Agenda:

- Adding and removing elements from DOM
- Modifying element's attributes
- Event Listeners
- Modifying CSS of the elements

In the last module on DOM we have learnt that how can we access the DOM elements. So, in this module with the help of knowledge from the last clas
we are going to learn that how we can add or remove elements from DOM.

Adding and Removing elements from DOM:

Adding an element:

In order to add an element into DOM, we first need to create an element for that,

We can use the JavaScript `document.createElement()` to create a new HTML element and attach it to the DOM tree.

The `document.createElement()` accepts an HTML tag name and returns a new `Node` with the `Element` type.

```
let element = document.createElement(htmlTag);
```

The above code will create a new HTML element of the type that we have specified in it's parameters.

Now, we can use the `appendChild()` method to add this element into our DOM.

Let's understand this with the example given below, where we will create a div element with some HTML content and then we will add it to our DOM.

So, let's consider that the HTML document before adding the div element is as shown below:

```
<!DOCTYPE html>
<html>
<head>
    <title>JS CreateElement Demo</title>
</head>
<body>

</body>
</html>
```

Now, we will use the `document.createElement()` to create a new `<div>` element as shown below:

```
let div = document.createElement('div');
```

We have created a div element, now let's add some HTML content to it using innerHTML():

```
div.innerHTML = '<p>CreateElement example</p>';
```

Now, to attach this div element to the document, we use the `appendChild()` method:

```
document.body.appendChild(div);
```

In the above code we have applied the appendChild() method on the body because here we want to add the div element that we have created to th
body of our document.

Now, joining all the above code, the HTML code becomes:

```
<!DOCTYPE html>
<html>
<head>
    <title>JS CreateElement Demo</title>
</head>
<body>
    <script>
        let div = document.createElement('div');
        div.innerHTML = '<p>CreateElement example</p>';
        document.body.appendChild(div);
    </script>
</body>
</html
```

Let's discuss in detail about **appendChild()** method:

As we have seen above that we used appendChild() method to connect the newly created element to our HTML document.

The `appendChild()` is a method of the `Node` interface. The `appendChild()` method allows you to add a node to the end of the list of child node of a specified parent node.

Syntax:

```
parentNode.appendChild(childNode);
```

In this method, the `childNode` is the node to append to the given parent node. The `appendChild()` returns the appended child.

If the `childNode` is a reference to an existing node in the document, the `appendChild()` method moves the `childNode` from its current position the new position.

Consider the example shown below:

Suppose you have given the following HTML document:

```
<ul id="menu">
</ul>
```

Now, we want to add the list items i.e.,

- elements to this list , now see the code below:

```
function createMenuItem(name) {
    let li = document.createElement('li');
    li.textContent = name;
    return li;
}
// get the ul#menu
const menu = document.querySelector('#menu');
// add menu item
menu.appendChild(createMenuItem('Home'));
menu.appendChild(createMenuItem('Services'));
menu.appendChild(createMenuItem('About Us'));
```

How it works:

- First, the `createMenuItem()` function create a new list item element with a specified name by using the `createElement()` method.
- Second, select the `<ul>` element with id `menu` using the `querySelector()` method.
- Third, call the `createMenuItem()` function to create a new menu item and use the `appendChild()` method to append the menu item to the `<ul>` element

Output:

```
<ul id="menu">
    <li>Home</li>
    <li>Services</li>
    <li>About Us</li>
</ul>
```

Removing an element:

In order to remove a child element from a node, we can use the `removeChild()` method , consider the syntax shown below:

```
let childNode = parentNode.removeChild(childNode);
```

The `childNode` is the child node of the `parentNode` that you want to remove. If the `childNode` is not the child node of the `parentNode` , th
method throws an exception. The `removeChild()` returns the removed child node from the DOM tree but keeps it in the memory, which can be use
later. If you don't want to keep the removed child node in the memory, you use the following syntax:

```
parentNode.removeChild(childNode);
```

The child node will be in the memory until it is destroyed by the JavaScript garbage collector. Let's take an example to understand it : Suppose you hav
given the following HTML markup, where a list is given as shown below:

```
<ul id="menu">
    <li>Home</li>
    <li>Products</li>
    <li>About Us</li>
</ul>
```

Now, consider that you want to remove the last child element:

```
let menu = document.getElementById('menu');
menu.removeChild(menu.lastElementChild);
```

How it works:

- First, get the `ul` element with the id `menu` by using the `getElementbyId()` method.
- Then, remove the last element of the `ul` element by using the `removeChild()` method.
The `menu.lastElementChild` property returns the last child element of the `menu` .

