Module-02

Document Object Model

1. Introduction to DOM

- DOM (Document Object Model) represents the structure of an HTML document as a tree of objects.
- JavaScript can access and manipulate these elements dynamically.

2. Selecting Elements in DOM

- **Using JavaScript:**
  - document.getElementById("id") – Selects an element by ID.
  - document.getElementsByClassName("class") – Selects elements by class.
  - document.getElementsByTagName("tag") – Selects elements by tag name.
  - document.querySelector("selector") – Selects the first matching element.
  - document.querySelectorAll("selector") – Selects all matching elements.
- **Using jQuery:**
  - $("selector") – Selects elements using CSS-like syntax.

VTUSYNC.IN

3. **Modifying Elements**

- **JavaScript:**
  - element.innerHTML = "New Content"; – Changes content inside an element.
  - element.textContent = "New Text"; – Updates only text content.
  - element.setAttribute("attribute", "value"); – Modifies attributes.
- **jQuery:**
  - $("selector").html("New Content"); ○ $("selector").text("New Text");

    ○ $("selector").attr("attribute", "value");

4. **Adding & Removing Elements**

- **JavaScript:**
  - document.createElement("tag") – Creates a new element.
  - parent.appendChild(child) – Adds a child element. ○

    parent.removeChild(child) – Removes an element.
- **jQuery:**
  - $("parent").append("<tag>New Content</tag>"); ○

    $("parent").prepend("<tag>New Content</tag>"); ○

    $("selector").remove();

VTUSYNC.IN

5. **Handling Events**

- **JavaScript:** ◦ element.addEventListener("event", function);
- **jQuery:**
    - $("selector").on("event", function() {});
- Example:

```
document.getElementById("btn").addEventListener("click", function() {
alert("Button Clicked!");
});
$("#btn").on("click", function() {
alert("Button Clicked!");
});
```

6. **Changing CSS Styles**

- **JavaScript:** ◦ element.style.property = "value";
- **jQuery:**
    - $("selector").css("property", "value");
- Example:

```
document.getElementById("box").style.backgroundColor = "blue";
$("#box").css("background-color", "blue");
```

7. **Animations & Effects in jQuery**

VTUSYNC.IN

- $("selector").fadeIn();
- $("selector").fadeOut();
- $("selector").slideUp();
- $("selector").slideDown();
- $("selector").animate({property: "value"}, duration);

8. AJAX and Fetching Data

- **JavaScript Fetch API:**

```
fetch("url")
   .then(response => response.json())
   .then(data => console.log(data));
```

- **jQuery AJAX:**

```
$.ajax({
   url: "url",
   method: "GET",
success: function(data) {
console.log(data);

   }
});
```

VTUSYNC.IN

9. Event Delegation

- Used for handling events dynamically on elements that may not exist at the start.
- **JavaScript:**

```
document.getElementById("parent").addEventListener("click", function(event) {
if (event.target.matches(".child")) {        console.log("Child clicked!");
   }
});
```

- **jQuery:**

```
$("#parent").on("click", ".child", function() {
console.log("Child clicked!");
});
```

10. Form Handling & Validation

- Accessing form values:

```
let inputValue = document.getElementById("input").value; let
inputValue = $("#input").val();
```

- Validating input fields: if (inputValue.trim() === "") {   alert("Field cannot be empty");

}

VTUSYNC.IN

1. DOM Node

- A **node** is any object in the **DOM tree**, such as:
  - **Element nodes** (<div>, <p>, <button>, etc.)
  - **Attribute nodes** (class, id, style)
  - **Text nodes** (content inside an element)
  - **Comment nodes** (<!-- This is a comment -->)

VTUSYNC.IN

2. Accessing Nodes

Using JavaScript

- document.getElementById("id") → Selects an element by ID.
- document.getElementsByClassName("class") → Selects elements by class name.
- document.getElementsByTagName("tag") → Selects elements by tag name.
- document.querySelector("selector") → Selects the first matching element.
- document.querySelectorAll("selector") → Selects all matching elements.

Using jQuery

- $("selector") → Selects elements using CSS-like syntax.

Example:

let node = document.getElementById("demo"); // JavaScript let

node = $("#demo"); // jQuery

3. Creating & Adding Nodes

Using JavaScript

- document.createElement("tag") → Creates a new element.
- parent.appendChild(child) → Appends a child node to a parent.
- parent.insertBefore(newNode, existingNode) → Inserts before another node.

