Kurt Leadley

Web231

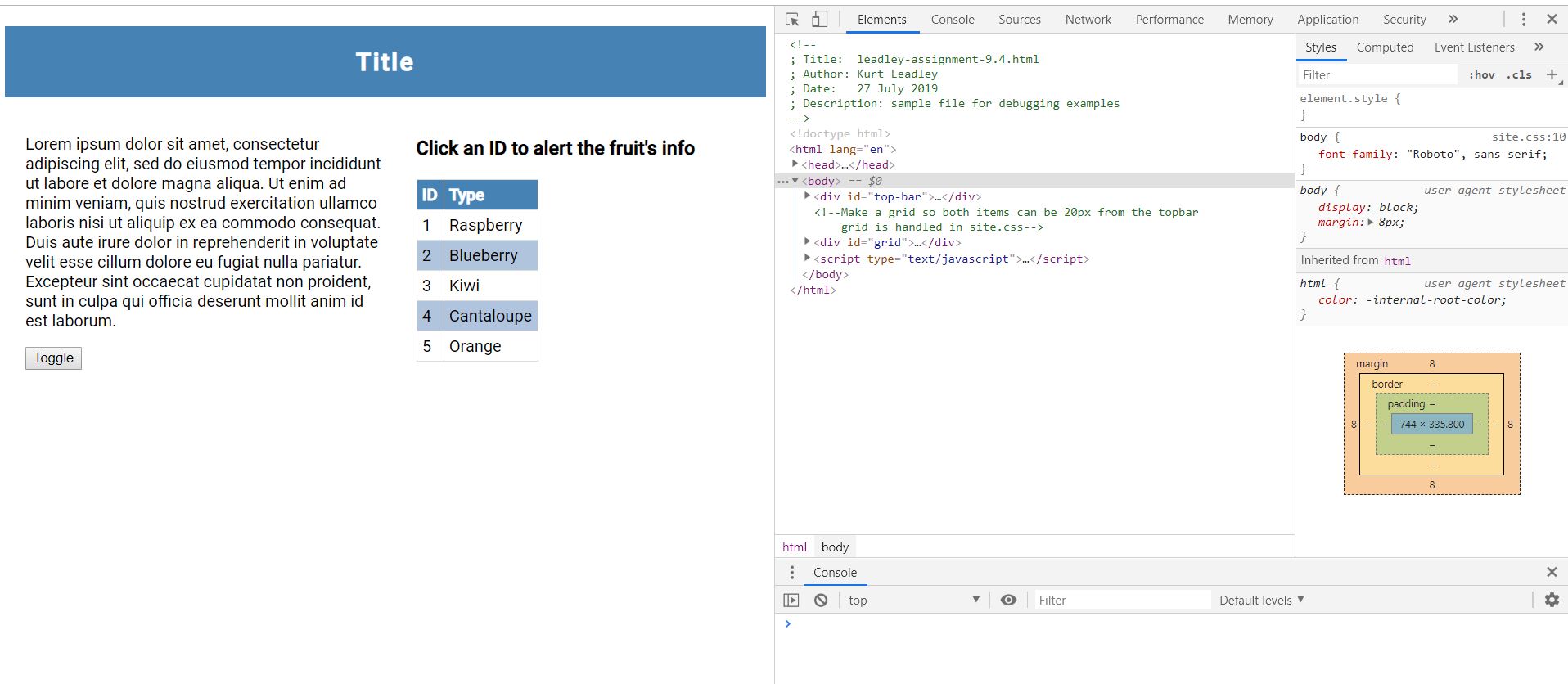
Assignment 9.4

July, 27th 2019

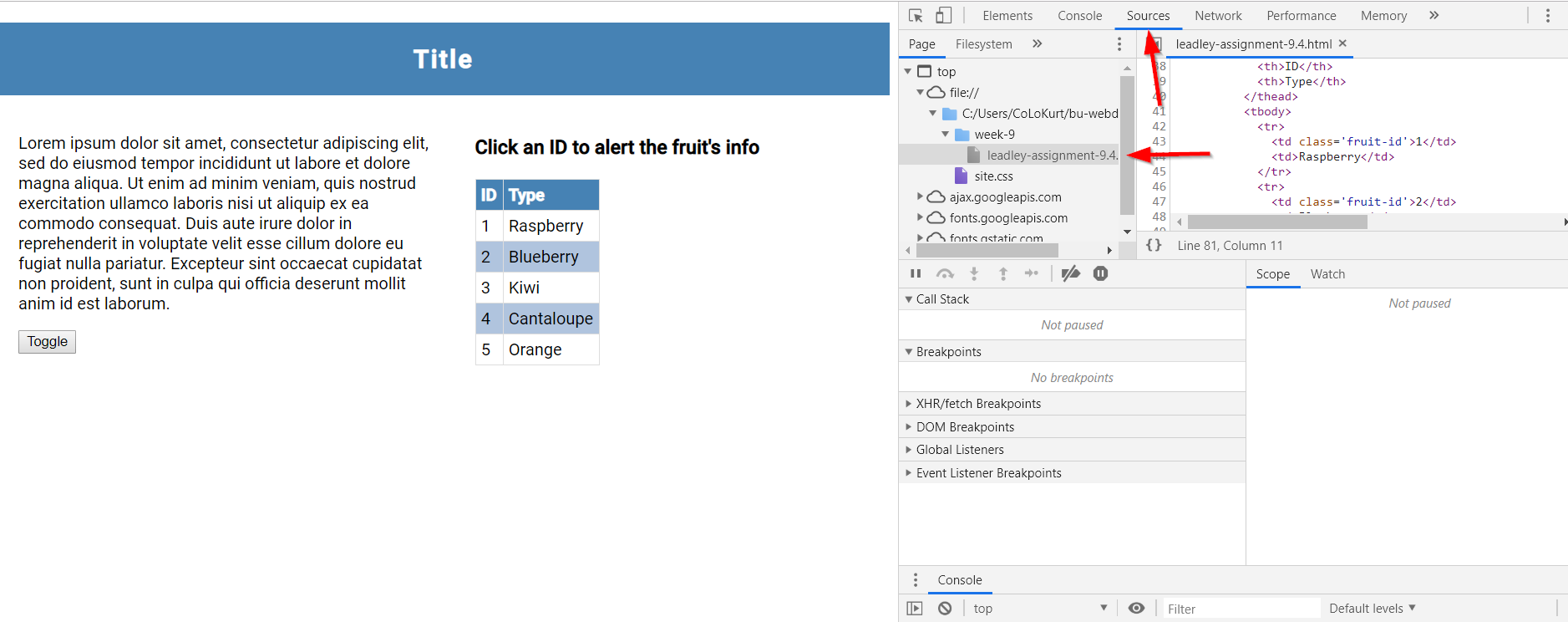
Client Side Debugging

## Break points and stepping through JavaScript code

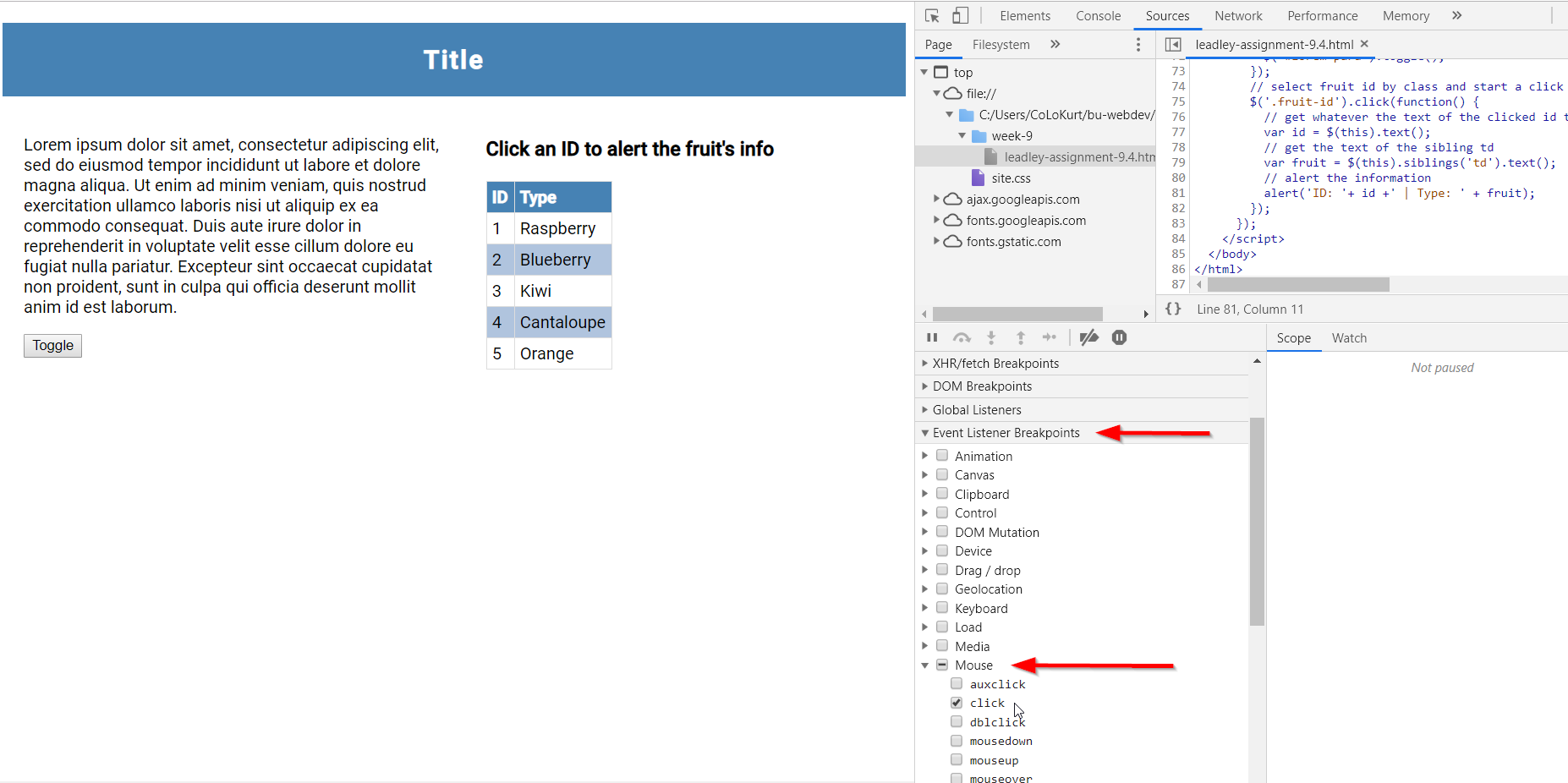
* Open up an HTML file that includes JavaScript code in Google Chrome
* Press F12



