HTML meets 🤝 JS :

Till now, we've built static web pages. That means our webpage did not interact with the user. With the interaction of Java-script, we can make our webpage interactive.

How to add Java-Script inside HTML:

JavaScript can be embedded within a page (internal script) or placed in an external script file (external script file with .js extension). However, for it to function in the browser, JavaScript must be enabled.

1. **Internal script:**
    a. Inside the **<head>** tag of our HTML document.
    b. Inside the **<body>** tag, as a last item.(before ending the body tag).

2. **External script:**
    c. Inside the **<head>** tag of our HTML document.
    d. Inside the **<body>** tag, as a last item.(before ending the body tag).

Difference between adding Java-script inside <head> tag and inside <body> tag:

The main difference lies in **when the JavaScript code is executed** and its impact on the **loading and rendering** of the web page.

JavaScript Inside `<head>` Tag:

- JavaScript inside the `<head>` tag is loaded and executed before the browser renders the page's HTML content.
- It is commonly used for:
    - Initializing variables or configurations needed early in the page's lifecycle.
    - Defining functions that will be executed later, triggered by events.
- While this approach allows JavaScript to load early, it can delay page rendering if the script is large or takes a long time to execute.

Example:

```
<!DOCTYPE html>

<html lang="en">


<head>

    <meta charset="UTF-8">

    <title>JavaScript in Head</title>

    <script>

        console.log("Welcome to JS in the <head> tag.");

    </script>

</head>

<body>

    <h1>Welcome to Chitkara</h1>

</body>

</html>
```

JavaScript Inside <body> Tag

- JavaScript inside the <body> tag is **loaded and executed after** the browser has loaded the entire HTML content.
- This ensures that all elements are available for manipulation by the script.
- Placing JavaScript at the **end of the <body> tag** can improve performance because the visible content is rendered first, enhancing the user's experience.

```
<!DOCTYPE html>

<html lang="en">



<head>

    <meta charset="UTF-8">

    <title>JavaScript in Body</title>

</head>

<body>

    <h1>Welcome to Chitkara</h1>

    <script>

        console.log("Hello, World! Script in the <body> tag.");

    </script>

</body>

</html>
```

Using an External JavaScript File

To keep your code organized, you can write JavaScript in a separate file with a `.js` extension, for example, `myscript.js`.

Example:

myscript.js

```
console.log("Welcome to JS");
```

After this we can include this file inside our HTML document.either inside the **<head>** tag or inside the **<body>** tag.

Example:

Inside the **<head>** tag:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <title>JavaScript in Body</title>

    <script src="myscript.js"></script>


</head>

<body>

    <h1>Welcome to Chitkara</h1>

</body>

</html>
```

Inside the **<body>** tag:

```html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <title>JavaScript in Body</title>
</head>

<body>

    <h1>Welcome to Chitkara</h1>


    <script src="myscript.js"></script>
</body>

</html>
```

Key Points to Remember:

1. **Use `<head>` for scripts that need to run early or initialize global variables/functions.**
2. **Use `<body>` for scripts that interact with or manipulate the HTML content.**
3. **Use the `defer` attribute to execute scripts in `<head>` after the page is fully loaded.**
4. **External JavaScript files help organize and reuse code across multiple HTML pages.**

Advantages of Using JavaScript as an External File:

- No repetition needed if the script is used by multiple pages.
- Updates can be made in one place, reflecting across all referencing pages.

- Keeps HTML pages cleaner and easier to read.
- External JavaScript files can be cached by the browser, leading to faster loading times for subsequent visits.
- Promotes separation of concerns, making it easier to maintain and understand both HTML structure and JavaScript logic.
- Facilitates collaboration in teams by allowing different members to work on HTML and JavaScript independently.

What is DOM? (Document Object Model):

DOM stands for **Document Object Model**. It is a programming interface for web documents, representing the structure of a webpage so that it can be accessed and manipulated using programming languages like JavaScript.

An Analogy to Understand DOM:

Imagine you're watching TV, and you don't like the show that's on. You want to change the channel or increase the volume. How do you do that?

- You use a **remote control**.
  The remote serves as a **bridge** that allows you to interact with the television.

Similarly, JavaScript acts as a "remote control" that allows you to interact with an HTML page via the DOM.

How the DOM Works

When a browser loads an HTML document, it creates the **DOM**, which is a **tree-like structure** of the document.

- The DOM represents each element of the webpage (such as headings, paragraphs, buttons) as **nodes** or **objects**.

- This structure allows JavaScript to:
    - Access elements in the document.
    - Modify their content, style, and structure.
    - Handle user events like clicks, key presses, etc.

Features of the DOM

The DOM is **object-oriented**, meaning it allows programming languages to interact with the page elements as objects.

The **HTML DOM** (specific to HTML documents) defines:

1. **HTML elements as objects:** Every element (e.g., `<div>`, `<button>`) is treated as an object.
2. **Properties of HTML elements:** Attributes and styles of elements can be accessed and modified (e.g., `innerHTML`, `className`).
3. **Methods to access elements:** Functions like `getElementById()` or `querySelector()` are used to find elements.
4. **Events for HTML elements:** Methods to handle interactions like clicks (`onclick`) or key presses (`onkeypress`).

Why is the DOM Useful?

The DOM enables developers to:

- Change the content of elements dynamically.
- Modify the style of elements.
- Add, remove, or rearrange elements on the page.
- React to user interactions, such as clicks or form submissions.

Example:

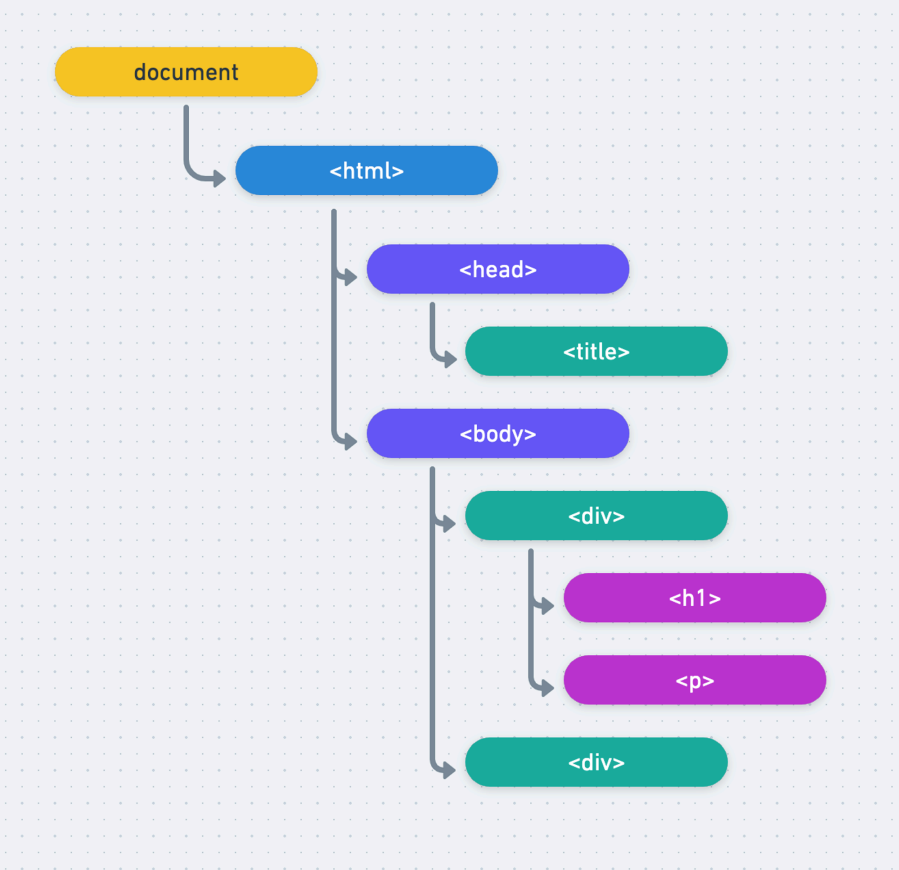
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>DOM Example</title>

</head>
<body>
    <h1>Understanding the DOM</h1>
    <button id="myButton">Click Me</button>
    <p id="myText">This is some text.</p>

    <script>
        // Accessing the button and text using the DOM
        const button = document.getElementById("myButton");
        const text = document.getElementById("myText");

        // Adding an event listener to change button color and text
content
        button.addEventListener("click", () => {
            button.style.backgroundColor = "green"; // Change button color
            text.textContent = "The text has been updated!"; // Change
text
        });
    </script>
</body>
</html>
```

Let's suppose you have this code

```
<html>
  <head>
    <title>JS is easy</title>
  </head>
```

```
  <body>

    <div>

      <h1>Title of page</h1>
      <p>Some description here</p>

    </div>

  </body>
</html>
```

If you open this in the browser, the web browser creates a DOM of the webpage when the page is loaded. The DOM model is created as a tree of objects like this:



Remember: console.log() in javascript is your best friend life long

console.dir():

The console.dir() is similar to the console.log() function only, but using this console.dir() function we can see an interactive listing of all properties of a specified JavaScript object. It shows the properties of the object in a collapsible tree format, allowing you to explore the object's structure in detail.
Here "dir" stands for "directory."

Example:

```
<!DOCTYPE html>
<html>

<head>
    <title>My Text</title>
</head>

<body>
    <h1>My Header</h1>
```

```
    <p>My Paragraph</p>


    <script>
        console.log(document);
        // console.dir(document);
    </script>

</body>

</html>
```

Properties and methods of the **document** object:

Inside this **document** object there are some properties and some methods are there using which we can access any other html element reference and we can manipulate it.

Example:

```
<!DOCTYPE html>
<html>


<head>
    <title>My Text</title>
</head>


<body>
    <h1>My Header</h1>
    <p>My Paragraph</p>



    <script>
```

```
        //Using children property is used to access immediate
children of an element.
        console.dir(document.children[0]); //html

        //Accesing the <body> element
        console.dir(document.children[0].children[1]); //body

        //Accessing the <h1> element
        console.dir(document.children[0].children[1].children[0]);
//h1

        //Manipulating the H1 element.
   document.children[0].children[1].children[0].innerHTML =
"<strong>I am Changed..</strong>";
        // document.children[0].children[1].children[0].innerText =
"<strong>I am Changed..</strong>";

    </script>

</body>

</html>
```

Difference between **innerText, innerHTML and textContent** properties:

1. `innerText`

- Represents only the **visible text** within an element, **ignoring hidden text** or extra spaces.
- **Excludes hidden elements** (e.g., those styled with `display: none`).
- Useful for working with **text content as it appears on the screen**.

