University of East Anglia
COMPUTING SCIENCES

Lab 5: JavaScript Events and Forms

Before attempting this lab, you will need to have completed **Lab 4: JavaScript and the DOM**.

Learning objectives

By completing this lab, you should be able to:

- Explain the event-based programming paradigm
- Give examples of HTML events that can be handled with JavaScript
- Describe the difference between the Event current target and target
- Write Event handlers using JavaScript
- Apply an event handler using Event Listener on an HTML element
- Test JavaScript code using the browser console
- Code HMTL form and form controls
- Apply CSS to style the web form
- Explain signposting for web usability
- Apply simple signposting on web pages

Background

Events are actions or occurrences that happen in the system you are programming, which the system tells you about so your code can react to them. For example, if the user clicks a button on a webpage, you might want to react to that action by displaying a dialog box. This is achieved by the system producing (or "firing") a signal of some kind that leads to an action (or “event handler”) that can be automatically taken (that is, some code running) when the event occurs.

Imagine in an airport, when the runway is clear for take-off, a signal is communicated to the pilot. As a result, the plane can safely take off. In the case of the Web, events are fired inside the browser window, and tend to be attached to a specific item that resides in it. The event might be trigged by an action done to a single element, or a set of elements, or while the HTML document is loaded in the current tab, or the entire browser window. There are many different types of events that can occur. For example:

- The user selects, clicks, or hovers the cursor over a certain element.
- The user chooses a key on the keyboard.
- The user resizes or closes the browser window.
- A web page finishes loading.
- A form is submitted.
- A video is played, paused, or finishes.
- An error occurs.

See MDN for more event references.

To react to an event, we attach an **event handler** to it. This is a block of code (usually a JavaScript function that we, as a programmer create) that runs when the event fires. When such a block of code is defined to run in response to an event, we say we are registering an event handler. There is another terminology called Event Listener. We register an event handler via an event listener. The event listener listens out for the event happening, and the event handler is the code that runs in response to the event happening.

When an event occurs, the browser will send a reference to the object onto which the event was dispatched, we called this the Event Object (MDN). We are interestd in two of the properties in this object, `currentTarget` and `target`. The `Event.currentTarget` is a reference to the currently registered target for the event. This means it is a reference to the HTML element where we register our event handler. The `Event.target` is a reference to the object to which the event is fired.

In this lab you will write event handler code and add the handlers to HTML elements. You will observe which element is the `currentTarget` and `target`.

Exercises

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

2. First, let us fix the toggle menu. Recall in Lab 3 (CSS media query), we have styled our hamburger menu nicely using CSS (Figure 1).

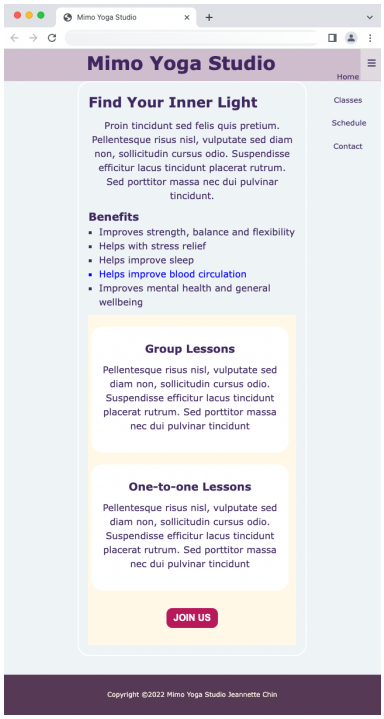


Figure 1. The output of index.html on screen size less than 768px.

We will write some code to make the navigation links only appear when the user clicks on the icon. This means, by default, the links are hidden. The links will only appear when the user clicks on the hamburger icon.

I. Using a text editor and create a new file named `hamburger.js`

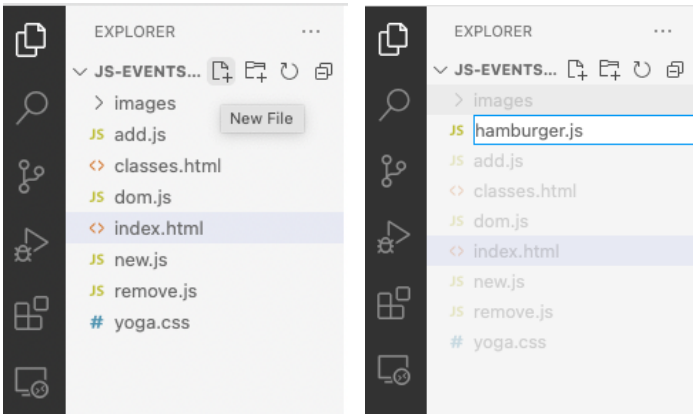


Figure 2. The screeshots of Visual Studio Code editor

If you are not using Visual Studio Code, make sure that this new file is saved in the same directory.