Example:

let newDiv = document.createElement("div"); newDiv.textContent

= "New Element"; document.body.appendChild(newDiv);

Using jQuery

- $("parent").append("<div>New Element</div>");
- $("parent").prepend("<div>New at Start</div>");

Example:

$("body").append("<div>New Element</div>");

VTUSYNC.IN

4. Removing & Replacing Nodes

Using JavaScript

- parent.removeChild(child) → Removes a node.
- parent.replaceChild(newNode, oldNode) → Replaces a node.

Example:

```
let toRemove = document.getElementById("remove");

toRemove.parentNode.removeChild(toRemove);
```

Using jQuery

- $("selector").remove(); → Removes an element.
- $("selector").replaceWith("<new element>"); → Replaces an element.

Example:

```
$("#remove").remove();

$("#old").replaceWith("<p>New Paragraph</p>");
```

VTUSYNC.IN

5. Cloning & Copying Nodes

Using JavaScript

- element.cloneNode(true) → Clones an element with all children (true) or only itself (false).

Example:

```
let original = document.getElementById("original"); let

clone = original.cloneNode(true);

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

Using jQuery

- $("selector").clone();

Example:

```
let clonedElement = $("#original").clone();

$("body").append(clonedElement);
```

VTUSYNC.IN

6. Navigating Nodes (Parent, Child, Sibling)

Using JavaScript

- element.parentNode → Gets the parent node.
- element.children → Gets all child elements.
- element.firstChild / element.lastChild → Gets the first/last child.
- element.nextSibling / element.previousSibling → Gets the next/previous node.

Example:

let parent = document.getElementById("child").parentNode; let

firstChild = document.getElementById("parent").firstChild;

Using jQuery

- $("selector").parent();
- $("selector").children();
- $("selector").first();
- $("selector").next();

Example:

let parent = $("#child").parent(); let

firstChild = $("#parent").children().first();

VTUSYNC.IN

7. Changing Node Content

Using JavaScript

- element.innerHTML = "New Content"; → Changes inner content.
- element.textContent = "New Text"; → Changes text only.

Using jQuery

- $("selector").html("New Content");
- $("selector").text("New Text");

Example: document.getElementById("demo").innerHTML =

"Updated Content";

$("#demo").html("Updated Content");

VTUSYNC.IN

8. Changing Node Attributes

Using JavaScript

- element.setAttribute("attribute", "value");
- element.getAttribute("attribute");
- element.removeAttribute("attribute");

Using jQuery

- $("selector").attr("attribute", "value");
- $("selector").removeAttr("attribute");

Example:

document.getElementById("demo").setAttribute("class", "new-class");

$("#demo").attr("class", "new-class");

1. Updating Element Content

- **innerHTML (JS) / .html() (jQuery)** → Changes the inner HTML, including formatting.
- **textContent (JS) / .text() (jQuery)** → Updates text content only (ignores HTML tags).
- **Example:**

document.getElementById("demo").innerHTML = "<b>Bold Text</b>"; // JavaScript

$("#demo").html("<b>Bold Text</b>"); // jQuery

VTUSYNC.IN

2. Updating Element Attributes

- **setAttribute("attr", "value") (JS) / .attr("attr", "value") (jQuery) →** Modifies an attribute.
- **getAttribute("attr") (JS) / .attr("attr") (jQuery)** → Retrieves an attribute value.
- **removeAttribute("attr") (JS) / .removeAttr("attr") (jQuery)** → Deletes an attribute.

- **Example:**

```
document.getElementById("myImage").setAttribute("src", "new.jpg"); //
JavaScript

$("#myImage").attr("src", "new.jpg"); // jQuery
```

3. Handling Events

- **JavaScript: addEventListener("event", function)** → Binds an event to an element.
- **jQuery: .on("event", function)** → Binds an event (can handle multiple events).
- **Common events:** click, mouseover, keydown, change.
- **Example:**

```
document.getElementById("btn").addEventListener("click", function() {
alert("Clicked!");
});


