University of East Anglia
COMPUTING SCIENCES

Lab 4: JavaScript and the DOM

Before attempting this lab, you will need to have completed **Lab 3: CSS Media Queries and Responsive Design**

Learning objectives

By completing this lab, you should be able to:

- Describe JavaScript programming language and its usage
- Examine JavaScript variables and control loops.
- Examine the DOM of an HTML document.
- Explain what is meant by “walking the DOM”
- Link an external JavaScript file to a HTML document
- Write JavaScript code to create a new element and insert it into a file
- Write JavaScript code to remove an element from a file
- Test JavaScript code using browser console

Background

JavaScript (JS) is a lightweight, interpreted, or just-in-time compiled programming language with first-class functions. While it is most well-known as the scripting language for Web pages, many non-browser environments also use it, such as Node.js, Apache CouchDB and Adobe Acrobat. JavaScript is a prototype-based (MDN), multi-paradigm, single-threaded, dynamic language. It supports object-oriented, imperative, and declarative (e.g. functional programming) styles.

Do not confuse JavaScript with the Java programming language. Both "Java" and "JavaScript" are trademarks or registered trademarks of Oracle in the U.S. and other countries. However, the two programming languages have very different syntax, semantics, and use. For this module we will be using the 6th edition of the language (sometimes known as ECMAScript 2015 or ES6), introduced in 2015, added many new features. We will be using JavaScript to manipulate our HTML documents.

The Document Object Model (DOM) (MDN) is a programming interface for HTML (and XML) which allows programs and scripts to change the document’s structure, style and content. It is structured in a hierarchical format, which can be visualised using a tree diagram. Each branch of the tree ends in a node, and each node contains objects. DOM methods allow programmatic access to the tree. With them, we can change the document's structure, style, or content. Nodes can also have event handlers attached to them. Once an event is triggered, the event handlers get executed.

Accessing the DOM

We don't have to do anything special to begin using the DOM. We use the API directly in JavaScript from within what is called a script, a program run by a browser.

When we create a script, whether inline in a `<script>` element or included in the web page, we can immediately begin using the API for the document or window objects to manipulate the document itself, or any of the various elements on the web page (i.e. the descendant elements of the document).

In this lab, we will use JavaScript to interact with the DOM for our `index.html`.

Exercises

1. Create a new folder in your student U: drive called `js-dom`. Copy all the files in `css-media` folder to this new folder.

2. Open this `js-dom` folder in your Visual Studio Code editor.

Part One

3. On Visual Studio Code editor, create a new file called `variables.js`

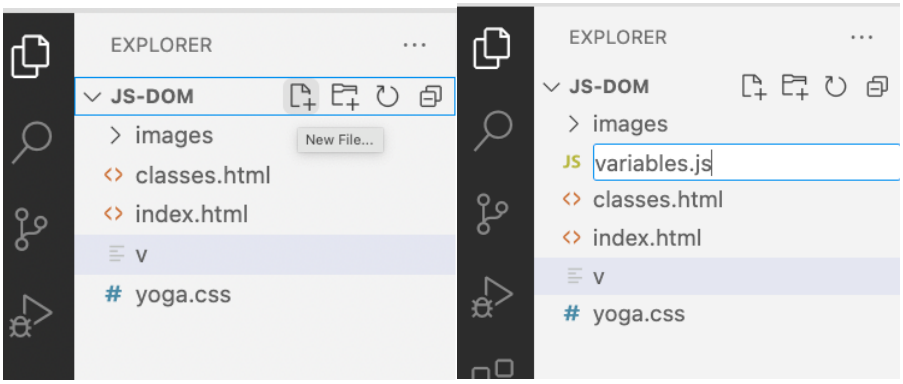


Figure 1. VS Code Editor

4. In your `variables.js` file:
    a) Create a variable type `var` called myVariable and assign the value as "Web Programming is fun!" (Figure 2)

Figure 2. variables.js

    b) Write a JS code to print the value to the web console (Figure 3):

```
JS variables.js > ...
1   var myVariable = 'Web Programming is fun! ';
2   console.log(myVariable);
```

