Day32(events, eventListener , input events, value , eventobject , target , submitevent , form , preventdefault)

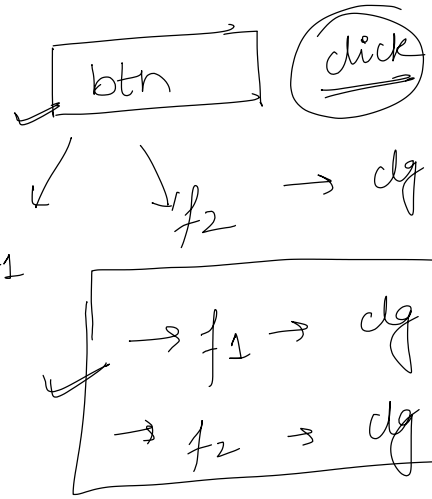
Wednesday, 26 April 2023 7:41 PM

Events

inline way

<button onclick="kaam()"> Click </button>

event handling property



addEventListener( )

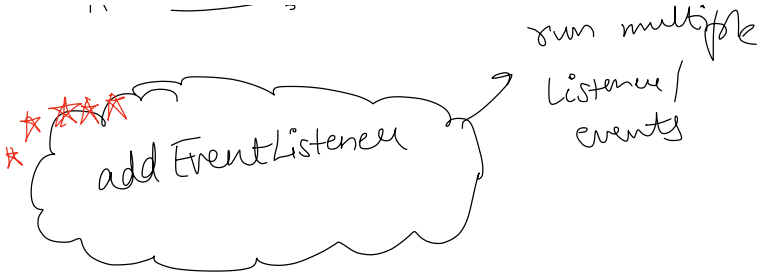
event trigger — Kaam

eg;

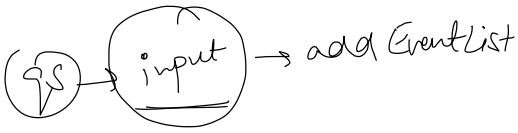
```
body
btn.addEventListener('click', function() {
    clg('Hi');
    body.style.bgC = "plum";
});
```

Imp

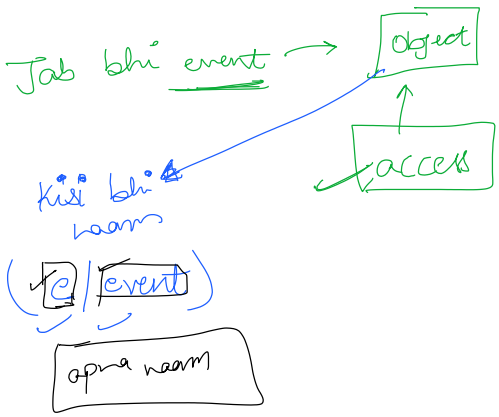
Best ?

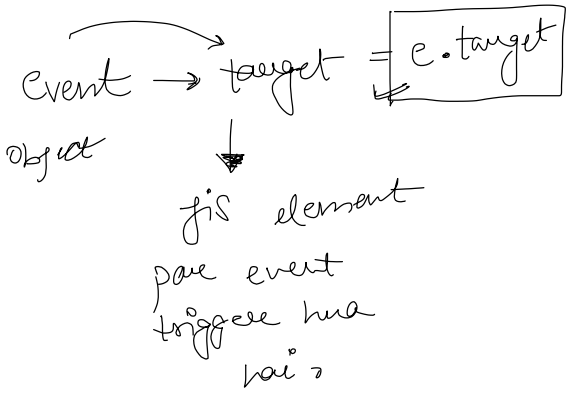


<input type="text">

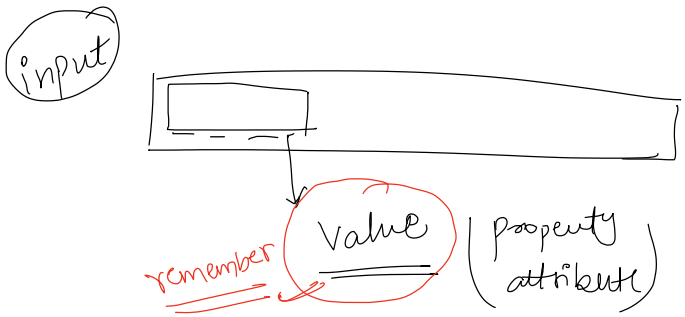


Whenever an event is run / executed along with the function / event, an object is also attached with it which we can have access of.