Now, consider that you want to remove all the child elements: To remove all child nodes of an element, you use the following steps:

- Get the first node of the element using the `firstChild` property.
- Repeatedly removing the child node until there are no child nodes left.

The following code shows how to remove all list items of the `menu` element:

```
let menu = document.getElementById('menu');
while (menu.firstChild) {
    menu.removeChild(menu.firstChild);
}
```

In these ways we can add or remove the element from our HTML document, now let's see that how we can modify element's attributes:

Modifying element's attributes:

Let's first discuss the connection between HTML attributes and DOM properties: When the web browser loads a HTML page, it generates th
corresponding DOM objects based on the DOM nodes of the document. For example, if a page contains the following `input` element:

```
<input type="text" id="username">
```

The web browser will generate an `HTMLInputElement` object. The `input` element has two attributes:

- The `type` attribute with the value `text` .
- The `id` attribute with the value `username` .

The generated `HTMLInputElement` object will have the corresponding properties:

- The `input.type` with the value `text` .
- The `input.id` with the value `username` .

Now, to access both standard and non-standard attributes, you use the following methods:

- `element.getAttribute(name)` – get the value of the attribute
- `element.setAttribute(name, value)` – set the value of the attribute
- `element.hasAttribute(name)` – check for the existence of the attribute
- `element.removeAttribute(name)` – remove the attribute

Now, let's understand the working of these methods one by one in detail: **getAttribute() method:** To get the value of an attribute on a specified elemer you call the `getAttribute()` method of the element:

```
let value = element.getAttribute(name);
```

Parameters

The `getAttribute()` accepts an argument which is the name of the attribute from which you want to return the value.

Return value

If the attribute exists on the element, the `getAttribute()` returns a string that represents the value of the attribute. In case the attribute does n exist, the `getAttribute()` returns `null` . Consider the example below:

```
<!DOCTYPE html>
<html>
<body>

    <a href="https://www.javascripttutorial.net"
        target="_blank"
        id="js">JavaScript Tutorial
    </a>

    <script>
        let link = document.querySelector('#js');
        if (link) {
            let target = link.getAttribute('target');
            console.log(target);
        }
    </script>
</body>
</html>
```

Output:

```
_blank
```

How it works:

- First, select the link element with the id `js` using the `querySelector()` method.
- Second, get the target attribute of the link by calling the `getAttribute()` of the selected link element.

setAttribute() method:

To set a value of an attribute on a specified element, you use the `setAttribute()` method:

```
element.setAttribute(name, value);
```

Parameters

The `name` specifies the attribute name whose value is set. It's automatically converted to lowercase if you call the `setAttribute()` on an HTM element. The `value` specifies the value to assign to the attribute. It's automatically converted to a string if you pass a non-string value to the method.

Return value

The `setAttribute()` returns `undefined` . Note that if the attribute already exists on the element, the `setAttribute()` method updates the valu otherwise, it adds a new attribute with the specified `name` and `value` . Typically, you use the `querySelector()` or `getElementById()` to sele an element before calling the `setAttribute()` on the selected element. Consider the example shown below:

```
<!DOCTYPE html>
<html>
<body>
    <button id="btnSend">Send</button>
    <script>
        let btnSend = document.querySelector('#btnSend');
        if (btnSend) {
            btnSend.setAttribute('name', 'send');
        }
    </script>
</body>
</html>
```

Output:

```
<button id="btnSend" name="send">Send</button>
```

How it works:

- First, select the button with the id `btnSend` by using the `querySelector()` method.
- Second, set the value of the `name` attribute to `send` using the `setAttribute()` method.

hasAttribute() method:

To check an element has a specified attribute or not, you use the `hasAttribute()` method:

```
let result = element.hasAttribute(name);
```

Parameters

The `hasAttribute()` method accepts an argument that specifies the name of the attribute that you want to check.

Return value

The `hasAttribute()` returns a Boolean value that indicates if the element has the specified attribute. If the element contains an attribut the `hasAttribute()` returns true; otherwise, it returns `false` .

removeAttribute() method:

The `removeAttribute()` removes an attribute with a specified name from an element:

```
element.removeAttribute(name);
```

Parameters

The `removeAttribute()` accepts an argument which is the name of the attribute that you want to remove. If the attribute does not exis the `removeAttribute()` method wil not raise an error.