2. `innerHTML`

- Retrieves or sets the **HTML content** of the specified element, including any **HTML tags** or elements.
- Provides access to both text and the **HTML structure** within the element.

- Commonly used when you need to **dynamically manipulate HTML content** or insert new HTML elements.

3. `textContent`

- Retrieves or sets the **entire text content** of an element, **including hidden text** (e.g., elements with `display: none`) and extra spaces.
- Ignore any HTML structure or formatting.
- Ideal for working with raw text content.

Example:

```
<!DOCTYPE html>
<html lang="en">


<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Document</title>
</head>


<body>

        <p id="p1">

            This is the Line 1


                This is the Line 2


            <strong>Welcome to js</strong>

            <span style="display: none;">This is the hidden
text</span>
```

```
        </p>

    <script>


        let d1= document.children[0].children[1].children[0];
        console.log(d1);

        // console.log(d1.innerText);
        // console.log(d1.textContent);
        // console.log(d1.innerHTML);


    </script>

</body>

</html>
```

- **Use `innerText` for visible text only.**
- **Use `innerHTML` when you need access to or want to modify the HTML structure.**
- **Use `textContent` for raw text content, including hidden text.**

Note: Accessing the reference of any element by the above way can be a tedious process. However, the **document** object provides several methods to make this task easier:

```
// Returns a reference to the element by its ID.
document.getElementById("someid");

// Returns an array-like object of all child elements that have all of the given class names.
document.getElementsByClassName('someclass');

// Returns an HTMLCollection of elements with the given tag name.
document.getElementsByTagName("li");
```

// Returns the first element within the document that matches the specified group of selectors.

document.querySelector(".someclass");

// Returns a list of elements within the document that match the specified group of selectors.

//Here we can target any element by using any kinds of selectors or combinators.

document.querySelectorAll('.class, #id');

Examples:

document.getElementById("someid");

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>

<body>

    <h1 align="center">Welcome to Chitkara</h1>

    <div class="box box1">Item1</div>
    <div class="box box2">Item2</div>
    <div class="box box3">Item3</div>
    <div class="box box4">Item4</div>

    <p id="p1">This is paragraph 1</p>


    <script>

        let p_element = document.getElementById("p1");
        console.log(p_element);
        // console.dir(p_element);
```

```
        // console.log(p_element.textContent);
        // console.log(p_element.innerText);
        // console.log(p_element.innerHTML);

        p_element.innerHTML = "<strong>This text is changed </strong>"


    </script>

</body>

</html>
```

document.getElementsByClassName('someclass');

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>

<body>

    <h1 align="center">Welcome to Chitkara</h1>


    <div class="box box1">Item1</div>
    <div class="box box2">Item2</div>
    <div class="box box3">Item3</div>
    <div class="box box4">Item4</div>

    <p id="p1">This is paragraph 1</p>

    <script>
```

```
        let boxes = document.getElementsByClassName("box");

      console.log(boxes);//HTMLCollection(4) array of all elements whose
class is "box";

        console.log(boxes[1]); //2nd div element
        console.log(boxes[1].textContent); // Item2


        //to get the text content for all the elements whose class name is
box:
        for (let i = 0; i < boxes.length; i++) {
            console.log(boxes[i].textContent);
        }


        //using forEach function
        // boxes.forEach(element => {
        //     console.log(element.innerText);
        // });

    </script>

</body>

</html>
```

Note: **HTMLCollections** object is the array-like but not the Java-script array, Unlike Java-script arrays, this object do not have **forEach** or other functions like push(), pop() function.

To make it work with **forEach**, we can convert this object to a **Java-script array**.

Example:

```
console.log(boxes instanceof Array); //false

//Converting HTMLCollection to traditional Java-script array
```

```
boxes= Array.from(boxes);

console.log(boxes instanceof Array); //true

boxes.forEach(element => {
    console.log(element.textContent);
});
```

document.getElementsByTagName("tagname");

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>

<body>

    <h1 align="center">Welcome to Chitkara</h1>


    <div class="box box1">Item1</div>
    <div class="box box2">Item2</div>
    <div class="box box3">Item3</div>
    <div class="box box4">Item4</div>

    <p id="p1">This is paragraph 1</p>




    <script>

        let divs= document.getElementsByTagName("div");
```

```
        console.log(divs);
        console.log(divs[0]);
        console.log(divs[0].textContent);

        //Iterating using traditional for loop
        // for(let i=0;i<divs.length;i++){
        //     console.log(divs[i].textContent);
        // }


        //Iterating using forEach function
        divs= Array.from(divs);
        divs.forEach(d => console.log(d.textContent));



    </script>

</body>

</html>
```

document.querySelector(".someclass");

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>

<body>

    <h1 align="center">Welcome to Chitkara</h1>
```

```
    <div class="box box1">Item1</div>
    <div class="box box2">Item2</div>
    <div class="box box3">Item3</div>
    <div class="box box4">Item4</div>

    <p id="p1">This is paragraph 1</p>
    <p id="p2">This is paragraph 2</p>




    <script>

        let id_p1_element = document.querySelector("#p1");

        console.log(id_p1_element);
        console.log(id_p1_element.textContent);

        //here it will return only first element.
        let b = document.querySelector(".box");
        console.log(b);
        console.log(b.textContent); //Item1


    </script>

</body>

</html>
```

document.querySelectorAll('.class, #id');

Case1:

```
  //It will target all the elements whose class name is box.
```

```
let boxes= document.querySelectorAll(".box");

 console.log(boxes); //NodeList
 console.log(boxes[0]); // 1st div element (1st div node)
 console.log(boxes[0].textContent); // Item1
```

Case2:

```
//It will target all the elements whose class name is box or the id is p1.
let boxes= document.querySelectorAll(".box, #p1");

 console.log(boxes); //NodeList
 console.log(boxes[4]); // 5th node element which is p element
 console.log(boxes[4].textContent); // This is paragraph 1
```

Note: The **querySelectorAll()** method returns a **NodeList** object, which **resembles an array but isn't strictly an array**. While it lacks typical array methods like **push(), pop()** etc, it does possess its own forEach() method, allowing iteration. Moreover, the NodeList from querySelectorAll() can be converted into a standard JavaScript array using the Array.from() method.

Example:

```
<!DOCTYPE html>
<html lang="en">


<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>

<body>


    <h1 align="center">Welcome to Chitkara</h1>

    <div class="box box1">Item1</div>
```

```
    <div class="box box2">Item2</div>
    <div class="box box3">Item3</div>
    <div class="box box4">Item4</div>


    <p id="p1">This is paragraph 1</p>
    <p id="p2">This is paragraph 2</p>

    <script>

        //It will target all the elements whose class name is box or the
id is p1.
        let boxes = document.querySelectorAll(".box, #p1");

        console.log(boxes instanceof Array); //false

        boxes.forEach(e => console.log(e.textContent));

        //converting to the normal array
        boxes = Array.from(boxes);


        console.log(boxes instanceof Array); //true

        boxes.forEach(e => console.log(e.textContent));

    </script>

</body>

</html>
```

HTML Nodes vs Elements:

- In the HTML DOM (Document Object Model), an HTML document is a collection of nodes with (or without) child nodes.

- Nodes are element nodes, text nodes, and comment nodes.

- An Element is one specific type of node..(**All the HTML elements are the Nodes but all the Nodes are not an HTML element)**

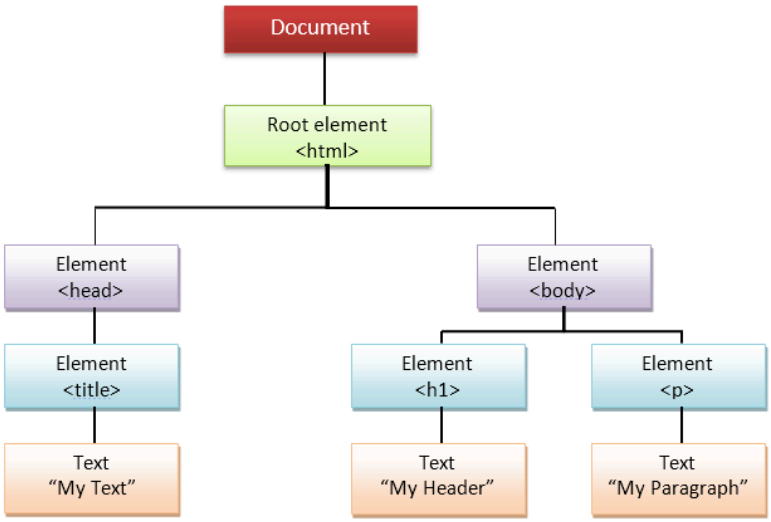
Note: In the DOM, nodes serve as a conceptual superclass representing entities within an HTML document. They provide a common interface and properties/methods shared by all node types. Element nodes, text nodes, and comment nodes are specific implementations or subclasses of the generic node concept, each with unique characteristics and behaviors.

```
console.log(boxes[0] instanceof Node);  //true
console.log(boxes[0] instanceof Element);  //true
```

Example:

```
<!DOCTYPE html>
<html>

<head>
    <title>My Text</title>
</head>

<body>
    <h1>My Header</h1>
    <p>My Paragraph</p>
</body>

</html>
```

Nodes structure of the above html:



Grab children/parent/siblings:

Example:

```
<!DOCTYPE html>
<html lang="en">

<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
</head>


<body>

   <div>
      <p id="p1">paragraph 1</p>
      <p>paragraph 2</p>
   </div>

   <script>
```

```
    let p1= document.getElementById("p1");
    let div= p1.parentNode;

    let childrenOfdiv= div.children;

    console.log(childrenOfdiv);

    console.log(childrenOfdiv[0].innerText); //paragraph 1

   console.log(childrenOfdiv[0].nextElementSibling.textContent); //paragraph 2

  </script>

</body>

</html>
```

Create new DOM Elements:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Document</title>
</head>

<body>

    <h1>Welcome to Chitkara</h1>


    <script>
```

```
            var newHeading = document.createElement('h1');
            var newParagraph = document.createElement('p');

            // create text nodes for new elements
            var h1Text = document.createTextNode('This is a nice header
    text!');
            var pText = document.createTextNode('This is a nice
    paragraph text!');

            // attach new text nodes to new elements
            newHeading.appendChild(h1Text);
            newParagraph.appendChild(pText);

            // Append newly created elements to the body
            document.body.appendChild(newHeading);
            document.body.appendChild(newParagraph);

