[IS216] Extra Exercises - Week 5 - Javascript DOM

Objectives

- To understand how dynamic, interactive web pages can be built using JavaScript
- To use JavaScript to build dynamic, interactive HTML web pages.

Instructions

- Questions with no asterisk mark are easy peasy.
- Questions marked with * are slightly challenging.
- Questions marked with ** are challenging.
- Questions marked with *** are very challenging.

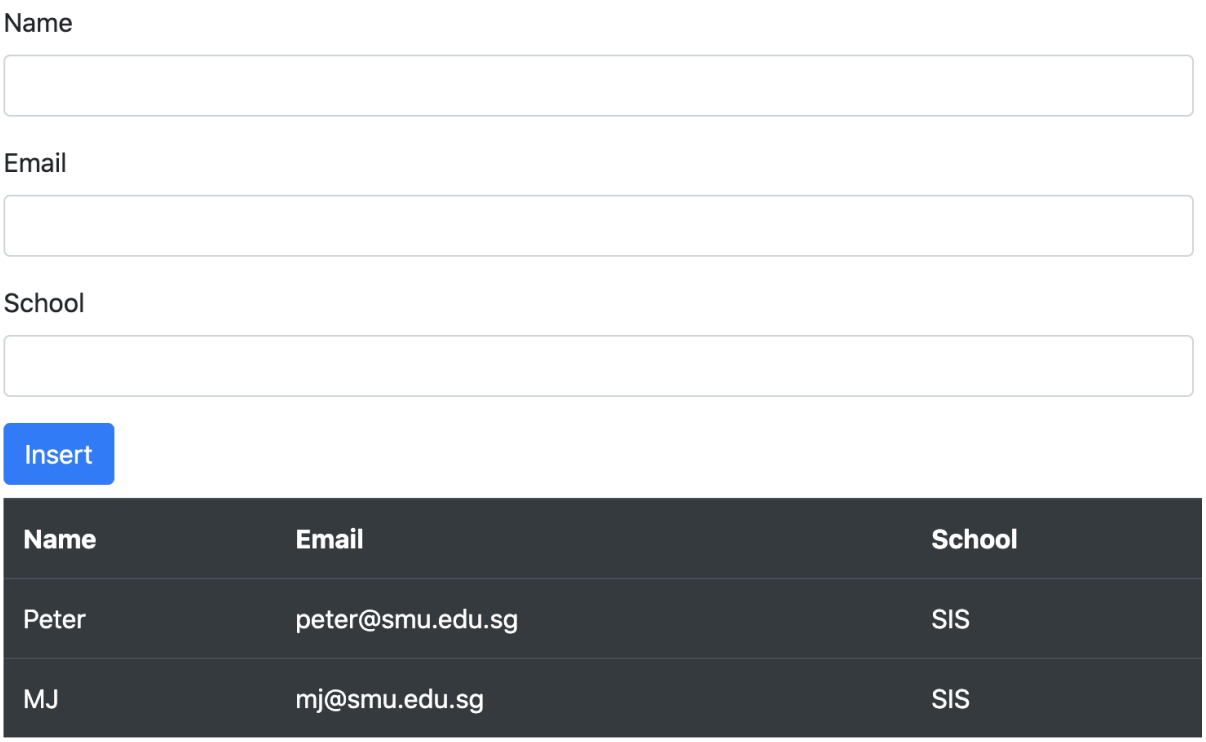
NOTE: If you spot any mistakes/errors in the questions, please contact your instructors by email and state the issues. We will try to address it as soon as possible.

Question 1 – Students Table (**)

Go to `student_table` directory.

```
|-- student_table
      |-- table.html (to do)
```

`table.html` contains a form that accepts the user's name, email, and school information. It contains a table that shows the information of two students initially.

