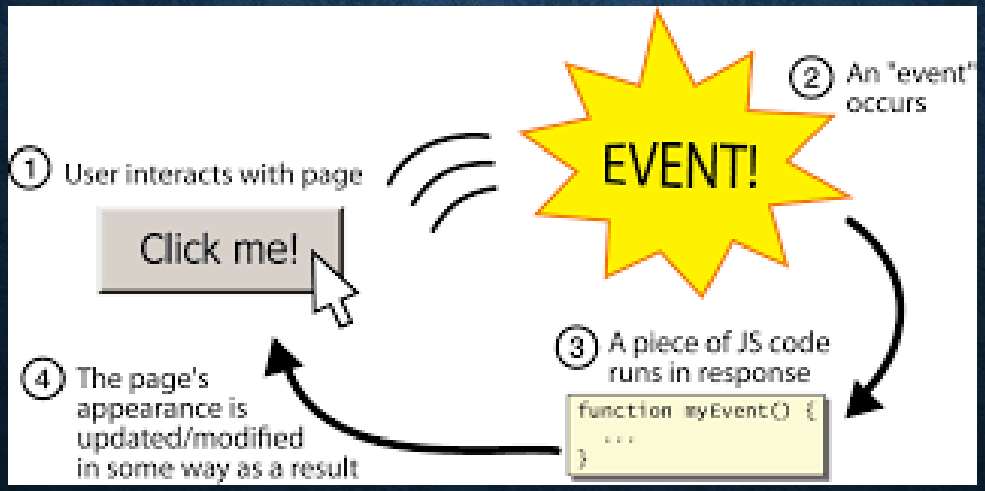
# WEB TECHNOLOGY LECTURE - 08

# EVENTS

- JavaScript **events** allow you to interact with user actions that make web pages **dynamic and interactive**.
- An event is a signal that something has happened in the browser, like:
  - A button is clicked (click event)
  - A key is pressed (keydown event)
  - A form is submitted (submit event)
  - The page loads (load event)



# HANDLE EVENTS

1. Inline HTML Event: You can add an event directly in HTML (not recommended as it get mixed with HTML)

<button onclick="alert('Button clicked!')">Click Me</button>

2. JavaScript Event Listener (Best Practice):

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

3. Using directly the onclick Property:

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

# EVENT OBJECT

- The event object gives extra details about the event.

```
document.addEventListener("click", function(event) {
    console.log("Event type:", event.type);
    console.log("X position:", event.clientX);
    console.log("Y position:", event.clientY);
});
```

- The following is useful for handling shortcuts and form validation.

```
document.addEventListener("keydown", function(event) {
    console.log("Key pressed:", event.key);
});
```

# PREVENT DEFAULT BEHAVIOR

- Stops the default action of an element.
- For example:
  - Prevent anchor (<a>) from navigation.

<a src="https://www.google.com">click here </a>

```
document.getElementById("myLink").addEventListener("click", function(event) {
    event.preventDefault(); // Prevents the link from navigating
    alert("Link clicked, but it won't navigate!");
});
```

Form validation Example:

- When user clicks the submit button, we can check whether the input field are valid or not. If valid, only then we want to allow user to submit the form.
- For example, explore this page.

# MOUSE EVENTS

| Event | Attribute | Description |
|---|---|---|
| Click | onclick | The event occurs when the user clicks on an element |
| Dblclick | ondblclick | The event occurs when the user double-clicks on an element |
| mousedown | onmousedown | The event occurs when a user presses a mouse button over an element |
| mousemove | onmousemove | The event occurs when a user moves the mouse pointer over an element |
| mouseover | onmouseover | The event occurs when a user mouse over an element |
| mouseout | onmouseout | The event occurs when a user moves the mouse pointer out of an element |
| mouseup | onmouseup | The event occurs when a user releases a mouse button over an element |

# KEYBOARD EVENTS

| Event | Attribute | Description |
|---|---|---|
| keydown | onkeydown | The event occurs when the user is pressing a key or holding down a key |
| keypress | onkeypress | The event occurs when the user is pressing a key or holding down a key |
| Keyup | onkeyup | The event occurs when a keyboard key is released |

