DOM and Event Handling in JavaScript

Dr Harshad Prajapati
14, 21 Nov 2023

Events and Event Handling

What is an Event?

- Event is a notification of something has occurred in the system that we are programming.
- Events are fired inside browser window.
- Events finally get delivered to a single element or a set of elements.

Events in HTML Document

- Examples of HTML events:
  - The user selects, clicks, or hovers the cursor over a certain element.
  - The user resizes or closes the browser window.
  - When a web page has loaded
  - When an image has been loaded
  - When an input field is changed
  - When an HTML form is submitted
  - An error occurs.
  - A video is played, paused, or ends.
  - The user presses a key on the keyboard.

Ways to Handle Events in a Webpage

- There are two ways to handle events in a webpage:
  - Using DOM manipulation API.
  - Using HTML attributes.

How to react to an event: Event Handling using DOM API

- There are two major steps:
  - Write event handler or event listener.
  - Attach event handler with event.

How to react to an event: Event Handling using DOM API

- We need to write our logic in form of a function for an event in which we are interested.
  - That function is called event handler or event listener.
  - The form of event handler could be:

```
function myEventHandler(event) { ... }
```

```
function myEventHandler() { ... }
```

- We need to attach or register our event handler with specific event.
  - This is called registering an event handler.

How to react to an event: Event Handling using DOM API

- To register, we use addEventListener() method on source element.
  - element.addEventListener(event, function, useCapture);
    - The name of the event as string.
    - The event handler function itself or reference to it.
    - The third parameter is how the event is to be propagated.
      - Whether event capturing (true) or event bubbling (false).
- For example, we can attach event handler changeBackground for a button element btn for click event.
  - `btn.addEventListener("click", changeBackground);`

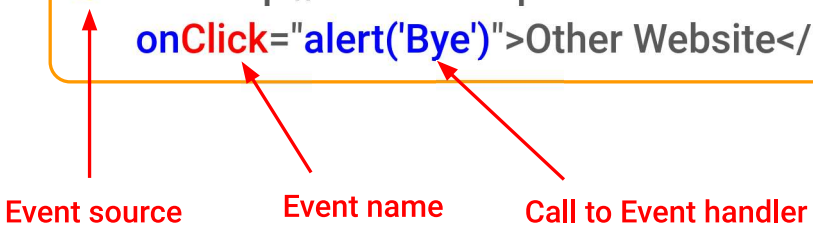
Event source — Event name — Event handler name

Removing Listener using DOM API

- We need to remove event handler if
  - The desired work gets completed.
  - Keeping event handler may affect to other parts of the page.
- To remove a registered listener, we use removeEventListener().
  - The parameters are same as in addEventListener().
- For example, we can remove event handler changeBackground from a button element btn for click event.
  - `btn.removeEventListener("click", changeBackground);`

Event Handling and Registration using HTML Attributes

- We can specify event handlers as event attributes in the HTML tags.
- The attribute names typically take the form onxxx where xxx is the event name.
- The event handler is passed as a string as a function call to inbuilt function or a function defined in JavaScript that is present or included in the page.
- For example:
  - `<a href="http://www.example.com" onClick="alert('Bye')">Other Website</a>`



Uses of JavaScript and its Events

- Generate HTML content at runtime (on the fly).
- Handling time, clock (for example, timer for test/exam).
- Slideshow of images.
- Render real time data (for example, cricket scorecard).
- Asynchronous communication (AJAX).

Uses of Event Handling

- User Interaction:
  - Click Events: Triggered when a user clicks on an element, such as a button or a link.
  - Mouse Events: Include events like mouseover, mouseout, mousedown, and mouseup.
  - Keyboard Events: Capture user keyboard inputs, like keydown, keyup, and keypress.
- Interactivity and UX:
  - Hover Events: Change styles or trigger actions when a user hovers over an element, enhancing the user experience.

Uses of Event Handling

- Form Handling:
  - Submit Events: Handle form submissions to perform validation or initiate actions before submitting data to the server.
  - Change Events: Triggered when the value of a form element changes, useful for real-time validation or updating other elements.
- Browser Events:
  - Window Events: Handle events related to the browser window, such as resize, scroll, and focus events.
- Error Handling:
  - Error Events: Capture and handle errors that may occur during the execution of JavaScript code.

Other Uses of Event Handling

- Document Loading:
- Animation and Transitions:
- Drag and Drop:
- Media Events:
- WebSocket Events:

Mostly used events

- Mouse related events:
  - onmousedown: called when the mouse button was pressed.
  - onmouseup: called when the mouse button was released.
  - onmousemove: called when the mouse was moved.
  - onmouseover: called when mouse pointer moves over the element.
  - onmouseout: called when the mouse moved off of this element.
- Keyboard related events:
  - onkeydown: called when a key was pressed down.
  - onkeypress: called when a key was pressed.
  - onkeyup: called when a key was pressed.

Mostly used events

- Form related events:
  - onsubmit: called when submit button is clicked.
  - onreset: called when the reset button is clicked.

Mostly used events

- Form fields related events:
  - onchange:
    - called when a control loses focus.
    - called while the value of its contents has changed.
  - onblur: called when a form field lost the focus (when focus moves to another field).
  - onfocus: called when control receives focus.
  - onclick: called when this item is clicked.
  - ondblclick: called when the item is double-clicked.

Mostly used events

- Other events:
  - onabort: called when an image failed to load.
  - onload: called after the object (iframe, image, script) finished loading.
  - onbeforeunload:
    - Called just before the user is navigating away from the page.
    - Called when the user closes a browser or tab.
  - onunload:
    - Called when the user is leaving the current page. The onunload event occurs after onbeforeunload event.
  - onresize: called when the window or frame was resized by the user.

onError event handler and Error object

- The onerror event handler:
  - The onerror event is used to handle errors that occur during the loading of an external resources:
    - image, script, or stylesheet.
  - The onerror is associated with the window object.
  - The Error object:
    - Default object for representing an exception.
    - Each Error object has a name and message properties.

Add Many Event Handlers to the Same Element

- Example:
  - element.addEventListener("click", myFunction);
    element.addEventListener("click", mySecondFunction);

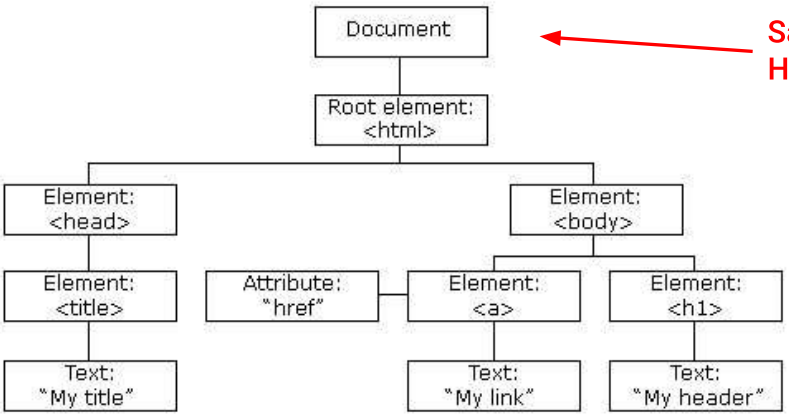
Add Events of Different Types to the Same Element

- Example:
  - element.addEventListener("mouseover", myFunction);
    element.addEventListener("click", mySecondFunction);
    element.addEventListener("mouseout", myThirdFunction);

JavaScript and DOM

Document Object Model (DOM)

- DOM is a model for describing documents in a tree-structure.
- It was designed to provide uniform access to structured documents in diverse applications (parsers, browsers, editors, databases).



Sample DOM tree for an HTML Document

History of DOM

- Originally, the Document Object Model (DOM) and JavaScript were tightly bound.
- Each major browsers (i.e., IE and Netscape) had their own overlapping DOM implementation.