Figure 3. variables.js

a) On VS Code editor, open your `index.html` file. To add or link this `variables.js` file - create a `script` element and place it inside the `head` block; the value of `src` is "`variables.js`" (Figure 4)

```
<head>
<meta name="viewport" content="width=device-width, initial-scale=1">
<title>Mimo Yoga Studio</title>
<link rel="stylesheet" href="yoga.css">
<script src="https://kit.fontawesome.com/67bd86ffdc.js" crossorigin="anonymous"></script>
<script src="variables.js"></script>
</head>
```

Figure 4. index.html

b) Verify the output:

i. Save your `index.html` file.

ii. Open your `index.html` using Chrome (Figure 5):

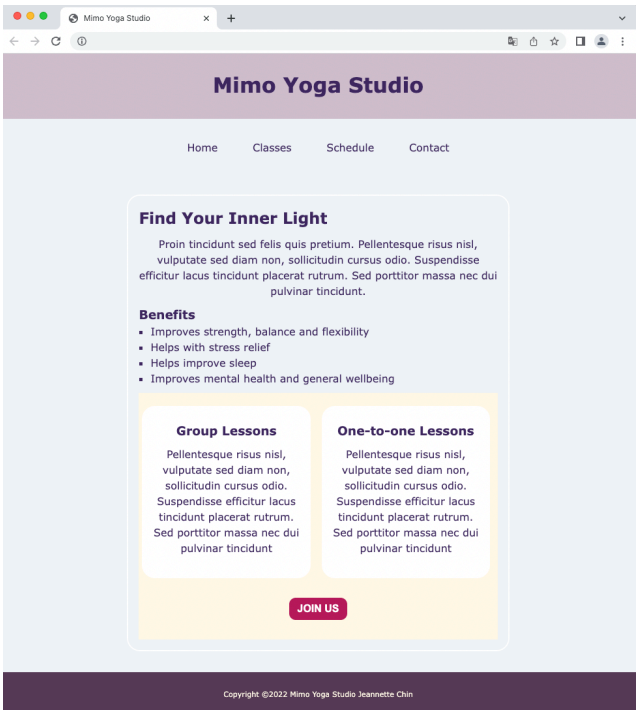


Figure 5. index.html

iii. On any part of the page, right click the mouse and select the "Inspect" option (Figure 6):

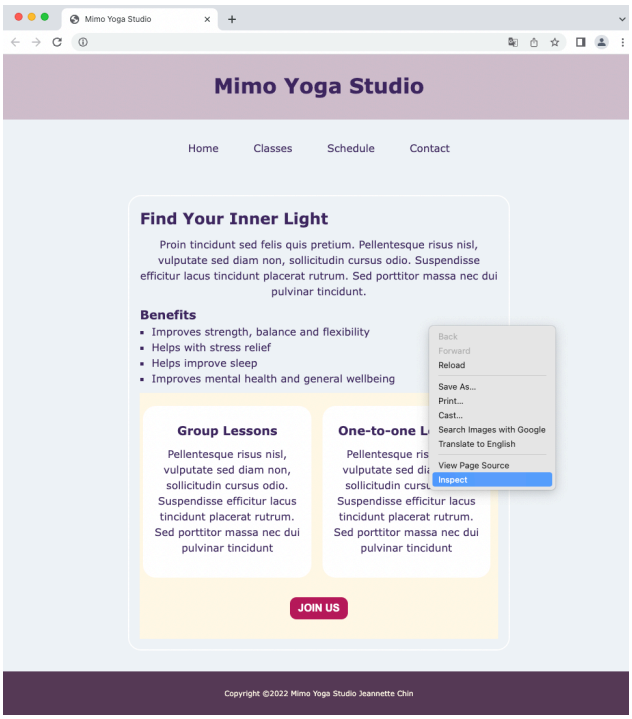


Figure 6. index.html

iv. On the 'Inspect' window, select the "Console" tab and verify the output:

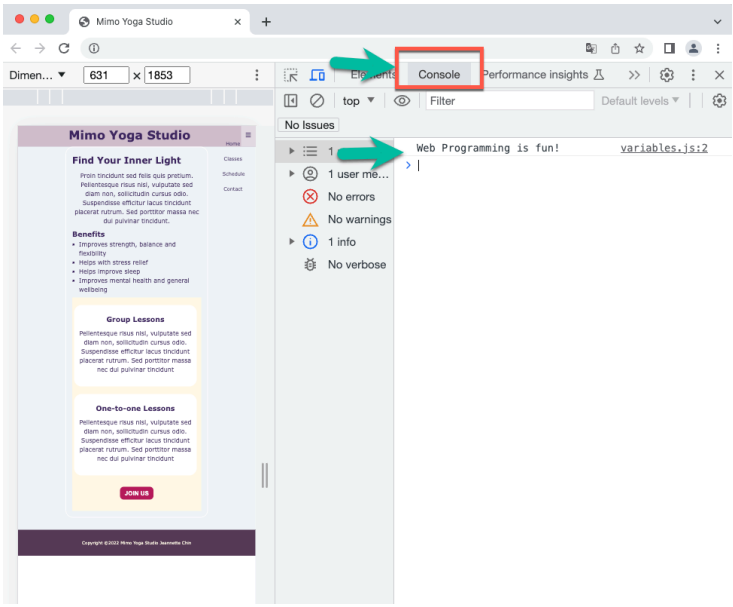


Figure 7. index.html

c) Now, in your `variables.js` file, create a second variable type `let` called `letVar` and assign the value as “This is let variable” (Figure 8):

```
3
4   let letVar = 'This is let variable';
```

Figure 8. variables.js

d) Write JavaScript code to print this value on Web console and verify the output using Chrome (Figure 9): [**HINT**: refer to task 4(b) above]

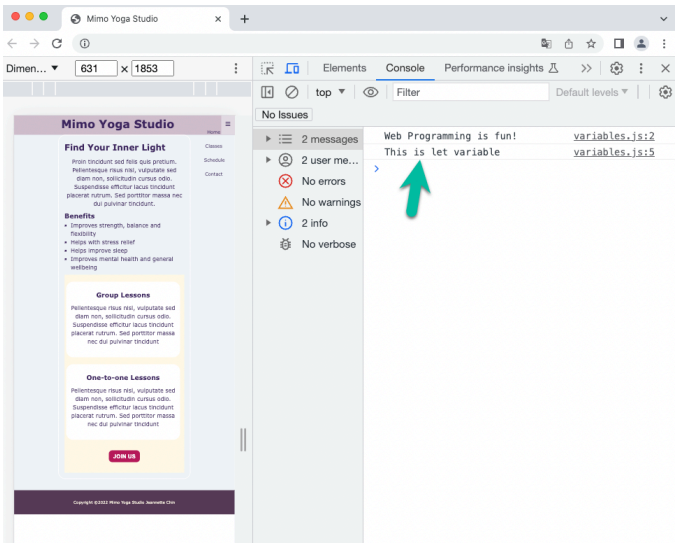


Figure 9. The output on Chrome - index.html

e) Back to your `variables.js` file, create a variable type `const` called `constVar` and assign the value as “This is const variable” and print this value on Web console (Figure 10):

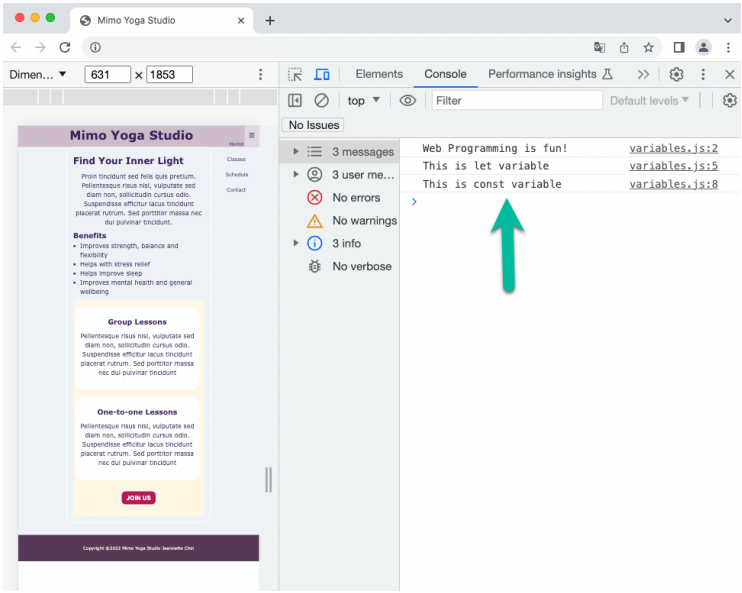


Figure 10. The output on Chrome - index.html

f) Now, let's us print 5 outputs. We will print `myVariable` 5 times. We will use a `for` loop for this task:

```
10  for (let i = 0; i< 5; i++){
11      console.log(myVariable);
12  }
```