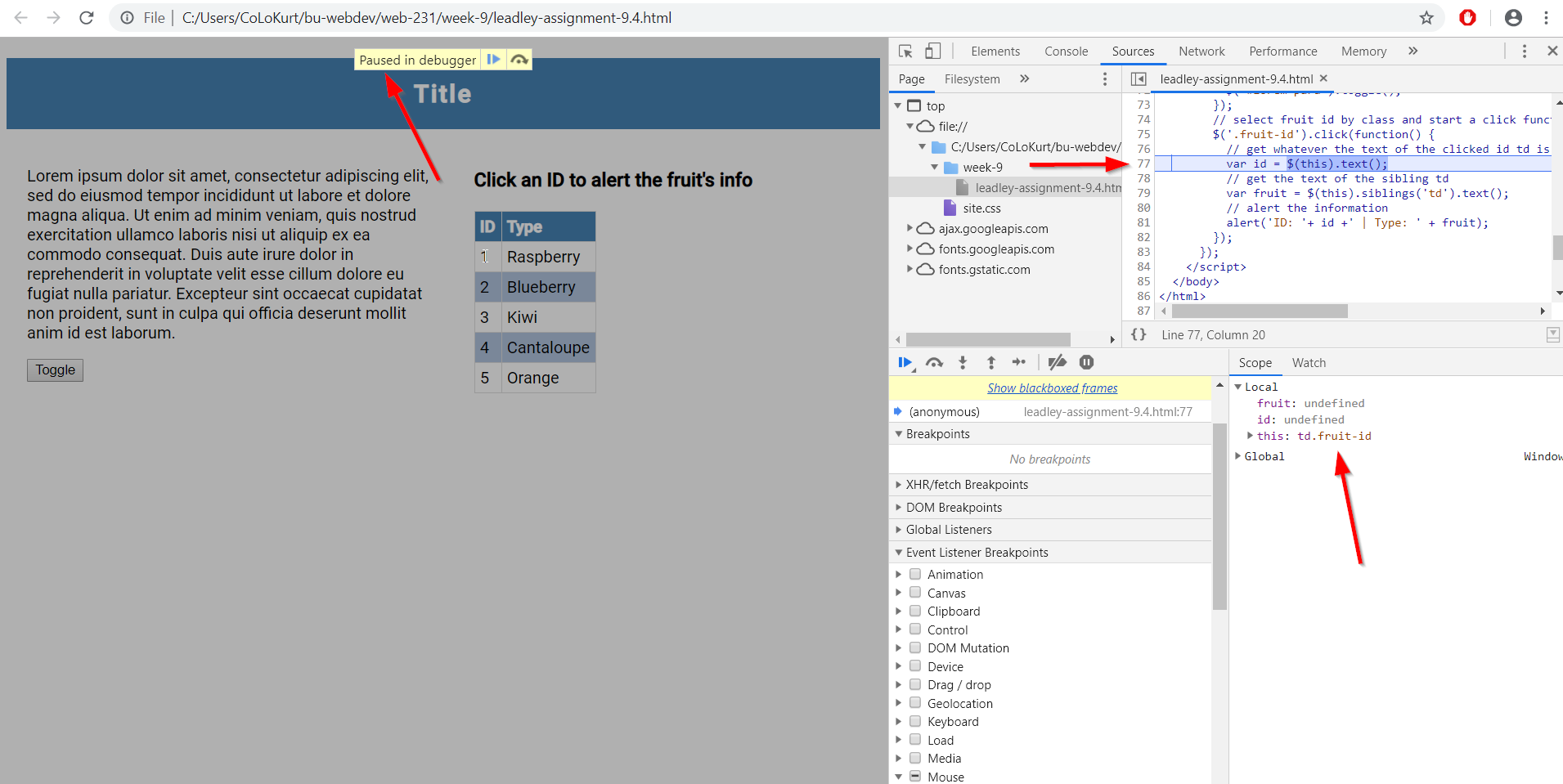
* Click the “Sources” tab
* Four window pane sections should appear in the debugging area. In the upper left pane, choose your HTML file. The file’s code should appear in the upper right pane



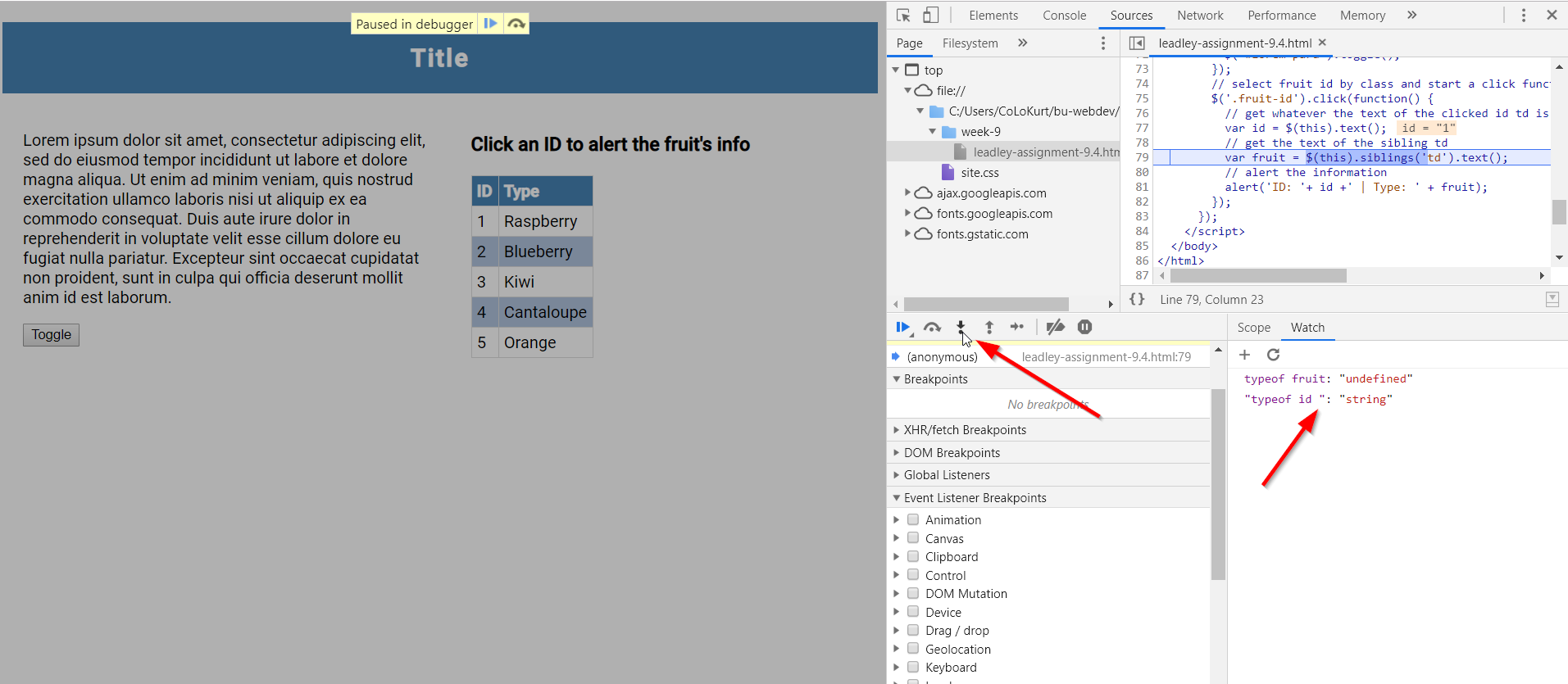
* In this particular example, no meaningful code is executed without a mouse click event occurring. So we will set up a break point based on mouse clicking
* In the lower left pane, we can choose our type/s of breakpoints. Drop down the “Event Listener Breakpoints”
* Select “Mouse” and the check the “click” option to set up the event listener breakpoint



