Add HTML Element using JS

- Way 1 `.innerHTML` - Performance Issue
- Not to use because for adding new element, all the old will have to be re-rendered always. Can be used when instead of adding, innerHTML has to be changed

```
const todoList = document.querySelector(".todo-list")
todoList.innerHTML += "<li>Todo 2</li>"
todoList.innerHTML += "<li>Todo 3</li>"
```

- Way 2 `.createElement("tag") .createTextNode("text") .append`

```
const newTodoItem = document.createElement("li") // create tag
const newTodoItemText = document.createTextNode("Todo 2") // create text
newTodoItem.append(newTodoItemText) // add text in tag // create element

const todoList = document.querySelector(".todo-list") // get list
todoList.append(newTodoItem) // append new item in the list
```

- Way 3 `.createElement("tag") .textContent`

```
const newTodoItem = document.createElement("li") // create tag
newTodoItem.textContent = "Todo 2" // create text // change is here

const todoList = document.querySelector(".todo-list") // get list
todoList.append(newTodoItem) // append new item in the list
```

Remove HTML element using JS

`.remove`

```
const todo1 = document.querySelector('.todo-list li'); // first li
todo1.remove();
```

- `.prepend` - adds in starting
- append, prepend - adds inside ul
- after, before - adds outside ul

Insert before ul

```
const newTodoItem = document.createElement("li");
newTodoItem.textContent = "Todo 2";
const todoList = document.querySelector(".todo-list");
todoList.before(newTodoItem);
```

Insert after ul

```
const newTodoItem = document.createElement("li");
newTodoItem.textContent = "Todo 2";
const todoList = document.querySelector(".todo-list");
todoList.after(newTodoItem);
```

Insert adjacent elements

`element.insertAdjacentHTML (where, html)`

- beforebegin ~ before
- afterbegin ~ prepend
- beforeend ~ append
- afterend ~ after

```
const todoList = document.querySelector(".todo-list");
todoList.insertAdjacentHTML("beforeend", "<li>Todo 2</li>");
```

Clone nodes

`element.cloneNode(true);`

- To append as well as prepend

```
const ul = document.querySelector(".todo-list");
const li = document.createElement("li");
li.textContent = "New Todo";

ul.append(li);
ul.prepend(li);
```

- Generally dono likhe to jo baad me likha vahi override karke kaam me aayega - solution is clone node and use
- true means - deep cloning - child node (text node) bhi clone ho

```
const ul = document.querySelector(".todo-list");
const li = document.createElement("li");
li.textContent = "New Todo";

const li2 = li.cloneNode(true);

ul.append(li);
ul.prepend(li2);
```

Static List v/s Live List

- Static List - querySelector -> generally used bcoz jo tha initially vahi rakhti extra nhi
- Live List - getElementsBySomething

```
<ul class="todo-list">
    <li>Item 1</li>
    <li>Item 2</li>
    <li>Item 3</li>
    <li>Item 4</li>
    <li>Item 5</li>
</ul>
```

```
const ul = document.querySelector(".todo-list");
const listItems1 = ul.querySelectorAll("li"); /// NodeList(5) [li,li,li,li,li] //
static list
const listItems2 = ul.getElementsByTagName("li"); /// NodeList(5) [li,li,li,li,li]
// live list

const 6thLi = document.createElement("li");
6thLi.textContent = "item 6";
ul.append(6thLi);

console.log(listItems1);
/// NodeList(5) [li,li,li,li,li] // 6 li's on webpage but not in list
console.log(listItems2);
/// HTMLCollections(6) [li,li,li,li,li, li] // 6 li's on webpage b and in list
```

How to get the dimension of element?

`.getBoundingClientRect()`

- Get height & width

```
const sectionTodo = document.querySelector(".section-todo");

// Get complete info
const info = sectionTodo.getBoundingClientRect();
console.log(info); /// {bottom,height,left,right,top,width,X,V}

// Get specific info
const heightInfo = sectionTodo.getBoundingClientRect().height; /// 433.547684648
```

Events

- Something happenning - click, button press, hover

3 ways to add event - Event Fire

- Way 1 - HTML - `onclick="JS Code"`

```
<button class="btn btn-headline" onclick="console.log('Clicked')">
```

- Way 2 - JS - function on property -`ele.onclick = function(){}`

```
const btn = document.querySelector(".btn-headline");
console.dir(btn) /// {onclick: null, onblur:, onchange:.......}
btn.onclick = function(){
    console.log("Clicked");
}
console.dir(btn) /// {onclick: f() {..}, onblur:, onchange:.......}
```