Figure 11. The code in variables.js

g) Save the file.
h) Verify this output in Chrome (Figure 12):

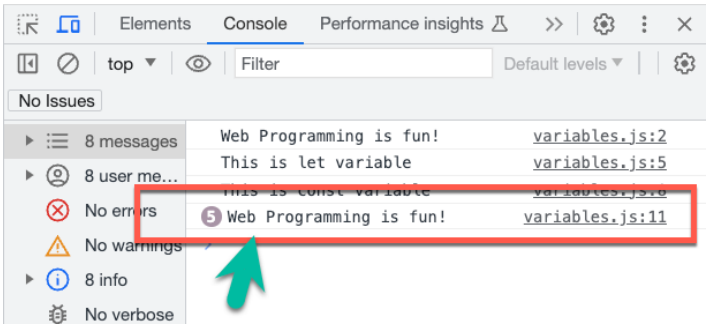


Figure 12. The output in Chrome

i) Notice that the output has only one line with an indicator "5" at the beginning. This is because when the outputs are identical, Chrome only shows one output. The indicator tells us the number of the identical outputs. To see all 5 outputs, we can concatenate the index in

the string, in my case, I concatenate the index at the end of the string (line 11, Figure 13):

```
10    for (let i = 0; i< 5; i++){
11        console.log(myVariable + i );
12    }
```

Figure 13. The code in variables.js

j) Refresh Chrome and your output should be similar to Figure 14:

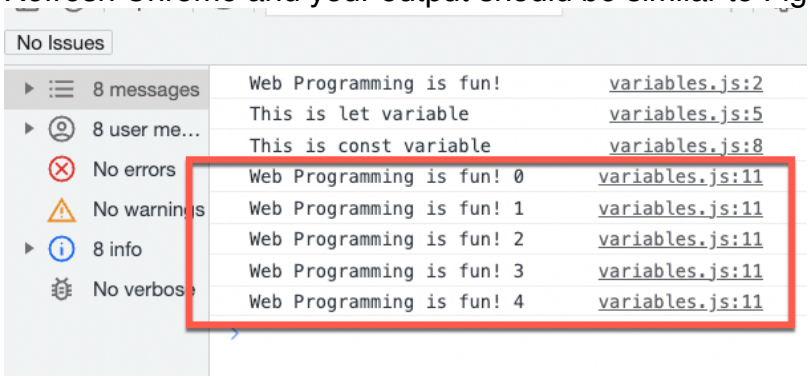


Figure 14. The output in Chrome

k) Now, the code below is a while loop and print the output of variable `letVar` 3 times (Figure 15). Try this yourself.

```
14
15    let number = 0;
16    while (number < 3){
17        console.log(letVar + number);
18        number = number + 1;
19    }
```

Figure 15. The code in variables.js

l) Verify your output using Chrome (Figure 16):