# FRAME/OBJECT EVENTS

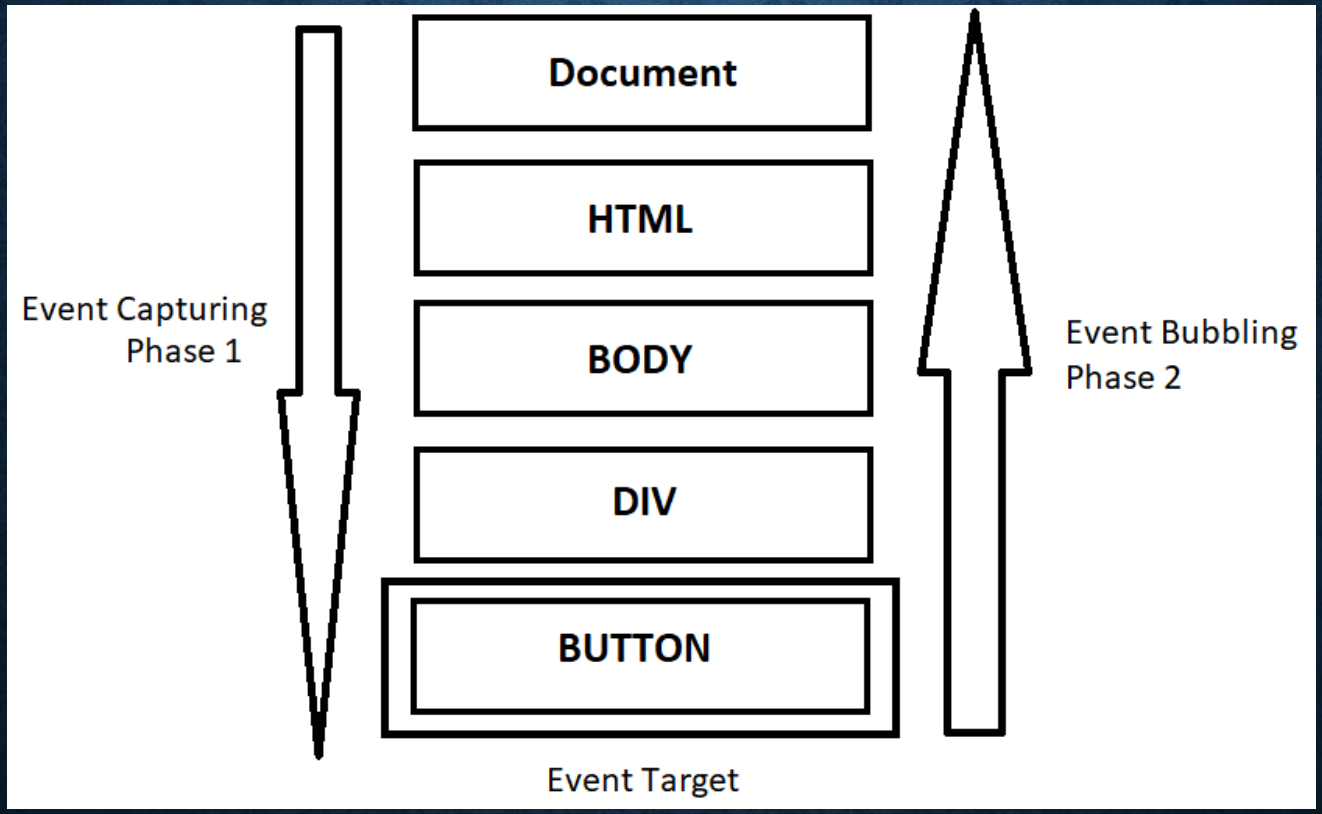
| Event | Attribute | Description |
|---|---|---|
| Abort | onabort | The event occurs when an image is stopped from loading before completely loaded (for <object>) |
| Error | onerror | The event occurs when an image does not load properly (for <object>, <body> and <frameset>) |
| Load | onload | The event occurs when a document, frameset, or <object> has been loaded |
| Resize | onresize | The event occurs when a document view is resized |
| Scroll | onscroll | The event occurs when a document view is scrolled |
| Unload | onunload | The event occurs when a document is removed from a window or frame (for <body> and <frameset>) |

