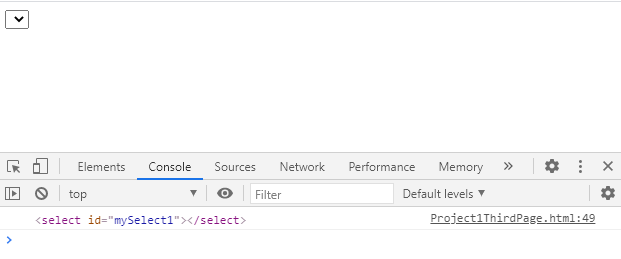
Now replace the code on lines 44, through 50, with the following.



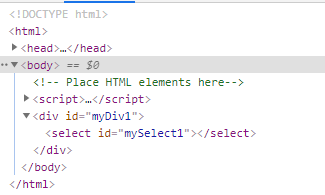
Here we are using “querySelector()” find our “select list” and setting the variable “element” to its “Object Element.” We then use a “while” loop to check for the existence of a “firstChild” and if one exists, we remove it. The loop will terminate when the test for “element.firstChild” returns null. Simple method to implement.

Save and run the code in your browser.

In the “Console” tab all that we see for “select1” is the <select> tag itself.



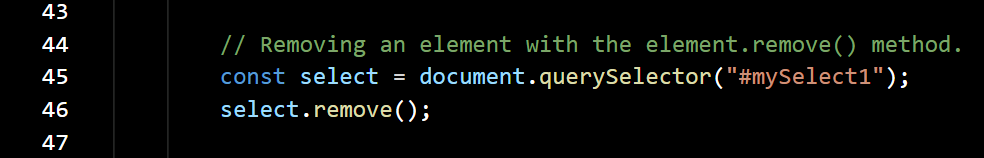
If we examine the “Elements” tab:



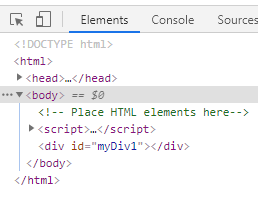
we see is the <div> which contains the <select> element. As an experiment in the above code on line 45, change “#mySelect1” to “#myDiv1” and run the code. Then look at the “Elements” tab again.

As we saw in “Lecture 6,” to create an “Array” from an array like “NodeList” we can use the “traditional” *Array.prototype.slice.call([node list name])* or the “modern” syntactic sugar *Array.from([node list name]).*

We have a similar “syntactic sugar” for removing node elements. Change the code on lines 44, through 49, with the following.

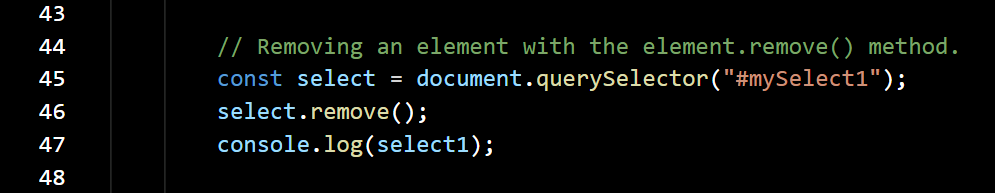


That is about as simple as it gets. No parent nodes, no iterating through children nodes – just “point and shoot.” Save the code and run it in the browser, then open the “Elements” tab and inspect the results.

With one “querySelector()” and one “node.remove()” we have removed both the <select> and all its children, the <option> tags. From the “DOM.”

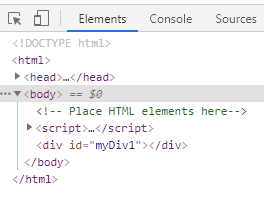
It hides all the methods we used above, but it does the job.

Just one more thing to try with removing nodes. Let us just add a *console.log(select1)* to the above code:

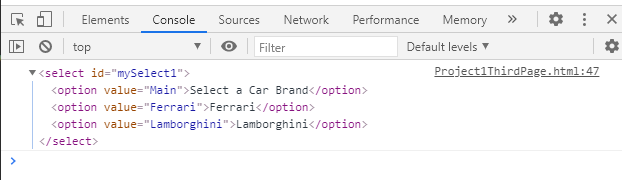


Run it in the browser:

Inspecting the “Elements” tab shows “select1” has been removed from the “DOM.”



But checking the “Console” tab shows that after “select1” is removed “console.log()” can still write it out, it’s in memory but not part of the “DOM”:

Notice the “console.log()” was run on line 47, while the “select.remove()” was run on line 46.

If we try “const mySelect = document.querySelector("#mySelect1"),” we will get “null” showing the “select1” is no longer in the “DOM.”

For the remainder of the lecture refer to “LetsBuildSelectLists.html.”