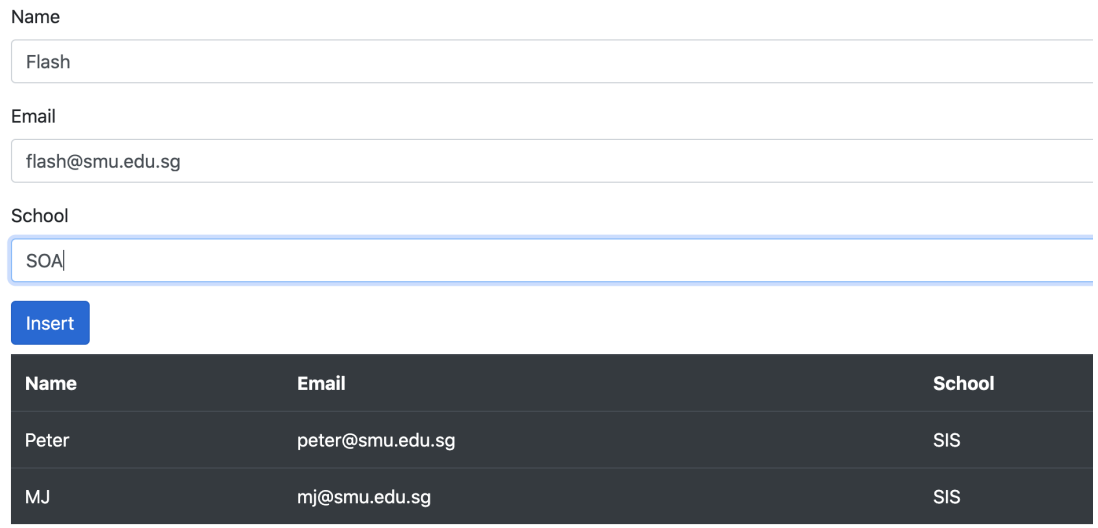


Add code in `table.html` such that the JavaScript function `insert()` is executed when the "insert" button is clicked. The `insert()` function gets the values from the name, email, and school input fields in the form and add those values as a row in the table.

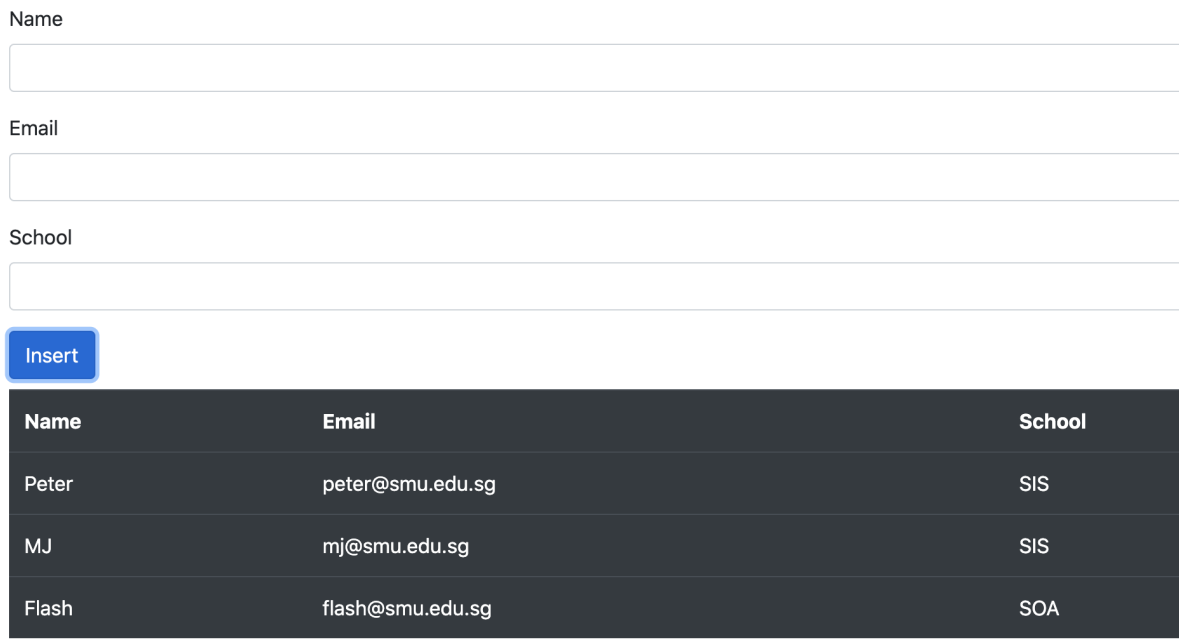
Hint: explore built-in functions `insertRow()` and `insertCell()`

For example,

The user enters the information of a student:



Then, when the user clicks the “insert” button, the new student’s information is added at the last row of the table. Notice that the input fields become empty after that.