$("#btn").on("click", function() {
alert("Clicked!");
});
```

VTUSYNC.IN

4. Removing Event Listeners

- **JavaScript: removeEventListener("event", function)**
- **jQuery: .off("event")**
- **Example:**

document.getElementById("btn").removeEventListener("click", myFunction);

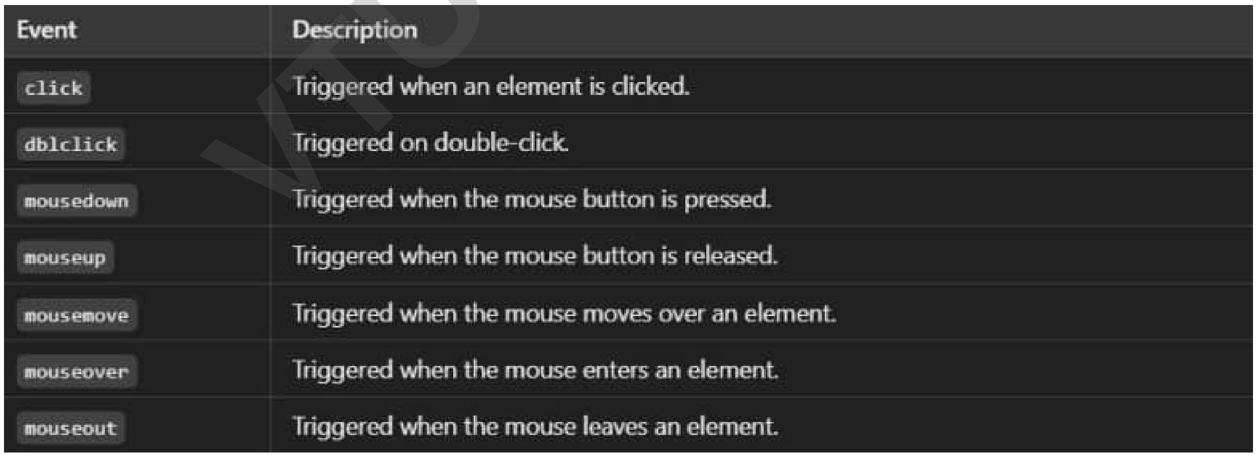
$("#btn").off("click");

Different Types of Events in JavaScript & jQuery

Events in JavaScript and jQuery allow developers to handle user interactions effectively. Below are the key types of events:

1. Mouse Events

These events respond to user actions with the mouse.



Event	Description
`click`	Triggered when an element is clicked.
`dblclick`	Triggered on double-click.
`mousedown`	Triggered when the mouse button is pressed.
`mouseup`	Triggered when the mouse button is released.
`mousemove`	Triggered when the mouse moves over an element.
`mouseover`	Triggered when the mouse enters an element.
`mouseout`	Triggered when the mouse leaves an element.

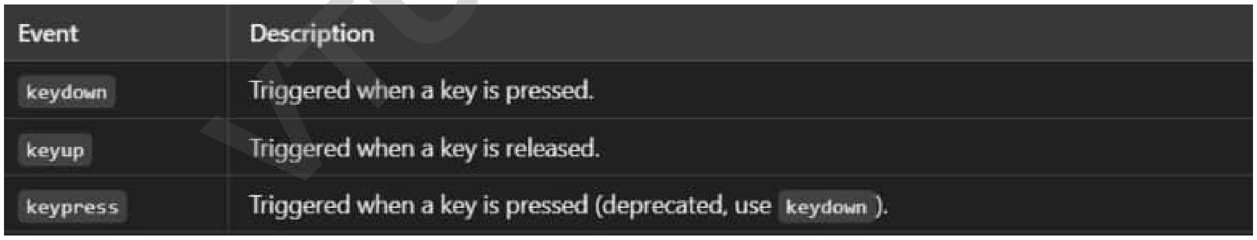
VTUSYNC.IN

```
document.getElementById("btn").addEventListener("click", function() {

alert("Button clicked!");

});


// jQuery

$("#btn").on("click", function() {

alert("Button clicked!");

});
```

VTUSYNC.IN

2. Keyboard Events

These events respond to keyboard interactions.



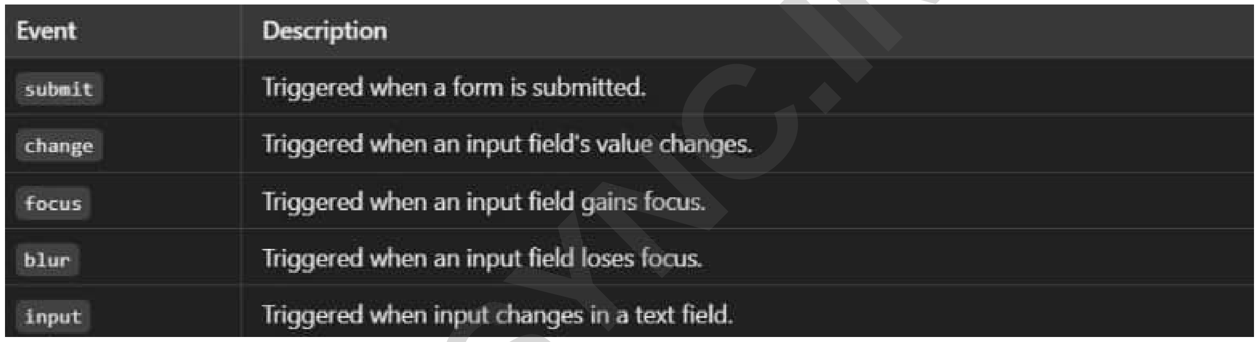
Event	Description
`keydown`	Triggered when a key is pressed.
`keyup`	Triggered when a key is released.
`keypress`	Triggered when a key is pressed (deprecated, use `keydown`).