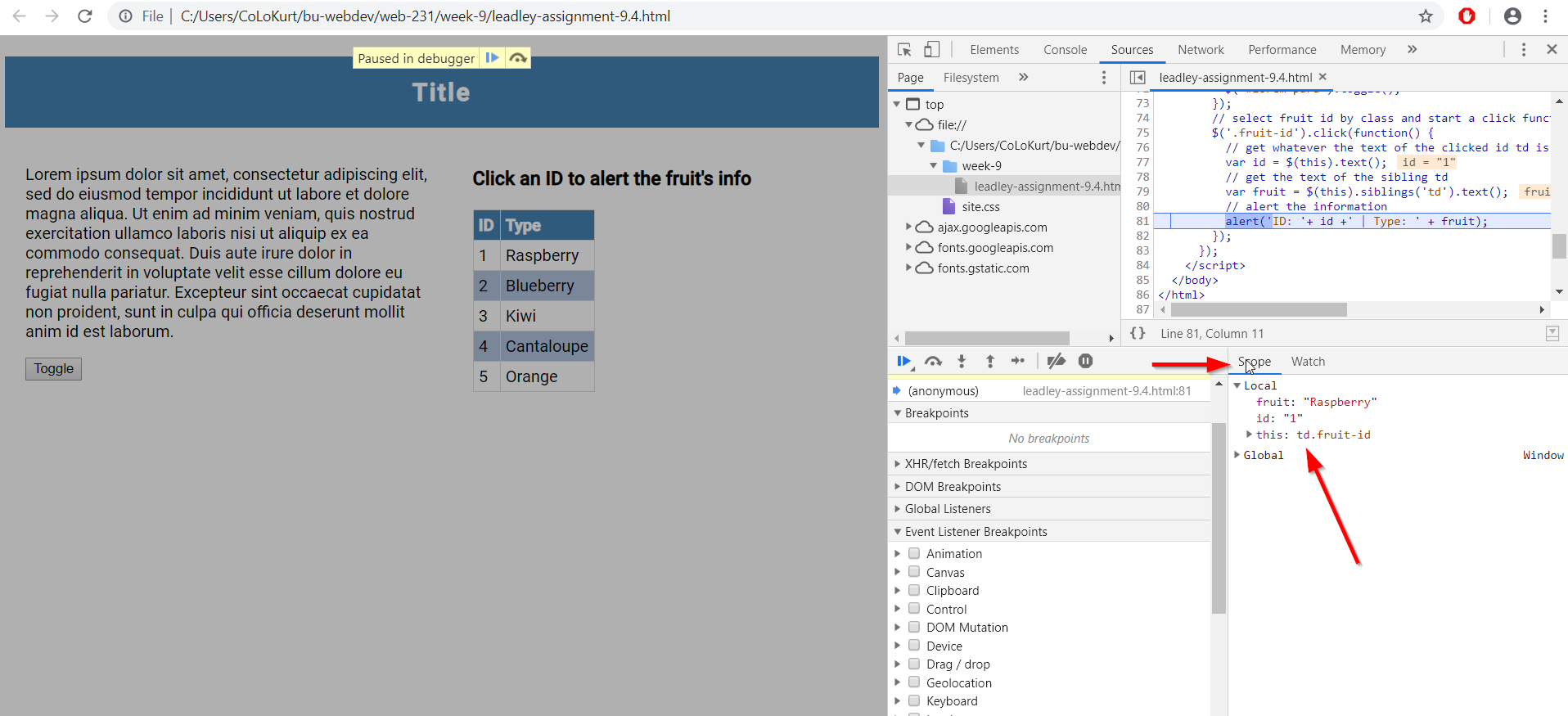
* We are now ready to test out our breakpoint and step through some code
* Hover over the “1” in the “ID” column of the table and click with your mouse
* You will see that the script ran up to line 77 of the code in the upper right pane. This is because the code stops on the first line of code on the *$(.fruit-id).click* function.
* In the bottom right pane, you can see the scope of the local and global variables. Right now both of our declared variables are still undefined because we haven’t executed line 77 of the code or beyond that line yet. We will step through the functions code line by line in a moment.



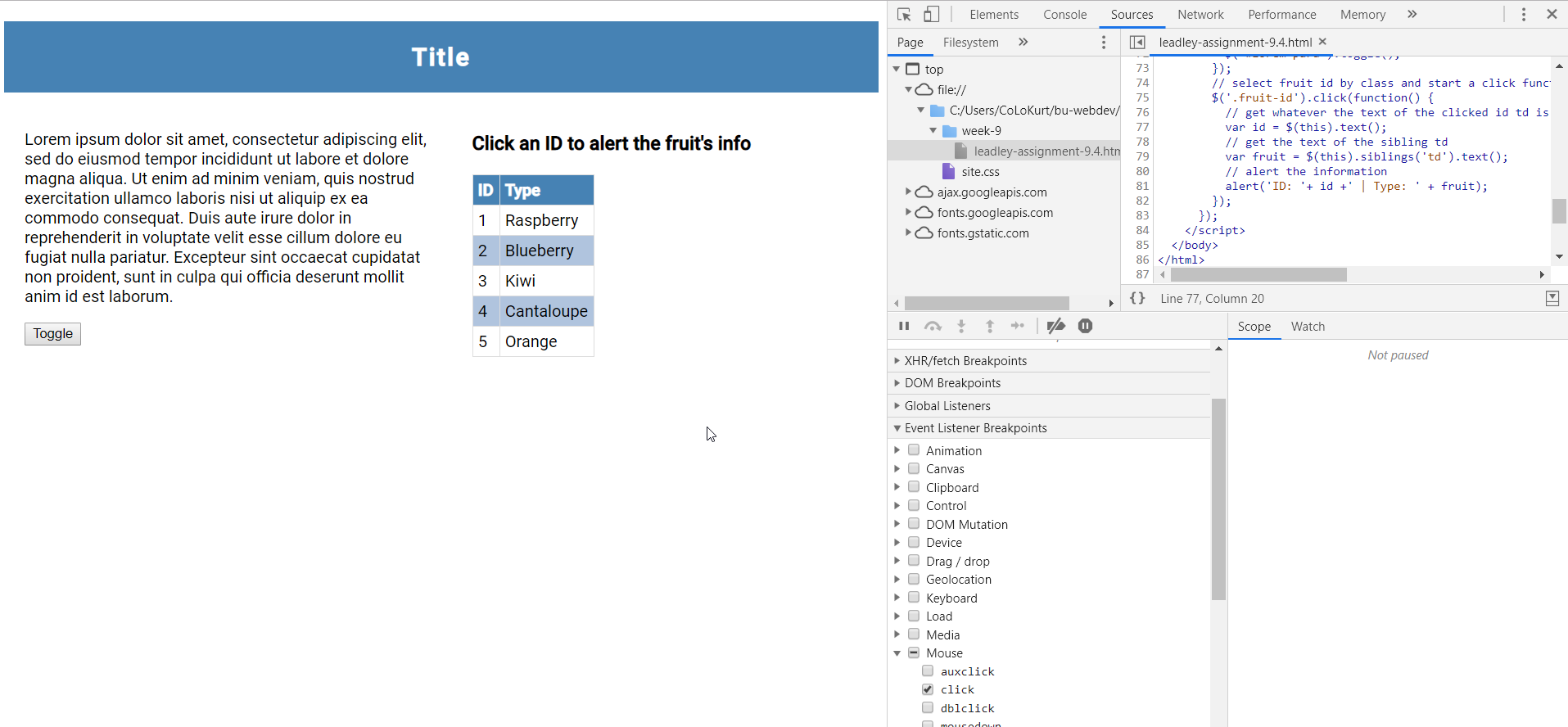
* Before we step through the functions code, we are going to set up a variable watch to find the type of the variable
* Click on the “Watch” tab in the lower right pane.
* Click the “+” symbol to add an expression
* Add *typeof fruit*
* Add *typeof id*
* We are now ready to step through the code
* In the bottom left pane, there are 7 icons in a row near the top of the pane. These are different actions that can be chosen to dictate how the script will continue to execute.
* Choose the down arrow that reads (on hover) “Step into next function call”



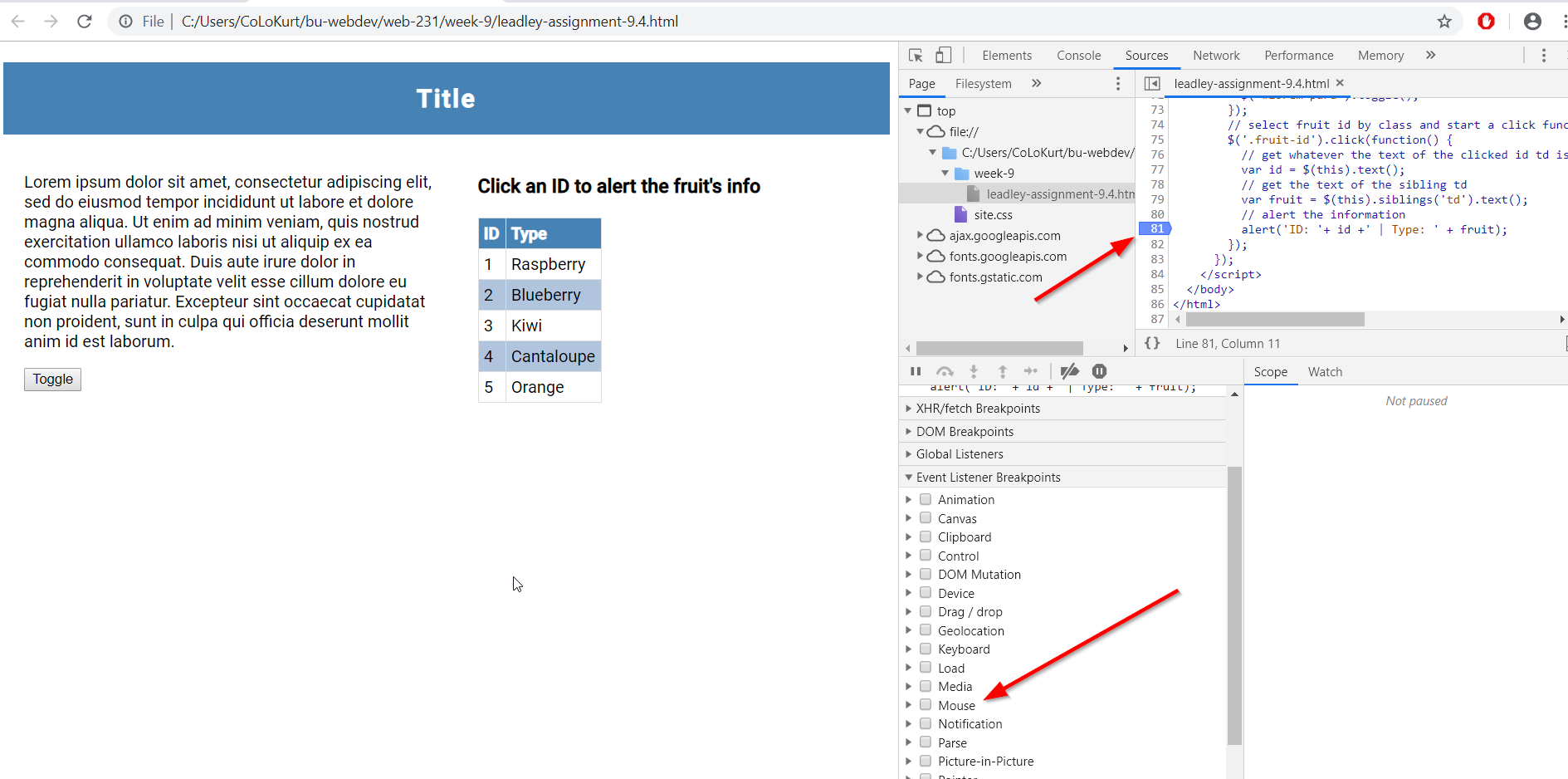
* You can see in the “Watch” tab that typeof id is now defined and fruit is still undefined. This is because the code is now stopped on line 79 of the code. This was the next line of code in the function call.
* Press the down arrow again. You will now see that fruit now also has a typeof. Click on the “Scope” tab in the lower right pane and you can now see the values of the fruit and id variables.