II. We are interested in the element where the hamburger icon is coded in our HTML document. For me, the icon is coded inside an anchor `<a>` tag with an ID called “hamburger”, as shown in Figure 3 below. If this is not the case for you, then create this ID now in both `index.html` and `classes.html`.

```
15   <header>
16       <h1>Mimo Yoga Studio</h1>
17       <a id="hamburger">
18           <i class="fa-solid fa-bars"></i>
19       </a>
20   </header>
```

Figure 3. The code in index.html

III. For the default behaviour where we want our navigation links to be hidden, we will use CSS to do this job. In both of your `index.html` and `classes.html`, add a class attribute called "mainNav" to the `ul` element, as shown in Figure 4 below:

```
22   <nav id="mainNav">
23       <ul class="mainNav">
24           <li class="nav"><a href="index.html">Home</a></li>
25           <li class="nav"><a href="classes.html">Classes</a></li>
26           <li class="nav"><a href="schedule.html">Schedule</a></li>
27           <li class="nav"><a href="contact.html">Contact</a></li>
28       </ul>
29   </nav>
```

Figure 4. The code in index.html

IV. We want this default behaviour when the hamburger menu icon appears. Recall from Lab 3 (CSS Media Query) we coded that the hamburger menu icon only appears when the screen size is less than 768px. So, we will continue our configuration using the same media query we created in that lab. Now open the `yoga.css` file in the editor and scroll down to the media query

a. Create a rule: for CSS class **mainNav**, the `height` is 0, and the value of the `overflow` property is `hidden`.

b. Create a new CSS class named **showNav** with a rule to define the `height` to 300px.

c. Save the file.

d. Now test the `index.html` using Chrome. Resize the browser window so that it is smaller than 768px. You will see that the navigation links are disappeared as shown in Figure 5. This is the default behaviour we want.

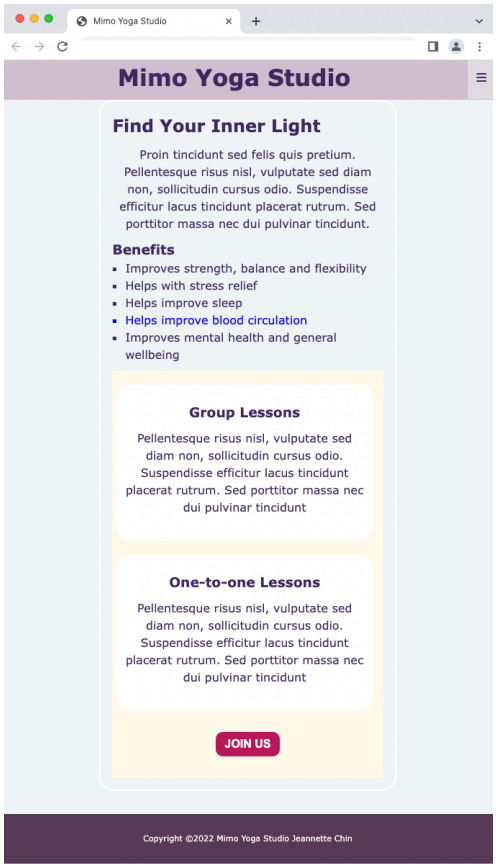


Figure 5. The output of index.html with screen size less than 768px. The navigation links near the hamburger menu icon are hidden by default

V. Next, we will write some JavaScript code to listen for user clicks. In your `hamburger.js` file:

   a. write some code to pick up the element with an **ID** called "hamburger" and store the value in a let variable called "hamburger". [**Hint:** use `document.querySelector()` method]

   b. register a 'click' event hander called "toggleNav" to this hamburger element [**Hint:** use the `addEventListener()` method]. The "toggleNav" is our event handler we have yet to write.

   c. now, scroll up and at the beginning of the file (`hamburger.js`) write a **function** named "toggleNav". NOTE: This function should be placed above the code where we register the event handler, or the browser will complain.

      In this "toggleNav" event handler function –

      i. Write some code to pick up the element with a **class** called "mainNav" and store the value in a let variable

called “links” [**Hint:** use `document.querySelector()` method]

ii. We will toggle the CSS class for this element to “showNav” by using the `classlist.toggle()` method (MDN), as shown in Figure 6 below:

```
5    links.classList.toggle('showNav');
```

Figure 6. The code in hamburger.js

iii. Save the file

d. Now we need to add this `hamburger.js` to our `index.html` to see the effect. Add this file to your `index.html` and `classes.html` now [**Hint**: use `<script>` tag inside `<head>` block]

e. Save the files and test them using Chrome.

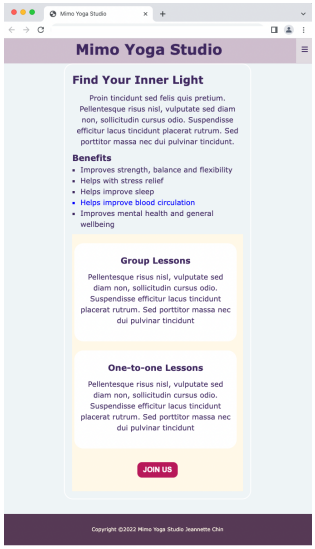


Figure 7. default behaviour

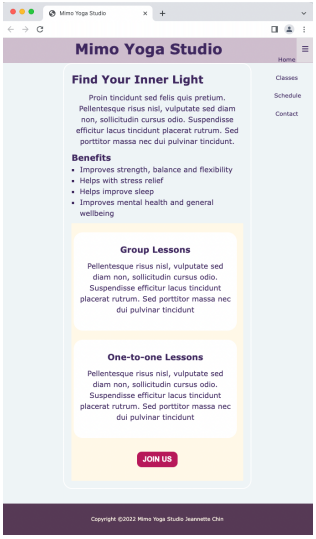


Figure 8. Main navigation links appear after user clicks the hamburger menu icon

VI. Create some CSS style rules such that each navigation link has a background colour of #ebe8eb and the background colour will change to #f4f0f4 when user mouseover it, as shown in Figure 9 and Figure 10. [**Hint**: you may need to tweak your existing styles to make it look good]