DOM Now

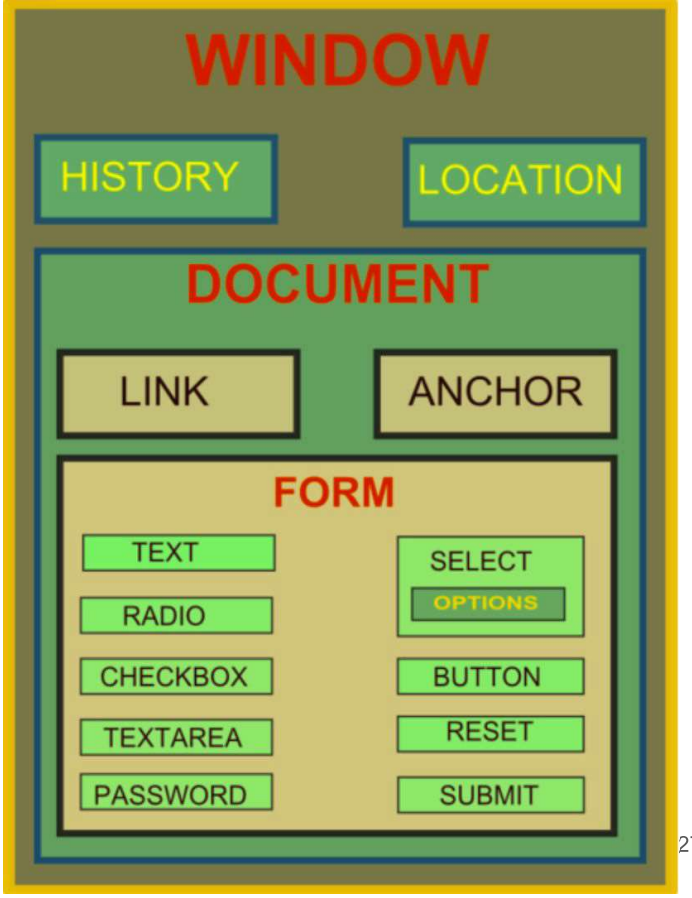
- Now, the DOM is a separate standard, and is available in other languages:
  - Java, Server side JavaScript, Python, etc.
- The DOM defines a standard for accessing and manipulating documents.
  - Documents could be HTML, XML.
- DOM API is an object-based, language-neutral API for XML and HTML documents:
  - HTML DOM defines a standard way for accessing and manipulating HTML documents.
  - The XML DOM defines a standard way for accessing and manipulating XML documents.

What is DOM? and What is HTML DOM?

- The DOM is a W3C (World Wide Web Consortium) standard.
- The DOM defines a standard for accessing documents.
- The W3C DOM standard is separated into 3 different parts:
  - Core DOM - standard model for all document types.
  - XML DOM - standard model for XML documents.
  - HTML DOM - standard model for HTML documents.
- What is HTML DOM?
  - The HTML DOM is a standard object model and programming interface for HTML.

HTML DOM structure

- Browser builds a model of the webpage, called document object.
- This document object includes everything (as objects) that is part of the webpage.
- These objects placed inside document object can be accessed from script using DOM API.
- Objects are in a hierarchy.
- The window is the parent for a given web page.



Uses of DOM API

- Find HTML elements in HTML Document.
- Change HTML elements in HTML Document.
- Add HTML elements in HTML Document.
- Delete HTML elements in HTML Document.
- Attach and detach event handlers to and from HTML elements.

Modern libraries or frameworks do all these things transparently with less complex code.

Manipulating HTML DOM with JavaScript

- With the HTML DOM, JavaScript can access and change all the elements of an HTML document.
  - JavaScript can change the HTML elements in the page.
  - JavaScript can change the HTML attributes in the page.
  - JavaScript can change the CSS styles in the page.
  - JavaScript can remove existing HTML elements and attributes.
  - JavaScript can add new HTML elements and attributes
  - JavaScript can react to all existing HTML events in the page.
  - JavaScript can create new HTML events in the page.

HTML DOM: Finding HTML Elements

- There are several ways to find HTML elements:
  - Finding HTML element by id.
    - document.getElementById(id)
  - Finding HTML elements by tag name.
    - document.getElementsByTagName(name)
  - Finding HTML elements by class name.
    - document.getElementsByClassName(name)

HTML DOM: Finding HTML elements

- Finding HTML elements by CSS selectors.
  - If we want to find all HTML elements that match a specified CSS selector (id, class names, types, attributes, values of attributes, etc), use the querySelectorAll() method.
  - const x = document.querySelectorAll("p.intro");

Property: innerHTML

- The innerHTML is a property of any document element that contains all of the text within that element (including html source code)
  - innerHTML considers everything as a string, not as DOM objects.
- It is not part of the DOM standard.
  - It is not a standard property.
  - But widely supported
  - Before DOM came, it was used to manipulate web pages.

HTML DOM: Changing HTML Elements

- We can change HTML content using innerHTML property of element:
  - element.innerHTML = <new html content>
  - document.getElementById(id).innerHTML = <new html content>
- Change the attribute value of an HTML element:
  - element.setAttribute(attribute, value)
- Change the style of an HTML element
  - element.style.property = <new style>

Example: Find and Manipulate HTML Elements

dom-manipulation1.html

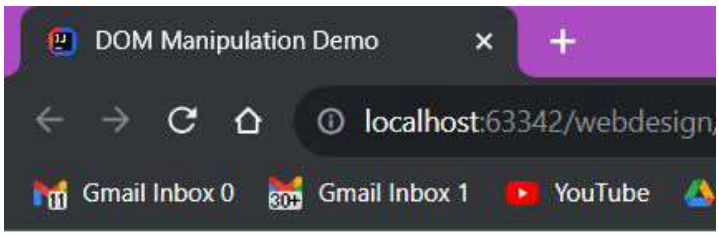
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>DOM Manipulation Demo</title>
    <script>
        1 usage
        function changeParaStyles(){
            const paraElements = document.getElementsByTagName("p");
            for (paragraphEl of paraElements){
                paragraphEl.style.color = "blue";
            }
        }
    </script>
</head>
<body>
    <p>First Paragraph ...</p>
    <p>Second Paragraph ...</p>
    <p>Third Paragraph ...</p>
    <input type="button" onclick="changeParaStyles()" value="Change Paragraph Color"/>
</body>
</html>
```

Find by tag name

Change style

Manipulate the elements

Run Application

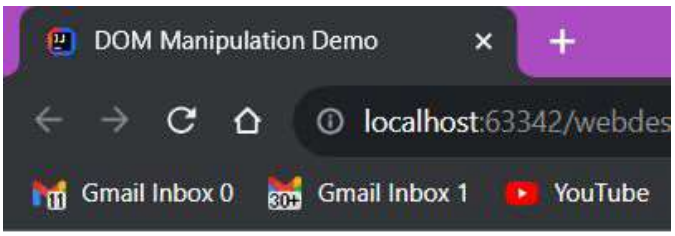


First Paragraph ...

Second Paragraph ...

Third Paragraph ...

Change Paragraph Color



First Paragraph ...

Second Paragraph ...

Third Paragraph ...

Change Paragraph Color

Color of all paragraphs change to blue

Click this button

dom-manipulation2.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>DOM Manipulation Demo</title>
    <script>
        1 usage
        function changePara1Styles(){
            const paraEl = document.getElementById("para-1");
            paraEl.style.color = "blue";
        }
        1 usage
        function changePara1Content(){
            const paraEl = document.getElementById("para-1");
            paraEl.innerHTML = "Updated First Paragraph ..."
        }
        1 usage
        function changeParaStyles(){
            const paraElements = document.querySelectorAll("p.highlight");
            for (paragraphEl of paraElements){
                paragraphEl.style.color = "red";
            }
        }
```