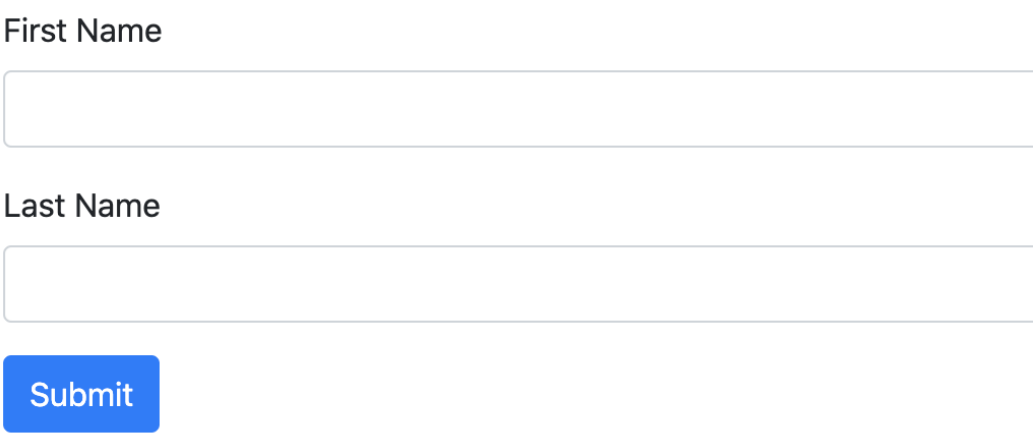


Question 2 – Form (*)

Go to `form` directory.

```
|-- form
      |-- form.html (DO NOT MODIFY)
      |-- form.js (to do)
```

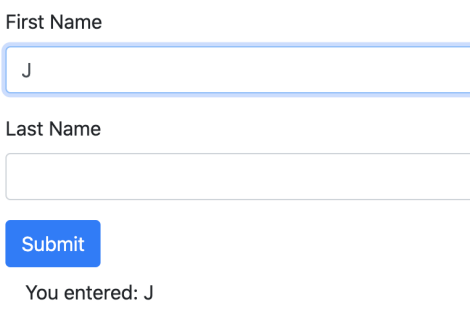
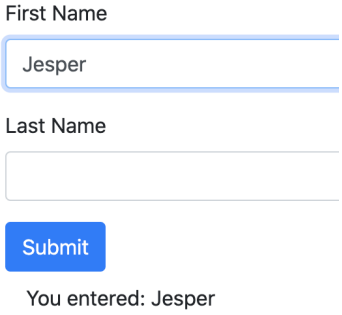
`form.html` contains a form that accepts the user's first name and last name.

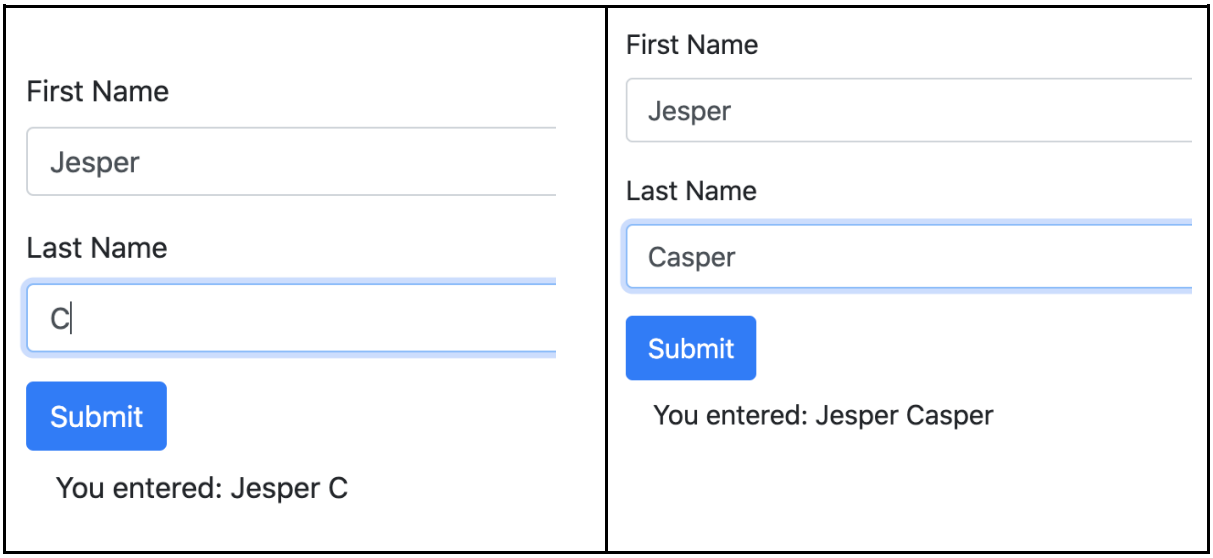


Task A: Write a JavaScript function called `getName()` in `form.js`, which gets the values of the first and last names currently entered in the input fields and displays them in the paragraph with id="name", given in `form.html`.