            // shortcut
            document.body.append(newHeading, newParagraph);


            //OR simpler way

            var newHeading2 = document.createElement('h2');

            // Setting the text to the h2 element.
            newHeading2.innerText = "This is the heading 2"

            document.body.appendChild(newHeading2);

        </script>

    </body>

    </html>
```

Get/Set text to elements:

```
<!DOCTYPE html>
```

```html
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>


<body>

    <h1>Welcome to Chitkara</h1>



    <script>

        // Set operation

        let div1 = document.createElement("div");
        let div2 = document.createElement("div");
        let div3 = document.createElement("div");

        div1.innerText = `Hello

                        <h4>World</h4>`;

        /*
        <div>
          "Hello "
          <br>
          <br>
          "<h4>World</h4>"
        </div>
        */

        div2.textContent = `Bye

                        <h4>World</h4>`
```

```
        /*
        <div>Bye

                        <h4>World</h4>
        </div>
        */

        div3.innerHTML = `Whatsup
                        <h4>World</h4>`
        /*
        <div>Whatsup
        World</div>
        */

        document.body.append(div1);
        document.body.append(div2)
        document.body.append(div3)

    </script>

</body>

</html>
```

Add/Remove/Toggle/Check Classes the elements:

Here we need to make use of **classList** property of the element.

Example:

```
        // grab element on page you want to use
        var firstHeading = document.getElementById('firstHeading');

        // will remove foo if it is a class of firstHeading
        firstHeading.classList.remove('foo');

        // will add the class 'anotherClass' if one does not already exist
        firstHeading.classList.add('anotherclass');
```

```
        // add or remove multiple classes
        firstHeading.classList.add('foo', 'bar');
        firstHeading.classList.remove('foo', 'bar');

        // if visible class is set remove it, otherwise add it
        firstHeading.classList.toggle('visible');

        // will return true if it has class of 'foo' or false if it does
not
        firstHeading.classList.contains('foo');
```

Example:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>

    <style>
        .bgcolor_border {

            background-color: aqua;
            border: 2px solid red;
        }

        .font_size32 {
            font-size: 32px;
        }


        .text_color {
            color: blue;
        }
    </style>
```

```
</head>

<body>

    <p id="p1" class="text_color">This is the paragraph</p>

    <script>

        let p1_element = document.getElementById("p1");

        p1_element.classList.remove("text_color");

        // p1_element.classList.add("bgcolor_border");
        // p1_element.classList.add("font_size32");

        p1_element.classList.add("bgcolor_border", "font_size32");

    </script>

</body>

</html>
```

Add/Remove attributes:

Example1:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
```

```
    <style>
        .bgcolor_border {

            background-color: aqua;
            border: 2px solid red;
        }

        .font_size32 {
            font-size: 32px;
        }


        .text_color {
            color: blue;
        }
    </style>

</head>

<body>

    <p class="item" title="This is item 1" id="item1">Item 1</p>

    <script>

        let p_element = document.querySelector("#item1");

        console.log(p_element.getAttribute("title")); //This is item 1

        p_element.removeAttribute("class"); // class attribute will be
removed..inspect the element

        p_element.setAttribute("title", "This is the new title");
//modifying the existing attribute

  p_element.setAttribute("title2", "This is the title2"); //adding a new
attribute
```

```
    </script>

</body>

</html>
```

Example2:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>

</head>


<body>


    <img src="f1.jpg" alt="This is a flower image" height="200px"
width="200" class="image">


    <script>


        let img_element = document.querySelector(".image");

        console.log(img_element.getAttribute("src")); //f1.jpg

        img_element.setAttribute("src", "b1.jpg");

        img_element.setAttribute("height", "400px");
        img_element.setAttribute("width", "400px");


    </script>
```

```
</body>

</html>
```

Remove element:

```
el.remove();

parent.removeChild(element);
```

Example:

```
<!DOCTYPE html>
<html lang="en">




<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>

</head>


<body>


    <h2 id="heading2">This is a heading 1</h2>


    <div class="container">

        <p id="p1">This is a paragraph 1</p>
        <p id="p2">This is a paragraph 2</p>
```

```
    </div>


    <script>
        let h2_element= document.querySelector("#heading2");
        //h2_element.remove();


        let container_element= document.querySelector(".container");
        let p1_element = document.getElementById("p1");
        container_element.removeChild(p1_element);

    </script>

</body>

</html>
```

Modifying style:

```
      itm1.style.color = 'yellow';
      itm1.style.backgroundColor = 'red';
```

Example:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
```

```
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>

</head>


<body>


    <h2 id="heading2">This is a heading 1</h2>

    <script>

        let h2_element= document.querySelector("#heading2");


        h2_element.style.color = 'yellow';
        h2_element.style.backgroundColor = 'red';
        h2_element.style.border = "10px solid green"

       //applying multiple styles at once
      //h2_element.style.cssText = "color: yellow; background-color: red;
      border: //10px solid green;";



    </script>


</body>

</html>
```

Events handlers:

Event handlers allow you to react when specific actions (events) occur on a web page or its elements.

There are many different types of events that can occur. For example:

- The user selects a certain element or hovers the cursor over a certain element.
- The user clicks a button
- 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.

A comprehensive list of events can be found here:
https://developer.mozilla.org/en-US/docs/Web/Events

When an event occurs we can define a handler function to get executed.

Common Events

Events are categorized based on their source or nature. Here are some common examples:

Window Events

These occur when something happens to the browser window:

- **onload**: Triggered when a page is completely loaded.
- **onunload**: Triggered when a page is closed or replaced by another page.

User Events

These occur when users interact with page elements (via mouse, keyboard, etc.):

- **Mouse Events**: `click`, `dblclick`, `mousedown`, `mouseup`, `mouseover`, `mousemove`, `mouseout`
- **Keyboard Events**: `keypress`, `keydown`, `keyup`
- **Form Events**: `focus`, `blur`, `submit`, `reset`, `change`
- **Other Events**: `select` (text selection), `resize` (browser window resized)

How to Use Event Handlers

1. **Inline Event Handlers**
    - **Events are directly written into the HTML attributes.**
    - **Requires prefixing the event with `on` (e.g., `onclick`).**

Example 1: Inline Events

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Inline Event Example</title>
</head>

<body>
    <h2 id="heading2">This is Heading 1</h2>
    <button id="btn" onclick="fun1();">Click Here</button>

    <script>
        function fun1() {
            console.log("Button is clicked");
            const h2_element = document.getElementById("heading2");
            h2_element.innerHTML = "The Text is changed";
            h2_element.style.backgroundColor = "aqua";
            h2_element.style.border = "10px solid green";
        }
    </script>
</body>

</html>
```

Note: Inline events can clutter HTML and are not recommended for modern web development.

Here the html code will be polluted with events.

2. Event Listeners in Script

- **Use JavaScript to separate HTML and script logic.**
- **Modern method: Use `addEventListener()` for flexibility.**

Example2: events inside the script

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>

</head>


<body>


    <h2 id="heading2">This is a heading 1</h2>

    <button id="btn">Click Here</button>


    <script>



        // const button = document.getElementById("btn");
```

```
        // button.addEventListener("click", function () {
        //     console.log("Button is clicked");
        //     const h2_element=document.getElementById("heading2");
        //     h2_element.innerHTML="The Text is changed" ;
        //     h2_element.style.backgroundColor = "aqua" ;
        //     h2_element.style.border = "10px solid green";
        // });


        //using arrow function

        const button = document.getElementById("btn");

        button.addEventListener("click",() => {
            console.log("Button is clicked");
            const h2_element=document.getElementById("heading2");
            h2_element.innerHTML="The Text is changed" ;
            h2_element.style.backgroundColor = "aqua" ;
            h2_element.style.border = "10px solid green";
        });


    </script>


</body>

</html>
```

Example: To change the image if mouse is hover on it.

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
```

```
</head>


<body>

    <h1>Image changes on mouse hover</h1>

    <img src="b1.jpg" alt="background image" height="200px" width="200px"
id="image">

    <script>


        const image_element= document.getElementById("image");

        image_element.addEventListener("mouseover", () =>{

                //image_element.src="b2.jpg";
                image_element.setAttribute("src", "b2.jpg");
        });


        image_element.addEventListener("mouseout", () =>{

            image_element.src="b1.jpg";
        });

    </script>

</body>

</html>
```

Using Event Object:

- **Inside the handler function we can take parameters also, this parameter will represent the event object.**

- **The event object contains information about the event that occurred (e.g., type of event, target element).**

Example:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>

<body>
    <h1>Event handler with parameter</h1>
    <button class="btn">Click Here</button>
    <script>

        const btn_element = document.querySelector(".btn");
        // Using handler function with event parameter explicitly
        btn_element.addEventListener("click", function (e) {
            console.log("Button is clicked");
            console.log(e); // PointerEvent object
            console.log(e.target); // button element
        });

        // Alternatively, accessing event object directly (not
recommended)
        // btn_element.addEventListener("click", function () {
        //     console.log("Button is clicked");
        //     console.log(event);
        //     console.log(event.target);
        // });



    </script>
```

```
</body>

</html>
```

Note: Use the `target` property of the event object to get the element that triggered the event.

Event Capturing vs Bubbling:

When an event is triggered on an element, it propagates through the DOM in two distinct phases:

1. **Event Capturing** (capture phase)
2. **Event Bubbling** (bubble phase)
3.

1. Event Bubbling (Default Behavior)

- The event starts at the target element (where the event originated) and travels **upwards** through its ancestors in the DOM hierarchy, all the way to the root element.
- **Order of traversal:** Target → Parent → Grandparent → Root.
- **Default behavior** in most browsers.

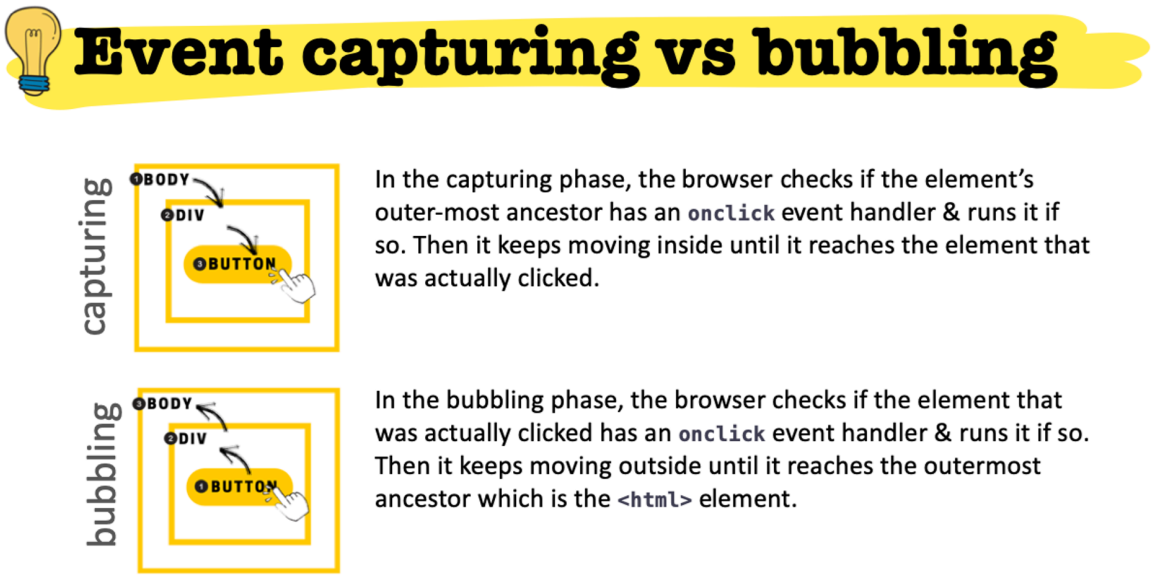
```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>