Find by id

Change style of a particular element

Find by id

Change innerHTML of a particular element

Find elements by CSS selector

Manipulate the elements

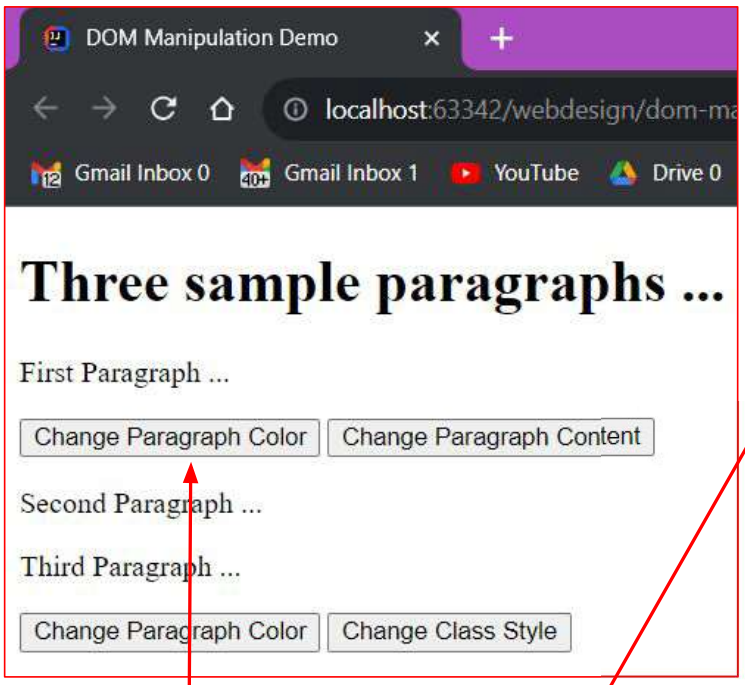
dom-manipulation2.html

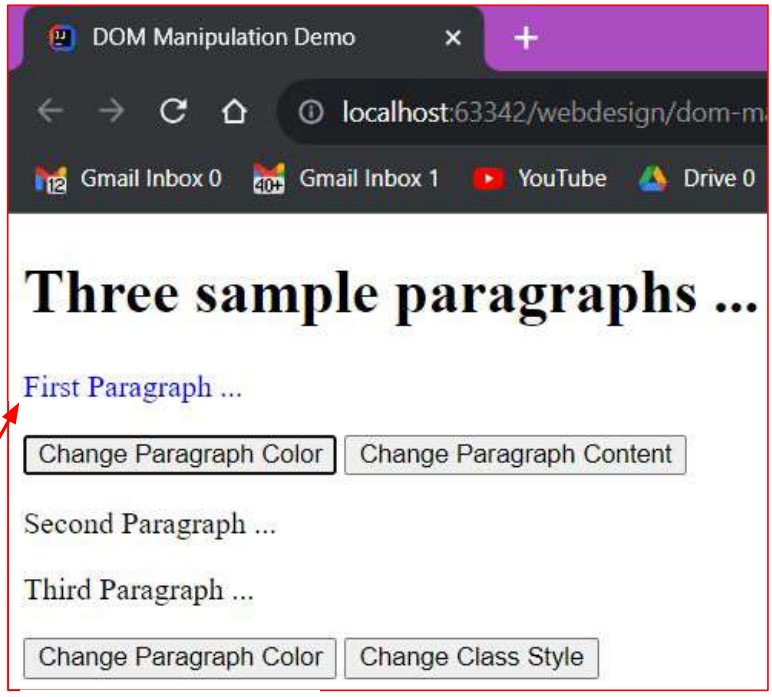
Find elements by class name

```
        function changeClassStyles(){
            const highlightEls = document.getElementsByClassName("highlight");
            for (highlightEl of highlightEls){
                highlightEl.style.color = "red";
            }
        }
    </script>
</head>
<body>
    <h1 class="highlight">Three sample paragraphs ...</h1>
    <p id="para-1">First Paragraph ...</p>
    <input type="button" onclick="changePara1Styles()" value="Change Paragraph Color"/>
    <input type="button" onclick="changePara1Content()" value="Change Paragraph Content"/>
    <p class="highlight">Second Paragraph ...</p>
    <p class="highlight">Third Paragraph ...</p>
    <input type="button" onclick="changeParaStyles()" value="Change Paragraph Color"/>
    <input type="button" onclick="changeClassStyles()" value="Change Class Style"/>
</body>
</html>
```