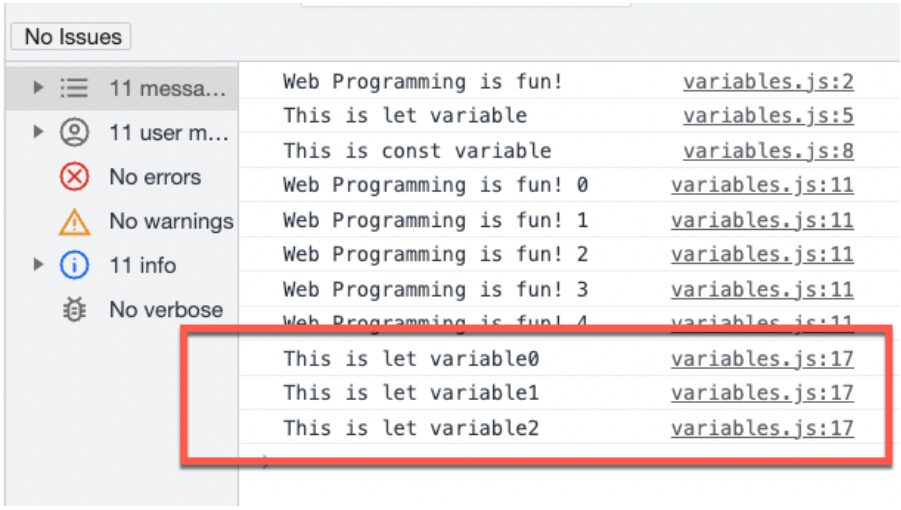


Figure 16. The output in Chrome

m) Now, try change the value of these variables and test them again. For example, the code in Figure 17 below is changing the value of `myVariable` in line 4, and print the new value in line 5 :

```
JS variables.js > ...
1   var myVariable = 'Web Programming is fun! ';
2   console.log(myVariable);
3
4   var myVariable = 'Web Programming is sooooo fun!';
5   console.log(myVariable);
6
```

Figure 17. The code in variables.js

n) The output on Chrome is:

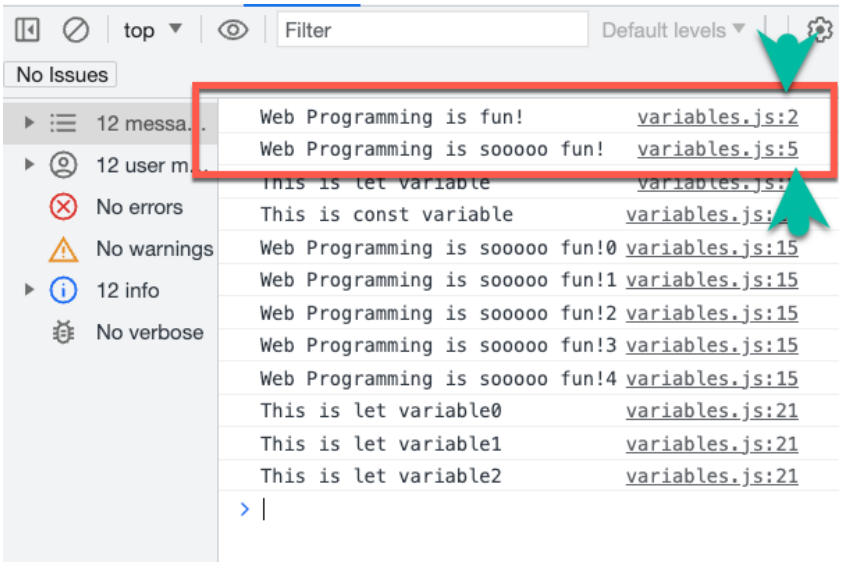


Figure 18. The output

o) Now in your code, using the code I have provided above as guidance, change the value for `letVar` and `constVar`, and verify the outputs. Are there any errors? If so, why?

Make sure the above exercises are completed before moving on to the following DOM manipulation tasks.

Part Two

5. Now on Visual Studio Code editor, create a new file called `dom.js` .

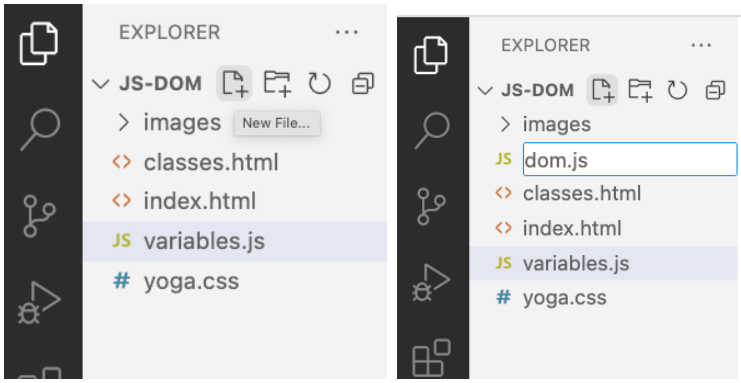


Figure 19. VS Code editor

6. Walking the DOM: Type the below JavaScript code (Figure 20) in your `dom.js`, and save the file.

```
JS dom.js > ...
1   /// See lab worksheet for code explanation
2   function theDOMElementWalker(node){
3       if (node.nodeType == 1){
4        console.log(node)
5        node = node.firstChild;
6        while(node){
7         console.log(node.nodeType);
8         theDOMElementWalker(node);
9         console.log("Value: " + node.textContent);
10        node = node.nextSibling;
11       }
12      }
13    }
14    let list = document.querySelector('main');
15    theDOMElementWalker(list);
16
```