    <style>
        #outer {
            border: 20px solid gold;
        }
```

```
        #middle {
            border: 20px solid red;

        }


        #inner {
            border: 20px solid green;

        }
    </style>


</head>

<body>

    <div id="outer">
        <div id="middle">
            <div id="inner">Click me</div>
        </div>
    </div>

    <script>


        let outer_element = document.getElementById("outer");
        let middle_element = document.getElementById("middle");
        let inner_element = document.getElementById("inner");



        outer_element.addEventListener('click', function () {
            console.log('Event captured at outer');
        });

        middle_element.addEventListener('click', function () {
            console.log('Event captured at middle');
        });
```

```
        inner_element.addEventListener('click', function () {
            console.log('Event captured at inner');
        });



    </script>

</body>

</html>
```

Note: Clicking on the inner element will log the following in order:

1. **Event bubbled at inner**
2. **Event bubbled at middle**
3. **Event bubbled at outer**



2. Event Capturing (Optional)

- The event starts at the root element and travels **downwards** through its descendants to the target element.
- **Order of traversal:** Root → Grandparent → Parent → Target.
- To enable capturing, set the third parameter of `addEventListener()` to `true` or pass `{ capture: true }`.

Example: Event Capturing:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>


    <style>
        #outer {
            border: 20px solid gold;
        }


        #middle {
            border: 20px solid red;

        }


        #inner {
            border: 20px solid green;

        }
    </style>
```

```
</head>

<body>

    <div id="outer">
        <div id="middle">
            <div id="inner">Click me</div>
        </div>
    </div>


    <script>

        let outer_element = document.getElementById("outer");
        let middle_element = document.getElementById("middle");
        let inner_element = document.getElementById("inner");



        outer_element.addEventListener('click', function () {
            console.log('Event captured at outer');
        }, true); // or {capture: true}

        middle_element.addEventListener('click', function () {
            console.log('Event captured at middle');
        }, true);

        inner_element.addEventListener('click', function () {
            console.log('Event captured at inner');
        }, true);



    </script>

</body>

</html>
```

Note: Clicking on the inner element will log the following in order:

1. **Event captured at outer**
2. **Event captured at middle**
3. **Event captured at inner**

Preventing Event Propagation

`stopPropagation()`

- **Stops the event from propagating further, either during the capturing or bubbling phase.**

Example:

```
inner.addEventListener("click", (e) => {
    console.log("Event captured at inner");
    e.stopPropagation(); // Prevents the event from reaching middle or
outer
});
```

Difference between e.stopPropagation() and e.preventDefault()

Both methods used in event handling in JavaScript, but they serve different purposes:

e.stopPropagation()

- It is used to stop the propagation of an event through the DOM hierarchy.
- It prevents the event from bubbling up to parent elements or capturing down to child elements.

e.preventDefault()

- It is used to prevent the default action associated with an event from occurring.
- For example, preventing a form submission, preventing a link from navigating to a new page, etc.

Example:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>

</head>

<body>
    <a href="https://example.com" id="link">Click me</a>

    <script>

        let link_element = document.getElementById('link')

        link_element.addEventListener('click', function (e) {
            console.log('Link clicked!');
            e.preventDefault(); // Prevent the default action of the link
        });


    </script>

</body>

</html>
```

Event delegation using a global listener:

- Instead of adding individual event listeners to multiple elements, you can add a single listener to a common parent.
- Use the event's `target` property to identify the clicked element.

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Multiple Paragraphs Event Handlers</title>
</head>

<body>
    <p>Paragraph 1</p>
    <p>Paragraph 2</p>
    <p>Paragraph 3</p>
    <p class="p1">Paragraph 4 (click me)</p>

    <script>
        document.addEventListener("click", function (e) {
            if (e.target.matches(".p1")) {
                console.log(`${e.target.textContent}: Click event handler
for .p1 triggered`);
            }

            // Another `if` block to target specific text content
            if (e.target.textContent === "Paragraph 2") {
                console.log(`Paragraph 2 clicked!`);
                alert("You clicked on Paragraph 2!");
            }
        });
    </script>
</body>
```

```
</html>
```

Note: The `matches()` method in JavaScript is used to check if a given element matches a specified CSS selector.

Student Activity:

Problem 1. https://codepen.io/drupalastic/pen/rNPdjZq?editors=1010

Problem 2. https://codepen.io/drupalastic/pen/XWOEpoo?editors=1011

Solution 2:

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <title>beforeunload Example</title>

  <style>
    #colorBox {
      width: 200px;
      height: 200px;
      background-color: black;
      margin: 20px;
    }
```

```
    .button {
      margin: 5px;
    }
  </style>

</head>

<body>

  <div id="colorBox"></div>
  <button class="btn">RED</button>
  <button class="btn">YELLOW</button>
  <button class="btn">GREEN</button>
  <button class="btn">RESET</button>

  <script>
    let colorBox = document.getElementById("colorBox");
    let btns = document.querySelectorAll(".btn");

    btns.forEach((btn) => {
      btn.addEventListener("click", function (e) {
        console.dir(e.target.innerText);
```

```
        if (e.target.innerText == 'RESET') {
          colorBox.style.backgroundColor = 'blue';
        } else {
          colorBox.style.backgroundColor =
e.target.innerText.toLowerCase();
        }
      });
    });
  </script>
</body>
</html>
```

Problem 3: https://codepen.io/drupalastic/pen/MWLVJxB?editors=1010

Solution 3:

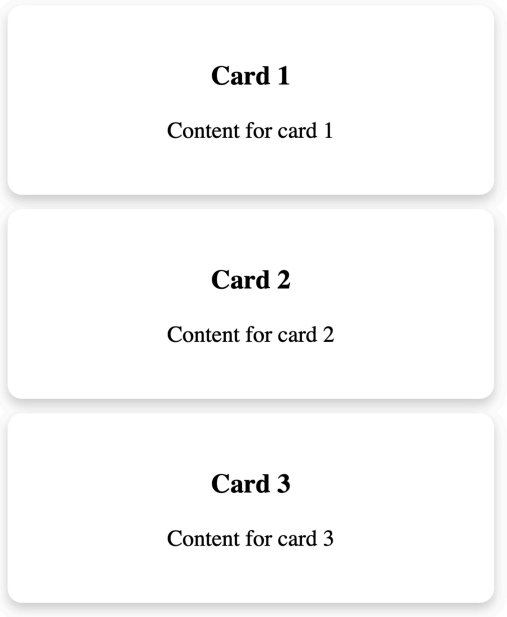
```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
```

```
<style>
  .box {
    width: 200px;
    height: 200px;
    background-color: black;
    margin: 20px;
  }

  .button {
    margin: 5px;
  }

  .red {
    background-color: red;
  }
  .yellow {
    background-color: yellow;
  }

  .green {
    background-color: green;
  }
</style>
```

```
</head>

<body>

  <div class="box"></div>

  <button class="btn">RED</button>
  <button class="btn">YELLOW</button>
  <button class="btn">GREEN</button>

  <script>

    let colorBox = document.querySelector(".box");
    let btns = document.querySelectorAll(".btn");

    btns.forEach((btn) => {
      btn.addEventListener("click", function (e) {
        console.log(e.target.innerText);

        // Remove existing color classes
        colorBox.classList.remove('red', 'yellow', 'green');
```

```
        // Add new color class based on the clicked button
        colorBox.classList.add(e.target.innerText.toLowerCase());

      });
    });
  </script>
</body>

</html>
```

Problem4: creating cards dynamically based on Java-script objects:



Solution:

```
<!DOCTYPE html>
<html lang="en">


<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>Document</title>
  <style>

    .card {
      box-shadow: 0 4px 8px 0 aqua;
      width: 300px;
      padding: 20px;
      text-align: center;
      background-color: white;
      border-radius: 10px;
      margin-bottom: 10px;
    }
  </style>
</head>
```

```
<body>
  <div id="card-container"></div>

  <script>

    const cards = [
      { title: "Card 1", content: "Content for card 1" },
      { title: "Card 2", content: "Content for card 2" },
      { title: "Card 3", content: "Content for card 3" },
    ];

    function createCard(obj) {
      let card = document.createElement("div");
      card.classList.add("card");

      let title = document.createElement("h3");
      title.innerText = obj.title;

      let content = document.createElement("p");
      content.textContent = obj.content;

      card.append(title);
      card.append(content);

      return card;
    }

    let container = document.getElementById("card-container");
```

```
    cards.forEach((cardObject) => {
      let card = createCard(cardObject);
      container.append(card);
    });