Task B: Add code in `form.js` that invokes the function `getName()` whenever a user writes something in any of the two input fields in `form.html`.

For example,

As soon as the user enters the character 'J' in First Name field: 	As the user enters the characters "Jesper" 
As the user enters the character 'C' in Last Name field:	As the user enters the characters "Casper"


First Name
Jesper
Last Name
C
Submit
You entered: Jesper C
First Name
Jesper
Last Name
Casper
Submit
You entered: Jesper Casper

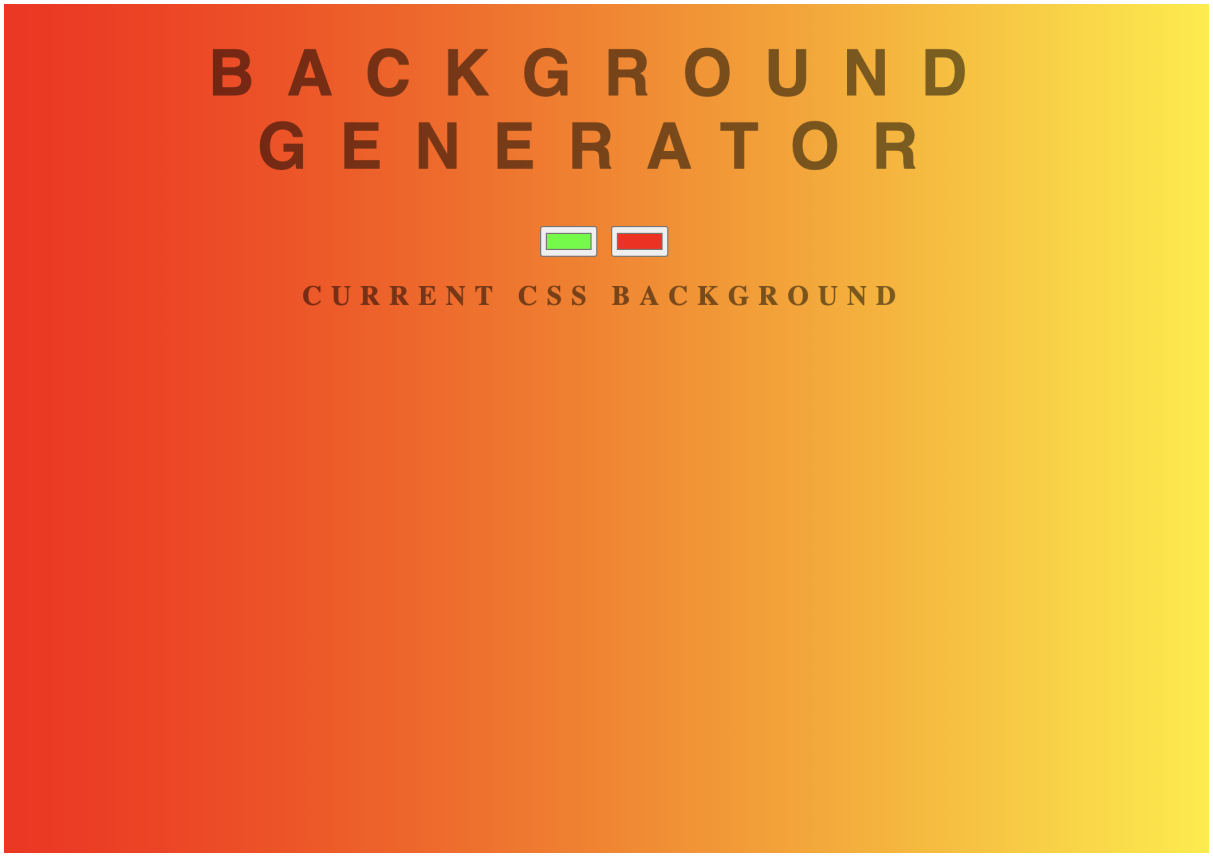
Question 3 – Background Generator (**)

Go to `background_generator` directory.

```
|--background_generator
      |-- script.js (to do)
      |-- background_generator.html  (to do)
      |-- style.css
```

Add HTML code in `background_generator.html` and Javascript code in `script.js` such that `background_generator.html` produces the following interactive behavior.

