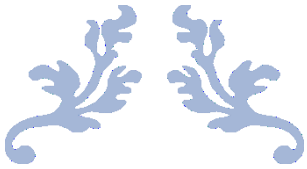


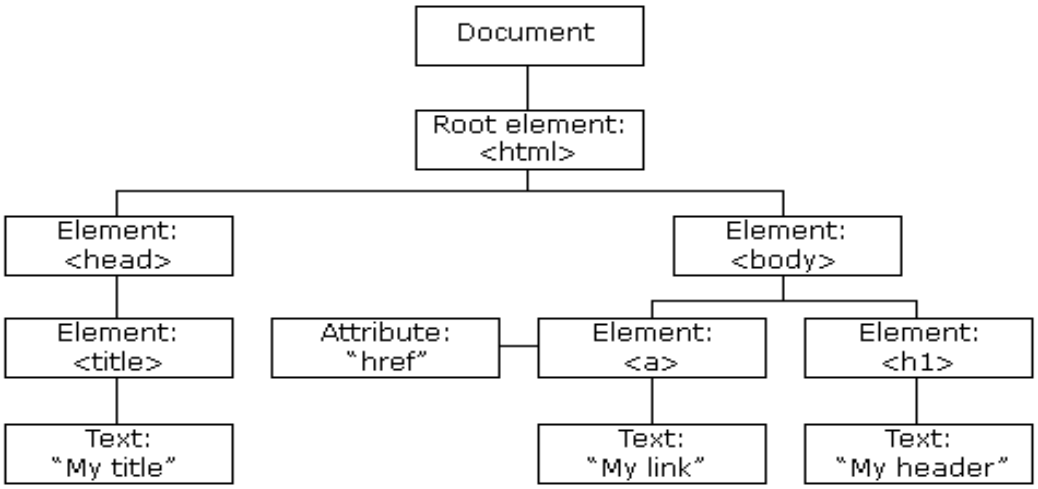
CHAPTER 9

JavaScript DOM Manipulation & Events

DOM (Document Object Model)

When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page. Every web page resides inside a browser window which can be considered as an object.

- ✓ The **HTML DOM** model is constructed as a tree of **Objects**.
- ✓ The HTML DOM Tree of Objects



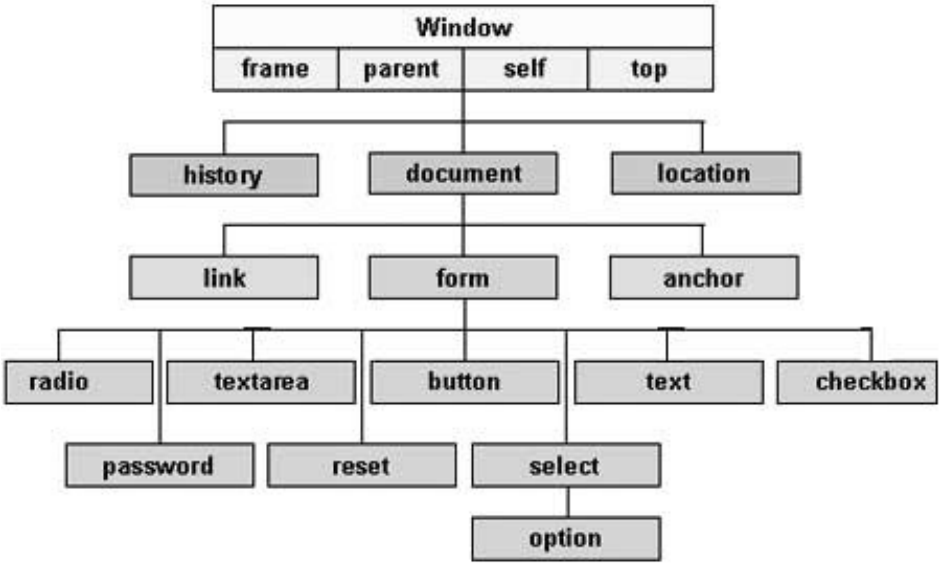
DOM is a "tree structure" representation created by the browser that enables the HTML structure to be easily accessed by programming languages for example the browser itself uses it to apply styling and other information to the correct elements as it renders a page, and developers like you can manipulate the DOM with JavaScript after the page has been rendered.

The way a document content is accessed and modified is called the **Document Object Model**, or **DOM**. The Objects are organized in a hierarchy. This hierarchical structure applies to the organization of objects in a Web document.