  </script>

</body>


</html>
```

Lab Assignment1:

Data:

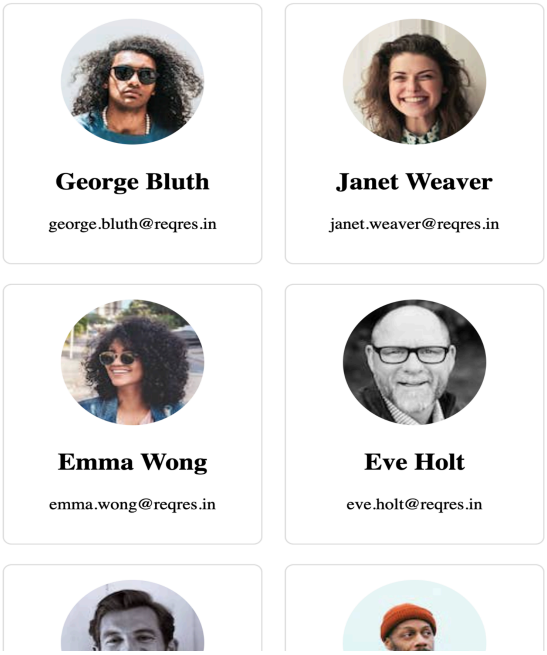
Use the following data:

```
let usersData = [
    {
        id: 1,
        email: "george.bluth@reqres.in",
        first_name: "George",
        last_name: "Bluth",
        avatar: "https://reqres.in/img/faces/1-image.jpg",
    },
    {
        id: 2,
        email: "janet.weaver@reqres.in",
        first_name: "Janet",
        last_name: "Weaver",
        avatar: "https://reqres.in/img/faces/2-image.jpg",
    },
    {
        id: 3,
        email: "emma.wong@reqres.in",
```

```
            first_name: "Emma",
            last_name: "Wong",
            avatar: "https://reqres.in/img/faces/3-image.jpg",
        },
        {
            id: 4,
            email: "eve.holt@reqres.in",
            first_name: "Eve",
            last_name: "Holt",
            avatar: "https://reqres.in/img/faces/4-image.jpg",
        },
        {
            id: 5,
            email: "charles.morris@reqres.in",
            first_name: "Charles",
            last_name: "Morris",
            avatar: "https://reqres.in/img/faces/5-image.jpg",
        },
        {
            id: 6,
            email: "tracey.ramos@reqres.in",
            first_name: "Tracey",
            last_name: "Ramos",
            avatar: "https://reqres.in/img/faces/6-image.jpg",
        },
    ];
```

Final Result:

This is the expected output



DOM:

This is the expected HTML output from your JS

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>

<body>


    <div id="userContainer">
        <div class="userCard">
            <img src="https://reqres.in/img/faces/1-image.jpg" alt="George
Bluth" />
            <h2>George Bluth</h2>
            <p>george.bluth@reqres.in</p>
```

```
            </div>
            <div class="userCard">
                <img src="https://reqres.in/img/faces/2-image.jpg" alt="Janet
Weaver" />
                <h2>Janet Weaver</h2>
                <p>janet.weaver@reqres.in</p>
            </div>
            <div class="userCard">
                <img src="https://reqres.in/img/faces/3-image.jpg" alt="Emma Wong"
/>
                <h2>Emma Wong</h2>
                <p>emma.wong@reqres.in</p>
            </div>
            <div class="userCard">
                <img src="https://reqres.in/img/faces/4-image.jpg" alt="Eve Holt"
/>
                <h2>Eve Holt</h2>
                <p>eve.holt@reqres.in</p>
            </div>
            <div class="userCard">
                <img src="https://reqres.in/img/faces/5-image.jpg" alt="Charles
Morris" />
                <h2>Charles Morris</h2>
                <p>charles.morris@reqres.in</p>
            </div>
            <div class="userCard">
                <img src="https://reqres.in/img/faces/6-image.jpg" alt="Tracey
Ramos" />
                <h2>Tracey Ramos</h2>
                <p>tracey.ramos@reqres.in</p>
            </div>
        </div>

    </body>

    </html>
```

CSS:

You don't need to write any CSS. You may use the CSS as it is.

```
        #userContainer {
            display: flex;
            flex-wrap: wrap;
            justify-content: center;
```

```
            gap: 20px;
            padding: 20px;
        }

        .userCard {
            border: 1px solid #ddd;
            border-radius: 8px;
            padding: 15px;
            text-align: center;
            width: 200px;
        }

        .userCard img {
            max-width: 100%;
            border-radius: 50%;
        }
```

Solution:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>

    <style>
        #userContainer {
            display: flex;
            flex-wrap: wrap;
            justify-content: center;
            gap: 20px;
            padding: 20px;
        }
```

```
        .userCard {
            border: 1px solid #ddd;
            border-radius: 8px;
            padding: 15px;
            text-align: center;
            width: 200px;
        }


        .userCard img {
            max-width: 100%;
            border-radius: 50%;
        }
    </style>


</head>


<body>


    <div id="userContainer"></div>

    <script>


        let usersData = [
            {
                id: 1,
                email: "george.bluth@reqres.in",
                first_name: "George",
                last_name: "Bluth",
                avatar: "https://reqres.in/img/faces/1-image.jpg",
            },
            {
                id: 2,
                email: "janet.weaver@reqres.in",
```

```
        first_name: "Janet",
        last_name: "Weaver",
        avatar: "https://reqres.in/img/faces/2-image.jpg",
    },
    {
        id: 3,
        email: "emma.wong@reqres.in",
        first_name: "Emma",
        last_name: "Wong",
        avatar: "https://reqres.in/img/faces/3-image.jpg",
    },
    {
        id: 4,
        email: "eve.holt@reqres.in",
        first_name: "Eve",
        last_name: "Holt",
        avatar: "https://reqres.in/img/faces/4-image.jpg",
    },
    {
        id: 5,
        email: "charles.morris@reqres.in",
        first_name: "Charles",
        last_name: "Morris",
        avatar: "https://reqres.in/img/faces/5-image.jpg",
    },
    {
        id: 6,
        email: "tracey.ramos@reqres.in",
        first_name: "Tracey",
        last_name: "Ramos",
        avatar: "https://reqres.in/img/faces/6-image.jpg",
    },
];



let container = document.getElementById("userContainer");
container.classList.add("userContainer");

function createUserCard(user) {
```

```
            let card = document.createElement("div");
            card.classList.add("userCard");


            let image_element = document.createElement("img");
            image_element.setAttribute("src", user.avatar);
            image_element.setAttribute("alt", user.first_name + " " +
user.last_name);

            let h2_element = document.createElement("h2");
            h2_element.innerText = user.first_name + " " + user.last_name;

            let p_element = document.createElement("p");
            p_element.innerText = user.email;


            card.append(image_element, h2_element, p_element);

            return card;

        }


        usersData.forEach(user => {
            let card = createUserCard(user);
            container.append(card);

        });

    </script>


</body>


</html>
```

`alert()` function:

- The alert() function displays a dialog box with a specified message and an OK button.
- It is often used to give the user some information or to prompt them for action.

Syntax:

```
alert("Hello! This is a alert message");
```

Example:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Document</title>
</head>

<body>


    <button id="btn">Click Here</button>

    <script>


        const btn_element = document.getElementById("btn");

        btn_element.addEventListener("click", () => {
            alert("This is the alert message");
        });

    </script>
```

```
</body>

</html>
```

`confirm()` function:

- The `confirm()` function displays a dialog box with a specified message, along with OK and Cancel buttons.
- It is used to get user confirmation for an action.

Syntax:

```
let result= confirm("Are you sure ?");

if(result === true){
    //user clicked on OK button
    //perform some logic
}
else{
    //user click on cancel button
    //do noting or handle cancellation
}
```

Example:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>

<body>


    <h1 id="heading">Welcome to Chitkara</h1>
```

```
    <button id="btn">Click Here to Delete the heading</button>

    <script>


        const btn_element = document.getElementById("btn");
        const header_element = document.getElementById("heading");


        btn_element.addEventListener("click", () => {

            let result = confirm("Are you sure ? to Delete this item");

            if (result === true) {
                console.log("User clicked on OK");
                //header_element.remove();
                document.body.removeChild(header_element);
                alert("Header is removed..");
            }
            else {
                console.log("User clicked on Cancel");
                alert("You have cancelled the deletion");
            }

        });

    </script>

</body>

</html>
```

`prompt()` function:

- The `prompt()` function displays a dialog box with a message prompting the user to input some text.

- It typically has an input field where the user can enter text and OK and Cancel buttons.
- If user Entered the value and press the OK button then this function will return that value in string format otherwise it returns the **null**.
- It is used to get user input, such as asking for a name, age, or any other information.

Syntax:

```
var userName = prompt("Please enter your name:");
if (userName !== null) {
    // User entered a name
    alert("Hello, " + userName + "! Welcome.");
} else {
    // User clicked Cancel or closed the prompt
    alert("You didn't enter your name.");
}
```

Example: Taking a year from the user and checking it is a leap year or not.

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>

<body>


    <button id="btn">Click Here to Check a Leap year</button>

    <script>
```

```
        const btn_element = document.getElementById("btn");

        btn_element.addEventListener("click", () => {


            // Prompt the user to enter a year
            var year = prompt("Enter a year:");

            if (year !== null) {
                // Convert the input to a number
                year = parseInt(year);

                // Check if the year is a leap year
                if ((year % 4 === 0 && year % 100 !== 0) || year % 400 ===
0) {
                    alert(year + " is a leap year.");
                } else {
                    alert(year + " is not a leap year.");
                }
            }
            else {
                alert("You have not entered year");
            }

        });

    </script>


</body>

</html>
```

Getting the input field value and displaying it on the screen:

- Here we will not use the **<form>** tag and **submit** button:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>

<body>

    <h1 align="center">Welcome to Chitkara</h1>

    <label for="username">Enter Username</label>
    <input type="text" id="username" name="username">


    <button id="btn">Click</button>


    <h2 id="display"></h2>


<script>


        const btn_element = document.getElementById("btn");

        btn_element.addEventListener("click", () => {

            let username_input_element =
document.getElementById("username");

            //to get the input field value
            let enteredValue = username_input_element.value;


            //checking the entered text is empty or not
```

```
            //remember required attribute will not work here, it will work
inside the <form> tag
            if(enteredValue.trim().length > 0){

                // let display_element=
document.getElementById("display");
                // display_element.innerText= "Welcome "+ enteredValue;

                //creating a new element and seting the text
                let h2_element= document.createElement("h2");
                h2_element.innerText = "Welcome "+enteredValue;

                document.body.appendChild(h2_element);

            }else{
                alert("Please enter the username");
            }

        });

    </script>


</body>

</html>
```

Student Activity: Creating a BMI calculator.

Solution:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
```

```
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>

<body>


    <h2>BMI Calculator</h2>


    <label for="height">Height (m):</label>
    <input type="number" id="height" step="0.01" placeholder="Enter height
in meters">

    <br><br>

    <label for="weight">Weight (kg):</label>
    <input type="number" id="weight" step="0.01" placeholder="Enter weight
in kilograms">
    <br><br>

    <input type="button" value="Calculate BMI" id="btn">


    <h4 id="result"></h4>


    <script>


        let btn_element = document.getElementById("btn");

        btn_element.addEventListener("click", () => {

            // Get height and weight values from the input fields
            var height =
parseFloat(document.getElementById('height').value);
            var weight =
parseFloat(document.getElementById('weight').value);
```

```
            // Check if height and weight are valid numbers
            if (isNaN(height) || isNaN(weight) || height <= 0 || weight <=
0) {
                alert('Please enter valid height and weight values.');

            }
            else {
                // Calculate BMI
                var bmi = weight / (height * height);

                // Display the calculated BMI
                document.getElementById('result').innerHTML = 'Your BMI
is: ' + bmi.toFixed(2);
            }



        });


    </script>



</body>

</html>
```

Lab Assignment3:

Create an application to accept the todo items from the user and add those items inside the page along with a checkbox and a delete button.