Manipulate the elements

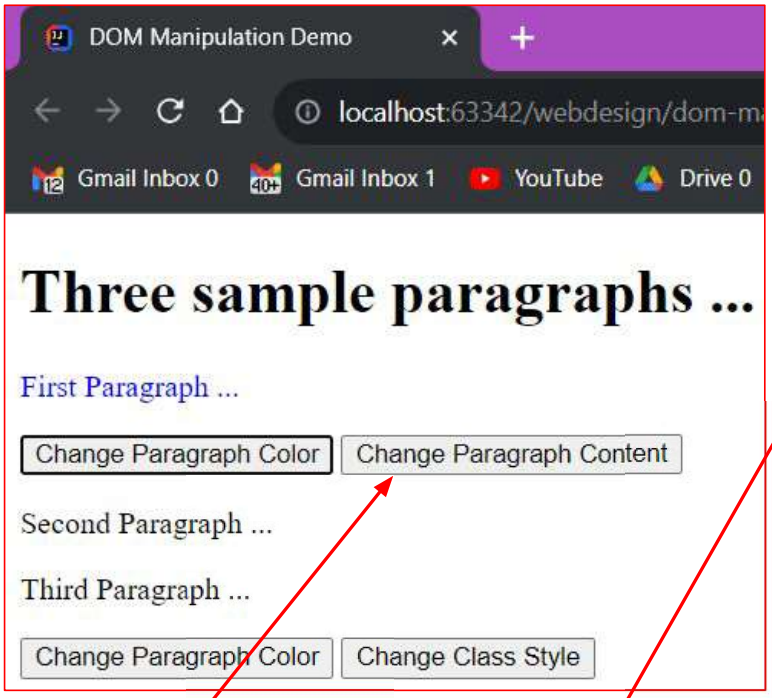
Run Application

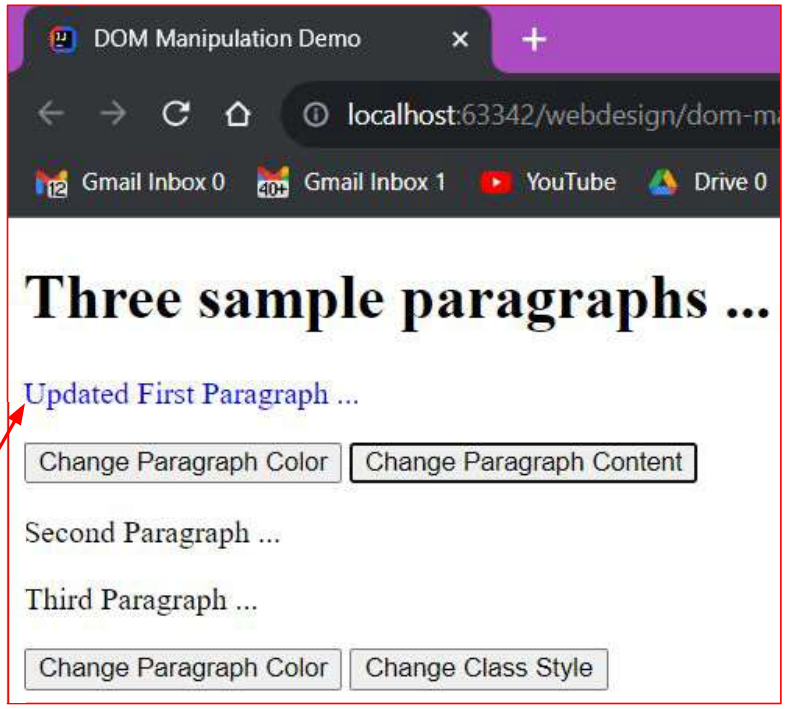




Click this button

Color of only first paragraph changes

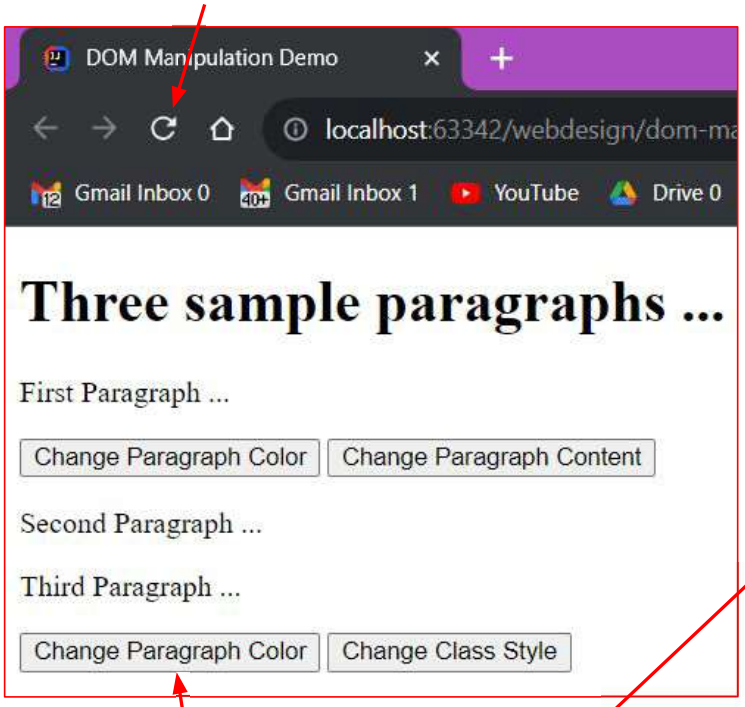


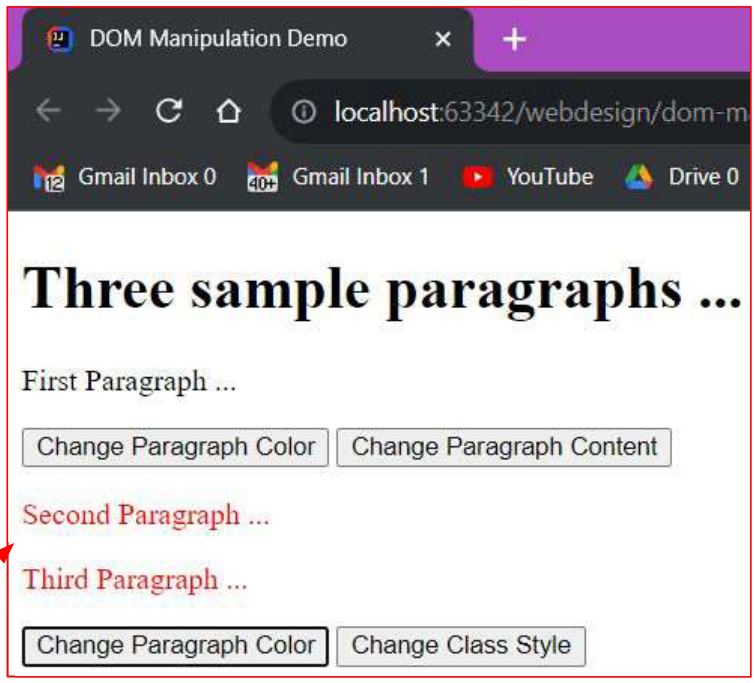


Click this button

Content of only first paragraph changes

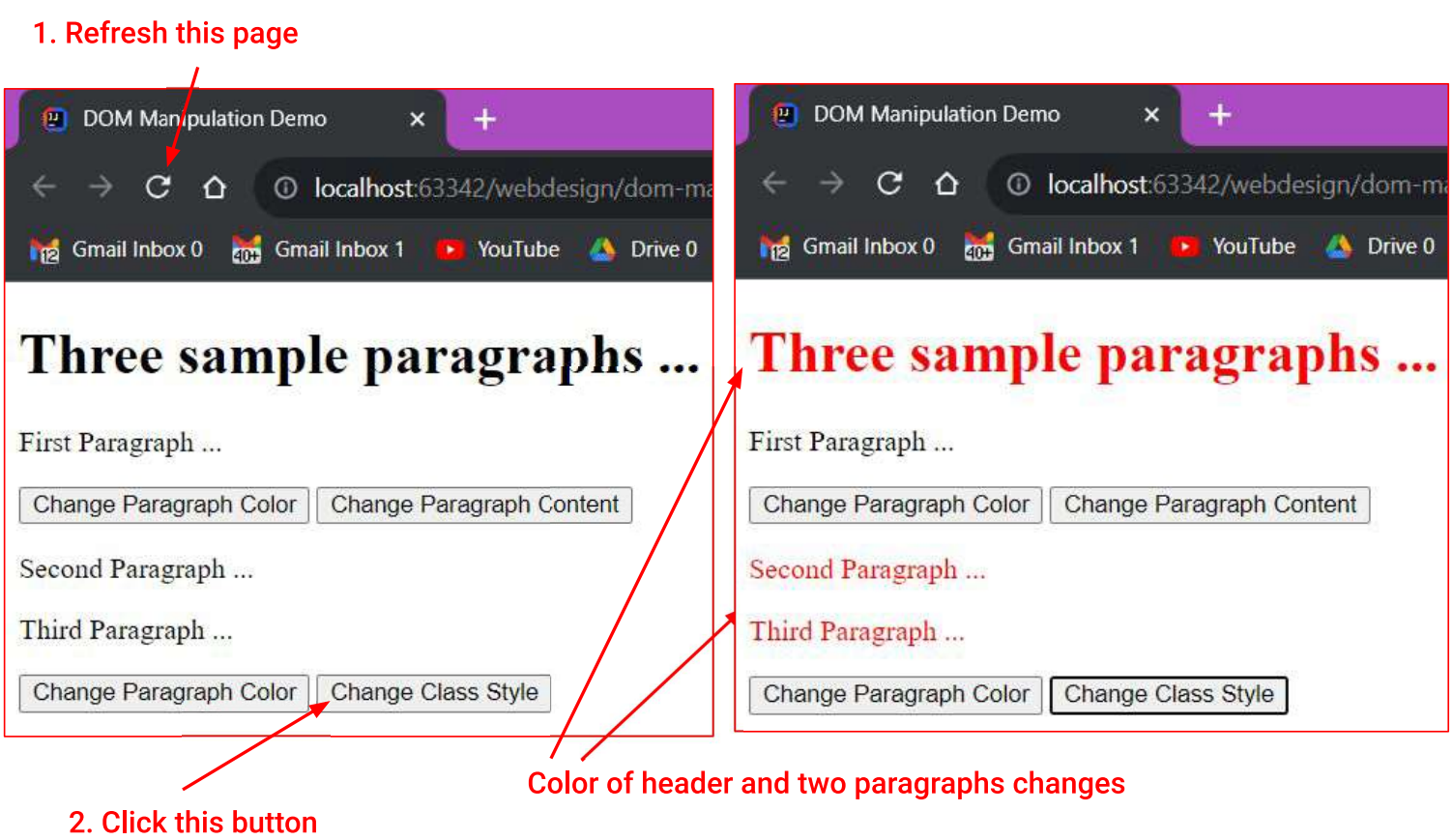
1. Refresh this page





2. Click this button

Color of two paragraphs changes



HTML DOM: Adding and Deleting HTML Elements

- Create an HTML element:
  - document.createElement(element)
- Remove an HTML element:
  - document.removeChild(element)
- Add an HTML element:
  - document.appendChild(element)
- Replace an HTML element:
  - document.replaceChild(new, old)
- Write into the HTML output stream:
  - document.write(text)

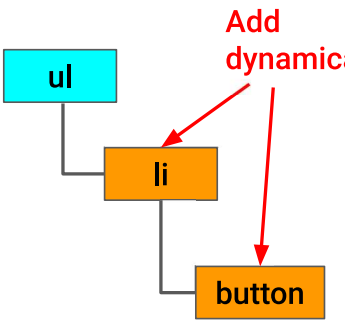
Example: Adding and Deleting HTML Elements

dom-manipulation3.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>DOM Manipulation Demo</title>
    <script>
        1 usage
        function addPart() {
            // Create a new <li> under <ul>
            const ulEl = document.getElementById("parts");
            let liEl = document.createElement("li");

            // Assign value of input to new <li>
            let inputEl = document.getElementById("part");
            liEl.textContent = inputEl.value+" ";