- **Window object** − Top of the hierarchy. It is the outmost element of the object hierarchy.
- **Document object** − Each HTML document that gets loaded into a window becomes a document object. The document contains the contents of the page.
- **Form object** − Everything enclosed in the <form>...</form> tags sets the form object.
- **Form control elements** − The form object contains all the elements defined for that object such as text fields, buttons, radio buttons, and checkboxes.

DOM Manipulation follows this sequence:
document → select element → modify content / attributes / style

Here is a simple hierarchy of a few important objects −



The **HTML DOM** is an **API** (Programming Interface) for **JavaScript**:

- ✓ JavaScript can add/change/remove HTML elements
- ✓ JavaScript can add/change/remove HTML attributes
- ✓ JavaScript can add/change/remove CSS styles
- ✓ JavaScript can react to HTML events
- ✓ JavaScript can add/change/remove HTML events

The Document object has various properties that refer to other objects which allow access to and modification of document content.

Topic / Method	Description	Basic Requirement	Syntax
document object	Represents the entire HTML document	HTML page loaded in browser	document.property/method
getElementById()	Finds ONE element using unique id	Element must have id attribute	document.getElementById("id")
getElementsByClassName()	Finds ALL elements with same class	Elements must share same class	document.getElementsByClassName("class")
querySelector()	Selects FIRST matching element (CSS selector)	Valid CSS selector	document.querySelector("selector")

Topic / Method	Description	Basic Requirement	Syntax
querySelectorAll()	Selects ALL matching elements	Valid CSS selector	document.querySelectorAll("selector")
innerHTML	Gets or sets HTML content inside element	Valid HTML element	element.innerHTML="text"
innerText	Gets or sets ONLY text (no HTML)	Valid HTML element	element.innerText="text"
setAttribute()	Sets/changes attribute value	Element reference	element.setAttribute(name,value)
getAttribute()	Gets attribute value	Existing attribute	element.getAttribute(name)
removeAttribute()	Removes attribute from element	Existing attribute	element.removeAttribute(name)
Style Manipulation	Changes CSS styles dynamically	Element reference	element.style.property=value

1. document object

What is it?

document is the **main entry point** to the HTML page.

It represents the **entire web page loaded in the browser**.

Why we use it

- Without document, JavaScript **cannot access HTML**
- All DOM methods come from document

When to use

- Whenever you want to **read, write, or modify HTML**

Example

```
<script>
document.write("Welcome to DOM");
</script>
```

Explanation

- document.write() writes content **directly to the page**
- Using document.write() after page load can overwrite the entire document.

2. getElementById()

What is it?

Selects **ONE element** using its **unique ID**

Why we use it

- Fastest and simplest DOM method
- IDs are unique → only one element returned

When to use

- When you know the element's **id**
- When only **one element** is needed

Example

```
<p id="p1">Hello</p>
<script>
document.getElementById("p1").innerHTML = "Hi";
</script>
```

Explanation

- Finds <p id="p1">
- Changes its content to "Hi"
- Returns null if ID does not exist

3. getElementsByClassName()

What is it?

Selects **ALL elements** with the same class name

Why we use it

- Classes are used for **grouping elements**
- Returns **HTMLCollection** (array-like)

When to use

- When multiple elements share **same class**
- When you want to apply **same change to many elements**

Example

```
<p class="c1">Text 1</p>
<p class="c1">Text 2</p>
<script>
var x = document.getElementsByClassName("c1");
x[0].innerHTML = "Changed";
</script>
```

Explanation

- x contains **both <p> elements**
- x[0] accesses the first one

Using for loop (IMPORTANT)

```
<p class="c1">Text 1</p>
<p class="c1">Text 2</p>
<script>
```

```
var x = document.getElementsByClassName("c1");
for (var i = 0; i < x.length; i++) {
   x[i].innerHTML = "Changed using for loop";
}
</script>
```

Why loop is needed

Because **multiple elements** are returned

Using for…of loop (IMPORTANT)

```
<p class="c1">Text 1</p>
<p class="c1">Text 2</p>
<script>
var x = document.getElementsByClassName("c1");
for (var x1 of x) {
   x1.innerHTML = "Changed using for loop";
}
</script>
```

4. querySelector()

What is it?