- **When the todo item is completed(checked) it should be striked and when we click on delete button, that todo item should be deleted with conformation.and one Edit button to edit the corresponding item.**

Solution:

```html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>


    <style>
        .completed {
            text-decoration: line-through;
        }
    </style>

</head>


<body>


    <label for="todoItem">Enter Todo Item</label>
    <input type="text" id="todoItem">


    <button id="btn">AddTodo</button>

    <br><br>


    <div class="todoContainer">

    </div>

    <script>


        function createTodoDiv(item) {

            const divElement = document.createElement("div");
```

```
const checkbox = document.createElement("input");
checkbox.setAttribute("type", "checkbox");


const span = document.createElement("span");
span.innerText = item;


const deleteButton = document.createElement("button");
deleteButton.innerText = "Delete";
deleteButton.style.backgroundColor = "red";

const editButton = document.createElement("button");
editButton.innerText = "Edit";
editButton.style.backgroundColor = "green";


//appending the checkbox, span and deleteButton
divElement.append(checkbox, span, editButton, deleteButton);


//based on checkbox, checked or not adding completed class
checkbox.addEventListener("change", (e) => {


    if (e.target.checked) {
        span.classList.add("completed");
    } else {
        span.classList.remove("completed");
    }


});
deleteButton.addEventListener("click", (e) => {


    let choice = confirm("Are you sure ?");
    if (choice === true) {
        let parentDiv = e.target.parentElement;
```

```
            parentDiv.remove();
        }


    });


    editButton.addEventListener("click", () => {

        const newTodo = prompt('Edit your todo:', span.innerText);
        if (newTodo !== null && newTodo.trim() !== '') {
            span.innerText = newTodo.trim();
        } else if (newTodo === '') {
            alert('Todo cannot be empty.');
        }



    });


    return divElement;

}



//adding a new todoItem along with checkbox and a delete button
document.getElementById("btn").addEventListener("click", () => {


    let todoItemInput = document.getElementById("todoItem");


    //input validation
    const todoItem = todoItemInput.value.trim();
```

```
            if (todoItem) {

                let divContainer =
document.querySelector(".todoContainer");

                let todoItemDiv = createTodoDiv(todoItem);

                divContainer.appendChild(todoItemDiv);

                //clear the input field
                todoItemInput.value = "";

            } else {
                alert("Please enter a valid todo item");
            }

        });

    </script>

</body>

</html>
```

Handling form submission:

Handling HTML form submission using JavaScript involves capturing form data when the user submits the form, preventing the default form submission behavior (which would cause a page reload), and then processing the data as needed.

Approach1: without using the <form> tag, in the above example:

- **Event Handling:** In the above approach, we attach an event listener to a regular button (`<input type="button">`) using the `click` event.
- **Data Retrieval:** We retrieve the height and weight values directly from the input fields using their IDs (`document.getElementById().value`).
- **Validation:** We perform validation checks to ensure that the entered values are valid numbers and greater than zero.
- **Submission:** There is no explicit form submission, and thus no page reload occurs.

Approach2: Using <form> tag

- **Form Structure:** In this approach, we wrap the input fields and the button inside a `<form>` tag**.**
- **Event Handling:** We attach an event listener to the **form element** using the `submit` event, instead of attaching it to a button.
- **Data Retrieval:** When the form is submitted, the browser automatically collects the values of all the input fields inside the form.
- **Preventing Default Submission:** Here we need to use `event.preventDefault()` inside the event handler to prevent the default form submission behavior, which would cause a page reload.
- **Validation:** Here the form validation of the input element (**required**) will work, but similar to the first approach, we can perform validation checks on the input values(any complex validation).
- **Submission:** Although we prevent the default submission behavior, we can still submit the form using JavaScript if needed, such as for **AJAX requests**.

Notes:

- When using a form, the browser provides additional functionalities like built-in validation, accessibility features..
- Handling form submission using JavaScript allows for more control over the submission process, enabling custom validation and asynchronous data processing.
- Both approaches are valid depending on the specific requirements and preferences of the developer.

By understanding the differences and trade-offs between the two approaches, developers can choose the one that best suits their needs for a particular application.

Creating a BMI calculator using form.

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>BMI Calculator with Form Submission</title>
</head>

<body>

    <h2>BMI Calculator</h2>

    <form id="bmiForm">
        <label for="height">Height (m):</label>
        <input type="number" id="height" name="height" step="0.01"
placeholder="Enter height in meters"
            required><br><br>

        <label for="weight">Weight (kg):</label>
        <input type="number" id="weight" name="weight" step="0.01"
placeholder="Enter weight in kilograms"
            required><br><br>

        <input type="submit" value="Calculate BMI">
    </form>

    <h4 id="result"></h4>

    <script>

            const form_element = document.getElementById("bmiForm");

     form_element.addEventListener("submit", (event) => {
```

```
            event.preventDefault(); // Prevent default form submission
behavior

            // Collect form data
            const formData = new FormData(event.target);

            const height = parseFloat(formData.get("height")); //here
height is the name attribute value
            const weight = parseFloat(formData.get("weight"));//here
weight is the name attribute value




             //Here we can also get the input field values individually
            // const height =
parseFloat(document.getElementById("height").value);
            // const weight =
parseFloat(document.getElementById("weight").value);




            // Check if height and weight are valid numbers
            if (isNaN(height) || isNaN(weight) || height <= 0 || weight <=
0) {
                alert('Please enter valid height and weight values.');
            } else {

                // Calculate BMI
                const bmi = weight / (height * height);

                // Display the calculated BMI
                document.getElementById('result').innerHTML = 'Your BMI
is: ' + bmi.toFixed(2);
            }
        });
    </script>

</body>
```

```
</html>
```

Note: using **FormData** object, we can gather all the form fields values by using **get()** function of this object.

Example:

formData.get("**name** of the input control");

`FormData` is somewhat similar to a map-like structure in that it stores key/value pairs.

- It has the following method:

1. **delete(name):** Removes the value associated with the given name.
2. **get(name):** Retrieves the first value associated with the given name.
3. **getAll(name):** Retrieves all values associated with the given name as an array.
4. **has(name):** Returns a boolean indicating whether the given name exists in the `FormData` object.
5. **set(name, value):** Sets the value associated with the given name, overwriting any existing value.
6. **entries():** Returns an iterator allowing iteration through all key/value pairs contained in the `FormData` object.
7. **keys():** Returns an iterator allowing iteration through all keys contained in the `FormData` object.
8. **values():** Returns an iterator allowing iteration through all values contained in the `FormData` object.
9. **append(key, value):** Add a new key-value pair.

We can use the forEach() function on the FormData object to iterate through all the key-value pairs.

Example:

```
        formData.forEach( (a,b) =>{

            console.log(a, b); // here value and then key will be
    printed


        } );
```

Converting the formData to the Java-script object:

```
const obj = {};
    formData.forEach((value, key) => {
        obj[key] = value;
    });
  console.log(obj);
```

Student Activity: Create a form which will collect the product related information like, **ProductId**, **ProductName**, **Price** and **Quantity**. And add the collected details inside a table below.

.

Solution:

```
<!DOCTYPE html>
<html lang="en">


<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
    <title>Product Information Form</title>
    <style>
        table,
        th,
        td {
            border: 2px solid black;


         /* It will merge the adjecent table cell border into a single
border */
            border-collapse: collapse;
            padding: 8px;
        }
    </style>
</head>

<body>

    <h2>Product Information Form</h2>

    <form id="productForm">


        <label for="productId">Product ID:</label>
        <input type="text" id="productId" name="productId"
required><br><br>

        <label for="productName">Product Name:</label>
        <input type="text" id="productName" name="productName"
required><br><br>

        <label for="price">Price:</label>
        <input type="number" id="price" name="price" step="0.01"
required><br><br>

        <label for="quantity">Quantity:</label>
        <input type="number" id="quantity" name="quantity"
required><br><br>

        <input type="submit" value="Add Product">
```

```
</form>

<h2>Product Information Table</h2>

<table id="productTable">
    <thead>
        <tr>
            <th>Product ID</th>
            <th>Product Name</th>
            <th>Price</th>
            <th>Quantity</th>
        </tr>
    </thead>

    <tbody>
    </tbody>


</table>


<script>

    const form_element = document.getElementById('productForm')


    function createTableRow(obj){

        const row_element = document.createElement("tr");

        const pid_column = document.createElement("td");
        pid_column.innerText = obj.productId;

        const pname_column = document.createElement("td");
        pname_column.innerText = obj.productName;

        const price_column = document.createElement("td");
        price_column.innerText = obj.price;

        const quantity_column = document.createElement("td");
```

```
            quantity_column.innerHTML = obj.quantity;

            //adding each column to the row_element.
            row_element.appendChild(pid_column);
            row_element.appendChild(pname_column);
            row_element.appendChild(price_column);
            row_element.appendChild(quantity_column);



//row_element.append(pid_column,pname_column,price_column,quantity_column)
;

            return row_element;

        }


        form_element.addEventListener('submit', function (event) {
            event.preventDefault(); // Prevent default form submission
behavior


            // Collect form data
            const formData = new FormData(event.target);

            //converting the formData to the JavaScript object
            const productObj = {};
            formData.forEach((value, key) => {
                productObj[key] = value;
            });

            let newProductRow = createTableRow(productObj);


            // Add product data to table
            const tableBody = document.querySelector('#productTable
tbody');
            tableBody.appendChild(newProductRow);
```

```
            //To clear the form field data
            event.target.reset();