```
document.addEventListener("keydown", function(event) {

console.log("Key pressed: " + event.key);

});
```

```
// jQuery

$(document).on("keydown", function(event) {

console.log("Key pressed: " + event.key);

});
```

3. Form Events

These events are related to form elements.



Event	Description
`submit`	Triggered when a form is submitted.
`change`	Triggered when an input field's value changes.
`focus`	Triggered when an input field gains focus.
`blur`	Triggered when an input field loses focus.
`input`	Triggered when input changes in a text field.

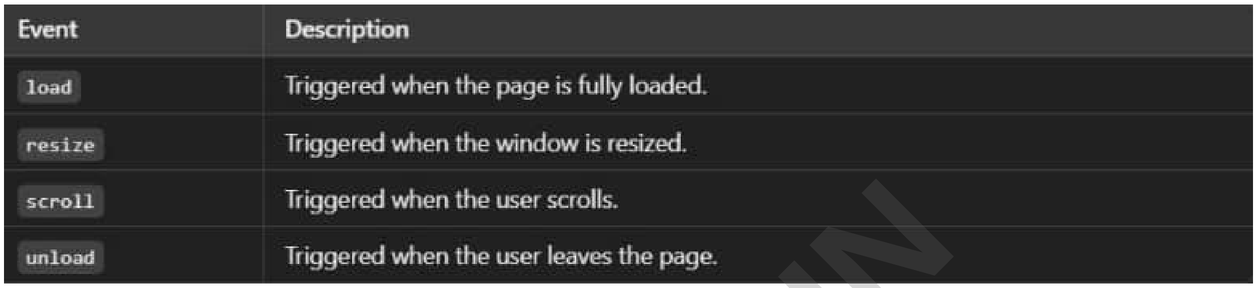
```
document.getElementById("myForm").addEventListener("submit",

function(event) {     event.preventDefault(); // Prevents form from

submitting     alert("Form submitted!");

});



// jQuery

$("#myForm").on("submit", function(event) {

event.preventDefault();


    alert("Form submitted!");
```

VTUSYNC.IN

```
});
```

4. Window Events

These events are triggered by the browser window.



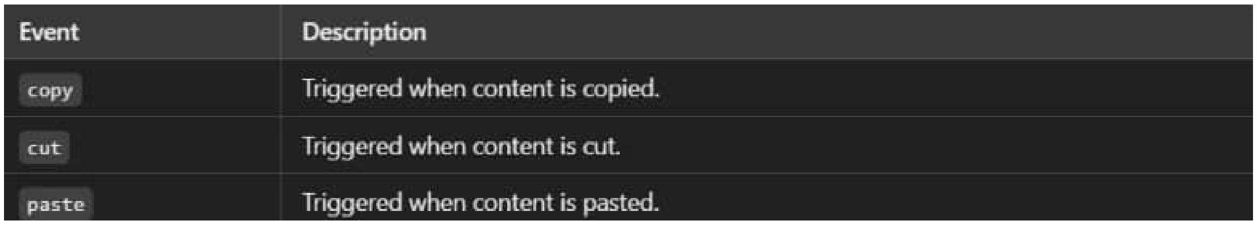
Event	Description
`load`	Triggered when the page is fully loaded.
`resize`	Triggered when the window is resized.
`scroll`	Triggered when the user scrolls.
`unload`	Triggered when the user leaves the page.

```
window.addEventListener("resize", function() {

console.log("Window resized!");

});



// jQuery

$(window).on("resize", function() {

console.log("Window resized!");

});
```

VTUSYNC.IN

5. Clipboard Events

These events handle copying and pasting actions.



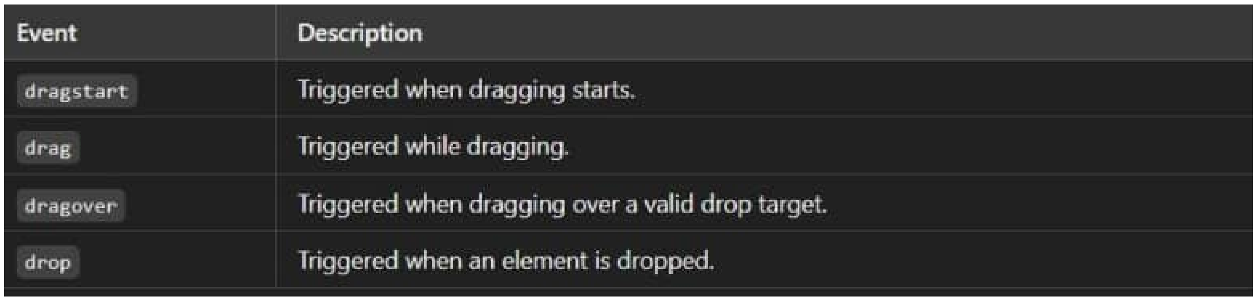
Event	Description
`copy`	Triggered when content is copied.
`cut`	Triggered when content is cut.
`paste`	Triggered when content is pasted.