Figure 9. The hamburger navigation links have a background colour of #ebe8eb

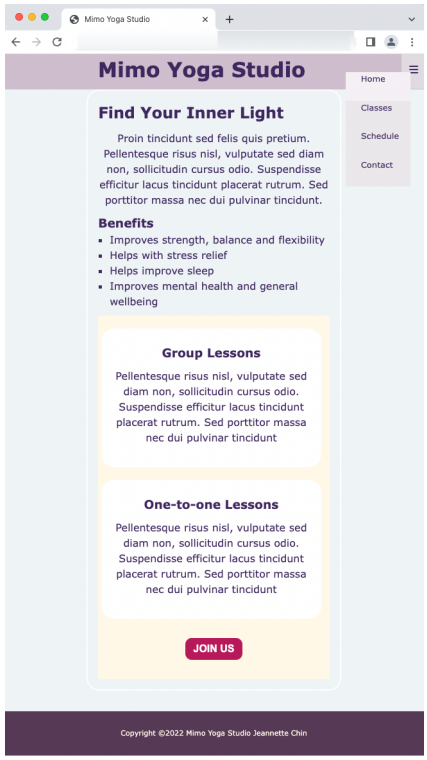


Figure 10. The hamburger navigation link changes its background colour when the user mouseovers it.

VII. Test both webpages and make sure the toggle menu function works.

3. Next, we will create a form and write some code to handle the form events. In your editor create a new page called `contact.html` as shown in Figure 11 and Figure 12 below:

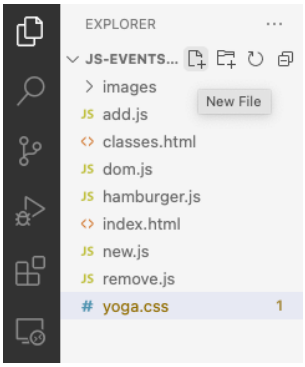


Figure 11. The screenshot of VSC editor

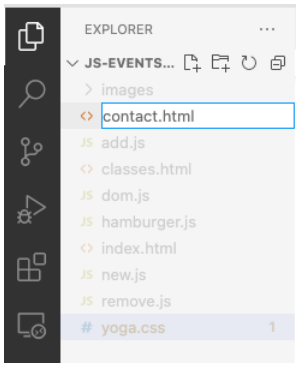
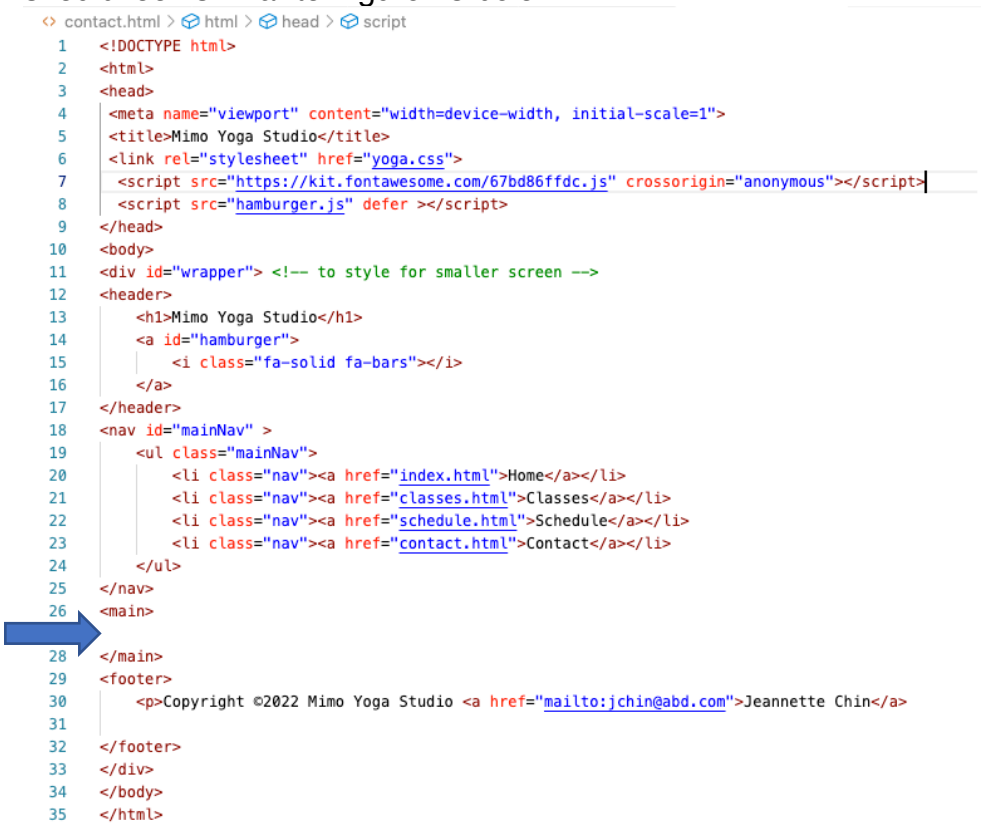


Figure 12. The screenshot of VSC editor

a) To save time, copy the code in `index.html` file and paste them to this new file

b) Delete all the content wrap inside the `<main>` element. Your code should look similar to Figure 13 below:



```
contact.html > html > head > script
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <meta name="viewport" content="width=device-width, initial-scale=1">
5  <title>Mimo Yoga Studio</title>
6  <link rel="stylesheet" href="yoga.css">
7   <script src="https://kit.fontawesome.com/67bd86ffdc.js" crossorigin="anonymous"></script>
8   <script src="hamburger.js" defer ></script>
9  </head>
10 <body>
11 <div id="wrapper"> <!-- to style for smaller screen -->
12 <header>
13     <h1>Mimo Yoga Studio</h1>
14     <a id="hamburger">
15         <i class="fa-solid fa-bars"></i>
16     </a>
17 </header>
18 <nav id="mainNav" >
19     <ul class="mainNav">
20         <li class="nav"><a href="index.html">Home</a></li>
21         <li class="nav"><a href="classes.html">Classes</a></li>
22         <li class="nav"><a href="schedule.html">Schedule</a></li>
23         <li class="nav"><a href="contact.html">Contact</a></li>
24     </ul>
25 </nav>
26 <main>
28 </main>
29 <footer>
30     <p>Copyright ©2022 Mimo Yoga Studio <a href="mailto:jchin@abd.com">Jeannette Chin</a>
31
32 </footer>
33 </div>
34 </body>
35 </html>
```

Figure 13. The code in contact.html

c) Change the title to “Mimo Yoga Studio : contact page”. Test this page using Chrome, your output should look similar to Figure 14. There is no content there yet!