        });
    </script>

</body>

</html>
```

Note: To properly align the form control values, make use of flex, grid or another table.

Note: Here after refreshing the page we can see the entire data is lost. To persist the previous record even though refreshing the page or restarting the application again, we need to make use of the **localStorage** object.

Student Assignment:

- Add one extra form field to take an image url/name from the user.
- Add one extra column inside the table which will display the product image.

localStorage:

https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage

What is meant by local storage?

Storing locally, right? Storing what ? ⇒ data or information. Storing where? ⇒ Locally ⇒ means where? On google drive? No. it'll be stored on our browser where we will run the app. Local storage can be used to store data as a mini database or a local database.

What is meant by database?

- The collection of data or the place where the data is stored is called database.
- Similarly local storage is something where data can be stored locally but it is limited by size, that means large amount of data cannot be stored.

Actual Definition:

localStorage is a property that allows JavaScript sites and apps to save key-value pairs in a web browser with no expiration date.

When to use local storage ?

- *You should only use local storage when storing insensitive information, i.e., we cannot store passwords and some sensitive information*

Where to see the local storage ?

- Open browser and click on inspect.
- Click `Application` and inside that you cna see Local Storage.
- That is where our data gets stored, we'll see how to store and get data.

What are the limitations ?

- The major limitations of local storage are:
    - *Insecure data (data is not secured, it can be hacked, i.e., someone who can access your system can have the data)*
    - *Limited storage capacity, i.e., images and videos cannot be stored, but we can store image and video urls.*

How does local storage work ?

- **To use localStorage in your web applications, there are four methods to choose from:**
    - ***setItem("key", value):*** *Add key and value to local storage*
    - ***getItem("key"):*** *This is how you get items from localStorage*
    - ***removeItem("key"):*** *Remove an item by key from localStorage*
    - ***clear():*** *Clear all local storage*

Note: localStorage stores only string, If we try to store number, boolean, array, objects, it will be stored in only string format.

Example:

```
localStorage.setItem("a",10);
localStorage.setItem("b",true);
localStorage.setItem("c","Welcome");

console.log(localStorage.getItem("a"));
console.log(localStorage.getItem("b"));
console.log(localStorage.getItem("c"));

console.log(typeof localStorage.getItem("a"));
console.log(typeof localStorage.getItem("b"));
console.log(typeof localStorage.getItem("c"));
```

Whenever we try to access a key which is not present in local storage, it'll return null.

Example:

```
console.log(localStorage.getItem("d")); //null

//to get a default value
console.log(localStorage.getItem("d") || "Hello"); //Hello
```

Storing the array and object in localStorage:

- As we know localStorage can only store the string, and if we try to store array or object, it will stored in string format only.

Example:

```
<script>

    let marks= [500,700,900,800,600];
```

```
let student = {
    roll: 10,
    name: "Ram",
    email: "ram@gmail.com"
};

localStorage.setItem("marksArr", marks);
localStorage.setItem("studentObj", student);

let m= localStorage.getItem("marksArr");
let s= localStorage.getItem("studentObj");

console.log(m); //500,700,900,800,600
console.log(s); // [object Object]


</script>
```

Here we can open the browser window and see the **application** tab while inspecting the element.

To solve this problem, we need to make use of **JSON object**, and store the array and objects in the form of JSON string.

What is JSON ? (Javascript Object Notation)

Actual Definition :

- JavaScript Object Notation (JSON) is a representation of structured data based on JavaScript object syntax.

JSON is most widely used

- Data is sent and received on Internet in JSON (mostly).
- It is based on Javascript objects.

Difference between JSON and JavaScript- Objects ?

1. Both follows the key-value format, but in JavaScript object double quote in key is optional where as in JSON it is mandatory.

2. JSON primarily used for data interchange between a server and a client, or between different systems. It's a lightweight format for storing and transmitting data. Where as JavaScript Objects Used within JavaScript code to represent complex data structures. JavaScript objects are fundamental to JavaScript programming and are extensively used to organize data and functionality.

3. JSON is designed to be language-independent, JSON is supported by many programming languages other than JavaScript. This makes it a popular choice for data exchange in web development and APIs. Whereas JavaScript objects are Native to JavaScript, JavaScript objects are not directly interoperable with other programming languages.

4. JavaScript objects can contain functions as values for properties. These functions are referred to as methods when attached to objects. Where as JSON doesn't support functions. If a JavaScript object contains functions, they are omitted during converting it to JSON format using **JSON.stringify()** function.

5. JavaScript objects can contain comments within the code, where as JSON does not support comments.

Example:

```
<script>


  let studentObj = {

      roll: 10,
      name: "Ram",
      married: false,
      email: "ram@gmail.com",  // this is a comment

      getDetails: function () {
          console.log("This is a function");
```

```
            console.log("Name is :", this.name);
        }
    }

    let studentJSON = {

        "roll": 10,
        "name": "Ram",
        "married": false,
        "email": "ram@gmail.com"

    }


</script>
```

Converting Java-script object or array to JSON string:

Here we need to make use of **JSON.stringify()** function.

Example:

```
<script>


    let studentObj = {

        roll: 10,
        name: "Ram",
        married: false,
        email: "ram@gmail.com",  // this is a comment

        getDetails: function () {
            console.log("This is a function");
            console.log("Name is :", this.name);
        }
    }
```

```
        let nums = [10, 50, 7, 8, 45, 44, 20, 15];


        //to convert JavaScript object to JSON string
        let studentJson = JSON.stringify(studentObj);

        //to convert the JavaScript array to JSON string
        let arrJson = JSON.stringify(nums);

       console.log(studentJson);
      // {"roll":10,"name":"Ram","married":false,"email":"ram@gmail.com"}

      console.log(arrJson);
      // [10,50,7,8,45,44,20,15]


    </script>
```

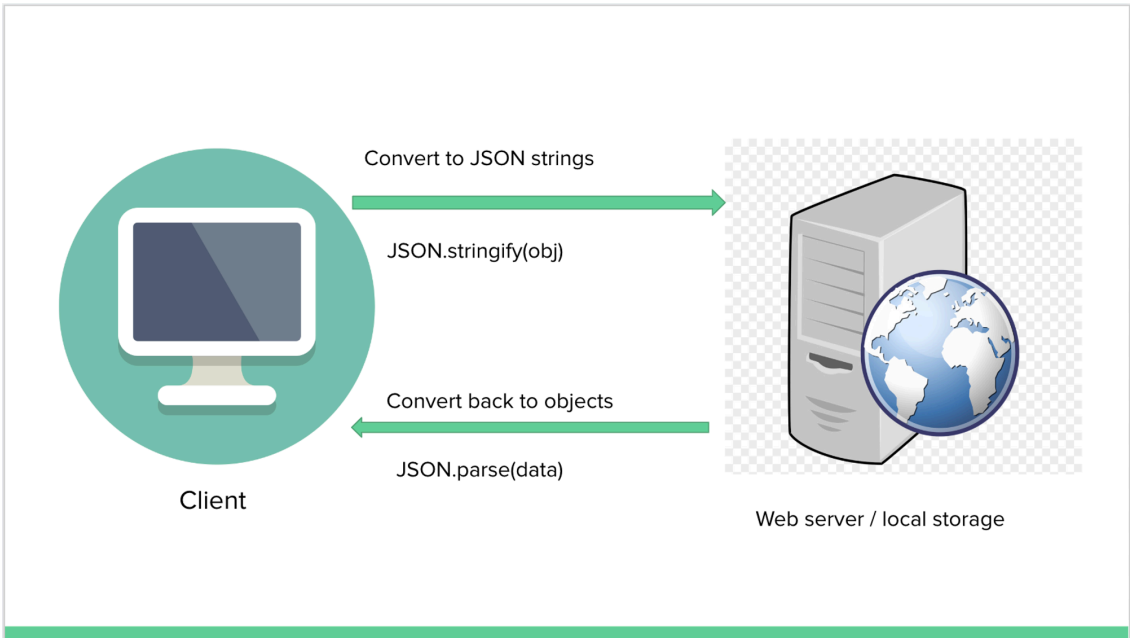
Converting JSON String or JavaScript object or array:

Here we need to make use of **JSON.parse()** function.

Example:

```
        //converting the above studentJson (JSON string) to the JavaScript
object.
        let stObj= JSON.parse(studentJson);

        //converting the above arrJson (JSON string) to the JavaScript
array.
        let arr= JSON.parse(arrJson);

        console.log(stObj);
        console.log(arr);
```

Note: Since localStorage will store any value inside the browser in the form of string, and when we want to get those value inside our application, we get those values in the form of string only, we can make use of JSON.stringify() and JSON.parse() function to store and get the arrays and JavaScript objects.



Converting String represented number to the actual number:

Here we can make use of one of the following approach:

1. **parseInt() / parseFloat()** function.
2. **Number()** constructor.

Example:

```
let stringNum = "100";

console.log(stringNum+10); // 10010

console.log(parseInt(stringNum) + 10); //110

console.log(Number(stringNum) + 10); //110
```

Note: If we want to convert any string which is not a number, then both will return **NaN.**

Example:

```
let stringNum = "abc";

console.log(parseInt(stringNum)); //NaN
console.log(Number(stringNum)); //NaN

let x= "10abc";

console.log(parseInt(x)); //10
console.log(Number(x)); //NaN
```

Note: we can also make use of **JSON.parse()** function to conver a string represented number in the original number.

Converting String represented boolean to the actual boolean:

Here we don't have parseBoolean() function. In this case we need to make use of JSON.parse() function.

```
let choice= "false";

if(choice){ // here string "false" is a truthy value
    console.log("Welcome");
}
else{
    console.log("Hello");
}


choice= JSON.parse(choice); //re-assingning the choice variable
```

```
if(choice){ // here choice value is false
    console.log("Welcome");
}
else{
    console.log("Hello");
}
```

Student Activity1:

Persisting username app: Build a simple web application that allows users to enter their name in a text box and save it. When the user revisits the page, the application should display the previously saved name.

Requirements

1. **Text Box:** A text box for users to enter their name.
2. **Submit Button:** A button to save the name to `localStorage`**.**
3. **Load Saved Name:** On page load, if a name is saved in `localStorage`, it should be displayed in the text box.
4. **Update Name:** Allow the user to update the saved name by entering a new name and clicking the submit button.

Solution:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>

<body>


    <h1 align="center">Welcome to localStorage</h1>
```

```
    <label for="username">Enter Username:</label>
    <input type="text" name="username" id="username">

    <button id="btn">Click</button>


    <h2 id="display"></h2>


    <script>

        let text = localStorage.getItem("enteredText") || "";

        display_element = document.getElementById("display");
        display_element.innerText = text;



        document.getElementById("btn").addEventListener("click", () => {

            input_element = document.getElementById("username");
            let userInputValue = input_element.value;

            localStorage.setItem("enteredText", userInputValue);
            display_element.innerText = userInputValue;


        });


    </script>


</body>

</html>
```

Student Activity2: create the following application to increase the like count and dislike count and store those counts to the **localStorage** so that those counts will not be lost when page reloaded.