- Way 3 - JS - method - `ele.addEventListener("property", function(){}`

```
const btn = document.querySelector(".btn-headline");
// 1 - Function alag se
function clickMe(){
    console.log("Clicked");
}
```

```
btn.addEventListener("click", clickMe);
// 2 - normal function in addEventListener
btn.addEventListener("click", function(){
    console.log("Clicked");});

// 3 -arrow function in addEventListener
btn.addEventListener("click", ()=>{
    console.log("Clicked")});
```

This keyword in addEventListener

```
const btn = document.querySelector(".btn-headline");

// 1 - Function alag se - button - kaun call kar rha
function clickMe(){
    console.log("Clicked");}
    console.log(this); /// <button class=""> </button>

// 2 - normal function - button - kaun call kar rha
btn.addEventListener("click",function(){
    console.log("Clicked");
    console.log(this); /// <button class=""> </button> // kaun call kar rha
});

// 3 -arrow function - window - one level up btn-> window(one level up line)
btn.addEventListener("click", ()=>{
    console.log("Clicked")});
    console.log(this); /// Window {......}
```

Click Event on Multiple Button

```
<div class="my-buttons button">
    <button> My Button 1 </button>
    <button> My Button 2 </button>
    <button> My Button 3 </button></div>
```

```
const allButtons = document.querySelectorAll(".my-buttons button");
// Normal loop
for(let i = 0 ; i < allButtons.length; i++){
    allButtons[i].addEventListener("click", function(){
        console.log(this);})} /// <button> My Button 1 </button> // this of
whichever btn we click
```

```
// For of loop
for(let button of allButtons){
    button.addEventListener("click", function(){
        console.log(this); /// <button> My Button 1 </button> // this of whichever
btn we click
        console.log(this.textContent);})} /// My Button 1

// forEach()
allButtons.forEach (function(button){
    button.addEventListener("click", function(){
        console.log(this);});}) /// <button> My Button 1 </button> // this of
whichever btn we click
```

Event Object

```
const firstButton = document.querySelector("#one"); //firstButton is element
firstButton.addEventListener("click", function(event){
    console.log(event); /// MouseEvent {......} // event is returned Object
})
```

- When event listener is added in any element :-

1. Js Engine --- line by line execute karta hai
2. Browser ---- JS Engine + Extra features
3. Browser ----- JS Engine + WebAPI

- When browser gets to know user performed event, it listens it! and do 2 works :-

1. Gives callback function to JS Engine
2. Browser gives information of Event alongwith Callback function -> in the form of Object

Properties in returned Event object

- Target - Element which triggered event
- currentTarget - Element, eventListener is attached to

```
const allButtons = document.querySelectorAll(".my-buttons button");

for(let button of allButtons){
    button.addEventListener("click",(e)=>{
        console.log(e.currentTarget); // jispar click karenge vo aayega
    })}
```

Keypress Event

keypress

```
const body = document.body;

body.addEventListener("keypress", (e) => {
  console.log(e); /// KeyboardEvent {........} // obj
  console.log(e.key); // Key which is pressed
});
```

Mouse Events

mouseover

```
const mainButton = document.querySelector(".btn-headline");
console.log(mainButton);
mainButton.addEventListener("mouseover", () => {
  console.log("Mouseover event ocurred!!!");
});
```

mouseleave

```
mainButton.addEventListener("mouseleave", () => {
  console.log("Mouseleave event ocurred!!!");
});
```

Event Capturing, Bubbling and Delegation

```
    <main>
      <div class="grandparent box">Grandparent
        <div class="parent box">Parent
          <div class="child box">Child</div>
        </div>
      </div>
    </main>
```

```
*{ padding:0;margin: 0;box-sizing: border-box; }
body{ font-family:arial,sans-serif; }
main{
  display:flex; justify-content:center; align-items: center; height:100vh; }
.box{min-width:100px; min-height: 100px; padding:5rem; color:white;
 font-weight: bold;  font-size: 2rem; cursor: pointer; }
.grandparent{ background: #b1d4e0; color:black; }
.parent{ background: #145da0; }
.child{ background: #0c2d48; }
```

```
const grandparent = document.querySelector(".grandparent");
const parent = document.querySelector(".parent");
const child = document.querySelector(".child");
```