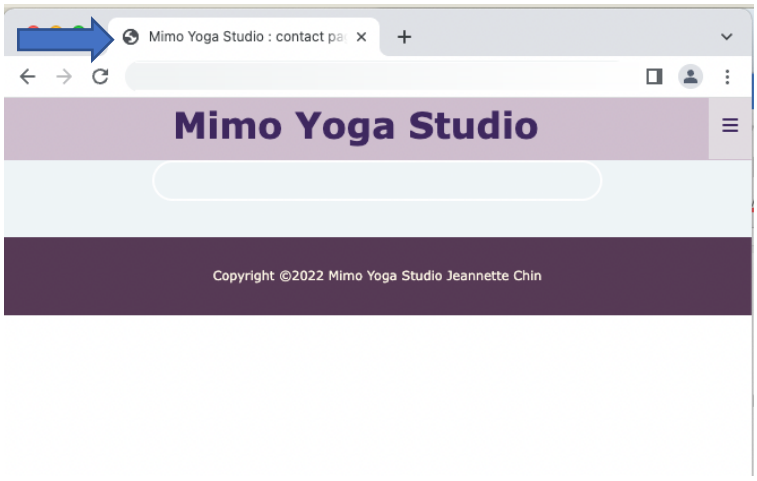


Figure 14. The output of contact.html

d) Next, we will create a web form on this page. In the body of the `<main>` element:
   a. Create a `h3` element with content “Contact Mimo Yoga Studio”

   b. Next, create a `div` element. We will place our `form` inside this `div` block.

   c. In the body of the `div` element that you have just created –
      i. Create a `p` element with this content - “required information is marked with an asterisk (*)”. Configure a class attribute named “form”. We will style this element using CSS later.

      ii. Create a `form` element and 2 attributes: the `method` and the `action` attribute to invoke server-side processing. For this exercise we will use POST `method` and leave the value of the `action` attribute blank. Configure an ID attribute named “form”.

      iii. Create 3 `input` text fields (MDN) for – Name, Email and Message respectively. Each text field should have an appropriate `label` (MDN) to indicate the name of the field. All fields are required. Use an asterisk (*) in the `label` for indication. For email, HTML5 `input` types (MDN) should be used in this exercise. For message, a bigger text box should be used [**Hint:** use `textarea`]. The `required` fields should be enforced by using HTML5 attribute and cater appropriately for accessibility [**Hint**: use `required` and `aria-required` attributes]

      iv. Create a submit `button` (MDN) with text “Send Now”. Configure the ID attribute for this button. Use the word “submit” as the ID.

      v. To improve accessibility, create a `fieldset` element (MDN) and place it inside the `form`. We would also like to add a `legend` element as well. The legend content should be “Contact form”. [**Hint**: `fieldset` element should be placed inside the `form` element, and make sure it is properly nested. The `legend` element should be places just after the `fieldset` element]

      vi. Ensure that your HTML tags are closing properly.

      vii. Now save the file.

viii. Test `contact.html` using Chrome.

ix. Figure 15 below is my output. Yours should look roughly similar. We will use CSS to style the form next.

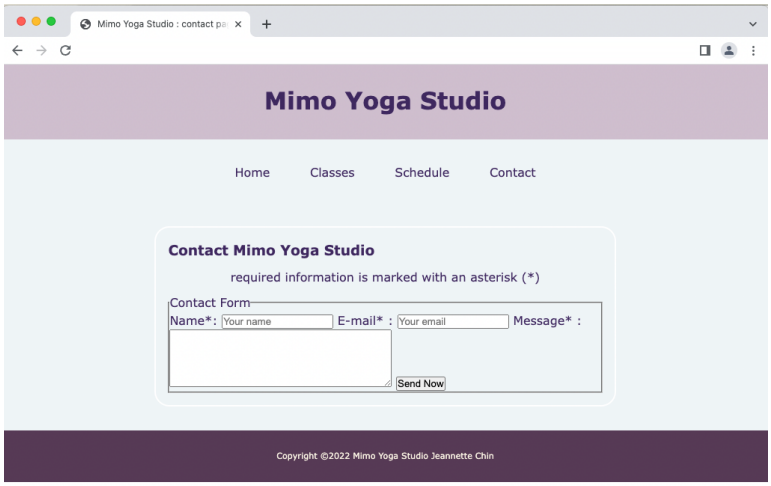


Figure 15. The initial output of contact.html

e) Now we will configure the look and feel of our form using CSS. In your `yoga.css` –

i. Create a CSS rule that affects all `p` elements that has been assigned with a class called "form" – `font-size` is `small`

ii. Create a CSS rule for ID called "form" – set the value of its `display` property to `flex`. (MDN)

iii. Create some CSS rules for `fieldset`:
   1. Configure the `display` property to `flex` (MDN)
   2. Configure the `flex-wrap` property to `wrap` (MDN)
   3. Margin – `top` and `bottom` is auto , `left` and `right` is 2% (MDN)
   4. Width is 100%
   5. Create a `border` with 20px, `solid` line and a colour of your choice (MDN) For colour, I use rgb() (MDN), you can use `hex` values if you wish.
   6. Configure the rounded border property to 5px [Hint: use `border-radius` property] (MDN)

iv. Create some CSS rules for `legend`:
   1. `padding` is 5px (MDN)
   2. `font-size` is medium (MDN)
   3. `word-spacing` is 4px (MDN)

Now save the file and refresh the page. Figure 16 below is my output. Depending on the choice of colour, your output should look similar.