            // Create a remove button for the new <li>
            let removeButton = document.createElement("button");
            removeButton.textContent = "Remove";
            removeButton.onclick = function () {
                ulEl.removeChild(liEl);
            };
```



Find by id

Create new <li> element

Find by id

Assign text content

Remove whole <li> from event handler

Create new <button> element

dom-manipulation3.html

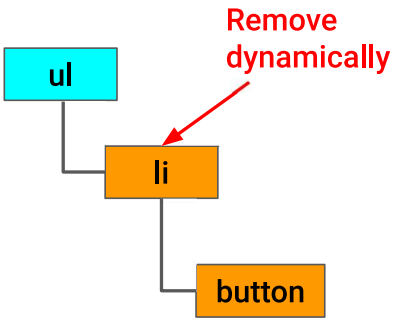
```
23              // Append the remove button to the new <li>
24              liEl.appendChild(removeButton);
25
26              // Append new <li> under <ul>
27              ulEl.appendChild(liEl);
28              inputEl.value = "";
29
30              // Bring focus to the input for the next entry
31              inputEl.focus();
32          }
33      </script>
34  </head>
35  <body>
36  <h1>Parts of a Computer</h1>
37  <ul id="parts">
38      <li>Monitor</li>
39  </ul>
40  Enter Part name: <input type="text" id="part" />
41  <input type="button" onclick="addPart()" value="Add Part"/>
42  </body>
43  </html>
```

Append remove <button> element to <li> element

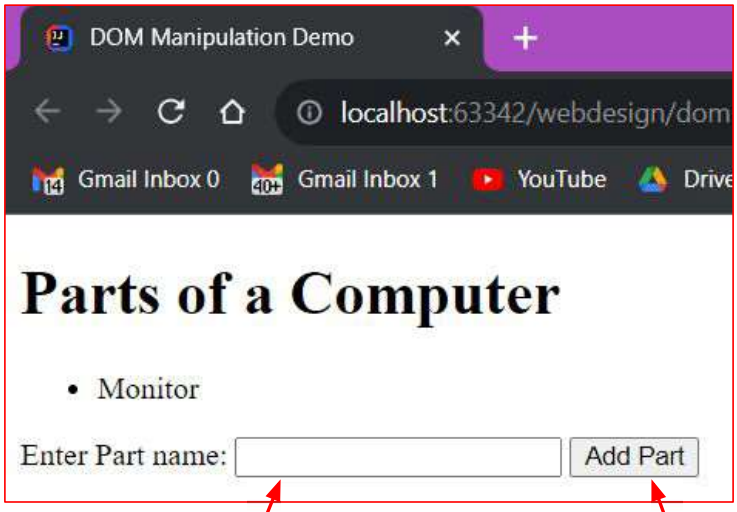
Append <li> element under <ul> element

Clear <input>

Bring focus into <input>



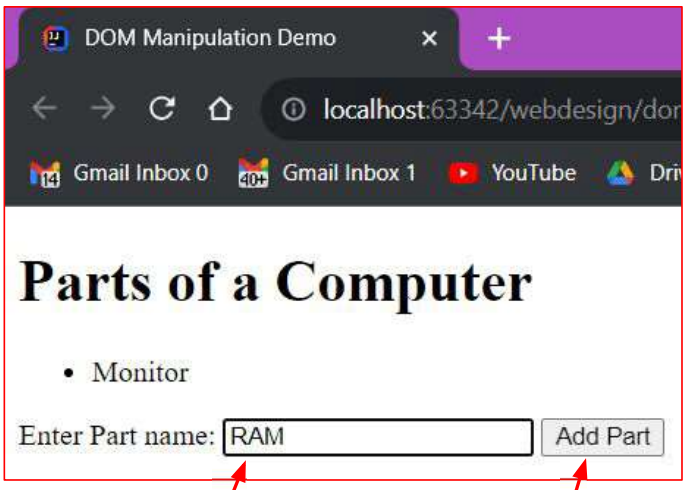
Run Application



1. We enter part name here

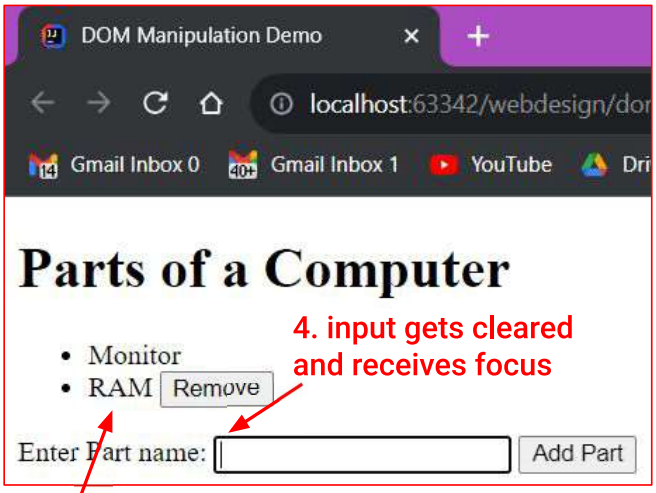
2. Then we click this button

Run Application



1. Enter part name

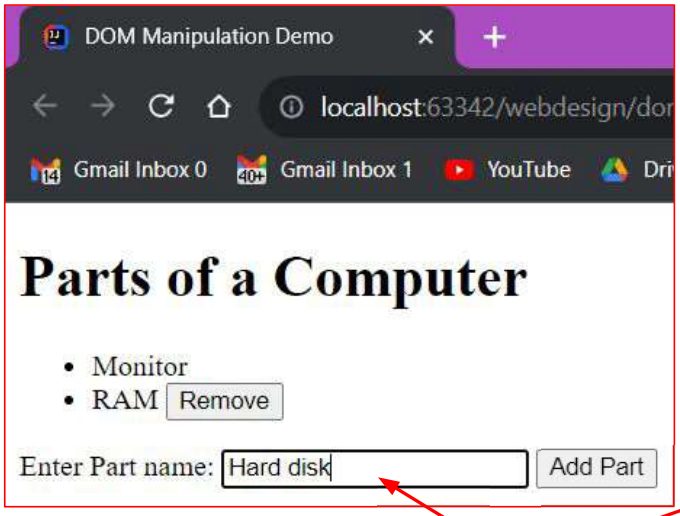
2. Click this button



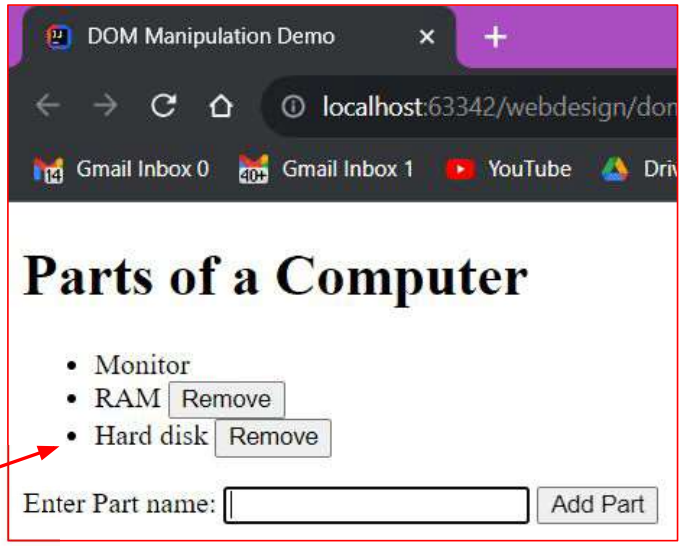
4. input gets cleared and receives focus

3. Part and Remove button gets added

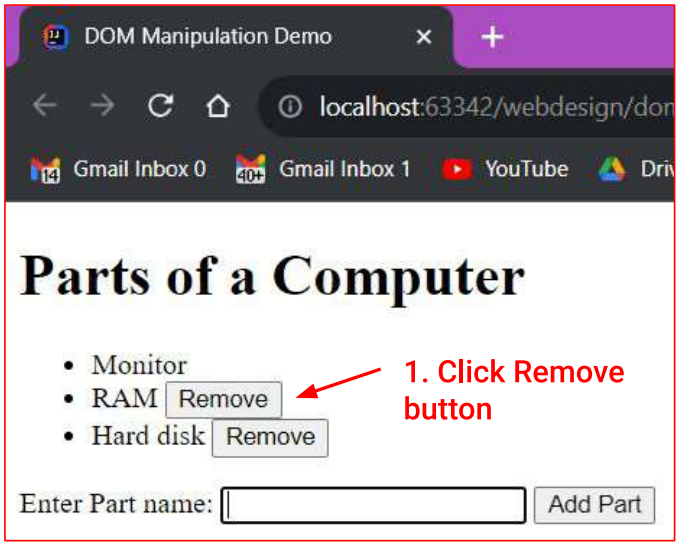
Run Application

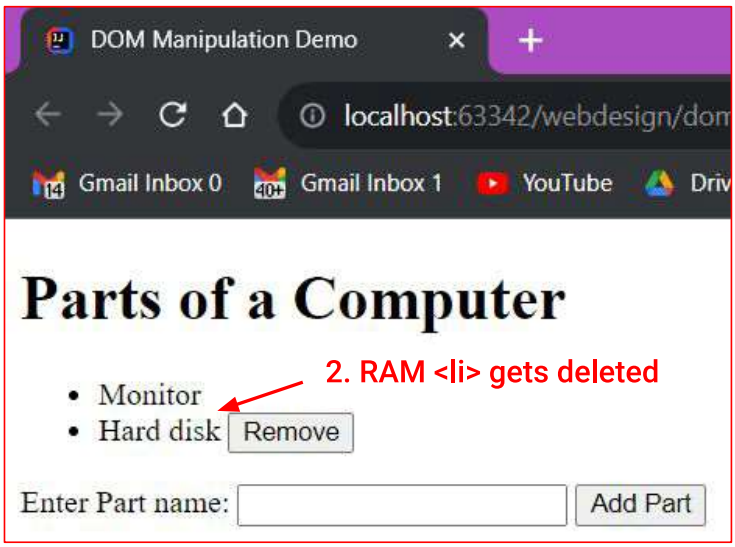


1. Enter second part



Run Application





Example: Form Handling

form-handling.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Form Handling Demo</title>
</head>
<body>
    <form id="myForm">
        <label for="name">Name:</label>
        <input type="text" id="name" name="name" required><br><br>

        <label for="phone">Phone:</label>
        <input type="tel" id="phone" name="phone" required><br><br>

        <label for="email">Email:</label>
        <input type="email" id="email" name="email" required><br><br>

        <input type="submit" value="Submit">
    </form>
```