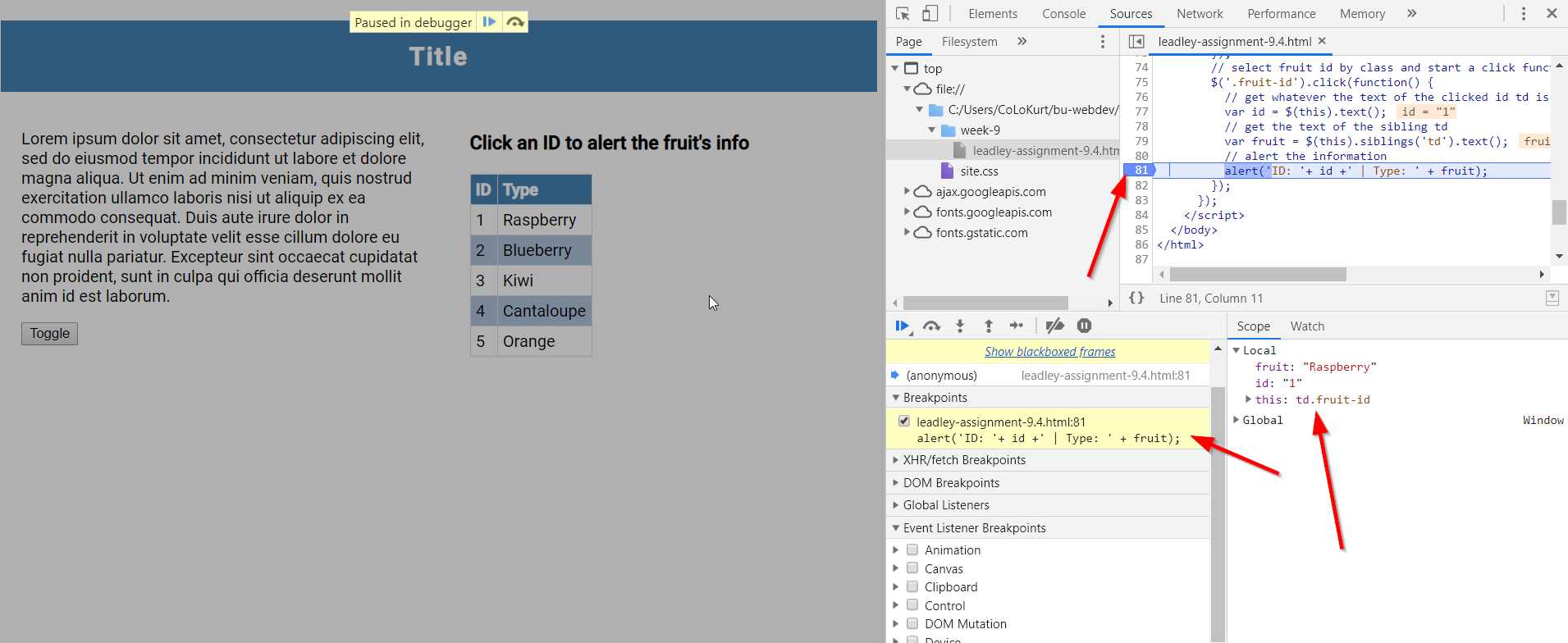
* In the bottom left pane, choose the left most arrow icon (the one that looks like a play button). This is the “Resume script execution” action. It will continue the code until another breakpoint is reached, or will complete execution if no more breakpoints are reached.



* If you are pretty certain where a problems exist in your code, you can set a breakpoint on particular lines of code, instead of stepping through line by line.
* To do this, in the upper right hand pane, click on the left of line number 81. The line should highlight blue.
* Also, remove the mouse click event listener in the bottom left pane, it is no longer needed



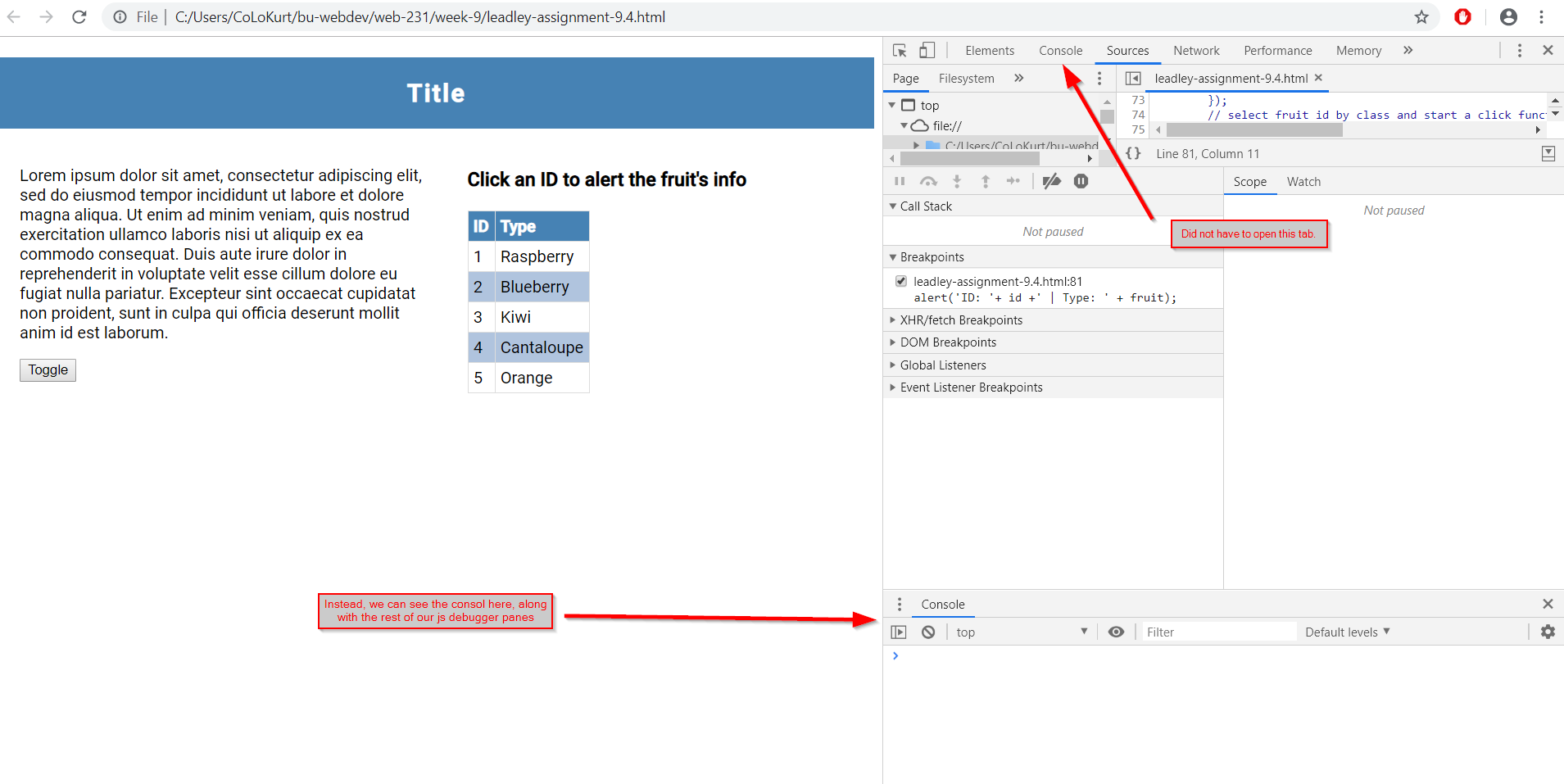
* Click on “1” in the “Id” column of the table again
* You can see the variables values rather quickly in this fashion.