Initially `background_generator.html` shows the following web page. It contains two color-type input fields. The initial color value of the first input field is `#00ff00` and the initial color value of the second input field is `#ff0000`.

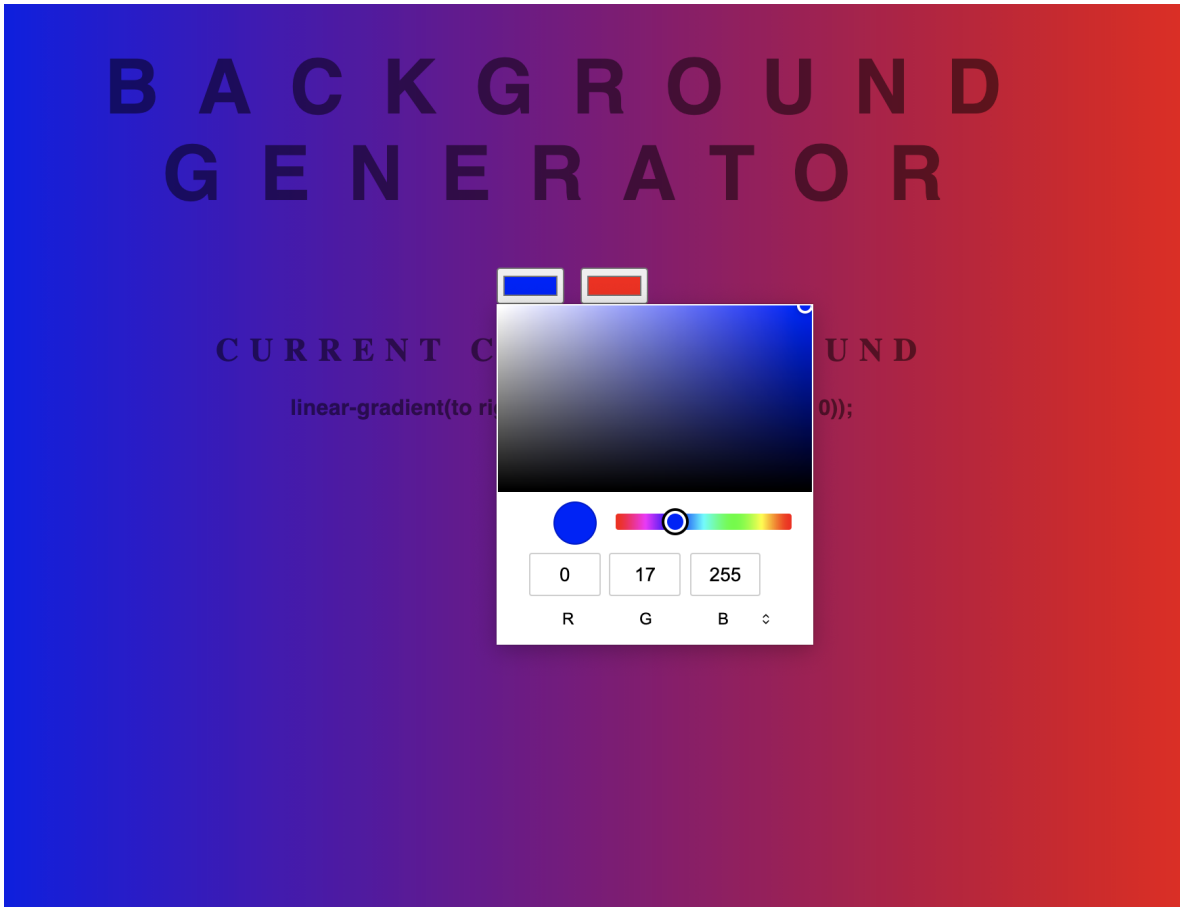


In style.css, the background property of HTML body is given as
`background: linear-gradient(to right, red , yellow);`
which specifies the linear-gradient background with two colors -- red and yellow.