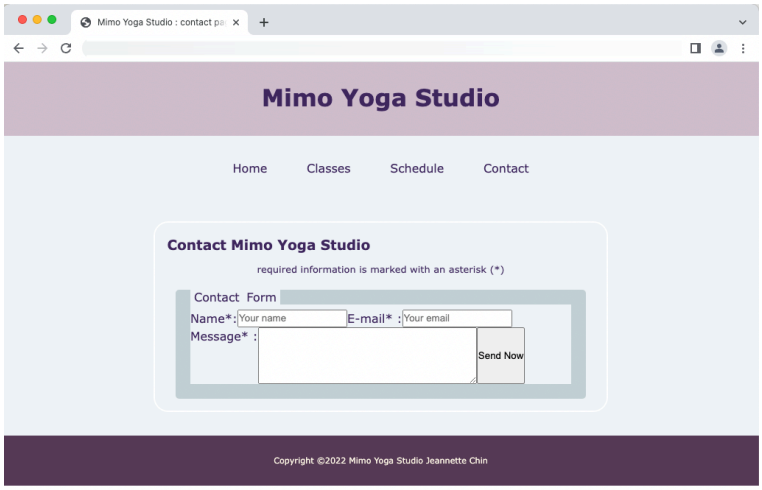


Figure 16. The output of contact.html

Resize the browser window to smaller than 768px, you will see that the contact form is rendered outside the main element, as shown in Figure 17 below. We will fix this later.

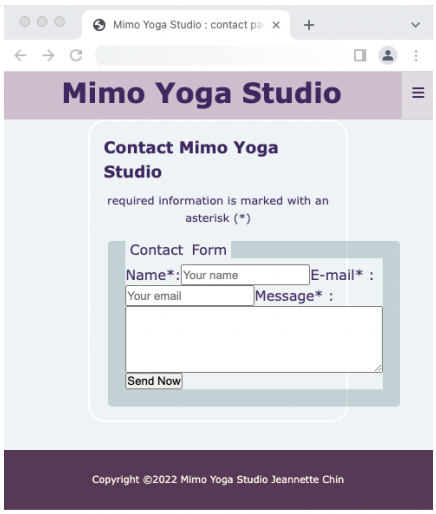


Figure 17. The output of contact.html on a screen size less than 768px

v. Now, create some CSS rules for both `input` and `textarea`:
   1. Configure the `display` property to `block`
   2. `Width` is 95% (MDN)
   3. `Padding` is 10px
   4. `Margin` – top and bottom is 0, left and right is 5px

5. Create a `border` with 1px, `solid` line and a colour of your choice
6. Create a rounded border of 5px
7. Configure the background colour to white (MDN)

vi. Next, we want these textboxes to react to mouseover. Using the CSS pseudo-class `hover` (MDN) configure a different background colour for `input` and `textarea`. [Hint: use `input:hover` and `textarea:hover`]

Now save the file and refresh the page. Figure 18 below is my output for you to compare.

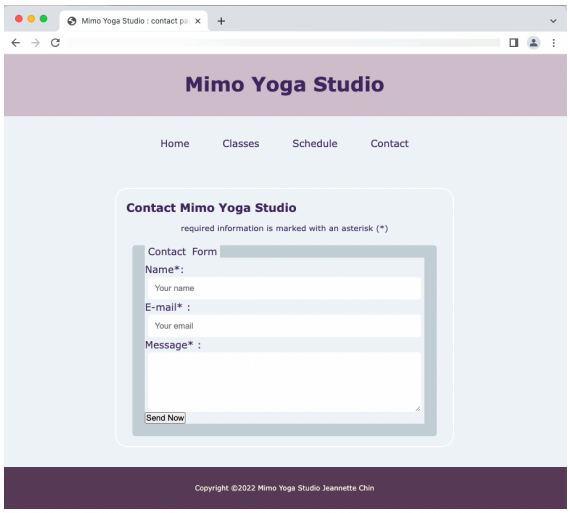


Figure 18. The output of contact.html

Resize the browser window to smaller than 768px, you will see that the contact form is rendered inside the main element, as shown in Figure 19 below.

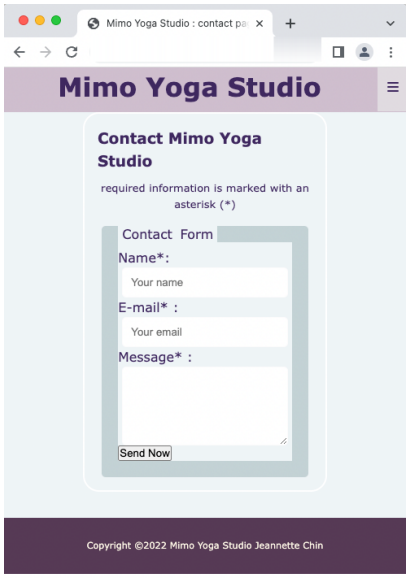


Figure 19. The output of contact.html on screen size smaller than 768px

vii. The labels look rather odd. We will fix this now. Create some CSS rules for `label`:
   1. Configure the `display` property to block
   2. `Width` is 100%
   3. `Margin` is 1%

   Save the file and refresh the page. Figure 20 below is my output for you to compare.