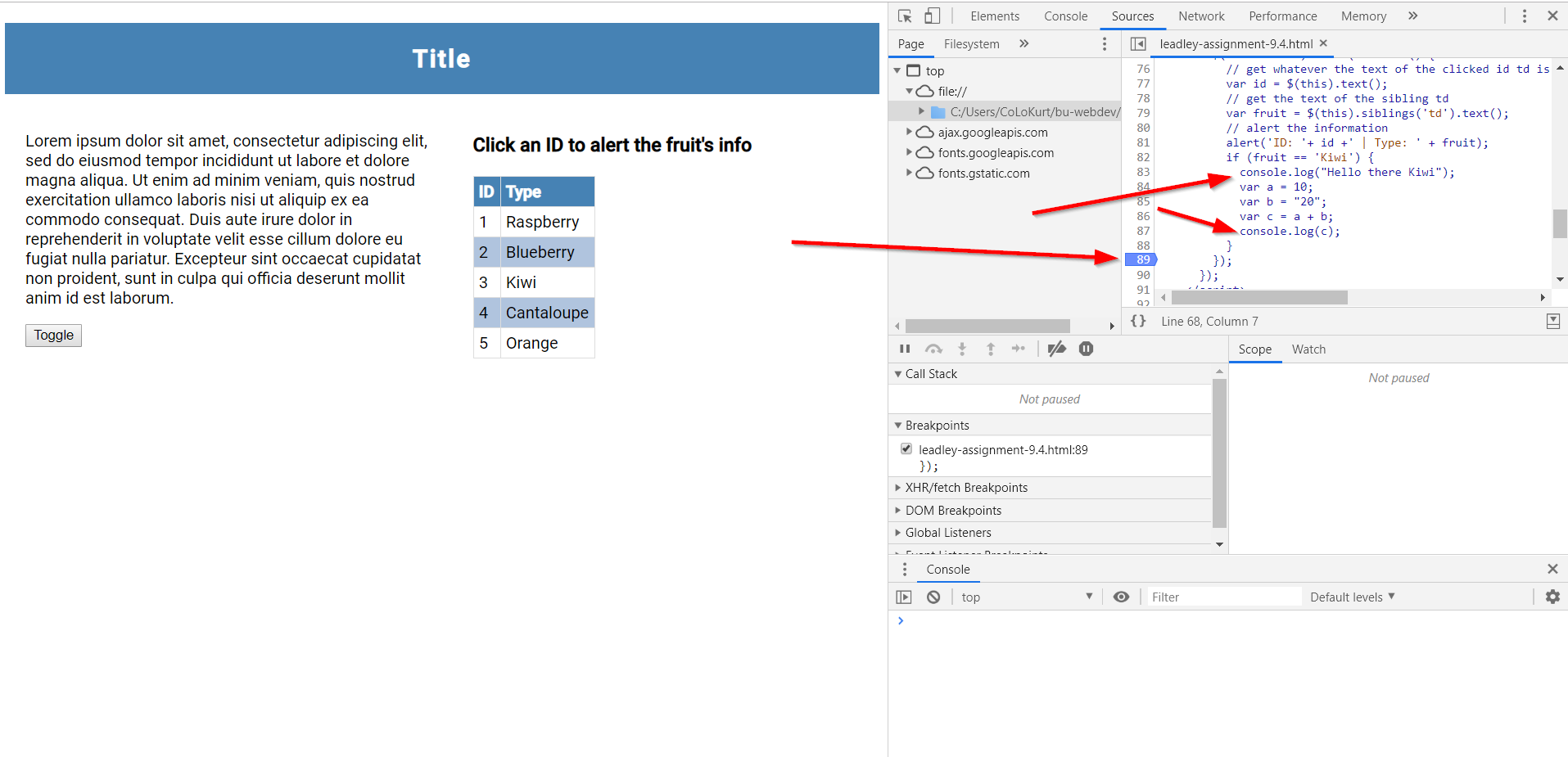
* Click the “Resume script execution” arrow like we did before to return the debugger to its initial state

## Using the Console to for debugging purposes

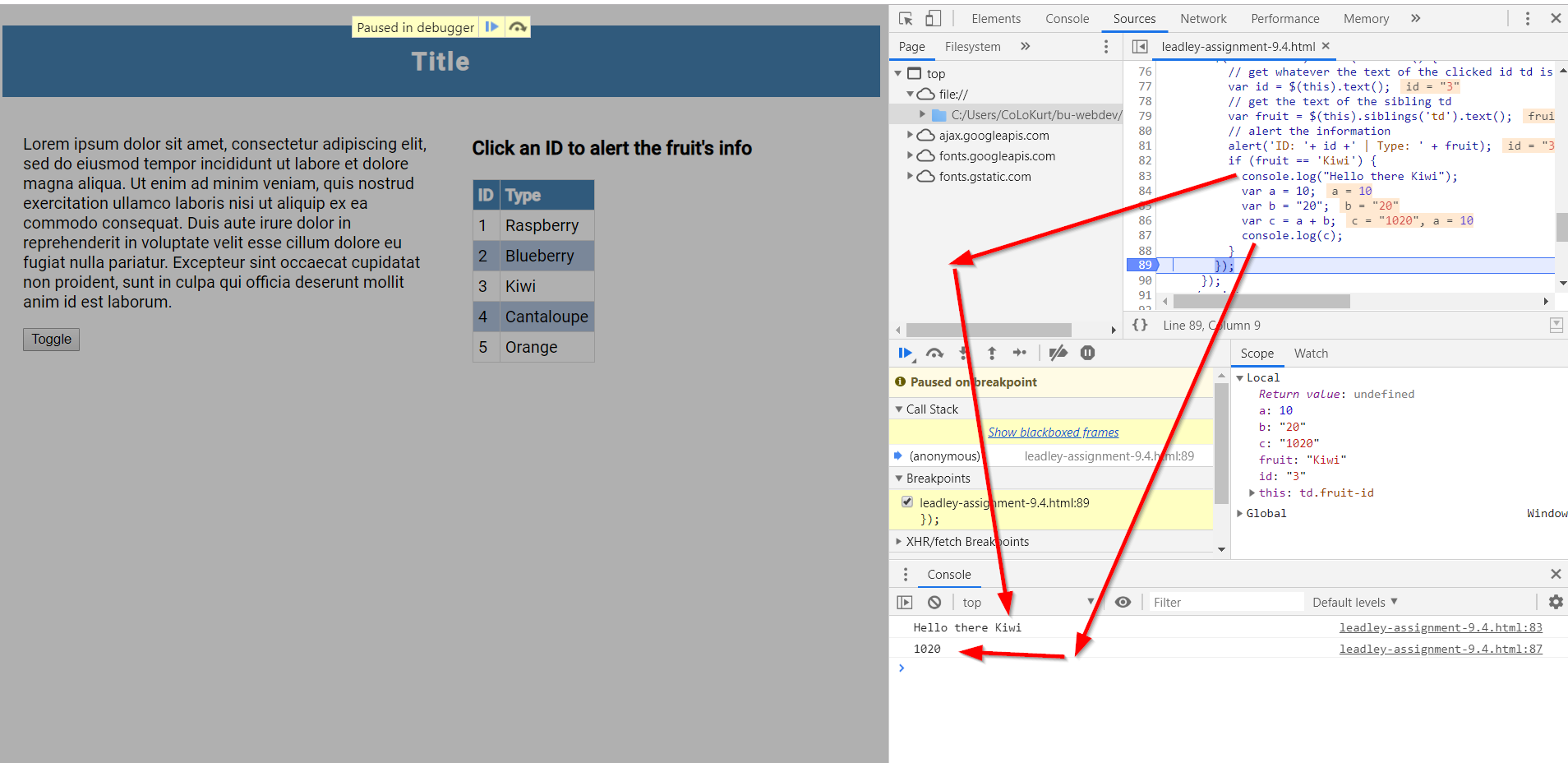
* Continuing on from where we left off, to open up the console window without clicking the console tab, press the “ESC” key. Make sure to click your cursor somewhere inside the debugger to make sure that area of the browser is active.



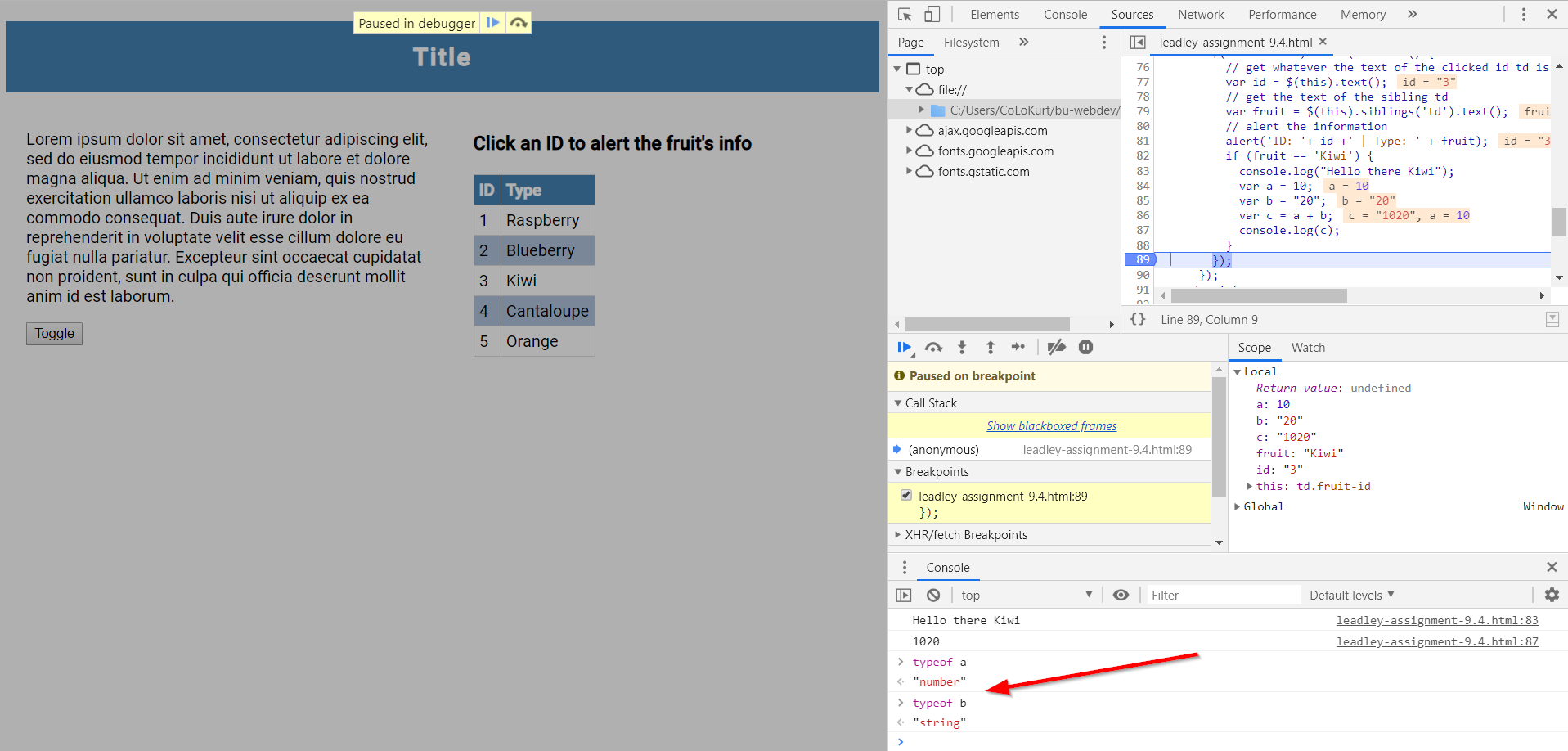
* Change the line break point to line 89
* I’ve added some “console.log” statements to the source file. They can viewed in the upper right pane in the debugger.