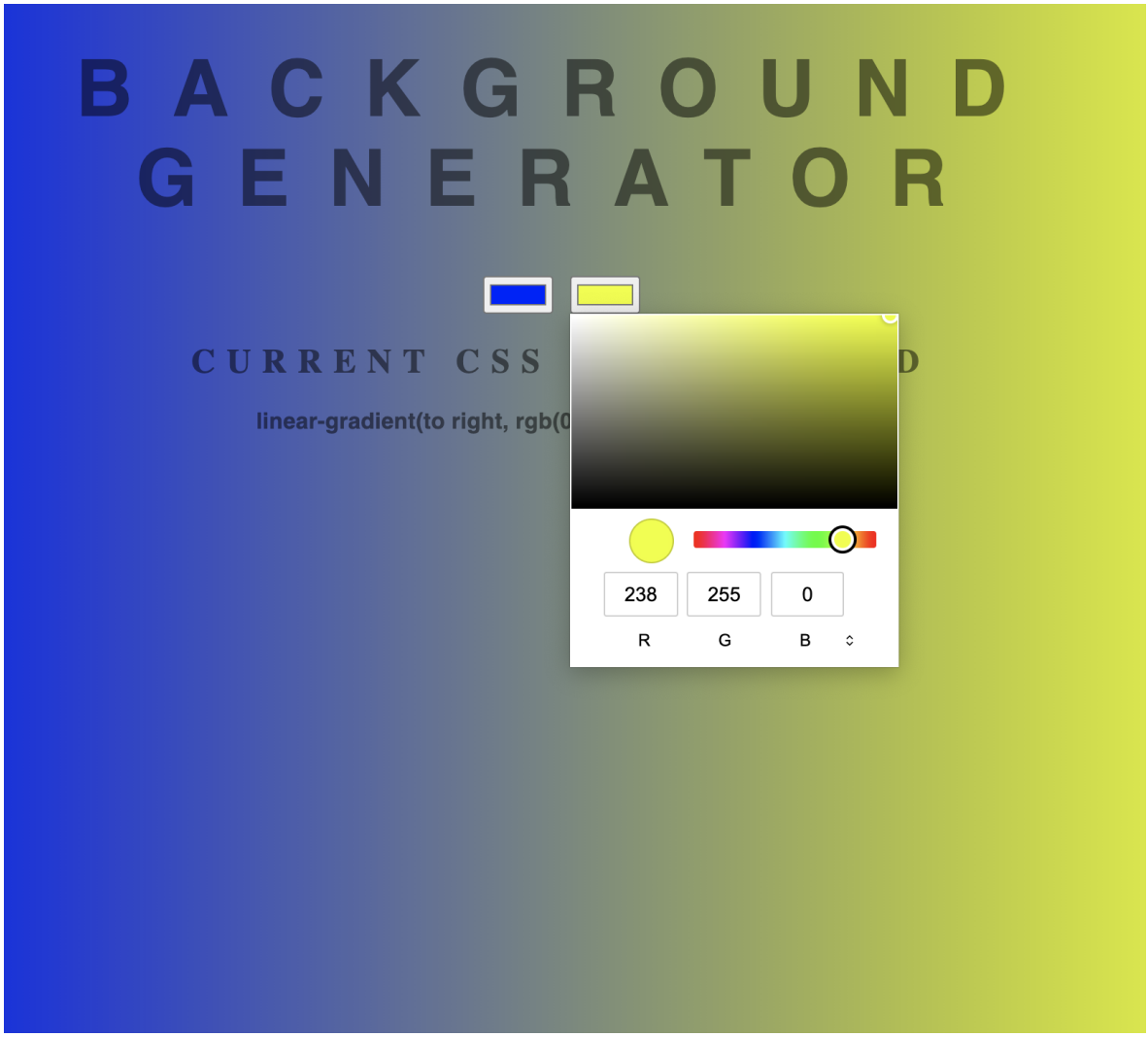
When the user selects a color from the first input field, the first color in linear-gradient background changes accordingly. Likewise, when the user selects a color from the second input field, the second color in linear-gradient background changes accordingly.