Selects the **FIRST matching element** using a **CSS selector**

Why it is powerful

- Uses **CSS rules**
- Can select:
  - tag (p)
  - class (.c1)
  - id (#p1)
  - combinations (div p, .box p)

When to use

- When selection is **complex**
- When you want **modern, readable code**

Example

```
<p>Paragraph</p>
<script>
document.querySelector("p").style.color = "red";
</script>
```

Explanation

- Finds the **first <p>**
- Changes text color to red

More querySelector() Examples

```
document.querySelector("#p1");     // by ID
```

```
document.querySelector(".c1");    // by class
document.querySelector("div p");  // p inside div
```

Even if multiple elements match, it returns **only the first one**

5. querySelectorAll()

What is it?

Selects **ALL matching elements** using CSS selectors

Why we use it

- More flexible than getElementsByClassName()
- Returns **NodeList**

When to use

- When selecting **multiple elements**
- When using **CSS selectors**

Example

```
<p>Para 1</p>
<p>Para 2</p>
<script>
var x = document.querySelectorAll("p");
x[1].innerHTML = "Second paragraph";
</script>
```

Explanation

- x contains all <p> elements
- x[1] refers to second paragraph

Using for loop

```
<script>
var x = document.querySelectorAll("p");
for (var i = 0; i < x.length; i++) {
  x[i].style.color = "blue";
}
</script>
```

Using for…of (Modern)

```
<script>
for (let p of document.querySelectorAll("p")) {
  p.style.fontSize = "20px";
}
</script>
```

6. innerHTML

What is it?

Changes **HTML content** inside an element

Why we use it

- Allows **HTML tags**
- Used for dynamic content

Example

```
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = "<b>Hello</b>";
</script>
```

Output: Text appears **bold**

7. innerText

What is it?

Changes **only text**, ignores HTML tags

Why we use it

- Prevents HTML rendering
- Safer for user input

Example

```
<p id="demo"></p>
<script>
document.getElementById("demo").innerText = "<b>Hello</b>";
</script>
```

Output

Hello

8. setAttribute()

What is it?

Adds or changes an attribute

When to use

- Changing src, href, class, etc.

Example

```
<img id="img1">
<script>
document.getElementById("img1").setAttribute("src", "image.jpg");
</script>
```

9. getAttribute()

What is it?

Reads an attribute value

Example

```
<p id="p1" title="Demo Text">Hello</p>

<script>
var a=document.getElementById("p1").getAttribute("title")
```

```
alert(a);
</script>
```

10. removeAttribute()

What is it?

Removes an attribute completely

Example

```
<p id="p1" title="Demo">Hello</p>
<script>
document.getElementById("p1").removeAttribute("title");
</script>
```

11. Style Manipulation

What is it?

Changes CSS using JavaScript

Why we use it

- Dynamic styling
- Respond to events

Example

```
<p id="p1">Hello</p>
<script>
document.getElementById("p1").style.backgroundColor = "yellow";
</script>
```

EVENT HANDLING

Event handling in JavaScript is the process of **detecting user actions or browser actions (events)** and **responding to them using JavaScript code**.

An **event** can be:

- Mouse click
- Key press
- Mouse movement
- Form submission
- Focus or blur of input field

JavaScript executes specific code when these events occur.

WHY DO WE USE EVENT HANDLING?

Event handling is used to make web pages **interactive and dynamic**.

It allows JavaScript to:

- Respond to user actions
- Validate form input
- Change content dynamically
- Change styles dynamically
- Control browser behavior

Without event handling, web pages would be **static**.

1. Function Call Without Parameter

Example

```
<script>
function down() {
   document.getElementById("test").innerHTML = "Mouse Down";
}
</script>
<p id="test" onmousedown="down()">test</p>
```

Explanation

- No argument is passed to the function.
- The function accesses the element using getElementById().
- Used when the element ID is fixed and known.

2. Function Call With “this”

Example

```
<script>
function up(x) {
   x.innerHTML = "Mouse Up";
}
</script>
<p onmouseup="up(this)">test</p>
```

Explanation

- this refers to the element that triggered the event.
- The element is passed to the function as a parameter.
- This method is reusable and cleaner.

3. Inline JavaScript (No Function Call)

```
<p onmouseover="this.style.color='red'; this.innerHTML='Mouse Over';">Test</p>
```

Explanation

- JavaScript is written directly inside the HTML event.
- this refers to the current element.
- Used only for very small logic or demonstrations.