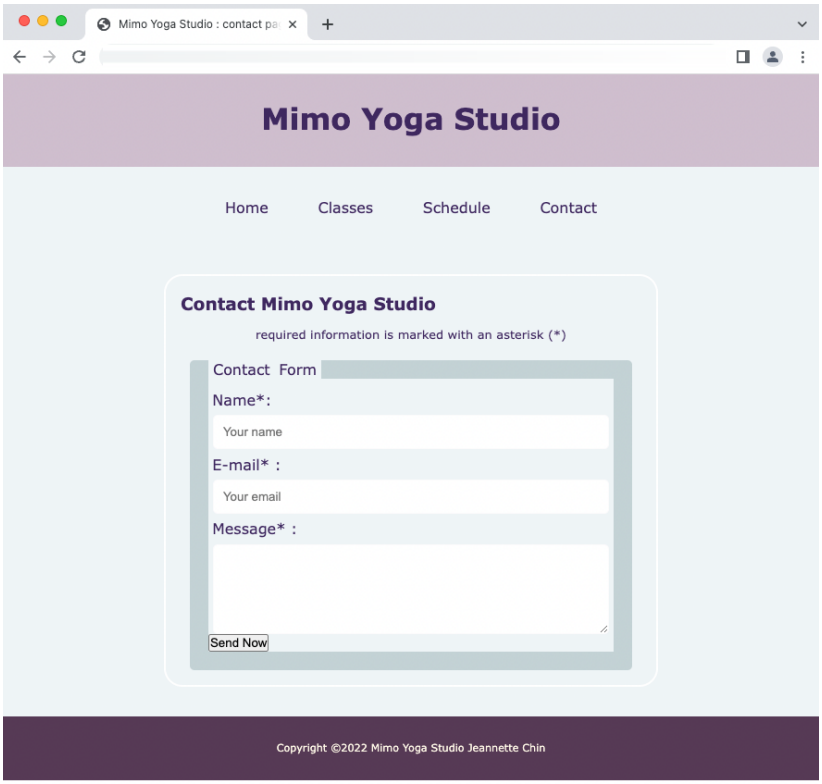


Figure 20. The output of contact.html

viii. Finally, we are getting there. The remaining submit `button` also required attention. Create some CSS rules for this button [**Hint:** use ID submit]:
   1. `Margin`: top and bottom is 2%, left and right is 35%
   2. `Min-Width` is 150px
   3. `Padding` is 5px
   4. Create a border with 1px, solid line and a colour of your choice
   5. Create a rounded border of 5px
   6. Configure the background colour of your choice
   7. Configure the text colour of your choice

Now save the file and refresh the page. Figure 21 below is my output for you to compare.

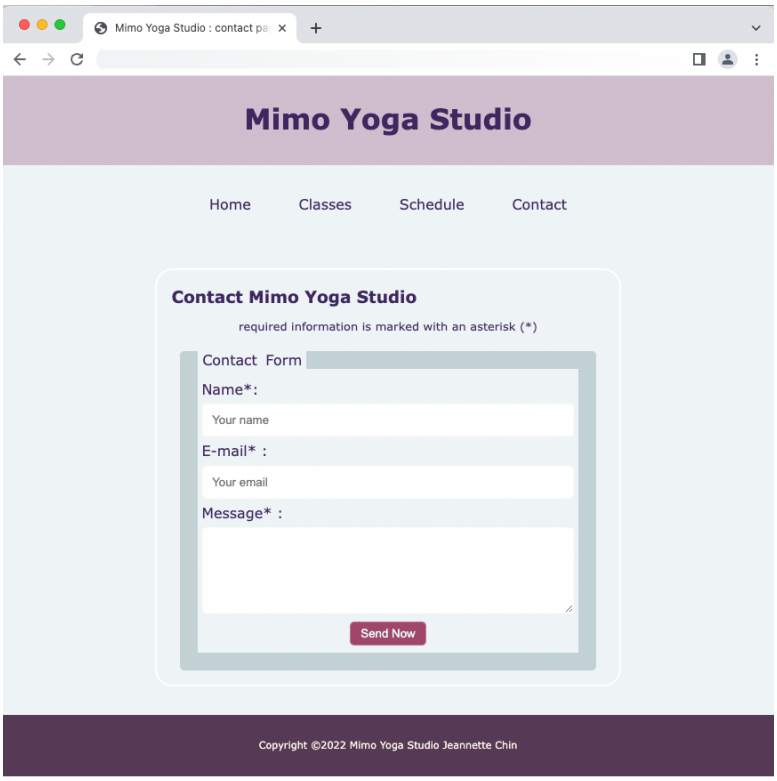


Figure 21. The output of contact.html

Figure 22 below shows the mobile view. The main components of the form are all fitting in nicely.