Figure 20. dom.js

Code explained:
The above code is a function called *theDOMElementWalker*. This function takes a Node (MDN) as parameter (Line 1). It then checks the type of the Node to see if it is an ELEMENT_NODE (MDN) (Line 2), if it is, the function will print the Node

information on the console (Line 3). It grabs the first child (MDN) of this Node and store the value in a variable called node (Line 4). Using a while loop (Line 5), the function prints the information of the Node type on the console (Line 6) , and calls itself recursively passing in the value of node as parameter (Line 7). It prints the content of the Node on the console (Line 8) and reassign the value of the variable node to the Node's next sibling (MDN).

From the above code, which element/node do we want to start traversing the DOM?

7. Now on the `index.html` file:
   a) To link this `dom.js` – HINT: use `script` element and place it inside the `head` section; the value of `src` is "`dom.js`".
   b) Save the `index.html` file.
   c) Verify the output on Google Chrome:
      a) Refresh Chrome. What do you see on this web console? Any errors?
      b) If you see errors similar to Figure 21 below, why?

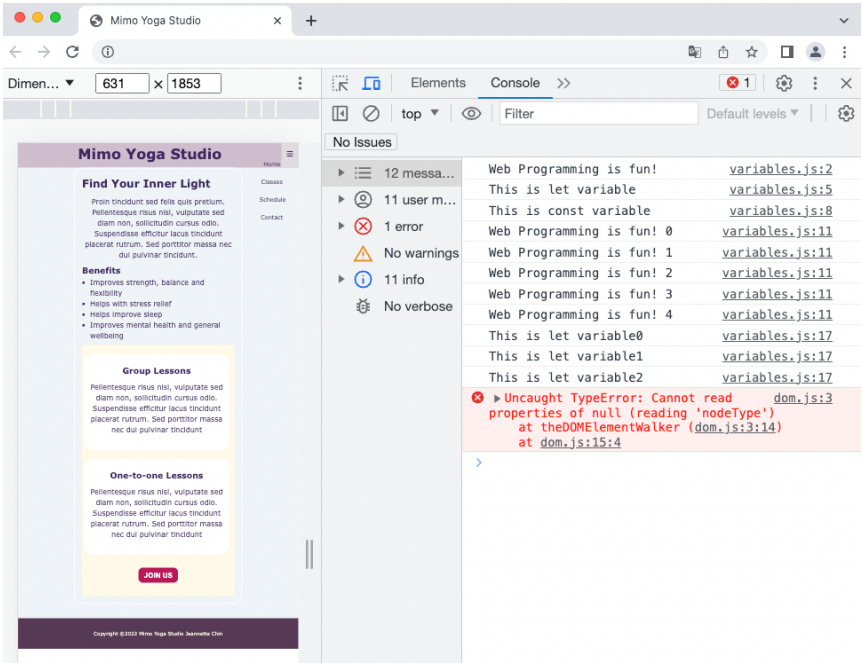


Figure 21. The output on index.html

   d) To fix this error, add a "defer" attribute in your script tag, as shown below:

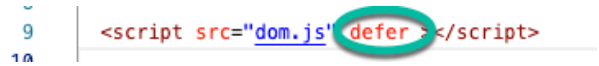


Figure 22. index.html

   e) Save the file and refresh the page. Now what do you see on the console?

f) Now modify the `dom.js` and choose another `element` to start traversing the DOM. I suggest `article` is a good start. Make sure you understand the output.

8. Dynamically create a new element and insert it to `index.html` "on the fly".
   a) On Visual Studio Code editor, create a new file called `new.js`.