```
document.addEventListener("copy", function() {

alert("Content copied!");

});



// jQuery

$(document).on("copy", function() {

alert("Content copied!");

});
```

VTUSYNC.IN

6. Drag & Drop Events

These events allow users to drag and drop elements.

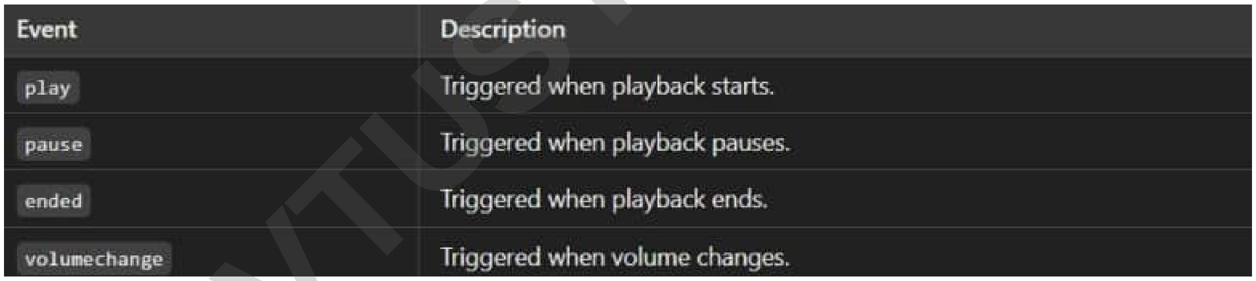


Event	Description
`dragstart`	Triggered when dragging starts.
`drag`	Triggered while dragging.
`dragover`	Triggered when dragging over a valid drop target.
`drop`	Triggered when an element is dropped.

```
document.getElementById("dragElement").addEventListener("dragstart",
function() {     console.log("Dragging started!");
});


// jQuery
$("#dragElement").on("dragstart", function() {
console.log("Dragging started!");
});
```

7. Media Events

These events are used for media elements like <audio> and <video>.



Event	Description
`play`	Triggered when playback starts.
`pause`	Triggered when playback pauses.
`ended`	Triggered when playback ends.
`volumechange`	Triggered when volume changes.

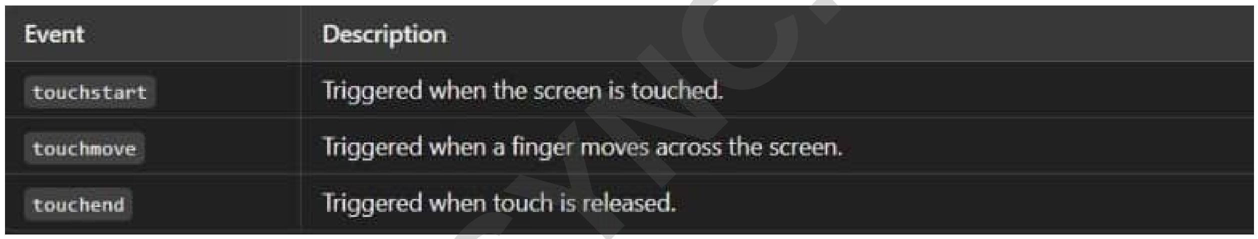
```
document.getElementById("myVideo").addEventListener("play", function() {
console.log("Video started playing!");

});
```

VTUSYNC.IN

```
// jQuery

$("#myVideo").on("play", function() {

console.log("Video started playing!");

});
```

8. Touch Events (Mobile Devices)

These events handle touch interactions on mobile devices.



Event	Description
`touchstart`	Triggered when the screen is touched.
`touchmove`	Triggered when a finger moves across the screen.
`touchend`	Triggered when touch is released.