We provide id to the form

form-handling.html

```html
    <script>
        document.getElementById("myForm").addEventListener("submit", function(event) {
            // Prevent the form from submitting the traditional way
            event.preventDefault();

            // Get values from the form
            var name = document.getElementById("name").value;
            var phone = document.getElementById("phone").value;

            // Display values on the console
            console.log("Name: " + name);
            console.log("Phone: " + phone);
            console.log("Email: " + document.forms["myForm"]["email"].value);
        });
    </script>
</body>
</html>
```

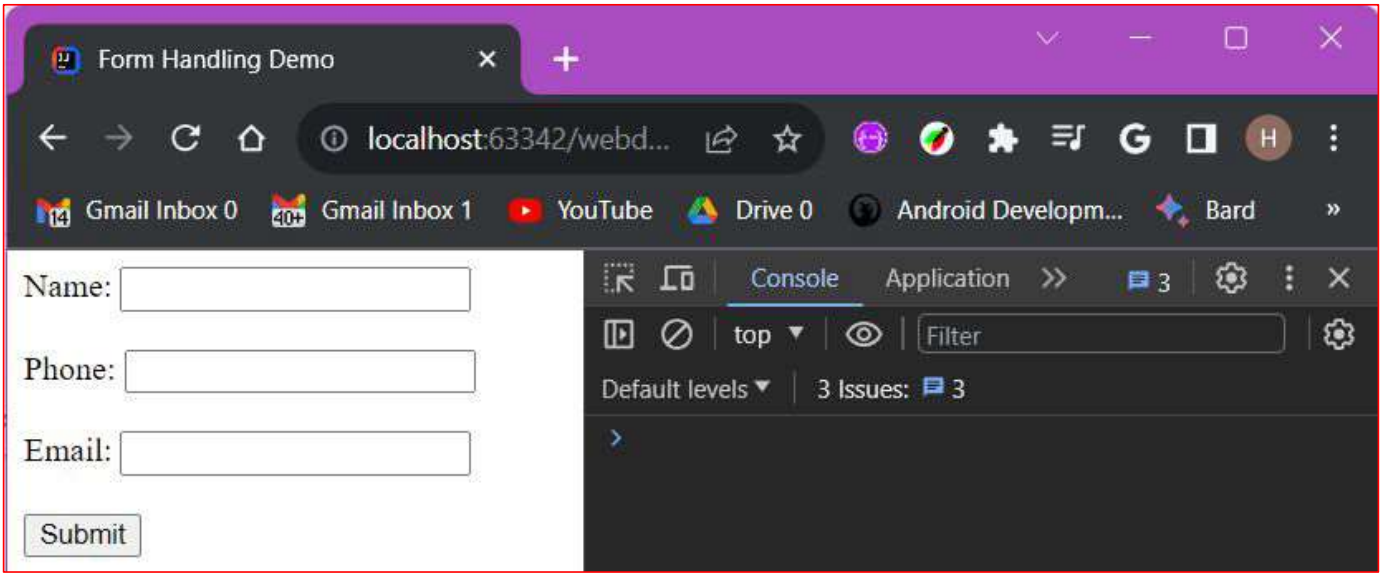
Handle form submit event

Prevent default behaviour of event

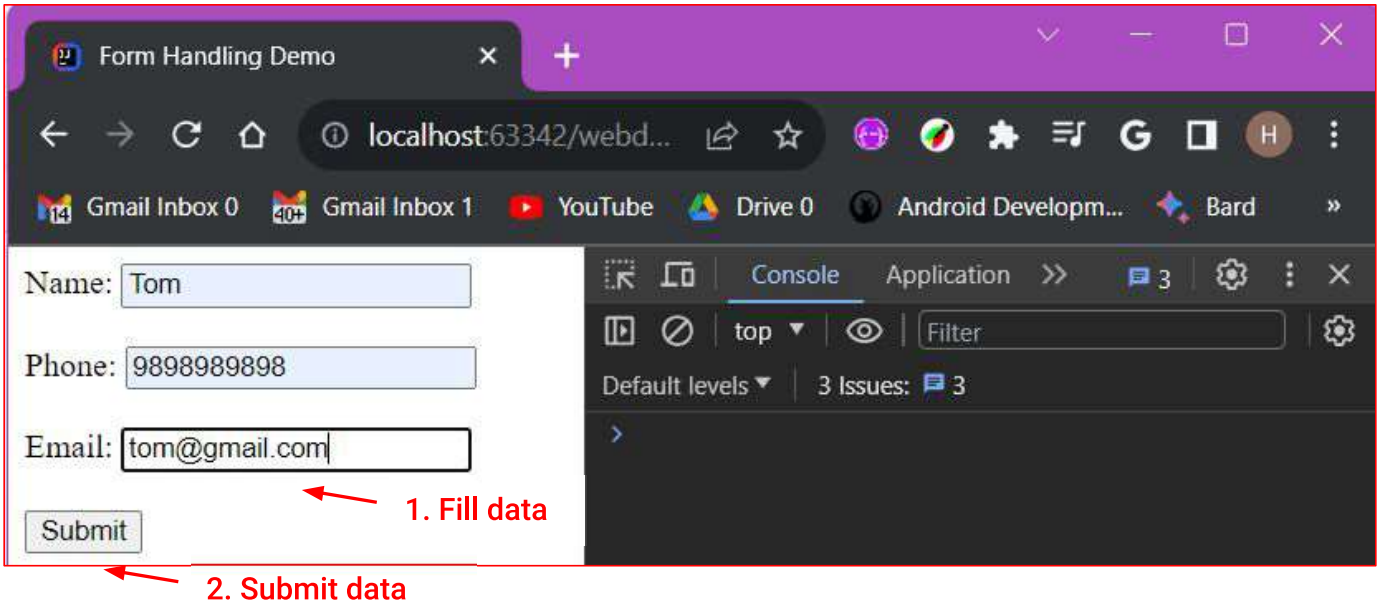
Find elements using ids.

Access form element using form object

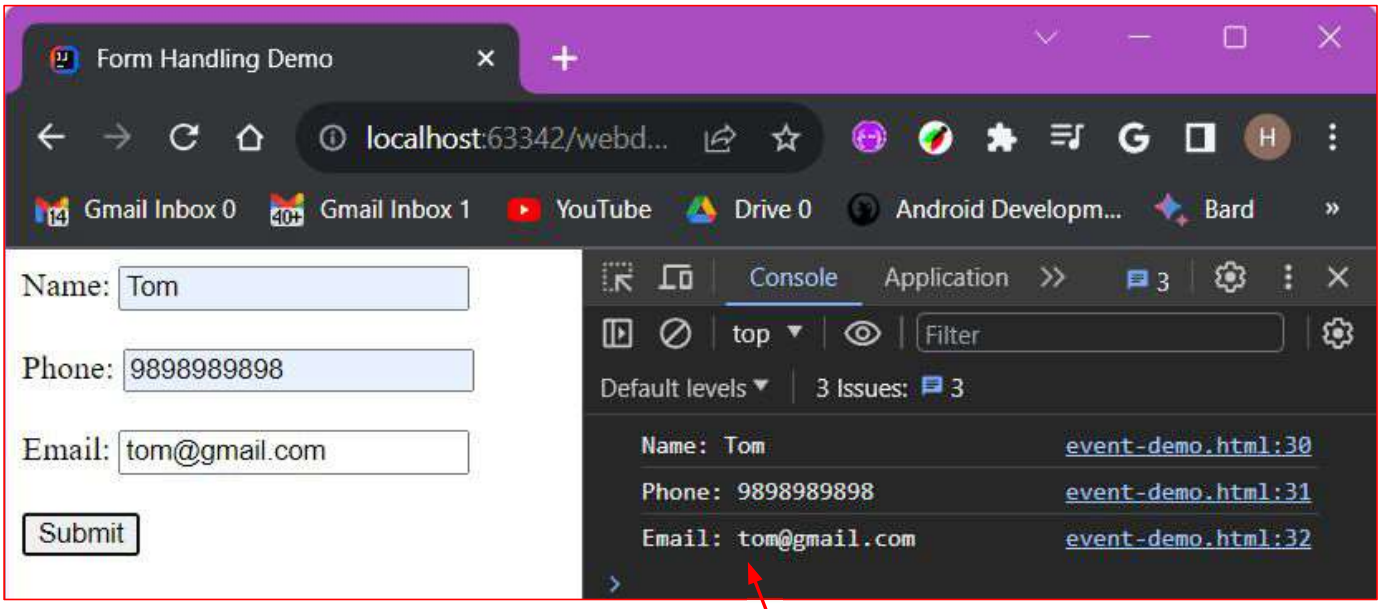
Run Application



Run Application



Run Application

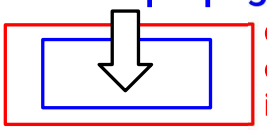


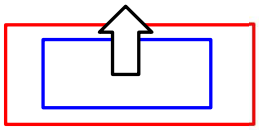
We can see filled (submitted) values

Event bubbling or event capturing

Event Bubbling or Event Capturing

- There are two ways of event propagation in the HTML DOM:
  - bubbling
  - capturing