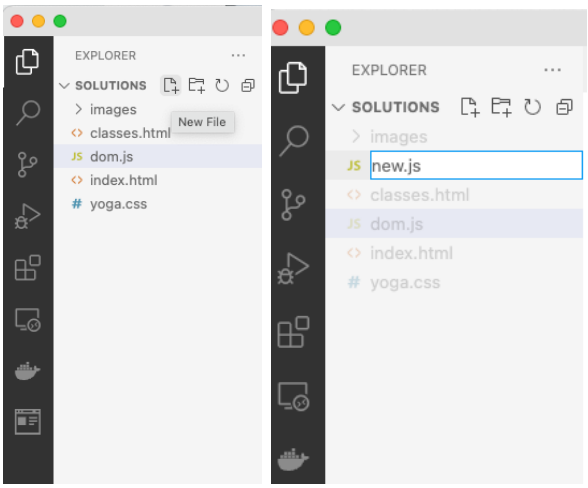


Figure 23. The screenshots of Visual Studio Code editor

   b) Write JavaScript code:
      i. to create a new `p` element, with the text content: "`Hi I am new P!`". The text colour of this new `p` element should be blue. This new `p` element should be inserted **just before** the `<section>` element in the `index.html` page – as shown in the figure below:

```
<main>
    <h2>Find Your Inner Light</h2>
    <p>Proin tincidunt sed felis quis pretium. Pellentes
    <h3>Benefits</h3>
    <ul class="benefits-list">
    <li>Improves strength, balance and flexibility</li>
    <li>Helps with stress relief</li>
    <li>Helps improve sleep</li>
    <li>Improves mental health and general wellbeing</li
    </ul>
<section>
    <article class="service">
        <h3>Group Lessons</h3>
        <p>Pellentesque risus nisl, vulputate sed diam n
    </article>
    <article class="service">
        <h3>One-to-one Lessons</h3>
        <p>Pellentesque risus nisl, vulputate sed diam n
    </article>
    <p>
        <button class="btn"> join us </button>
    </p>
</section>
```

Figure 24. index.html

Follow these steps:
      i. Step 1 - to create new element use `document.createElement()`:

```
1   //// create a new element
2   let newp = document.createElement("p");
```

Figure 25. code in new.js

ii. Step 2: The text colour can be configured via `Element.style.cssText` property (MDN) ;

```
3   //// configure some new style
4   newp.style.cssText = "color: blue;";
```

Figure 26. code in new.js

iii. Step 3: The content can be configured using `Element.textContent` property;

```
5   //// configure the content
6   newp.textContent = "Hi This is new P! ";
```

Figure 27. code in new.js

iv. Step 4: to select the `section` node we can use `document.querySelector()` method;

```
7   ///// grab the section element on the page
8   let sectionElement = document.querySelector("section");
```

Figure 28. code in new.js

v. Step 5: the insertion can be done by using `insertBefore()` method (MDN) but on the parent node of `section`.

```
9   //// attach the new p element before the section element
10  sectionElement.parentNode.insertBefore(newp, sectionElement);
```

Figure 29. code in new.js

ii. To link the script: in `index.html` – create a new `script` element and the value of its `src` attribute is `new.js` file. The `script` element should be placed inside the `head` block. Save your file.
iii. Test the `index.html` using Google Chrome.
iv. Your output should look similar to the figure below:

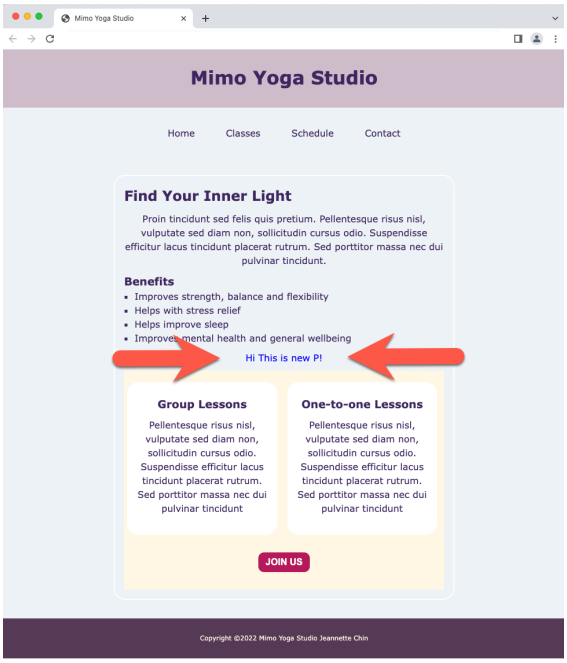
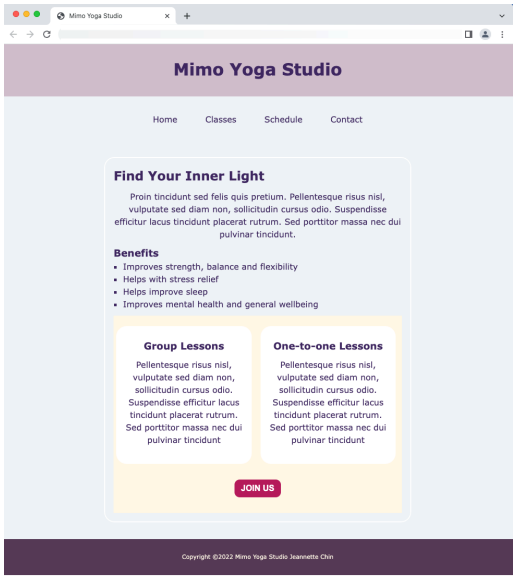