For example:

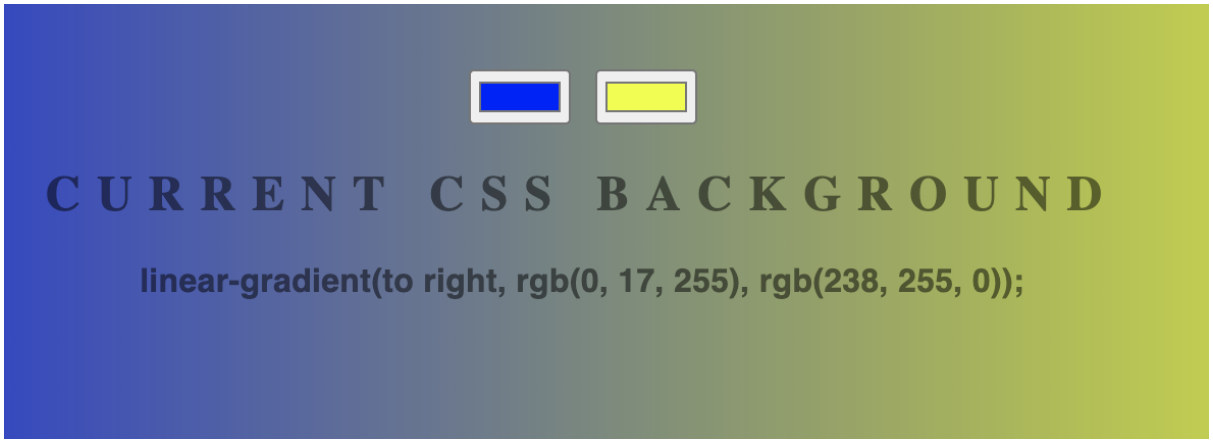
When the user selects ‘blue’-ish color from the first input field:



When the user selects ‘green’-ish color from the second input field:



Notice that the web page also shows the linear-gradient background property with currently-selected color values dynamically (when the user selects a color).

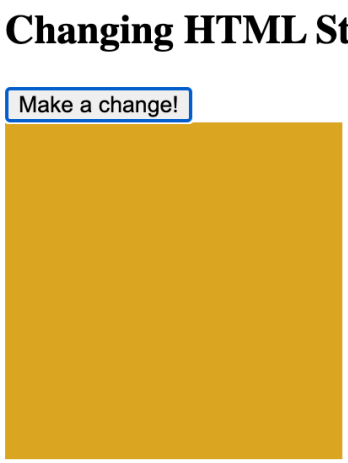


Question 4 – Change Style (**)

Go to `change_style` directory.

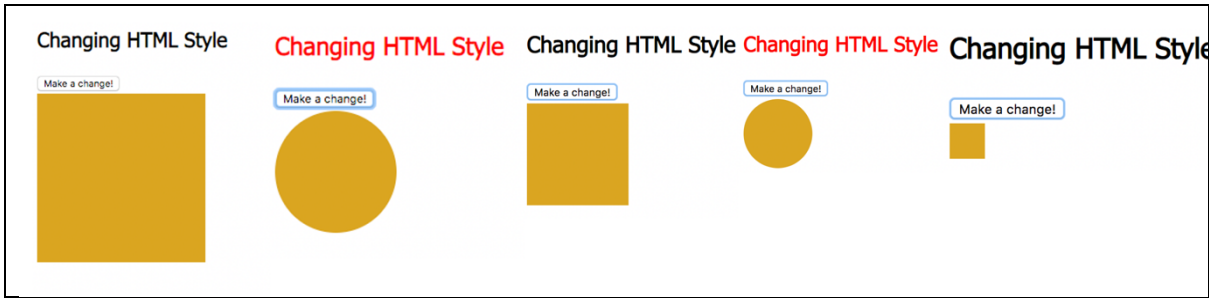
```
|-- change_style
      |-- change_style.html (to do)
```

`change_style.html` shows the following web page.



Task: Add CSS and Javascript code in `change_style.html` so that:

- When the "Make a change!" button is clicked, the web page changes the text color of the heading to red and it shows a circle. And when the button is clicked again, the web page changes the text color of the heading to black and it shows a square. This repeats for every button clicks.
- Everytime the button is clicked, the size of the square or the circle is also reduced by 10px.
- After clicking the button 20 times, the size of the shape is reset to its original size.