- If we have nested elements, and we trigger event on nested elements, then whose event handler should be called first nested element or outer element?
- Bubbling:
  - the innermost element's event is handled first and then the outer's.
  - It is default behaviour. If value for useCapture parameter is omitted.
- Capturing:
  - The outermost element's event is handled first and then the inner's.

event-types.html

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Event Type Demo</title>
6  </head>
7  <body>
8      <div id="outer">
9          <div id="inner">
10             Click me
11         </div>
12     </div>
13     <script>
           2 usages
14         function clickHandler(event){
15             console.log("event called on target = ",event.target);
16             console.log("event called on currentTarget = ",event.currentTarget);
17         }
18         document.getElementById("outer").addEventListener("click", clickHandler, true);
19         document.getElementById("inner").addEventListener("click", clickHandler, true);
20     </script>
21 </body>
22 </html>
```

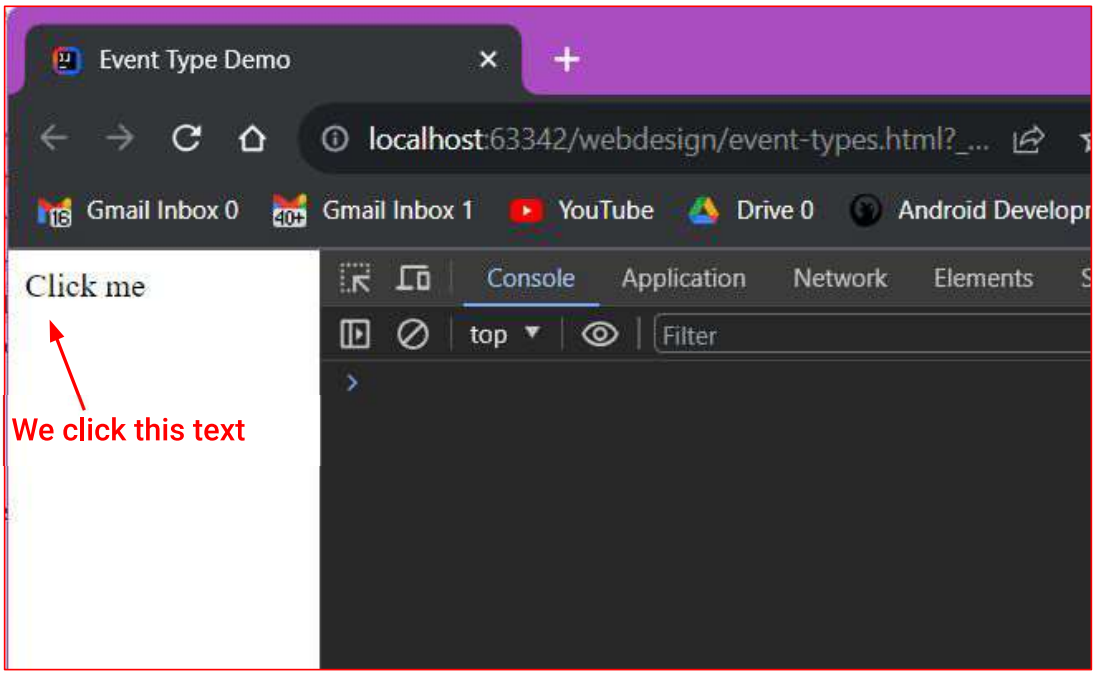
We print target

We print currentTarget

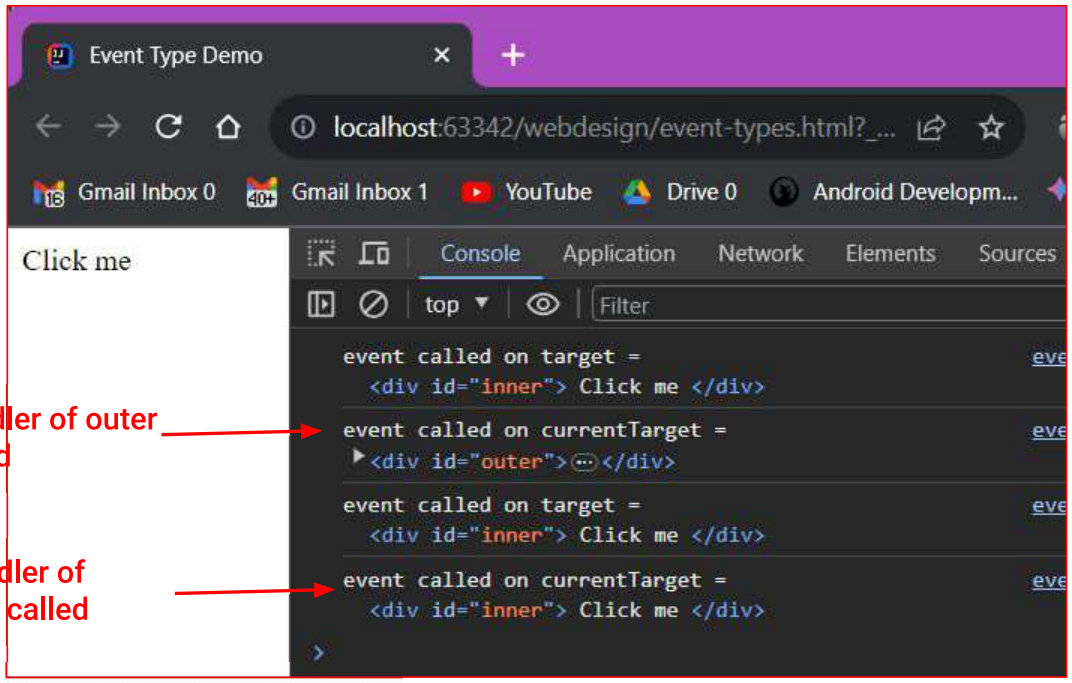
Register event handler for both

useCapture

Run Application



Run Application



First, handler of outer gets called

Then, handler of inner gets called

This behaviour is called event capturing.

event-types-1.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Event Type Demo</title>
</head>
<body>
    <div id="outer">
        <div id="inner">
            Click me
        </div>
    </div>
    <script>
        2 usages
        function clickHandler(event){
            console.log("event called on target = ",event.target);
            console.log("event called on currentTarget = ",event.currentTarget);
            }
        document.getElementById("outer").addEventListener("click", clickHandler);
        document.getElementById("inner").addEventListener("click", clickHandler);
    </script>
</body>
</html>
```