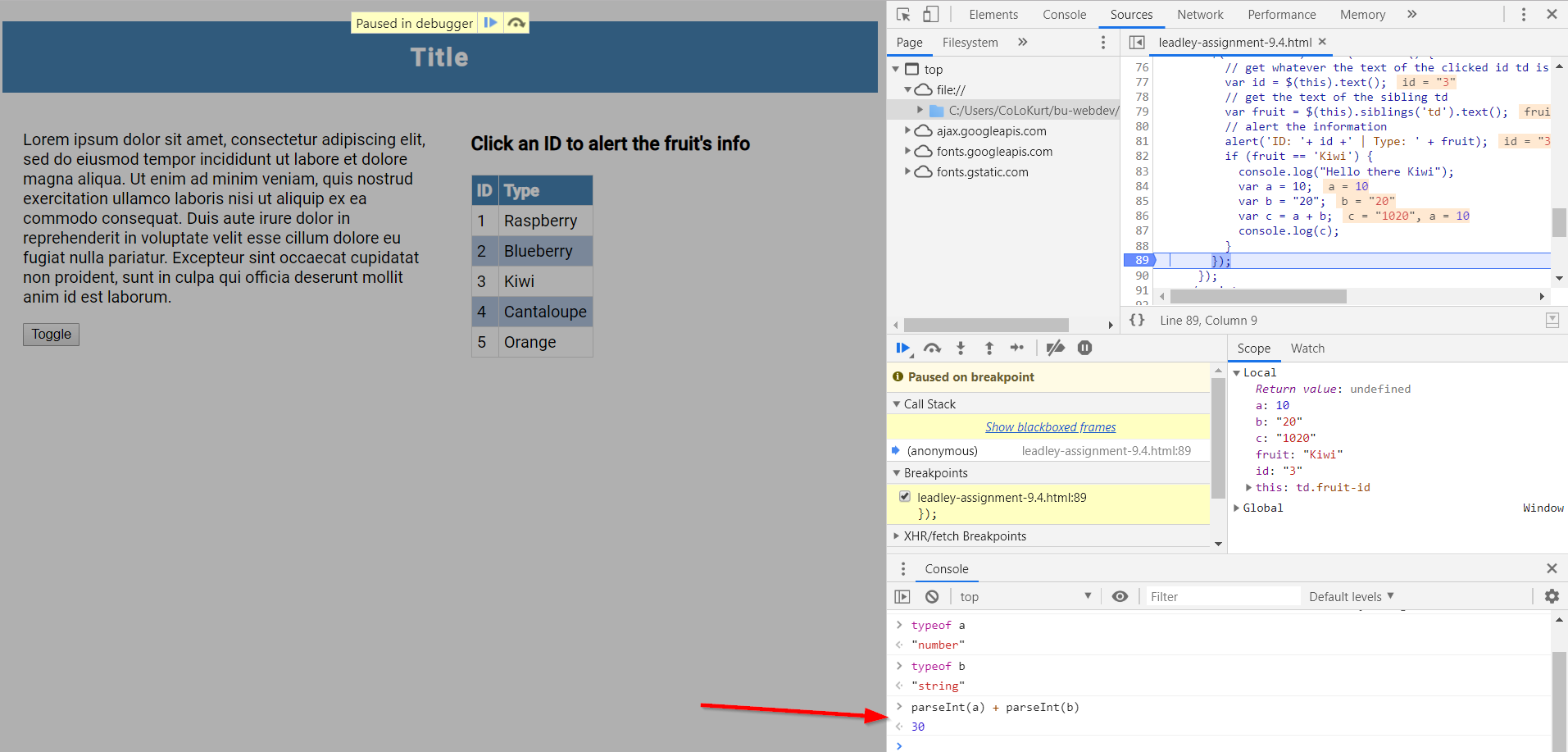
* Click on “3” in the “ID” column of the table
* You will get an alert, just click okay and then we should reach our breakpoint in the code.
* In your console window, you should see the output from the two console.log statements. Also, you can see the scope of all the variables at the break.



* As you can see in the console, a + b returns 1020. It should be 30. Let’s us the console to debug this.
* In the console window, type *typeof a* and hit enter. Then type *typeof b* and hit enter
* You can see that a is a number and b is a string.

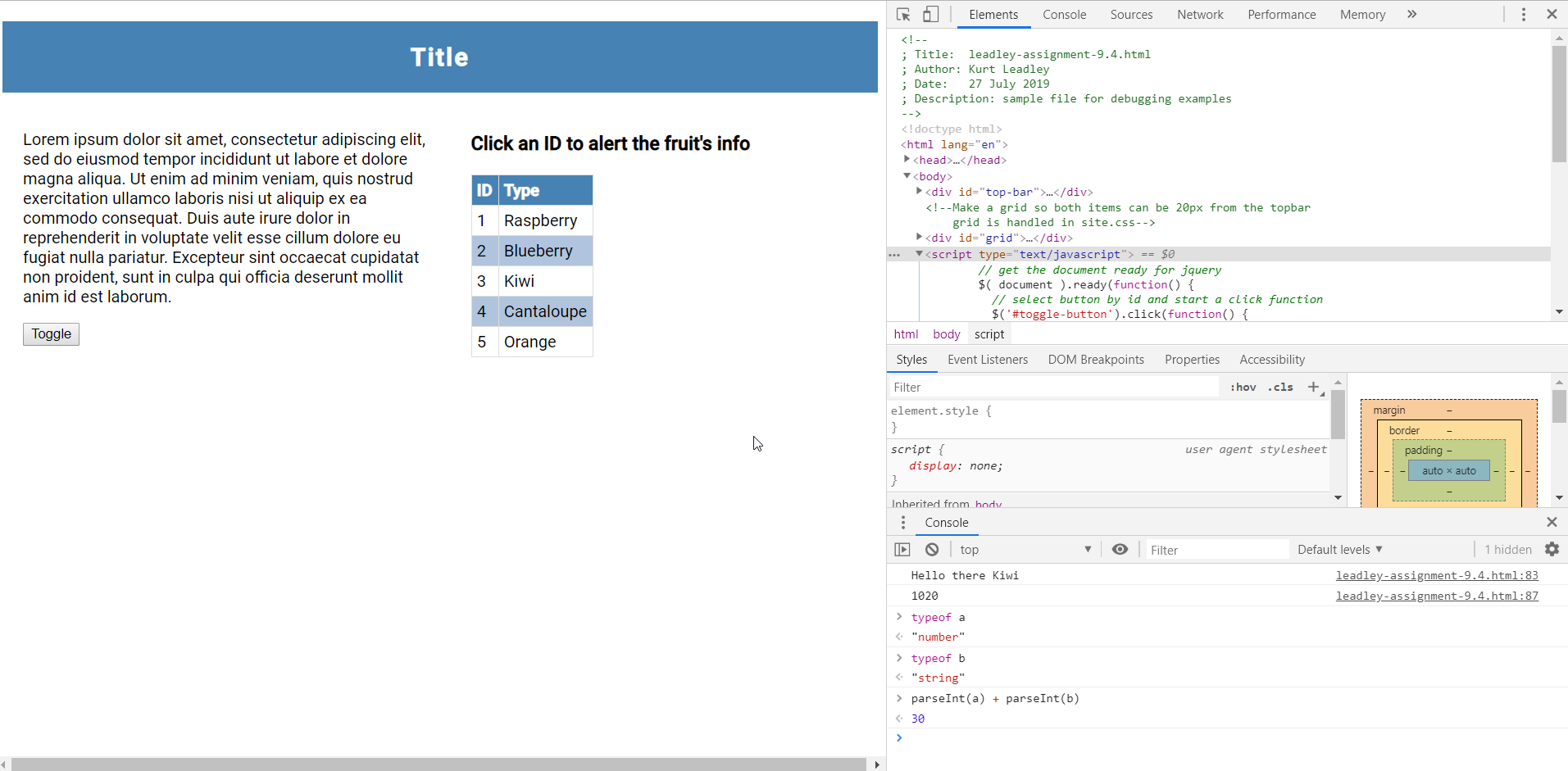


* We need to convert the variables to integers before applying arithmetic to them. Let’s test it out in the console before applying any code to the source file.
* In the console window, type *parseInt(a) + parseInt(b)* and press enter
* It should return 30