Return value

The `removeAttribute()` returns a value of `undefined` . So, in these ways we can modify the attributes on an element, Now, let's start to lea about EventListeners:

Event Listeners:

An event is an action that occurs in the web browser, which the web browser feedbacks to you so that you can respond to it. For example, when use click a button on a webpage, you may want to respond to this `click` event by displaying a dialog box. Each event may have an event handler which a block of code that will execute when the event occurs. An event handler is also known as an event listener. It listens to the event and executes whe the event occurs.

Event Flow:

For understanding Event flow consider the HTML document given below:

```
<!DOCTYPE html>
<html>
<body>
    <div id="container">
        <button id='btn'>Click Me!</button>
    </div>
</body>
```

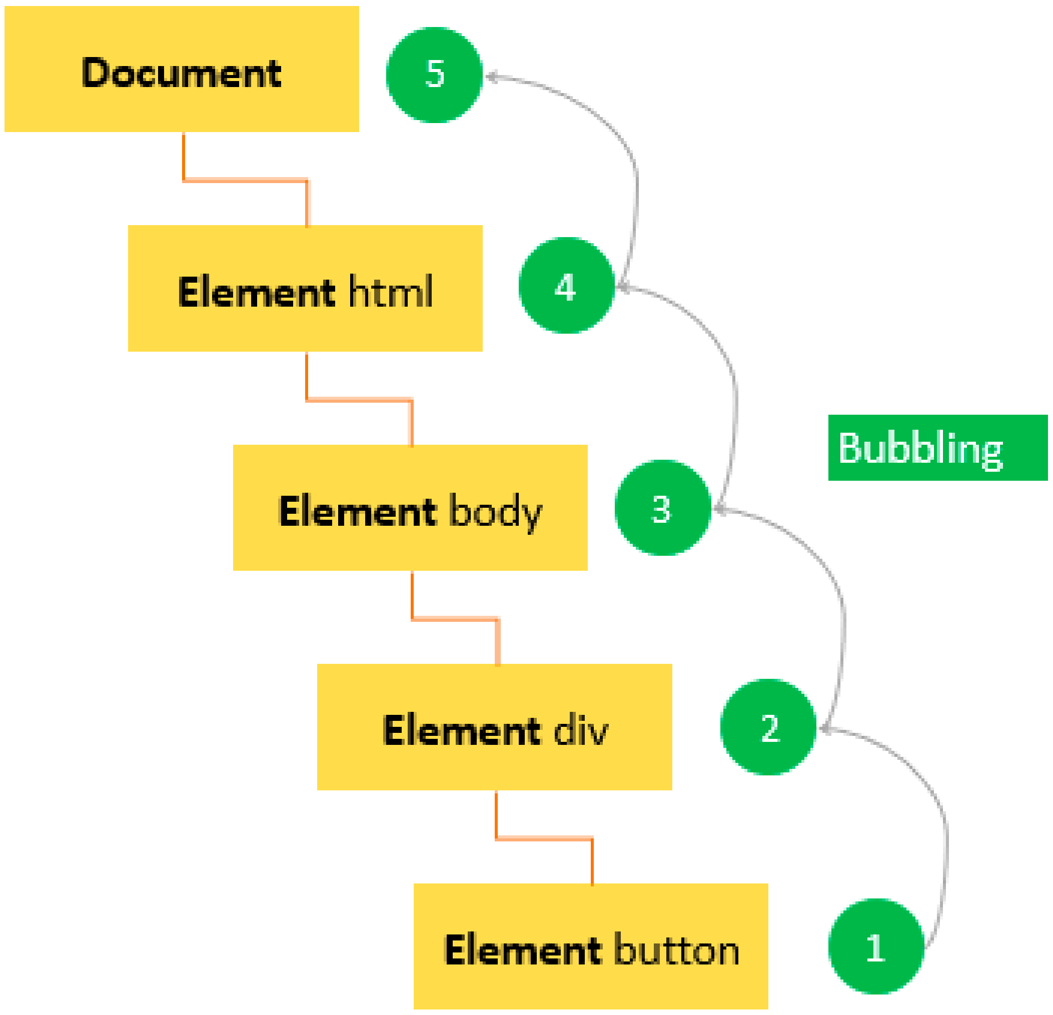
When you click the button, you're clicking not only the button but also the button's container, the `div`, and the whole webpage. Event flow explains th order in which events are received on the page from the element where the event occurs and propagated through the DOM tree. There are two ma event models: event bubbling and event capturing.