```
document.addEventListener("touchstart", function() {

console.log("Screen touched!");

});


// jQuery

$(document).on("touchstart", function() {

console.log("Screen touched!");

});
```

VTUSYNC.IN

1. Using JavaScript (addEventListener)

- The addEventListener(event, function) method is used to bind an event to an element.
- Syntax:

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

- Example:

```
document.getElementById("btn").addEventListener("click", function() {
alert("Button clicked!");
});
```

- Supports multiple event bindings for the same element.
- Does not override existing event listeners.

VTUSYNC.IN

2. Using JavaScript (onclick or Inline Event Handler)

- Assigning an event handler directly to the onclick property.
- Syntax: element.event = function;
- Example:

```
document.getElementById("btn").onclick = function() {
alert("Button clicked!");
};
```

- ⚠ **Limitations:**
  - Can only bind one event at a time (overwrites previous event).
  - Not suitable for dynamic elements.

3. Using jQuery (.on())

- The .on(event, selector, function) method is used to bind events dynamically.
- Syntax:

```
$(selector).on("event", function);
```

- Example:

```
$("#btn").on("click", function() {
alert("Button clicked!");
});
```

- Works on dynamically added elements.
- Can bind multiple events at once.

4. Using jQuery (.click(), .hover(), etc.) • Shortcut methods for specific events.

- Example:

VTUSYNC.IN

```
$("#btn").click(function() {

alert("Button clicked!");

});
```

- Simpler syntax but **does not work for dynamically created elements**.

---

5. Binding Multiple Events to the Same Element

- **JavaScript (addEventListener)**: let btn =

  document.getElementById("btn"); btn.addEventListener("mouseover",

  function() { console.log("Mouse over!"); }); btn.addEventListener("click",

  function() { console.log("Button clicked!"); });

- **jQuery (.on())**:

```
$("#btn").on("mouseover click", function() {

console.log("Mouse over or clicked!");


});
```

---

6. Binding Events to Multiple Elements

- **JavaScript (querySelectorAll)**:

  document.querySelectorAll(".buttons").forEach(button => {

VTUSYNC.IN

```
     button.addEventListener("click", function() {          alert("One of the buttons
     clicked!");
   });
});
```

- **jQuery (.each())**:

```
$(".buttons").each(function() {
$(this).on("click", function() {
alert("One of the buttons clicked!");
   });
});
```

---

7. Binding Events to Dynamically Added Elements

- **JavaScript (Event Delegation using document.addEventListener)**:

```
document.addEventListener("click", function(event) {
   if (event.target.classList.contains("dynamic-btn")) {
alert("Dynamically added button clicked!");
   }
});
```

- **jQuery (.on())**:

VTUSYNC.IN

```
$(document).on("click", ".dynamic-btn", function() {
alert("Dynamically added button clicked!"); });
```

- Works for elements added later via JavaScript/jQuery.

---

8. Removing an Event Binding

- **JavaScript (removeEventListener)**:

```
function myFunction() {
alert("Clicked!");
} document.getElementById("btn").addEventListener("click", myFunction);
document.getElementById("btn").removeEventListener("click", myFunction);
```

- **jQuery (.off())**:

```
$("#btn").on("click", function() { alert("Clicked!"); });
$("#btn").off("click"); // Removes click event
```

VTUSYNC.IN

Event Delegation in JavaScript & jQuery

Event Delegation

Event delegation is a technique in JavaScript where a single event listener is added to a parent element to handle events on multiple child elements, including dynamically added elements.

Efficient – Reduces memory usage by attaching fewer event listeners.

Handles Dynamic Elements – Works even when new elements are added dynamically.

Better Performance – Instead of attaching events to every element, it delegates handling to a common parent.

How Event Delegation Works?

1. Add an event listener to a common parent element.
2. Use event.target to check if the event was triggered by the intended child element.
3. Execute the event handler only if the correct child element is clicked.

1. Event Delegation Using JavaScript Example: Handling Clicks on a List of Items document.getElementById("list").addEventListener("click", function(event) {   if (event.target.tagName === "LI") {   alert("Clicked on: " + event.target.textContent);

}

});

The event listener is attached to the #list parent, but it works for all <li> elements inside it.

Works even for new <li> items added later.

VTUSYNC.IN

2. Event Delegation Using jQuery

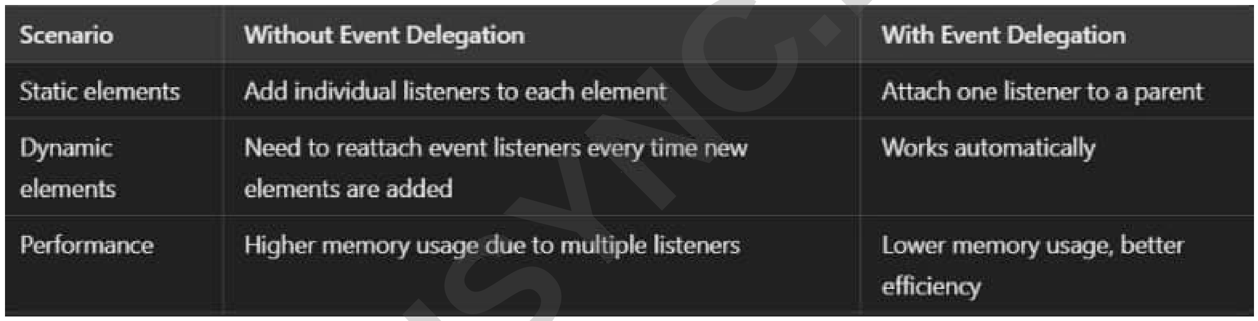
Example: Handling Clicks on Buttons inside a Parent Div