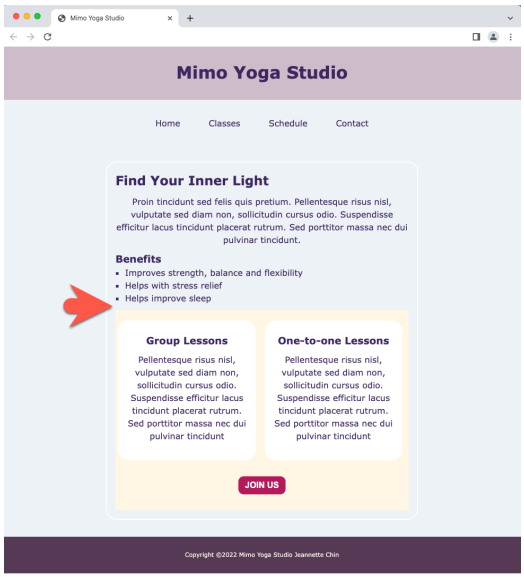
Figure 30. The output of index.html

v. If there are errors (e.g. similar to 7(c) above), fix them and refresh the page.

9. Write JavaScript code to remove the **last** `li` items from the "Benefits" list on `index.html` page. Your JavaScript code should be written as an external file called `remove.js`. Refer to the MDN for information on removing child elements: https://developer.mozilla.org/en-US/docs/Web/API/Node/removeChild. Your output should look like the screenshot below, that is, there should be 3 items left on the list instead of 4:



Before



After

Hint: Create an ID attribute for your *unordered* Benefits list; use `document.querySelector()` method to select this ID; access the children nodes from this element via the element's `.childNodes` property; use the `element.removeChild` method to remove the specific node.

10. Next, create a new file called `add.js`. Write some code to create a new `li` item with content "**Helps improve blood circulation**". Insert this new `li` item in the Benefits list but at the location *just before the last item* on this list. Your new `li` item should have blue text, as shown in figure below (Figure 32):

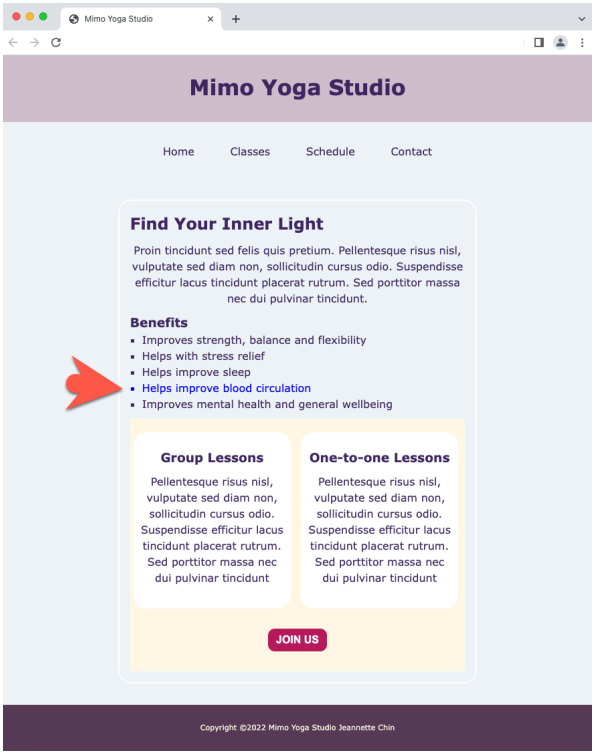


Figure 32. The output of index.html

Now that you have completed this lab, show your code and output to the lab tutor. They will sign the lab sign-off sheet if the lab has been satisfactorily completed.