Mouse events

Event Name	Description	When to Use	Simple Example
onclick	Fires when mouse click is completed	Button clicks	<button onclick="alert('Clicked')">Click</button>
onmousedown	Fires when mouse button is pressed	Detect press	Press
onmouseup	Fires when mouse button is released	Detect release	Release
onmouseover	Fires when mouse enters element	Hover effect	<h1 onmouseover="this.style.color='red'">Hover</h1>
onmouseout	Fires when mouse leaves element	Remove hover	<h1 onmouseout="this.style.color='black'">Out</h1>

onclick

- A JavaScript can be executed when an event occurs, like when a user clicks on an HTML element.
- To execute code when a user clicks on an element, add JavaScript code to an HTML event attribute:

Example:

```
<h2 onclick="alert(this.innerHTML)">Hello</h2>

If you click the heading, this refers to <h2>
```

onmousedown, onmouseup

- The onmousedown, onmouseup events are all parts of a mouse-click. First when a mouse-button is clicked, the onmousedown event is triggered, then, when the mouse-button is released, the onmouseup event is triggered.

- finally, when the mouse-click is completed, the onclick event is triggered (discussed above).

```
<div onmousedown="mdown(this)" onmouseup="mup(this)"
style="background-color:lightblue;width:70px;height:30px;padding:20px;">
Click Me</div>
```

```
<script>
function mdown(e) {
  e.style.backgroundColor = "yellow";
  e.innerHTML = "click";
}

function mup(e) {
  e.style.backgroundColor="pink";
  e.innerHTML="Release";
}
</script>
```

onmouseover and onmouseout

The onmouseover and onmouseout events can be used to trigger a function when the user mouses over, or out of, an HTML element:

```
<div onmouseover="mover(this)" onmouseout="mout(this)" >
Mouse Over Me</div>

<script>
function mover(e) {
  e.innerHTML = "Thank You";
  e.style.color="red"
}

function mout(e) {
  e.innerHTML = "Mouse Over Me"
  e.style.color="blue"
}
</script>
```

Form events

Event Name	When It Occurs
onfocus	When field gets focus
oninput	Immediately when value changes
onblur	When field loses focus
onchange	When value changes and focus is lost
onsubmit	When form is submitted

oninput

The oninput event occurs **immediately when the value of an input element changes** (while typing).

When to Use

- Live validation
- Auto formatting
- Real-time response

Syntax

```
<element oninput="myScript">
```

Example

```
<input type="text" oninput="this.value=this.value.toUpperCase()">
```

onchange

The onchange event occurs when **the value of an HTML element is changed**.

<*element* onchange="*myScript*">

Tip: This event is similar to the oninput event. The difference is that the oninput event occurs immediately after the value of an element has changed, while onchange occurs when the element loses focus, after the content has been changed. The other difference is that the onchange event also works on <select> elements.

onblur

The onblur event occurs when an **HTML element loses focus**.

<*element* onblur="*myScript*">

The onblur event is often used on input fields.

The onblur event is often used with form validation (when the user leaves a form field).

onfocus

The onfocus event occurs when an element gets focus.

The onfocus event is often used on input fields.

<*element* onfocus="*myScript*">

Remember:
The onBlur event is fired when you have moved away from an object without necessarily having changed its value.
The onChange event is only called when you have changed the value of the field and it loses focus.

onsubmit

The onsubmit event occurs when a form is submitted.

```
<form action="#" onsubmit="myFunction()">
  Enter name: <input type="text" name="fname" id="fname">
  <input type="submit" value="Submit">
</form>

<script>
function myFunction() {
   a =  document.getElementById("fname").value;
  alert("The form was submitted! Welcome " +  a);
}
</script>
```

Example:

Write an HTML and JavaScript program to perform the following tasks:

1. Add **three text input fields**.
2. In the **first text field**, convert the entered text into **uppercase** when the **value of the field is changed** using the onchange event.
3. In the **second text field**:
   - Change the **background color to yellow** when the field **gets focus** using the onfocus event.
   - Change the **text color to blue while typing** using the oninput event.
4. In the **third text field**, convert the entered text into **lowercase** when the field **loses focus** using the onblur event.

```
<script type="text/javascript">
function uppercase()
{
   var a = document.getElementById("fname");
   a.value = a.value.toUpperCase();
}
```

```
function lower(a)
{
   a.value = a.value.toLowerCase();
}
function inputText(x)
{
   x.style.color = "blue";
}
</script>
<!-- onchange -->
<input type="text" value="hello" id="fname" onchange="uppercase()">
<!-- onfocus + oninput -->
<input type="text" value="hello" onfocus="this.style.backgroundColor='yellow'"
oninput="inputText(this)">
<!-- onblur -->
<input type="text" value="HELLO" onblur="lower(this)">
```