```
$(document).on("click", ".child-btn", function() {     alert("Button clicked: " +

$(this).text());

});
```

The event is attached to document, but it listens for .child-btn clicks.

Works for dynamically added .child-btn elements.

VTUSYNC.IN



Scenario	Without Event Delegation	With Event Delegation
Static elements	Add individual listeners to each element	Attach one listener to a parent
Dynamic elements	Need to reattach event listeners every time new elements are added	Works automatically
Performance	Higher memory usage due to multiple listeners	Lower memory usage, better efficiency

4. When to Use Event Delegation?

When dealing with lists, tables, or elements created dynamically (e.g., <ul>, <table>, <div>)

When handling repetitive UI components (e.g., buttons inside cards, form fields inside a container)

When optimizing performance for high-interaction web pages

5. When Not to Use Event Delegation?

If the child elements have different event handlers that must be unique

If capturing event behavior is needed (i.e., focus, blur, mouseenter)

If immediate response is required (event delegation may have a slight delay in large DOM structures)

6. Real-World Example

Dynamically Adding and Handling Click Events

```
document.getElementById("addItem").addEventListener("click", function() {
let newItem = document.createElement("li");     newItem.textContent = "New
Item";     document.getElementById("list").appendChild(newItem);
});


// Event Delegation for Handling Clicks
document.getElementById("list").addEventListener("click", function(event) {
if (event.target.tagName === "LI") {        alert("You clicked on: " +
event.target.textContent);
   }
});
```

Clicking on newly added <li> items still triggers the event!

VTUSYNC.IN

Event Listeners in JavaScript & jQuery

What is an Event Listener?

An **event listener** is a function that waits for a specific event (like a click, hover, or keypress) to happen on an element and executes a function in response.

Detects user interactions (clicks, keyboard input, scrolling, etc.)
Enhances interactivity in web applications
Can be applied to multiple elements without modifying HTML

1. Adding Event Listeners in JavaScript

1.1 Using addEventListener()

The addEventListener() method attaches an event to an element without overwriting existing events.
Syntax:

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

- "event" → Type of event (e.g., "click", "mouseover", "keydown")
- function → The function to execute when the event occurs
- useCapture (optional) → true for event capturing, false for

bubbling **Example: Button Click Listener**

document.getElementById("btn").addEventListener("click",

function() {   alert("Button Clicked!");

VTUSYNC.IN

```
});
```

Supports multiple event listeners on the same element

More flexible than onclick

1.2 Using Inline Event Handlers (onclick, onmouseover)

You can directly assign an event to an element using properties like onclick.

Example:

```
document.getElementById("btn").onclick = function() {
alert("Button Clicked!");
};
```

⚠ Limitations:

- Overwrites any previously assigned event handlers
- Not ideal for handling multiple events

1.3 Using Anonymous vs Named Functions

Both **anonymous** and **named functions** can be used in event listeners.

Anonymous Function (Inline Function)

```
document.getElementById("btn").addEventListener("click", function() {
alert("Button Clicked!");
});
```

VTUSYNC.IN

Named Function

function showAlert() {

alert("Button Clicked!");

} document.getElementById("btn").addEventListener("click",

showAlert); **Advantage of Named Function:** Can be reused and

removed easily

---

2. Removing Event Listeners

Use removeEventListener() to detach an event listener.

document.getElementById("btn").removeEventListener("click", showAlert);

⚠ The function must be **named** when removing it. Anonymous functions cannot be

removed.

VTUSYNC.IN

3. Event Listeners in jQuery

3.1 Using .on() (Recommended)

The .on() method attaches an event listener dynamically.

$("#btn").on("click", function() {

alert("Button Clicked!");

});

Works even for dynamically added elements

Can bind multiple events

3.2 Using .click(), .hover(), .keypress() (Shortcut Methods)

```
$("#btn").click(function() {

   alert("Button Clicked!");

});
```

⚠ These methods **do not work on dynamically added elements**. Use .on() instead.

---

4. Event Propagation (Bubbling & Capturing)

Events travel through the DOM in **two phases**:

1☐**Capturing Phase** (Top to Bottom)

2☐**Bubbling Phase** (Bottom to Top) **Example: Bubbling Effect**