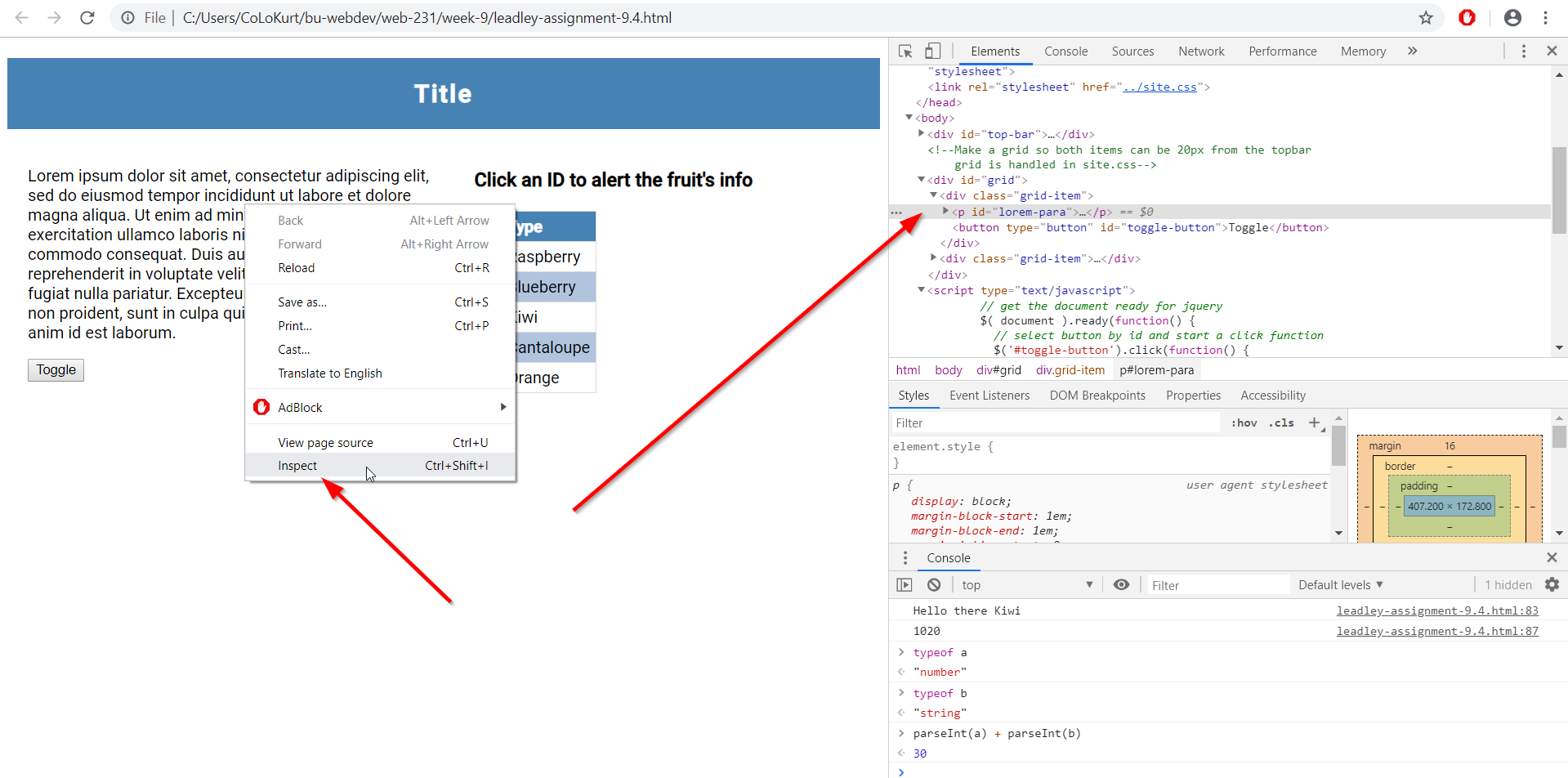


## Selecting HTML elements and using them in the console

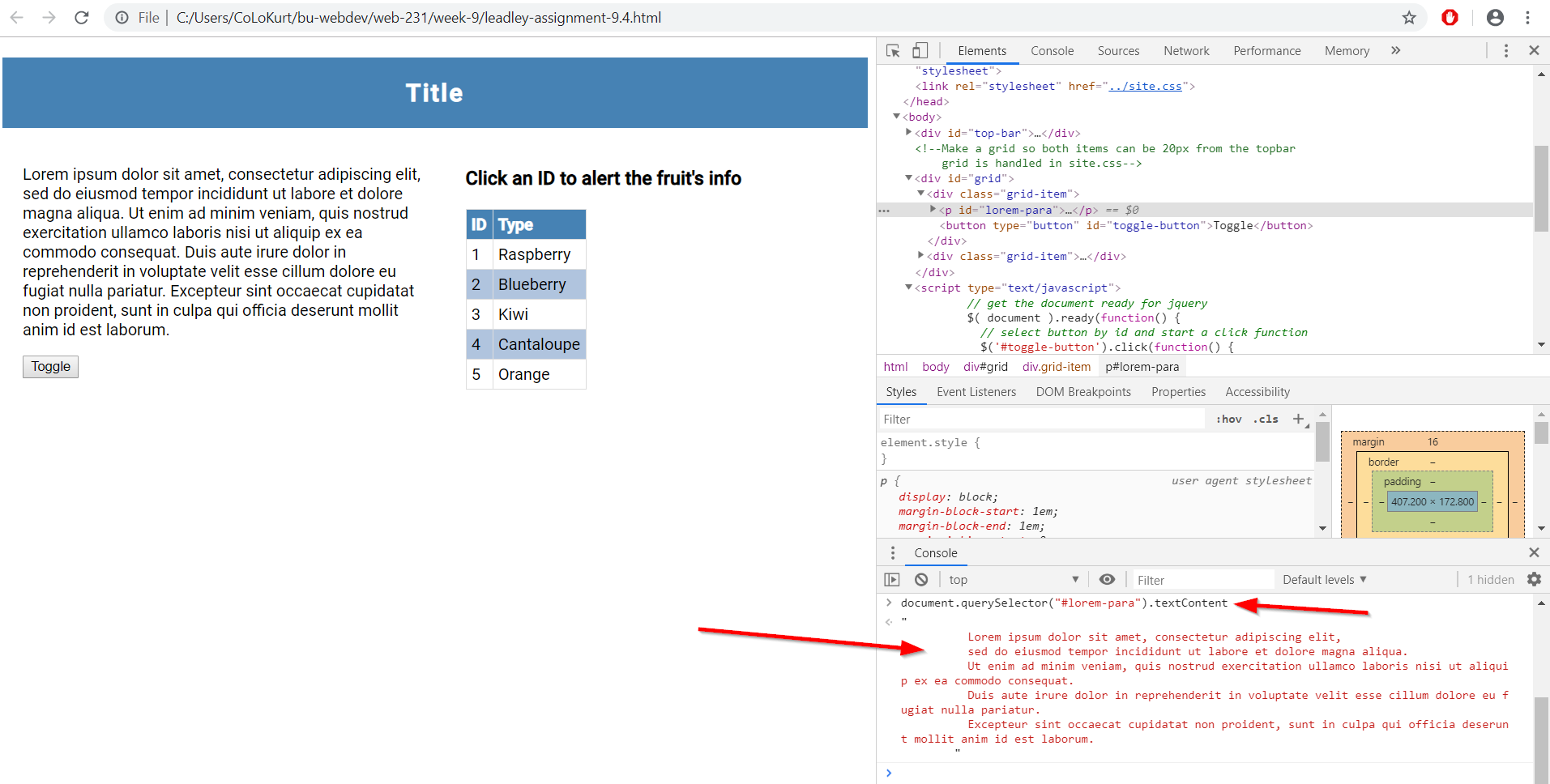
* Continuing where we left off, clear all breakpoints and finish code execution if you haven’t already.
* Click on the “Elements” tab



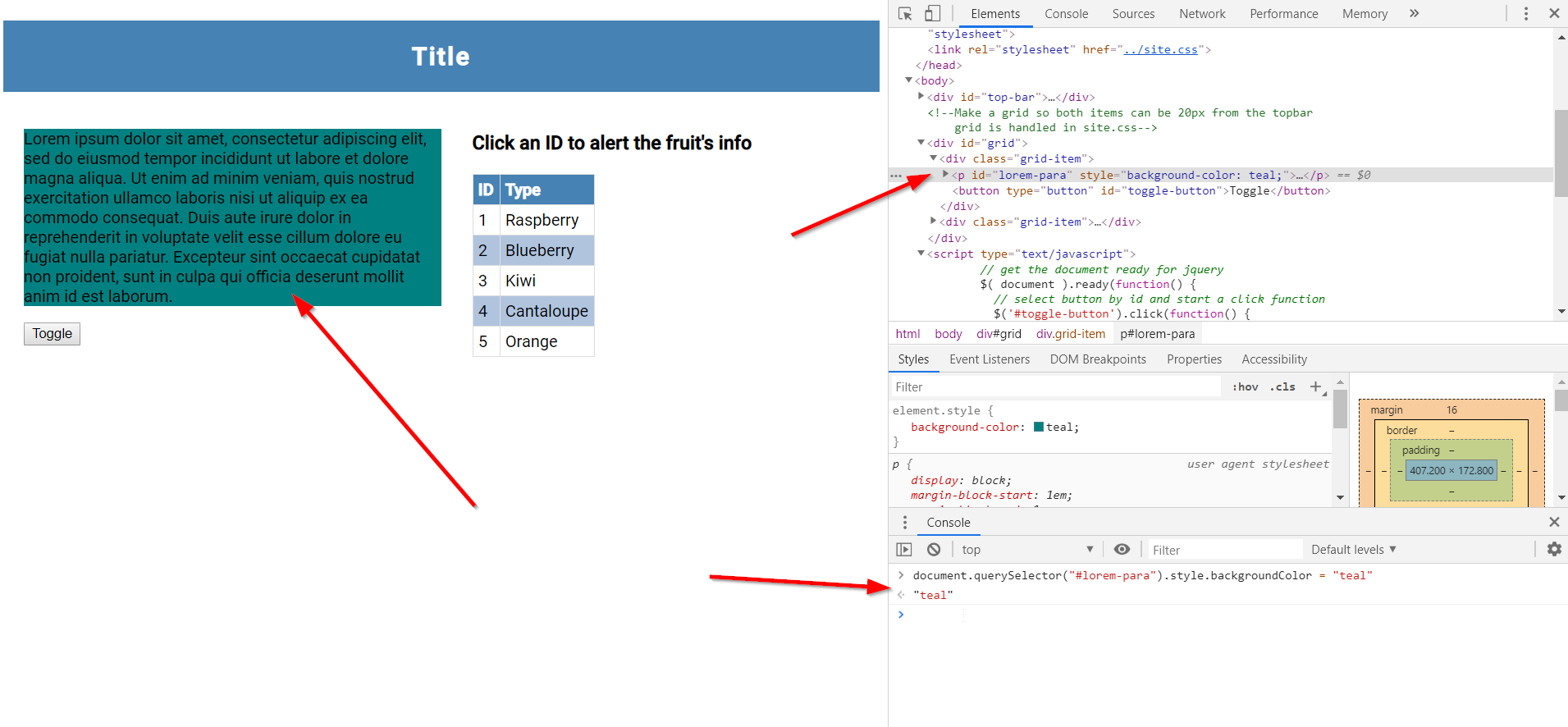
* Hover over the “Lorem ipsum…” paragraph text
* Right Click
* Choose “Inspect”
* You should see the element you chose to inspect highlighted in the “Elements” pane