Event Bubbling:

In the event bubbling model, an event starts at the most specific element and then flows upward toward the least specific element (the `document` even `window`). When you click the button, the `click` event occurs in the following order:

1. button
2. div with the id container
3. body
4. html
5. document

The `click` event first occurs on the button which is the element that was clicked. Then the `click` event goes up the DOM tree, firing on each noc along its way until it reaches the `document` object. Consider the picture shown below:



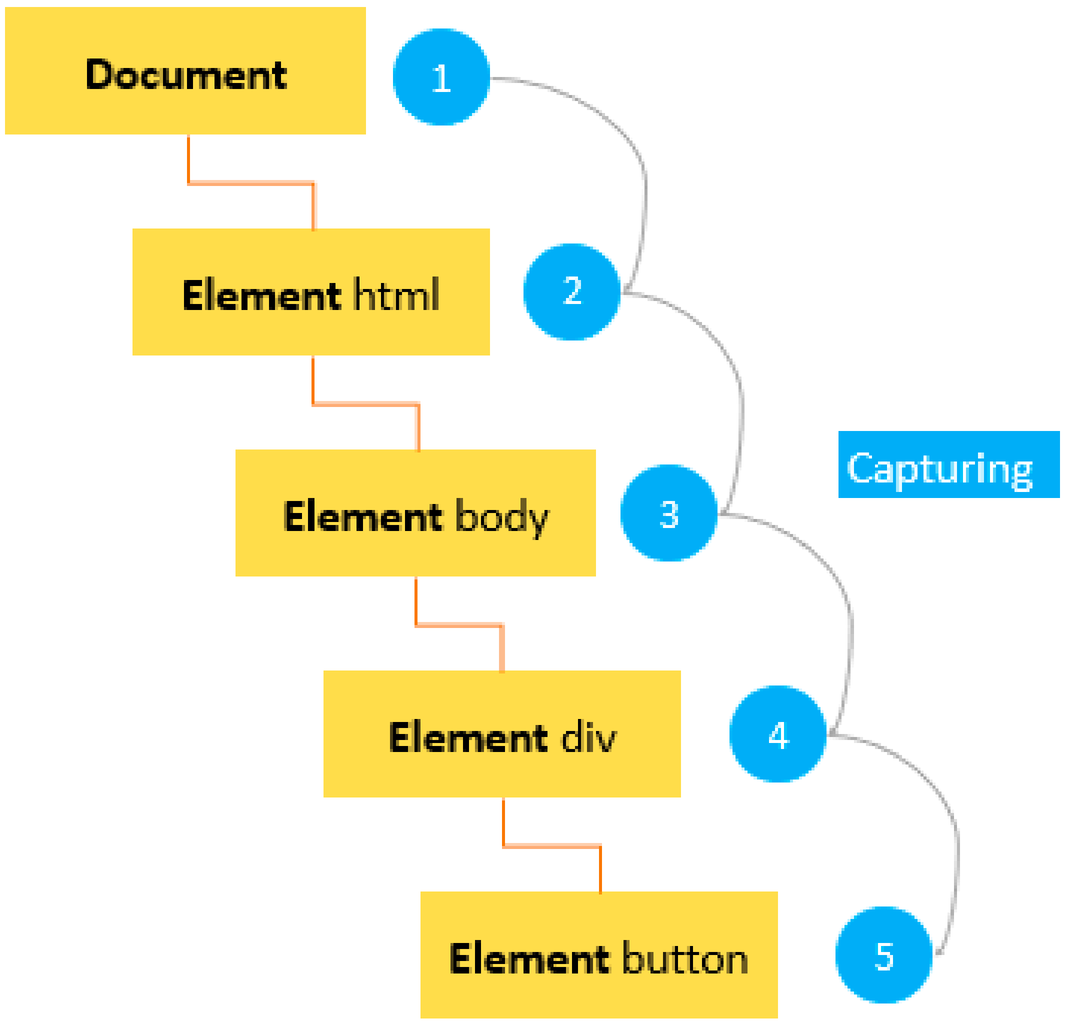
Event Capturing:

In the event capturing model, an event starts at the least specific element and flows downward toward the most specific element. When you click th button, the `click` event occurs in the following order:

1. document
2. html
3. body
4. div with the id container

5. button

Consider the picture shown below:



As we have learned above that whenever a event is fired a block of code is executed which is added to that event, that block of code is handled by eve handlers or event listeners. An event can be handled by one or multiple event handlers. If an event has multiple event handlers, all the event handle will be executed when the event is fired. We can handle the events in the following ways:

HTML Event Handler Attributes:

Event handlers typically have names that begin with `on` , for example, the event handler for the `click` event is `onclick` . To assign an eve handler to an event associated with an HTML element, you can use an HTML attribute with the name of the event handler. Consider the example below

```
<input type="button" value="Save" onclick="alert('Clicked!')">
```

In the above example code, whenever the button is clicked, the alert() method is fired which shows the alert with the text Clicked!. When you assig JavaScript code as the value of the `onclick` attribute, you need to escape the HTML characters such as ampersand (`&`), double quotes (`"`), les than (`<`), etc., or you will get a syntax error. An event handler defined in the HTML can call a function defined in a script. Consider the example show below:

```
<script>function showAlert() {
        alert('Clicked!');
    }
</script><input type="button" value="Save" onclick="showAlert()">
```

In this example, the button calls the `showAlert()` function when it is clicked. **Disadvantages of using HTML event handler attributes:** Assignir event handlers using HTML event handler attributes are considered as bad practices and should be avoided as much as possible because of tl following reasons: First, the event handler code is mixed with the HTML code, which will make the code more difficult to maintain and extend. Second, is a timing issue. If the element is loaded fully before the JavaScript code, users can start interacting with the element on the webpage which will caus an error. As assigning event handlers using HTML is a bad practice, so there are other methods let’s look at them:

addEventListener() method:

`addEventListener()` - registers an event handler syntax:

```
element.addEventListener(event, function, useCapture)
```

The `addEventListener()` method accepts three arguments:

- an event: It is a required parameter. It can be defined as a string that specifies the event's name.
- **function:**It is also a required parameter. It is a javascript functionwhich responds to the event occur.
- **useCapture:** It is an optional parameter. It is a Boolean type value that specifies whether the event is executed in the bubbling or capturing phase. Its possible values are **true** and **false**. When it is set to true, the event handler executes in the capturing phase. When it is set to false, the handler executes in the bubbling phase. Its default value is **false**.

Let's take an example to understand it better, consider that there is a button in our code with the id property as 'btn' and we want to add a eventListen at this button on click of the button:

```
let btn = document.querySelector('#btn');
btn.addEventListener('click',function(event) {
    alert(event.type); // click
});
```

In the above code, firstly we are selecting the button with querySelector method then we are assigning an eventListener to it using addEventListene method which is listening to the click event as we have passed the first parameter in the addEventListener method, then there is a anonymous functic as the second argument which is gets executed when the event is handled. Here, we are adding only a single event handler to handle a single event, b we can also add multiple event handlers to a single event. Let's understand this through the example given below:

```
let btn = document.querySelector('#btn');
btn.addEventListener('click',function(event) {
    alert(event.type); // click
});

btn.addEventListener('click',function(event) {
    alert('Clicked!');
});
```

removeEventListener() method:

`removeEventListener()` - removes an event handler The `removeEventListener()` removes an event listener that was added v the `addEventListener()` . However, you need to pass the same arguments as were passed to the `addEventListener()` . Consider the examp below:

```
let btn = document.querySelector('#btn');

// add the event listener
let showAlert = function() {
    alert('Clicked!');
};
btn.addEventListener('click', showAlert);

// remove the event listener
btn.removeEventListener('click', showAlert);
```

In the above code, firstly we have added an event listener using addEventLlistener method and then we have removed the event handler usir removeEventListener method. Note: In removeEventListener we cannot pass a anonymous function as a second argument as we have done in tl addEventListener method. There are so many events to which we can listen and add handlers:

Event Performed	Event Handler	Description
click	onclick	When mouse click on an element
mouseover	onmouseover	When the cursor of the mouse comes over the element
mouseout	onmouseout	When the cursor of the mouse leaves an element
mousedown	onmousedown	When the mouse button is pressed over the element

Event Performed	Event Handler	Description
mouseup	onmouseup	When the mouse button is released over the element
mousemove	onmousemove	When the mouse movement takes place.
Keydown & Keyup	onkeydown & onkeyup	When the user press and then release the key
focus	onfocus	When the user focuses on an element
submit	onsubmit	When the user submits the form
blur	onblur	When the focus is away from a form element
change	onchange	When the user modifies or changes the value of a form element
load	onload	When the browser finishes the loading of the page
unload	onunload	When the visitor leaves the current webpage, the browser unloads it
resize	onresize	When the visitor resizes the window of the browser