# FORM EVENTS

| Event | Attribute | Description |
|---|---|---|
| Blur | onblur | The event occurs when a form element loses focus |
| Change | onchange | The event occurs when the content of a form element, the selection, or the checked state have changed (for <input>, <select>, and <textarea>) |
| Focus | onfocus | The event occurs when an element gets focus (for <label>, <input>, <select>, textarea>, and <button>) |
| Reset | onreset | The event occurs when a form is reset |
| Select | onselect | The event occurs when a user selects some text (for <input> and <textarea>) |
| Submit | onsubmit | The event occurs when a form is submitted |

# EVENT PROPAGATION

- Bubbling (default): Child → Parent
- Capturing (use true): Parent → Child




- By default, events bubble up from the child to the parent.

# EVENT BUBBLING (DEFAULT BEHAVIOR)

- In **bubbling**, the event starts from the **target element** and moves **up** to its parent elements.

```
<div id="parent">
    <button id="child">Click Me</button>
</div>

<script>
document.getElementById("parent").addEventListener("click", function() {
    alert("Parent clicked!");
});

document.getElementById("child").addEventListener("click", function() {
    alert("Button clicked!");
});
</script>
```

- Expected Output (When Clicking the Button):

  (1)"Button clicked!" (child element fires first)

  (2)"Parent clicked!" (event bubbles up to parent)

# EVENT CAPTURING (TRICKLING DOWN)

- In **capturing**, the event starts from the **outermost parent** and moves **down** to the **target element.**

```
<div id="parent">
    <button id="child">Click Me</button>
</div>

<script>
document.getElementById("parent").addEventListener("click", function() {
    alert("Parent clicked first!");
}, true); // `true` enables capturing

document.getElementById("child").addEventListener("click", function() {
    alert("Button clicked!");
});
</script>
```

- Expected Output (When Clicking the Button):

  (1) "Parent clicked first!" (capturing starts from parent)

  (2) "Button clicked!" (event reaches child)

Explore this game to observe how bubbling and capturing are used.

# STOPPING EVENT PROPAGATION

- Sometimes, you don't want an event to **bubble up** or **capture down**. You can stop propagation using:
- event.stopPropagation()

```
document.getElementById("child").addEventListener("click", function(event) {
    alert("Button clicked!");
    event.stopPropagation(); // Stops event from reaching parent
});
```

- Now, parent can not capture the child's bubbling phase.

# REMOVING AN EVENT LISTENER

- Useful when you want to disable an event dynamically.

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

# TIME EVENTS: SETTIMEOUT

- JavaScript **time events** allow you to execute code **after a delay** or **at regular intervals**.
- This is useful for animations, scheduled tasks, and repeated execution.
- setTimeout() – Delay Execution
- Example: Show a message after 3 seconds

```
setTimeout(function() {
    alert("Hello, this is delayed!");
}, 3000); // 3000ms = 3 seconds
```

# TIME EVENTS: SETINTERVAL

- The setInterval() function executes a function repeatedly at a specified time interval.

```
setInterval(() => console.log("This runs every 2 seconds"), 2000);
```

- Task: Generate A Countdown Timer

# STOPPING TIMEOUT

- If you want to cancel a scheduled timeout before it executes, use clearTimeout().

```
let timer = setTimeout(() => alert("You won't see this!"), 5000);
clearTimeout(timer); // Cancels the timeout before it runs
```

- In the same way use If you want to stop an interval, use clearInterval().

# TO DO LIST

Revisit the TODO App as a combination of Events & DOM manipulation.

# JAVASCRIPT SUMMERY

- **JavaScript can react to events** - A JavaScript can be set to execute when something happens, like when a page has finished loading or when a user clicks on an HTML element
- **JavaScript can manipulate HTML elements** - A JavaScript can read and change the content of an HTML element
- **JavaScript can be used to validate data** - A JavaScript can be used to validate form input
- **JavaScript can be used to detect the visitor's browser** - A JavaScript can be used to detect the visitor's browser, and - depending on the browser - load another page specifically designed for that browser
- **JavaScript can be used to create cookies** - A JavaScript can be used to store and retrieve information on the visitor's computer (later).

# END OF JS