* In the “Elements” pane, hover over the highlighted p element with your mouse
* Right click -> Copy -> Copy JS path
* The element you selected now has its JS path copied to your clipboard
* In the “Console” pane, paste the path into the console.
* With this path, you can test out different actions you can apply to this element using JS.
* Type a single “.” and a list of methods that you can apply to the element should appear in a vertical list.
* Choose .textContent and hit enter to view the text content of the element

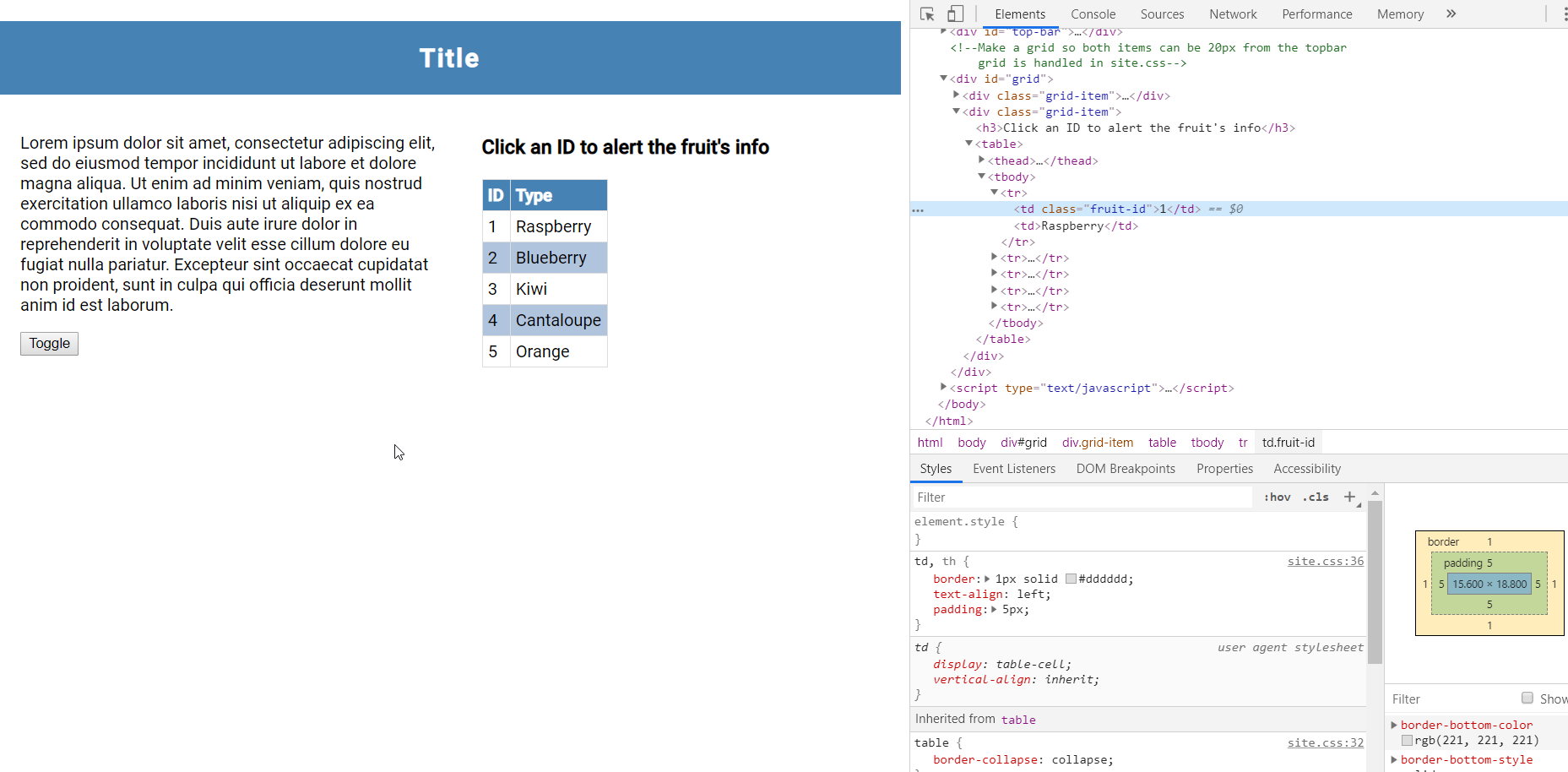


* You can style the element by pasting in JS path in and typing “.style” and then another “.” to bring up your styling options
* Choose background color and hit enter to color it

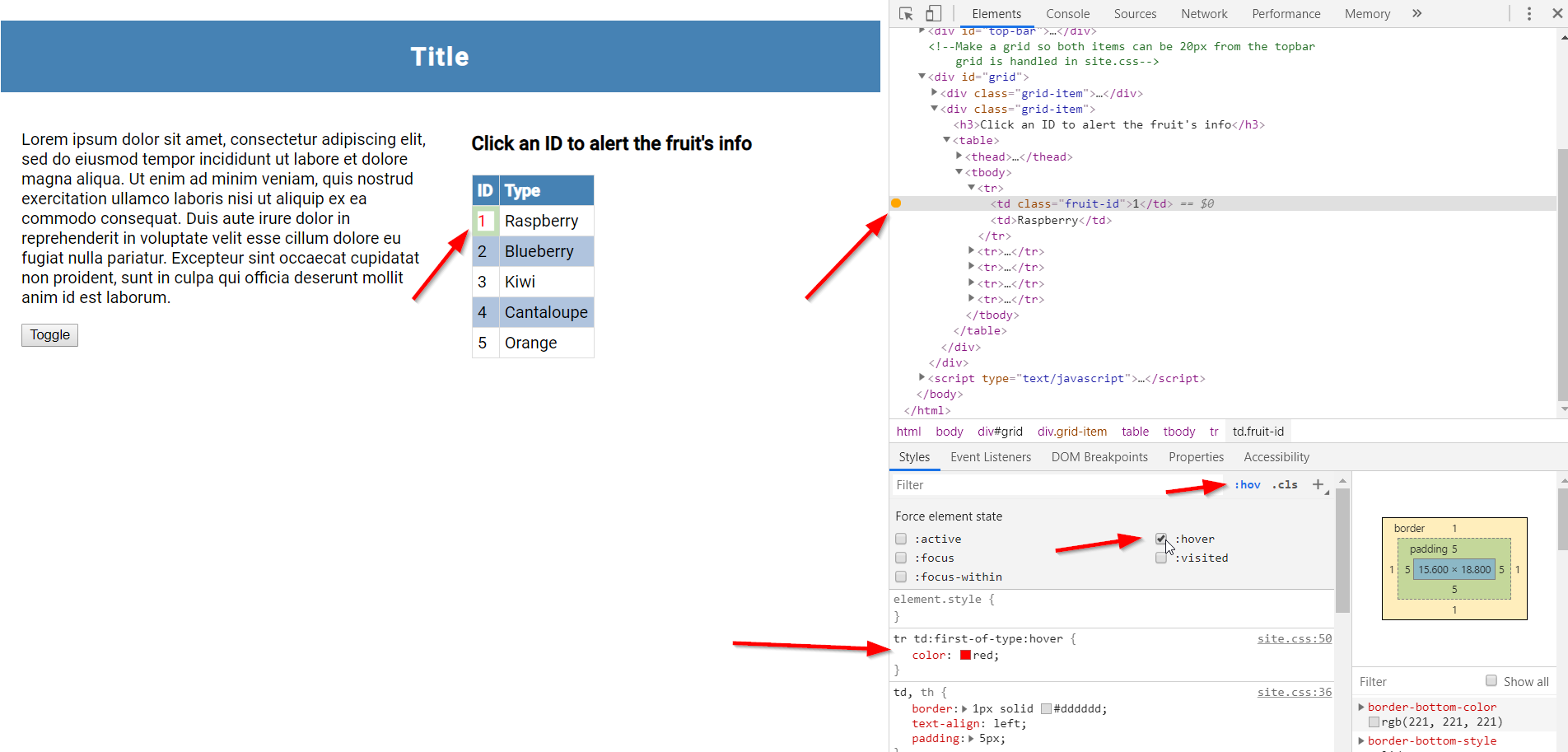


## Simulating CSS states on HTML elements

* Continuing where we left off, to simulate CSS states on HTML elements, right click “1” in the “ID” column of the table and choose “Inspect”
* The table cell should be highlighted in the Elements pane and its corresponding CSS in the “Styles” tab



* To mimic the CSS state of hover on this element, click :hov and then :hover in this “Styles” tab
* You should see in your browser the effect of the hover styling on the element (if one exists in your style sheet) along with the actual code in the “Styles” pane



* Now we are going to add a class to the Lorem ipsum paragraph
* Right click on the Lorem ipsum….. paragraph and click “Inspect”
* In the “Styles” tab, click “.cls” and then type “.hover-class” in the text field (or any other class you may have in your style sheet)
* The paragraph should be red and italic