Now, as we have learned about events and their handling let's start with modifying css: Let's start by learning that how we can set the inline styles:

Setting Inline Styles:

To set the inline style of an element, you use the `style` property of that element:

```
element.style
```

The `style` property returns the read-only `CSSStyleDeclaration` object that contains a list of CSS properties. For example, to set the color of a element to `red`, you use the following code:

```
element.style.color = 'red';
```

If the CSS property contains hyphens (`-`) for example `-webkit-text-stroke` you can use the array-like notation (`[]`) to access the property:

```
element.style.['-webkit-text-stock'] = 'unset';
```

To completely override the existing inline style, you set the `cssText` property of the `style` object. For example:

```
element.style.cssText = 'color:red;background-color:yellow';
```

Or you can use the `setAttribute()` method:

```
element.setAttribute('style','color:red;background-color:yellow');
```

Once setting the inline style, you can modify one or more CSS properties:

```
element.style.color = 'blue';
```

If you do not want to completely overwrite the existing CSS properties, you can concatenate the new CSS property to the `cssText` as follows:

```
element.style.cssText += 'color:red;background-color:yellow';
```

In this case, the `+=` operator appends the new style string to the existing one.

Manipulating CSS Classes:

The `className` is the property of an element. It returns a space-separated list of CSS classes of the element:

```
element.className
```

Suppose that you have the following `ul` element:

```
<ul id="menu" class="vertical main"><li>Homepage</li><li>Services</li><li>About</li><li>Contact</li></ul>
```

The following code shows the classes of the `ul` element on the Console window:

```
let menu = document.querySelector('#menu');
console.log(menu.className);
```

Output:

```
vertical main
```

To set a class to an element, you use the following code:

```
element.className += newClassName;
```

The `+=` operator adds the `newClassName` to the existing class list of the element. We can also use `classList` property of an element manipulate it's css, so let's understand it : The `classList` is a read-only property of an element that returns a live collection of CSS classes:

```
const classes = element.classList;
```

The `classList` is a `DOMTokenList` object that represents the contents of the element's class attribute. Even though the `classList` is read-onl but you can manipulate the classes it contains using various methods. Let's understand them directly through examples: **Getting css classes of a element:** Suppose we have given the following HTML snippet:

```
<div id="content" class="main red">JavaScript classList</div>
```

And if we want to get it's classes we can get as:

```
let div = document.querySelector('#content');
for (let cssClass of div.classList) {
    console.log(cssClass);
}
```

Output:

```
main
red
```

How it works:

- First, select the `div` element with the id `content` using the `querySelector()` method.
- Then, iterate over the elements of the `classList` and show the classes in the Console window.

Adding one or more classes to the class list of an element: To add one or more CSS classes to the class list of an element, you us the `add()` method of the `classList`. For example, the following code adds the `info` class to the class list of the `div` element with th id `content` :

```
let div = document.querySelector('#content');
div.classList.add('info');
```

The following example adds multiple CSS classes to the class list of an element:

```
let div = document.querySelector('#content');
div.classList.add('info','visible','block');
```

Removing element's classes: To remove a CSS class from the class list of an element, you use the `remove()` method:

```
let div = document.querySelector('#content');
div.classList.remove('visible');
```

Like the `add()` method, you can remove multiple classes once:

```
let div = document.querySelector('#content');
div.classList.remove('block','red');
```

So, in this way we can manipulate the classes hence css of any element. So, In this topic we have learnt about:

- How we can add and remove elements from a document ?
- How we can modify element’s attributes ?
- Events, their flow and their handling using Event listeners or handlers.
- And finally, how we can manipulate the CSS of an element.

Interview Questions

> Can you explain how to create custom HTML attributes, then get their values?

You can create custom HTML attributes by using the setAttribute() method. Once you have created a custom attribute, you can then get its value b using the getAttribute() method.

> How can we toggle between hiding and showing content using JavaScript?

There are a few different ways to toggle between hiding and showing content using JavaScript. One way would be to use the CSS style proper “display” and set it to “none” to hide the content, and then set it back to “block” to show it again. Another way would be to use the “visibility” style proper and set it to “hidden” to hide the content, and then set it back to “visible” to show it again.

> How can you change styles of a particular element using JavaScript?

You can change styles of a particular element using JavaScript by accessing the element’s style property. This property is an object that contains all the element’s style information. To change a specific style, you would simply set the appropriate property of the style object to the new value. F example, if you wanted to change the color of an element, you would do the following: element.style.color = “new color”;

Thank You !