Event Bubbling / Even Propagation

- Event bubbling is a method of event propagation in the HTML DOM API when an event is in an element inside another element, and both elements have registered a handle to that event. It is a process that starts with the element that triggered the event and then bubbles up to the containing elements in the hierarchy.
- `click -> parent of click -> parent of parent of click`
- what is clicked -> parent of what is clicked -> parent of parent of what is clicked

```
child.addEventListener("click",() => {
    console.log("You clicked on child");});
parent.addEventListener("click",() => {
    console.log("You clicked on parent");});
grandparent.addEventListener("click",() => {
    console.log("You clicked on grandparent");});
document.body.addEventListener("click",() => {
    console.log("You clicked on document.body");});
```

Output - click on child

- Click on child, child->parent->grand parent->body 's same event is also clicked

```
You clicked on child
You clicked on parent
You clicked on grandparent
You clicked on body
```

Output - click on parent

- Click on parent, parent->grand parent->body 's same event is also clicked

```
You clicked on parent
You clicked on grandparent
You clicked on body
```

Event Capturing

`Capturing events outermost parent of click -> parent of parent of click -> parent of click -> click`

- 3rd arg of addEventListener -> `true` - allow capturing
- Outermost parent of what is clicked to what is clicked -> Then bubbling

```
child.addEventListener("click",() => {
    console.log("capture !!!! child");},
     true);
parent.addEventListener("click",() => {
    console.log("capture !!!! parent");},
     true);
grandparent.addEventListener("click",() => {
    console.log("capture !!!! grandparent");},
    true);
document.body.addEventListener("click",() => {
    console.log("capture !!!! document.body");},
    true);
```

- Thus, Event capturing is the event starts from top element to the target element. It is the opposite of Event bubbling, which starts from target element to the top element.
- `Not Capturing events`
- 3rd arg of addEventListener -> `false` - not allowing capturing

```
child.addEventListener("click",() => {
    console.log("You clicked on child");});
parent.addEventListener("click",() => {
    console.log("You clicked on parent");});
grandparent.addEventListener("click",() => {
    console.log("You clicked on grandparent");});
document.body.addEventListener("click",() => {
    console.log("You clicked on document.body");});
```

`Capture -> Bubbling` Output - click on child

```
// capture - parent to click
capture !!!! body
capture !!!! grandparent
capture !!!! parent
capture !!!! child
// not capture ~ bubbling - click to parent
You clicked on child
You clicked on parent
You clicked on grandparent
You clicked on body
```

Practice Capturing and bubbling

```
grandparent.addEventListener("click",() => {
    console.log("capture !!!! grandparent");},
    true);
document.body.addEventListener("click",() => {
    console.log("capture !!!! document.body");},
    true);

child.addEventListener("click",() => {
    console.log("You clicked on child");});
parent.addEventListener("click",() => {
    console.log("You clicked on parent");});
// grandparent.addEventListener("click",() => {
//     console.log("You clicked on grandparent");});
// document.body.addEventListener("click",() => {
//     console.log("You clicked on document.body");});
```

`Capture -> Bubbling` Output - click on child

```
// capture - parent to click
capture !!!! body
capture !!!! grandparent
// not capture ~ bubbling - click to parent
You clicked on child
You clicked on parent
```

Event Delegation

```
grandparent.addEventListener("click", (e) => {
  console.log("Clicked");});
```

Output - click on grandparent

```
Clicked
```

Output - click on child -> child par to event bhi nhi

```
Clicked
```

- Capture -> Parent
- Click on child => true-grandparent -> true-parent -> true-child -> child -> parent -> grandparent(print) `No need to add different EventListener for child and parent`

Event object - Delegation

- Event Delegation is basically a pattern to handle events efficiently. Instead of adding an event listener to each and every similar element, we can add an event listener to a parent element and call an event on a particular target using the .target property of the event object.
- Make one Event listener for child, parent, grandparent -> event's target will get which one is clicked!
- `e.target will have jo click hoga vahi`
- When Child is clicked -> callback of grandparent run

```
grandparent.addEventListener("click", (e) => {
    console.log(e); /// MouseEvent{target : div.child.box}
    console.log(e.target); /// <div>"Child"</div>
    console.log(e.target.textContent); }); /// Child
```

- When parent is clicked -> callback of grandparent run

```
grandparent.addEventListener("click", (e) => {
    console.log(e); /// MouseEvent{target : div.parent.box}
    console.log(e.target); /// <div>"Parent"</div>
```

```
    console.log(e.target.textContent);}); /// Parent Child *** // parent ke andar
hi child hai
```