Keyboard Events

Event	Occurs When
onkeydown	A user presses a key
onkeypress	A user presses a key
onkeyup	A user releases a key

keydown

Fires when the user presses a key.

keypress

Fires when an actual character is being inserted in, for instance, a text input.

keyup

Fires when the user releases a key, *after* the default action of that key has been performed.

the deprecated keypress event, the keydown event is fired for all keys, regardless of whether they produce a character value.

Example: Write Html/js to perform the task as asked below.
Background color should turn red while key is down and it should turn blue while key is up.

```
<head>
<script type="text/javascript">
function fun1(id)
{
id.bgColor="blue";
}
function fun2(id)
{
id.bgColor="red";
}
</script>
</head>
<body onkeyup="fun1(this)"  onkeydown="fun2(this)">
</body>
```

Event Object & Handling Essentials

Property / Method	Description	Used For	Example
event.target	Element that triggered the event	Identifying element	html <button onclick="show(event)">Click</button> <script> function show(e){ alert(e.target.tagName); } </script>
event.type	Type of event	Debugging	html <button onclick="alert(event.type)">Click</button>
event.key	Key pressed (modern)	Keyboard input	html <input onkeydown="alert(event.key)">
event.keyCode	Numeric code of key pressed **(deprecated)**	Legacy keyboard handling	html <input onkeydown="alert(event.keyCode)">
event.button	Mouse button pressed (0 left, 1 middle, 2 right)	Mouse input detection	html <button onmousedown="check(event)">Click</button> <script> function check(e){ alert(e.button); } </script>
event.preventDefault()	Stops default behavior	Form / link control	html <form onsubmit="stop(event)"> <button type="submit">Submit</button> </form><script> function stop(e){ e.preventDefault(); } </script>

1. event.target

What it is

event.target refers to the **HTML element that triggered the event**.

Used for

- Identifying which element was clicked
- Handling multiple elements using one function

Example

```
<select onchange="show(event)">
```

```
 <option value="Red">Red</option>
 <option value="Blue">Blue</option>
</select>

<script>
function show(e) {
   alert(e.target.value);
}
</script>
```

2. event.type

What it is

event.type tells **which event occurred**.

Used for

- Debugging
- Handling multiple events using one function

Example

```
<button onclick="check(event)">Click</button>
<script>
function check(e) {
   alert(e.type);
}</script>
```

Explanation

- Clicking the button triggers a click event
- Output shows click

3. event.key

What it is

event.key returns **the key pressed by the user**.

Used for

- Keyboard input
- Validation
- Games and shortcuts

Example

```
<body onkeypress="keyCheck(event)">
<script>
function keyCheck(e) {
   alert("Key Pressed: " + e.key);
}
</script>
</body>
```

Explanation

- User presses a key
- Output displays the pressed key (A, Enter, etc.)

event.keyCode

✓ Get the value of the pressed keyboard key:

The keyCode property is deprecated.

Syntax

```
event.keyCode
```

Write script to display Unicode on key press.

```
<script >
function fun(e)
{ alert(e.keyCode); }
</script>
<body onkeypress="fun(event)" ></body>
```

4. event.preventDefault()

What it is

Stops the **default browser action**.

Used for

- Preventing form submission
- Stopping page reload
- Custom form validation

Example

```
<form onsubmit="stop(event)">
<input type="submit" value="Submit">
</form>
<script>
function stop(e) {
  e.preventDefault();
  alert("Form submission stopped");
}</script>
```

5) event.button

event.button tells you **which mouse button** was pressed during a mouse event (like mousedown, mouseup, click).

Value	Mouse Button
0	Left button
1	Middle button (wheel)
2	Right button

Example

```
<button onmousedown="checkButton(event)">
  Click with any mouse button
</button>

<script>
function checkButton(e) {
  if (e.button === 0) alert("Left mouse button");
  if (e.button === 1) alert("Middle mouse button");
  if (e.button === 2) alert("Right mouse button");
}
</script>
```

addEventListener()

addEventListener() is a JavaScript method used to **attach an event to an HTML element**.
It tells the browser:

"When this event happens, run this function."