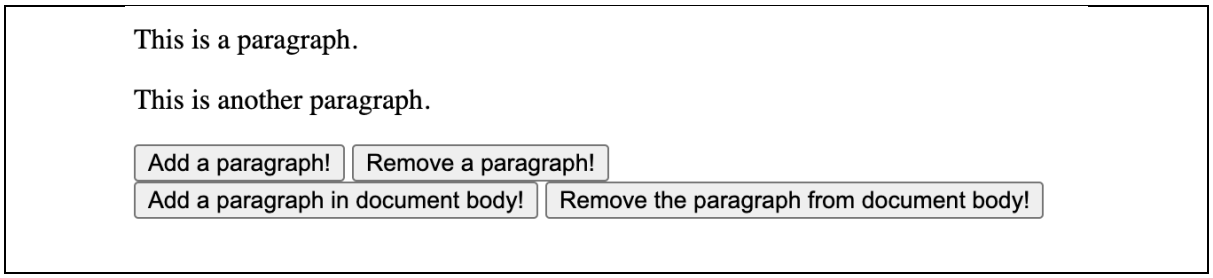


Question 5 – Add & Remove Paragraphs (**)

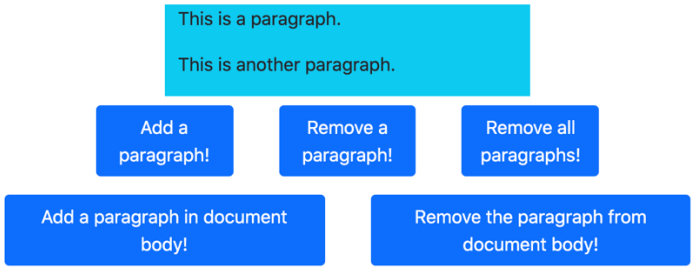
Go to `add_paragraph` directory.

```
|-- add_paragraph
      |-- add_para.html (to do)
```

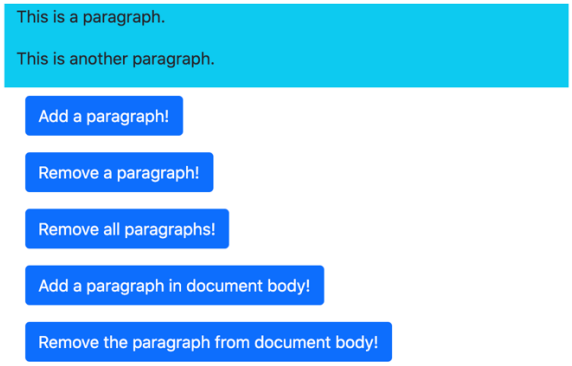
`add_para.html` shows the following web page.



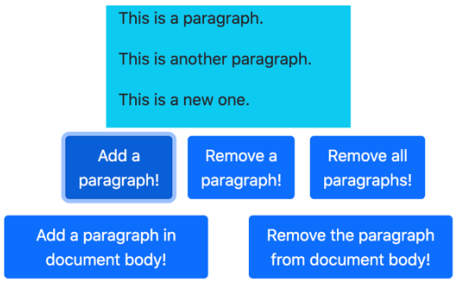
Task A: Add CSS/Bootstrap in `add_para.html` such that it shows the following web page. The content must be aligned in the centre.



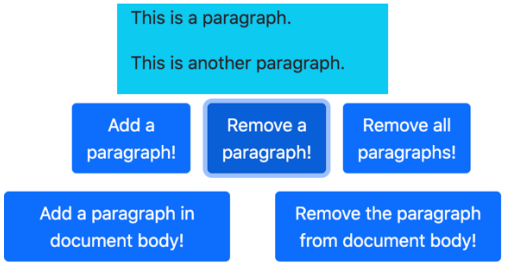
Use appropriate responsive setting such as when the web page is viewed in the “medium” device size, the contents are stacked over each other, as shown.



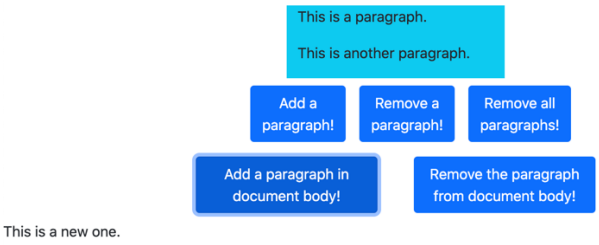
Task B: implement add() function such that when a user clicks on “Add a paragraph!” button, a paragraph is added into the highlighted region.



Task C: implement remove() function such that when the user clicks on “Remove a paragraph!” button, the last paragraph in the highlighted region is removed.



Task D: implement a new function such that when the user clicks on “Add a paragraph in document body!” button, a paragraph is added in the document body, outside the highlighted region:



Task E: implement a new function such that when the user clicks on “Remove the paragraph in document body!” button, the last paragraph in the document body, outside the highlighted region, is removed.