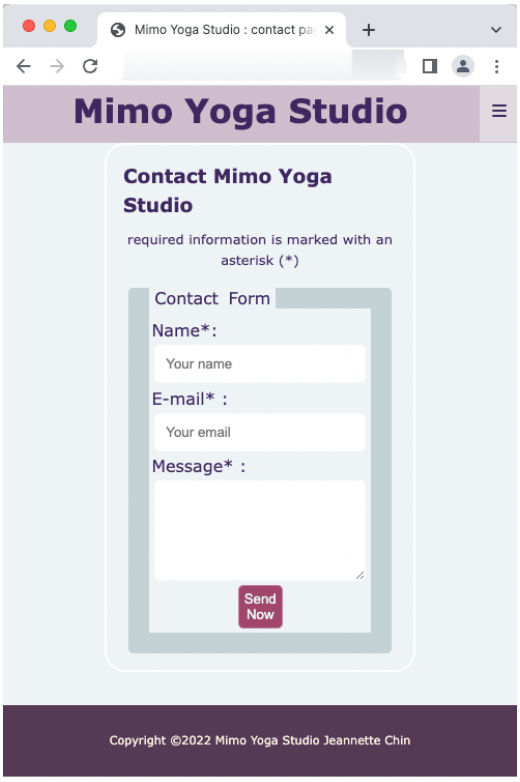


Figure 22. The output of contact.html on screen size smaller than 768px

However, for smaller screen size or mobile view, we should configure the font size appropriately using media query. Your next task to configure the font sizes using the media query you have created in previous lab.

f) Login to your fontawesome account and select appropriate web icons for this form. I have used 3 web icons – a message for my legend, envelop for my email and a right double arrow for the send button.

Figure 23 below is my final output:

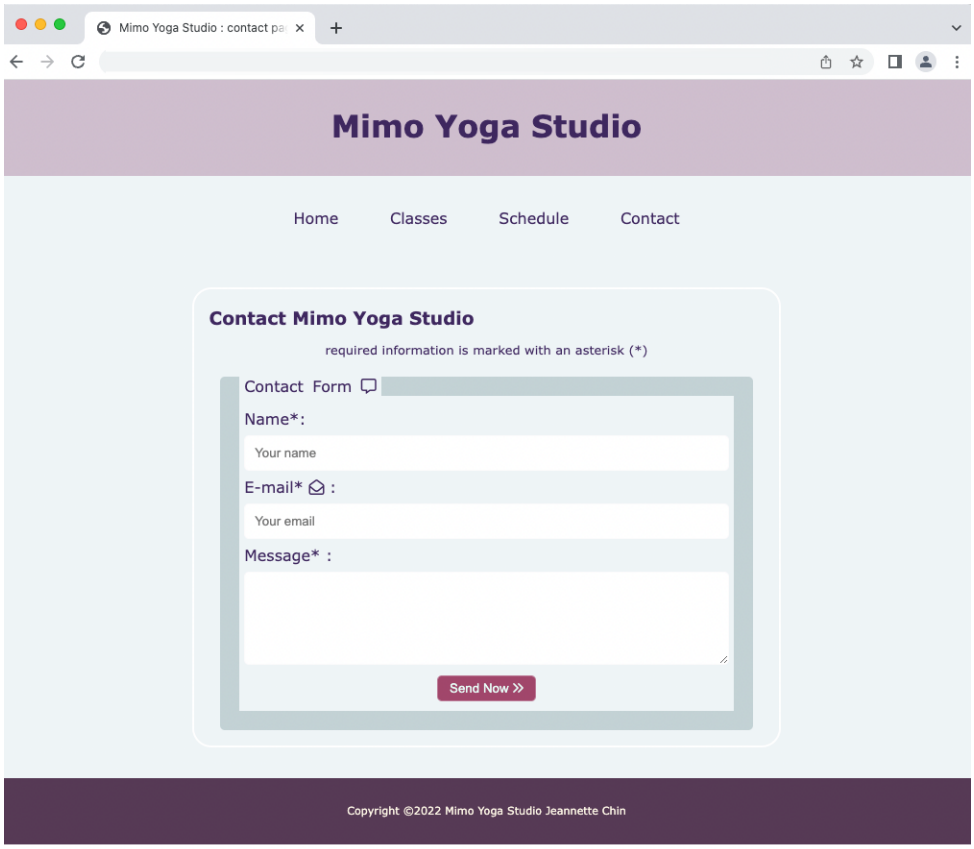


Figure 23. The final output of contact.html

g) Signposting is very important in terms of Web usability. It provides the information people need know to make their own decisions while they are on a website. For example, on which page they are currently visiting, and where they can go after. Often signposting refers to the navigation of the website. A good navigation system will give the visitors a better experience as the chances of them having to get lost or confused while visiting the website will be minimum. Observe our main navigation is in a prominent location (at the top) of the web pages. However, there is no indication on which page of the website the user is currently viewing. We will fix this by creating a class called "active".

1. For HTML pages - add an additional `class` called "*active*" to the navigation link. For example - on `index.html`, the "*active*" class should be added to the anchor tag link to this page, as shown in Figure 24.

```
<ul class="mainNav">
    <li class="nav active"><a href="index.html">Home</a></li>
    <li class="nav"><a href="classes.html">Classes</a></li>
    <li class="nav"><a href="schedule.html">Schedule</a></li>
    <li class="nav"><a href="contact.html">Contact</a></li>
</ul>
```

Figure 24. Add "active" class to navigation on index.html

2. Repeat the same for links go to `classes.html` and `contact.html` pages, as follows:

```
<ul class="mainNav">
    <li class="nav"><a href="index.html">Home</a></li>
    <li class="nav active"><a href="classes.html">Classes</a></li>
    <li class="nav"><a href="schedule.html">Schedule</a></li>
    <li class="nav"><a href="contact.html">Contact</a></li>
</ul>
```

Figure 25. Add "active" class to navigation on classes.html

```
19      <ul class="mainNav">
20          <li class="nav"><a href="index.html">Home</a></li>
21          <li class="nav"><a href="classes.html">Classes</a></li>
22          <li class="nav"><a href="schedule.html">Schedule</a></li>
23          <li class="nav active"><a href="contact.html">Contact</a></li>
24      </ul>
25  </nav>
```