Why do we use addEventListener()?

We use addEventListener() because:

- It keeps **HTML and JavaScript separate**
- It is **cleaner and more readable**
- Multiple events can be added to the same element
- It is the **modern and recommended** way to handle events

Syntax

```
element.addEventListener("event", function);
```

Example:

```
button.addEventListener("click", show);
```

Event Type	Event Name	Description
Keyboard	keydown	Fires when a key is pressed
Keyboard	keyup	Fires when a key is released
Keyboard	keypress	Fires when a character key is pressed (deprecated)
Mouse	click	Fires when mouse is clicked
Mouse	mousedown	Fires when mouse button is pressed
Mouse	mouseup	Fires when mouse button is released
Mouse	mouseover	Fires when mouse moves over an element
Mouse	mouseout	Fires when mouse moves out of an element

Example

```
<button id="btn">Click Me</button>
<script>
document.getElementById("btn").addEventListener("click", function() {
   alert("Button Clicked");
});
</script>
```

How it works

- getElementById("btn") selects the button
- addEventListener("click", ...) attaches the click event
- When button is clicked → alert appears

Examples

Example: Write JS to handle following mouse events

1) If mouse is over heading should turn yellow, If mouse goes out then it should turn black.

2) If find time button is clicked then show date and time information.

3) If button named “red” is clicked then background color should turn red, and button named “green” is clicked then background color should turn green

```
<html>
<head>
<script>
function fun(id){id.style.color = "yellow";}
function fun2(id){id.style.color = "black";}
function fun3(id)
{
d = new Date();
document.getElementById("demo").innerHTML = d;
}
function fun4()
{
id=document.getElementById("bd");
id.style.backgroundColor = "red";
}
function fun5()
{
id=document.getElementById("bd");
id.style.backgroundColor = "green";
}
</script>
</head>

<body id="bd">
<h1 onmouseover="fun(this)" onmouseout="fun2(this)">Hello</h1>
<input type="submit" value="Find Time" onclick="fun3(this)"/>
<p id="demo"></p>
<input type="submit" value="red" onclick="fun4()"/>
<input type="submit" value="green" onclick="fun5()"/>
</body>
</html>
```

Given a list of 5 <p class='text'> and also 5 elements, write code to:
(a) select all p tags
(b) append '(Updated)' to each element
(c) print the count of updated elements in console.

```
<p class="text">Paragraph 1</p>
<p class="text">Paragraph 2</p>
<p class="text">Paragraph 3</p>
<p class="text">Paragraph 4</p>
<p class="text">Paragraph 5</p>

<div class="text">Div 1</div>
<div class="text">Div 2</div>
<div class="text">Div 3</div>
<div class="text">Div 4</div>
<div class="text">Div 5</div>

<script>
   // (a) select all p tags
   var paragraphs = document.querySelectorAll("p.text");

   // (b) append '(Updated)' to each element
   for (var p of paragraphs) {
      p.innerHTML = `${p.innerHTML} (Updated)`;
   }

   // (c) print count in console
   console.log(paragraphs.length);
</script>
```

Create a form with two fields: username and email, along with a submit button.
1.When the form is submitted, prevent the default form submission and display an alert box with the message "Form submitted".
2. When the user makes changes to any field and then leaves it (loses focus), the background color of that field should change to yellow.
3. While user clicks on input field change background color to pink (only for username field).
4. While the user is typing in a field, the input text should appear in red color.
5. Implement this using only event attributes (oninput, onchange, onsubmit) in HTML, without using addEventListener.

```
<body>
<form onsubmit="myform(event)">
   <input type="text" onfocus="fun1(this)" onchange="fun2(this)" oninput="fun3(this)">
   <br><br>
   <label>Email:</label><br>
   <input type="email" onchange="fun2(this)" oninput="fun3(this)">
   <br><br>
   <input type="submit" value="Submit">
</form>
<script>
   function myform(e){
      e.preventDefault();
      alert("Form Submitted")
   }
   function fun1(x){
      x.style.backgroundColor='pink';
   }
   function fun2(x){
      x.style.backgroundColor='yellow';
   }
   function fun3(x){
      x.style.color='red';
   }
</script>
</body>
```