Likes : 0

Dislikes : 0

Like 👍 Dislike 👎

Solution:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <title>Image</title>
</head>

<body>
    <img height="200px"

src="https://cdn.theatlantic.com/media/img/photo/2019/03/national-puppy-da
y-photos/p01_556211362-1/original.jpg"
        alt="dog image" />
```

```
    <div>
        <p>Likes : <span id="likes"></span></p>
        <p>Dislikes : <span id="dislikes"></span></p>
    </div>
    <div>

      <!-- use win+. to create emoji -->
        <button id="likeBtn">Like👍</button>
        <button id="dislikeBtn">Dislike👎</button>
    </div>

    <script>
        let likeCount = localStorage.getItem('likeCount') || 0;
        let dislikeCount = localStorage.getItem('dislikeCount') || 0;

        document.querySelector("#likes").innerText = likeCount;
        document.querySelector("#dislikes").innerText = dislikeCount;

        document.getElementById("likeBtn").addEventListener("click", () =>
{

            likeCount++;
            localStorage.setItem('likeCount', likeCount);
            document.querySelector("#likes").innerText = likeCount;

        });

        document.getElementById("dislikeBtn").addEventListener("click", ()
=> {

            dislikeCount++;
            localStorage.setItem('dislikeCount', dislikeCount);
            document.querySelector("#dislikes").innerText = dislikeCount;
        });


    </script>

</body>
```

```
</html>
```

Student Activity3: Display the following Student object in the table, and store those objects in the localStorage so that these data will be displayed on the webpage even though we reload the webpage.

```
let students = [

    {
        roll: 10,
        name: "Ram",
        marks: 800
    },
    {
        roll: 12,
        name: "Rahul",
        marks: 810
    },
    {
        roll: 14,
        name: "Ramesh",
        marks: 850
    },
    {
        roll: 16,
        name: "Dinesh",
        marks: 700
    }

];
```

Solution:

```html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>

    <style>
        table,
        tr,
        th,
        td {
            border: 2px solid green;
            border-collapse: collapse;
            padding: 10px;
        }
    </style>

</head>

<body>


    <h1 align="center">Welcome to localStorage</h1>

    <table>
        <thead>
            <tr>
                <th>Roll</th>
                <th>Name</th>
                <th>Marks</th>
            </tr>
        </thead>

        <tbody id="tbody">

        </tbody>
```

```
    </table>

    <script>

        let students = [

            {
                roll: 10,
                name: "Ram",
                marks: 800
            },
            {
                roll: 12,
                name: "Rahul",
                marks: 810
            },
            {
                roll: 14,
                name: "Ramesh",
                marks: 850
            },
            {
                roll: 16,
                name: "Dinesh",
                marks: 700
            }

        ];


        let studentsJson = JSON.stringify(students);

        localStorage.setItem("tabledata", studentsJson);

        let studentsArr = JSON.parse(localStorage.getItem("tabledata")) ||
[];

        function createTableRow(obj) {

            let row = document.createElement("tr");
```

```
            let rollCol = document.createElement("td");
            let nameCol = document.createElement("td");
            let marksCol = document.createElement("td");

            rollCol.innerText = obj.roll;
            nameCol.innerText = obj.name;
            marksCol.innerText = obj.marks;

            row.append(rollCol, nameCol, marksCol);

            return row;

        }

        studentsArr.forEach(item => {

            let newRow = createTableRow(item);

            tbody_element = document.getElementById("tbody");

            tbody_element.append(newRow);
        });

    </script>


</body>

</html>
```

Student Activity3: Improving the product application, where we accepted the product details from the user using a form page and displayed those information on a table. (Using localStorage).

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Product Information Form</title>
    <style>
        table,
        th,
        td {
            border: 2px solid black;


            /* It will merge the adjecent table cell border into a single
border */
            border-collapse: collapse;
            padding: 8px;
        }
    </style>
</head>


<body>


    <h2>Product Information Form</h2>


    <form id="productForm">


        <label for="productId">Product ID:</label>
        <input type="text" id="productId" name="productId"
required><br><br>


        <label for="productName">Product Name:</label>
        <input type="text" id="productName" name="productName"
required><br><br>
```

```
        <label for="price">Price:</label>
        <input type="number" id="price" name="price" step="0.01"
required><br><br>

        <label for="quantity">Quantity:</label>
        <input type="number" id="quantity" name="quantity"
required><br><br>

        <input type="submit" value="Add Product">
    </form>

    <h2>Product Information Table</h2>

    <table id="productTable">
        <thead>
            <tr>
                <th>Product ID</th>
                <th>Product Name</th>
                <th>Price</th>
                <th>Quantity</th>
            </tr>
        </thead>

        <tbody>
        </tbody>

    </table>

    <script>

        const form_element = document.getElementById('productForm')
```

```
        function createTableRow(obj) {

            const row_element = document.createElement("tr");

            const pid_column = document.createElement("td");
            pid_column.innerText = obj.productId;

            const pname_column = document.createElement("td");
            pname_column.innerText = obj.productName;

            const price_column = document.createElement("td");
            price_column.innerText = obj.price;

            const quantity_column = document.createElement("td");
            quantity_column.innerHTML = obj.quantity;

            //adding each column to the row_element.
            row_element.appendChild(pid_column);
            row_element.appendChild(pname_column);
            row_element.appendChild(price_column);
            row_element.appendChild(quantity_column);



//row_element.append(pid_column,pname_column,price_column,quantity_column)
;

            return row_element;

        }

        //getting the product array from the localStorage
        let productsJson = localStorage.getItem("productsArray");
        let products = JSON.parse(productsJson) || [];

        // Populate table with existing products
        const tableBody = document.querySelector('#productTable tbody');
        products.forEach(product => {
```

```
            const row = createTableRow(product);
            tableBody.appendChild(row);
        });



        form_element.addEventListener('submit', function (event) {
            event.preventDefault(); // Prevent default form submission
behavior


            // Collect form data
            const formData = new FormData(event.target);

            //converting the formData to the JavaScript object
            const productObj = {};
            formData.forEach((value, key) => {
                productObj[key] = value;
            });

            //added the new product to the existing product list;
            products.push(productObj);

            //save this products array to the localStorage by stringify it
            localStorage.setItem("productsArray",
JSON.stringify(products));



            let newProductRow = createTableRow(productObj);


            // Add product data to table
            const tableBody = document.querySelector('#productTable
tbody');
            tableBody.appendChild(newProductRow);


            //To clear the form field data
            event.target.reset();
```

```
        });

    </script>


</body>


</html>
```

setTimeout() function**:**

The setTimeout() function allows you to execute a function or a block of code after a specified period of time (in milliseconds). It only runs the function once after the timer expires.

Syntax:

let timeoutID = setTimeout(cbfunction, delayinMiliSec);

Example:

```
    <script>

        setTimeout(function(){
            console.log("Welcome to Js");
        }, 2000);


        //OR using arrow function
        // setTimeout(() =>{
        //     console.log("Welcome to Js");
        // }, 2000);

        //OR supplying any external function
        // function foo(){
```

```
        //     console.log("Inside the function foo");
        // }
        // setTimeout(foo, 2000);

    </script>
```

Note: This setTimeout() function returns an unique id value, which can be used to cancel the timeout. This timeoutId need to be passed inside the **clearTimeout()** function.

Example:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>

<body>



    <button id="start">Click Here to add a heading after 2 second</button>
    <button id="stop">Stop</button>



    <script>

        const startButton = document.getElementById("start");

        let timeoutId;
```

```
        startButton.addEventListener("click", () => {

            timeoutId = setTimeout(() => {

                const h2Element = document.createElement("h2");
                h2Element.textContent = "Hello New Element is added";
                document.body.appendChild(h2Element);

            }, 2000);

        });

        document.getElementById("stop").addEventListener("click", () => {

            clearTimeout(timeoutId);
            console.log("Timeout is cancelled");
        });


    </script>

</body>

</html>
```

setInterval() function:

The **setInterval()** function allows you to execute a function or a block of code repeatedly, with a fixed time delay between each call.

Syntax:

```
let intervalID = setInterval(function, delayInMilliSec);
```

Example:

```
    <script>


    setInterval(function(){
        console.log("Welcome to JS");
    }, 1000);


    //Using arrow function
    // setInterval(() =>{
    //     console.log("Welcome to JS");
    // }, 1000);


    //Using any external function
    // function foo(){
    //     console.log("Inside the foo function");
    // }
    // setInterval(foo,1000);



</script>
```

Note: This setInterval() function also returns an unique id value, which can be used to cancel the Intervals. **intervalId** need to be passed inside the **clearInterval()** function.

Example:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
```

```
<body>


    <button id="start">Click Here to start the timer</button>
    <button id="stop">Stop timer</button>
    <button id="reset">Reset timer</button>


    <h3 id="header"></h3>


    <script>

        const startButton = document.getElementById("start");
        const headerElement = document.getElementById("header");


        let intervalId;
        let k = 0;

        startButton.addEventListener("click", () => {

            intervalId = setInterval(() => {

                headerElement.innerText = k++;

            }, 1000);

        });

        document.getElementById("stop").addEventListener("click", () => {

            clearInterval(intervalId);
            console.log("Interval is cancelled");
        });

        document.getElementById("reset").addEventListener("click", () => {

            k = 0;
```

```
            headerElement.innerText = k;
        });


    </script>

</body>

</html>
```

Example: Using LocalTime:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>

<body>

    <button id="start">Click Here to start the timer watch</button>
    <button id="stop">Stop timer watch</button>


    <h3 id="header"></h3>


    <script>

        const startButton = document.getElementById("start");
        const headerElement = document.getElementById("header");


        let intervalId;
```

```
        startButton.addEventListener("click", () => {

            intervalId = setInterval(() => {

                headerElement.innerText = new Date().toLocaleTimeString();

            }, 1000);

        });

        document.getElementById("stop").addEventListener("click", () => {

            clearInterval(intervalId);
            console.log("Interval is cancelled");
        });

    </script>

</body>

</html>
```