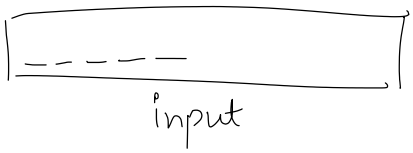


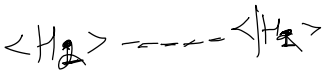


window.navigator.onLine => it is helpful in determining whether the user is online or offline.
Returns true => User online
Returns false => User offline



Question :

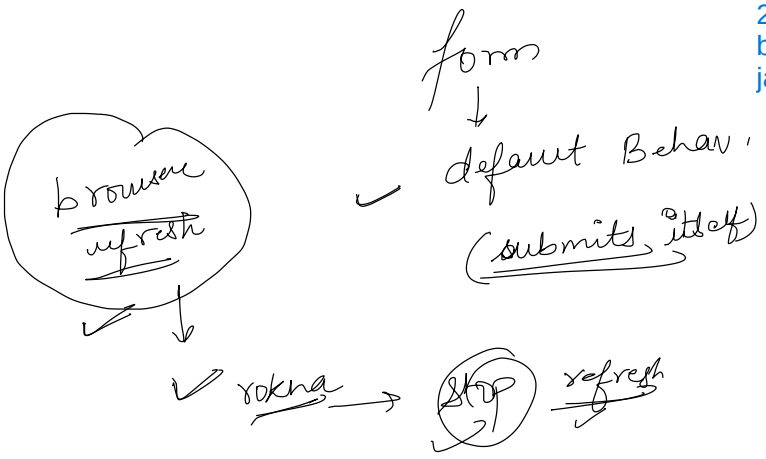




1. form ka default behaviour hota hai browser ko refresh karna (Kya hum isse rokh sakte hai ?) when we submit the data of the form.

2. event.preventDefault() => event ke default beha. ko rokh dega. mostly form ke sath use kiya jaata hai.

Now my browser will not be refreshed when we submit the data of the form.

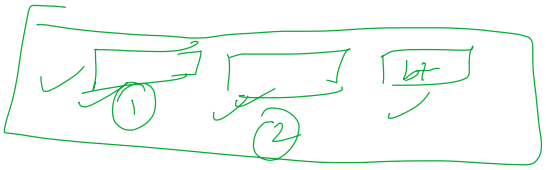


accessing elements of form

form → elements

property inside form

form ke ander elements name ki property hoti hai jiske through hum form ke elements ko access kar sakte hai.



JavaScript Code Parsing and Execution:

When the browser loads a webpage, it parses the HTML to create the DOM and then parses and executes the JavaScript code.
During this execution, if the code includes addEventListener calls, the browser registers these event listeners.

Event Listener Registration:

For each addEventListener call, the JavaScript engine registers the specified event listener (e.g., click, mouseover) to the corresponding DOM element.
The event listener is stored internally by the browser, associated with the specific element that it was added to.

Event Triggering:

When a user interacts with the webpage (e.g., clicks a button), the browser detects the event and creates an event object.
This event object contains details about the event (like the type of event, the target element, etc.).

Event Propagation:

The event undergoes three phases:

Capturing Phase: The event travels from the window object down to the target element.

Target Phase: The event reaches the target element where it was originally triggered.

Bubbling Phase: The event bubbles back up from the target element to the window object.

The event propagation phase gives elements along the way a chance to handle the event.

Event Handling:

During the target or bubbling phase (depending on how the listener was registered), the browser checks if the target element (or any of its ancestors during bubbling) has an event listener for this specific event.

If a matching event listener is found, the corresponding callback function is executed.

The function is added to the call stack and executed, performing the defined actions (e.g., logging a message, updating the UI).
Completion:

After the callback function executes, it is removed from the call stack, and the event loop continues, ready to handle the next event or operation.

Key Points:

Event listeners are registered during the initial parsing of the JavaScript code.

Only specified events are monitored on their respective elements.

Event propagation determines how the event moves through the DOM, and whether it is handled by the target or ancestor elements.

The corresponding callback is executed when a registered event is triggered on its associated element.