```
document.getElementById("parent").addEventListener("click", function() {

alert("Parent Clicked!");

}); document.getElementById("child").addEventListener("click",

function(event) {     alert("Child Clicked!");

});
```

Clicking on #child will trigger both listeners (child → parent).

VTUSYNC.IN

Stopping Event Bubbling

Use event.stopPropagation() to prevent bubbling.

document.getElementById("child").addEventListener("click", function(event) {

event.stopPropagation();   alert("Child Clicked, but parent won't trigger!");

});

5. Binding Multiple Events to an Element

Attach multiple event listeners using addEventListener() or .on(). **JavaScript Example** let btn = document.getElementById("btn");

btn.addEventListener("mouseover", function() { console.log("Mouse Over!"); });

btn.addEventListener("click", function() { console.log("Button Clicked!"); });

jQuery Example

$("#btn").on("mouseover click", function() {

console.log("Mouse Over or Clicked!"); });

VTUSYNC.IN

---

6. Event Delegation

Instead of attaching listeners to each element, attach it to a parent and use event.target to handle child elements dynamically.

JavaScript Example

```
document.getElementById("parent").addEventListener("click", function(event) {
if (event.target.tagName === "BUTTON") {        alert("Button inside parent
clicked!");
    }
});
```

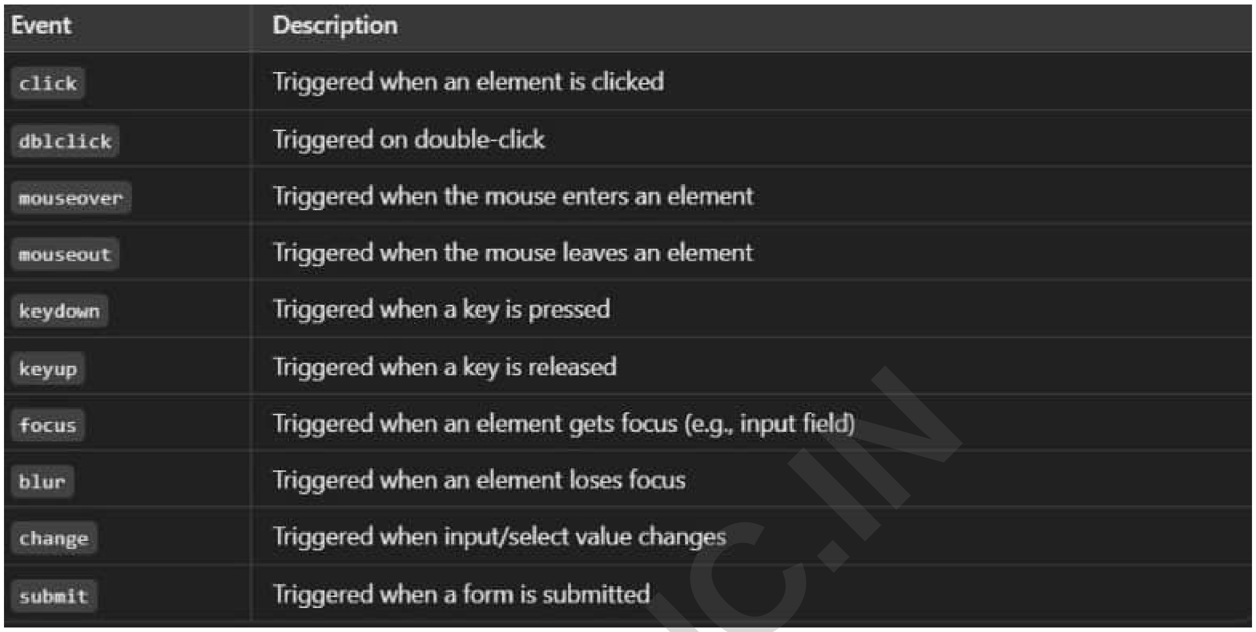
jQuery Example

```
$(document).on("click", ".child-btn", function() {
alert("Dynamically added button clicked!");
});
```

Best for handling dynamically generated elements.

VTUSYNC.IN

7. Common Event Listener Types



Event	Description
`click`	Triggered when an element is clicked
`dblclick`	Triggered on double-click
`mouseover`	Triggered when the mouse enters an element
`mouseout`	Triggered when the mouse leaves an element
`keydown`	Triggered when a key is pressed
`keyup`	Triggered when a key is released
`focus`	Triggered when an element gets focus (e.g., input field)
`blur`	Triggered when an element loses focus
`change`	Triggered when input/select value changes
`submit`	Triggered when a form is submitted

VTUSYNC.IN