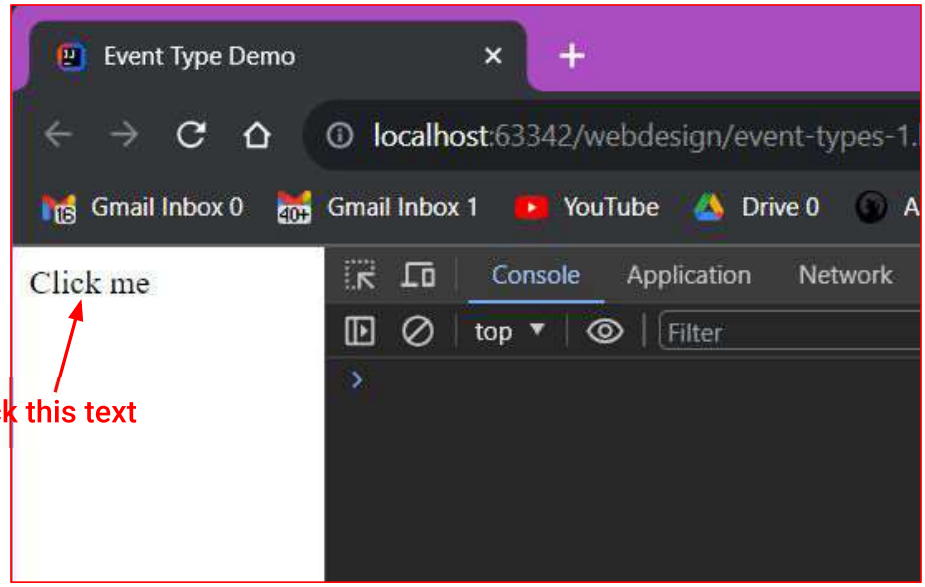
We print target

We print currentTarget

Register event handler for both

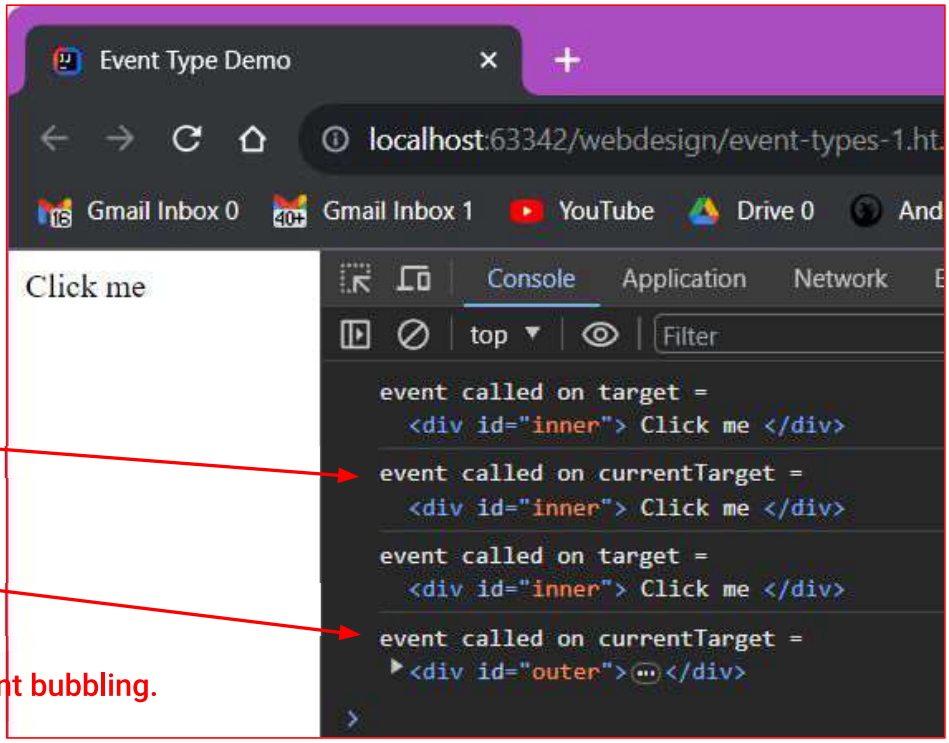
We do not specify third parameter

Run Application



We click this text

Run Application



First, handler of inner gets called

Then, handler of outer gets called

This behaviour is called event bubbling.

event-types-2.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Event Type Demo</title>
</head>
<body>
    <div id="outer">
        <div id="inner">
            Click me
        </div>
    </div>
    <script>
        2 usages
        function clickHandler(event){
            console.log("event called on target = ",event.target);
            console.log("event called on currentTarget = ",event.currentTarget);
        }
        document.getElementById("outer").addEventListener("click", clickHandler, false);
        document.getElementById("inner").addEventListener("click", clickHandler, false);
    </script>
</body>
</html>
```

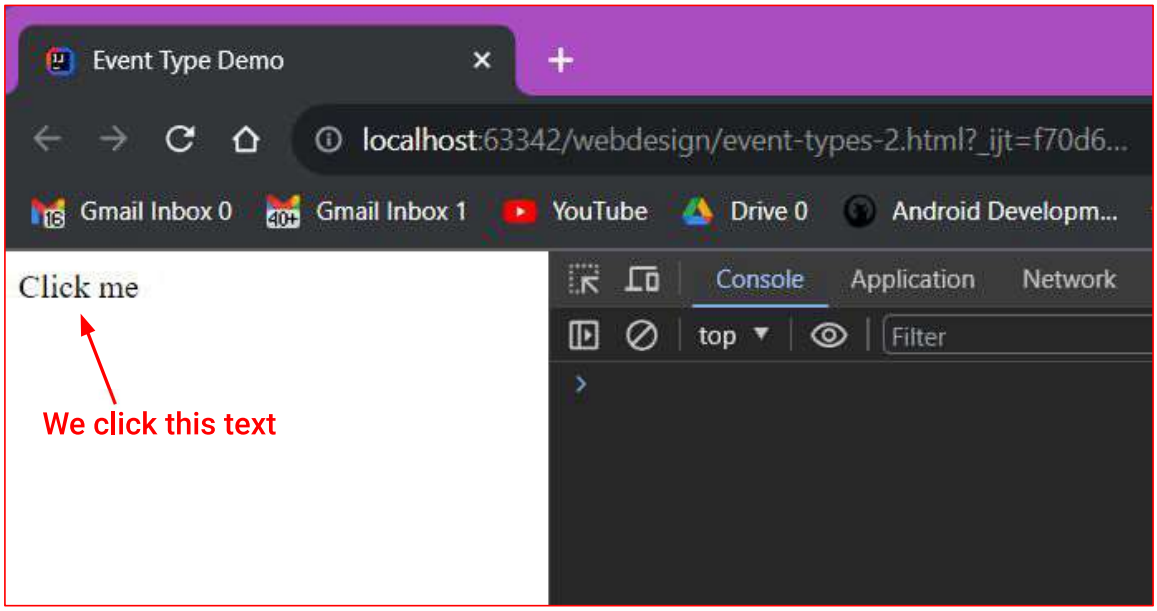
We print target

We print currentTarget

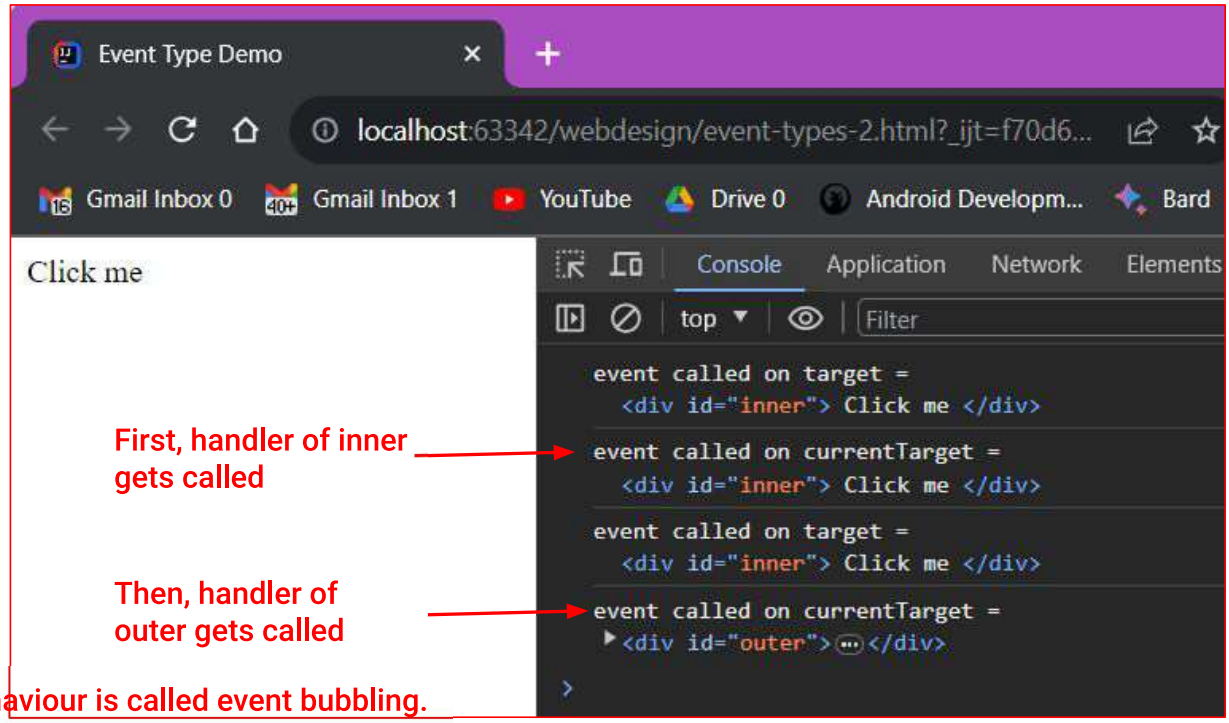
Register event handler for both

useCapture

Run Application



Run Application



This behaviour is called event bubbling.

References

- https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction
- https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model
- https://developer.mozilla.org/en-US/docs/Web/Events/Event_handlers