Figure 26. Add "active" class to navigation on contact.html

3. On `yoga.css`, create a `class` named `active`, and a rule to indicate the `background colour` is #bcc3b6 (or choose a colour of your choice). For me, I use `rgb(81,102,97, .3)` (MDN)

4. Save the files.

5. Test the web pages using Chrome. Figures 27 – 29 below are my outputs for you to compare.

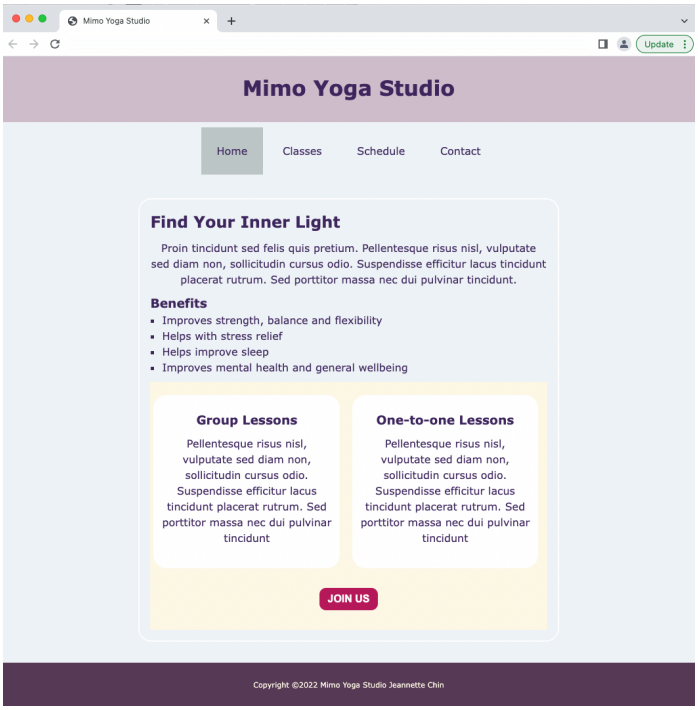


Figure 27. Signpost on index.html

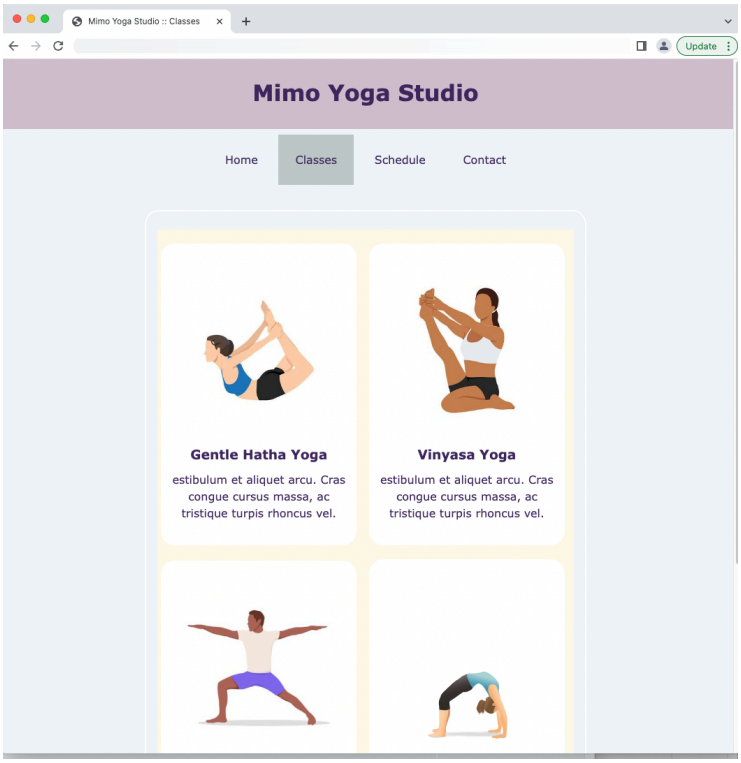


Figure 28. Signpost on classes.html

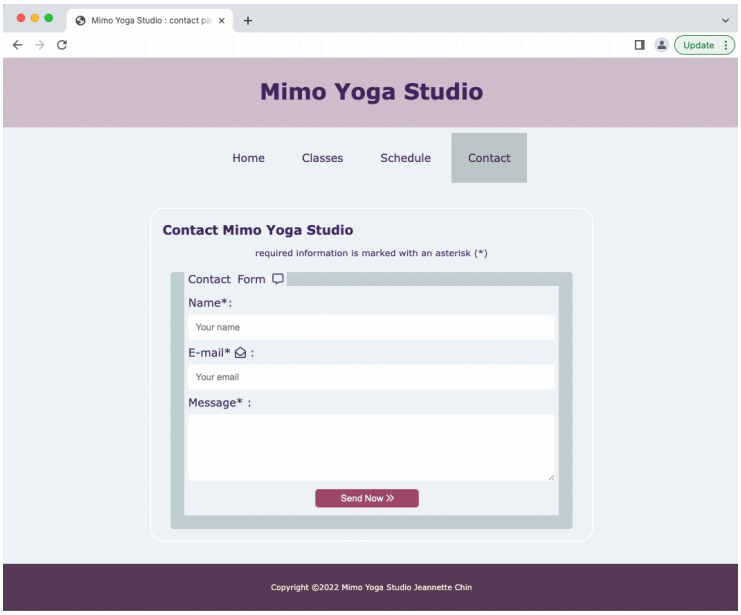


Figure 29. Signpost on contact.html

Now that we have created our form, we will write some JavaScript code to process this form later in the course.

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.