Events

Event Handling in JavaScript

Au Mau Duong
Technical Trainers

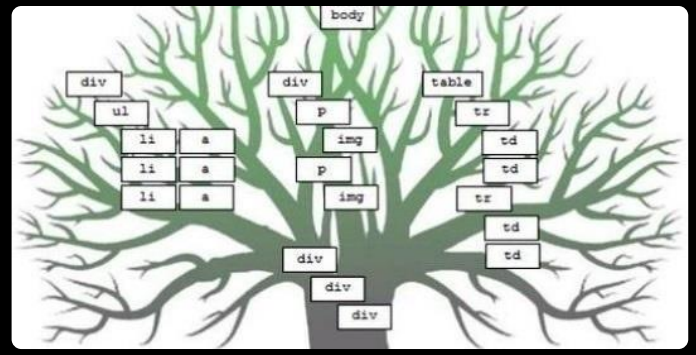
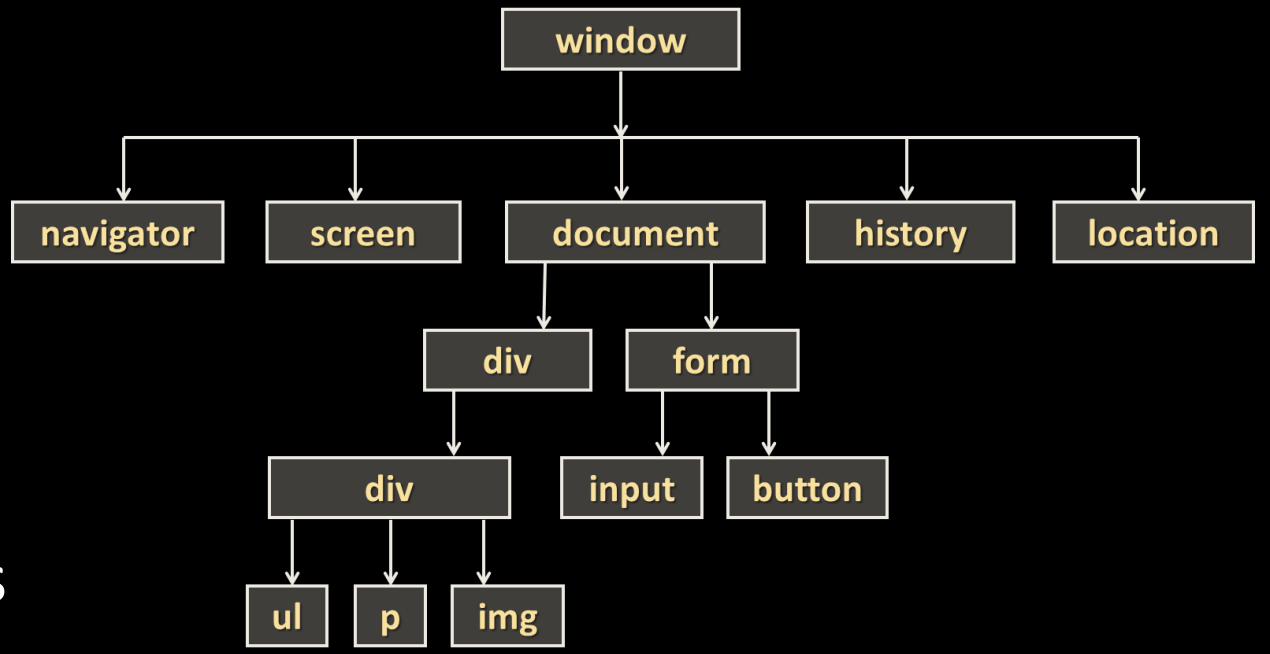


Table of Contents

Built-in Browser Objects (1)

- The browser provides some read-only data via:
  - `window`
    - The top node of the DOM tree
    - Represents the browser's window
  - `document`
    - Holds information of the current loaded document
  - `screen`
    - Holds the user's display properties
  - `navigator`
    - Holds information about the browser



Built-in Browser Objects (2)

- The browser provides some read-only data via:
  - `history`
    - The window.history object contains the browsers history

```
window.history.back()
```

```
window.history.forward()
```

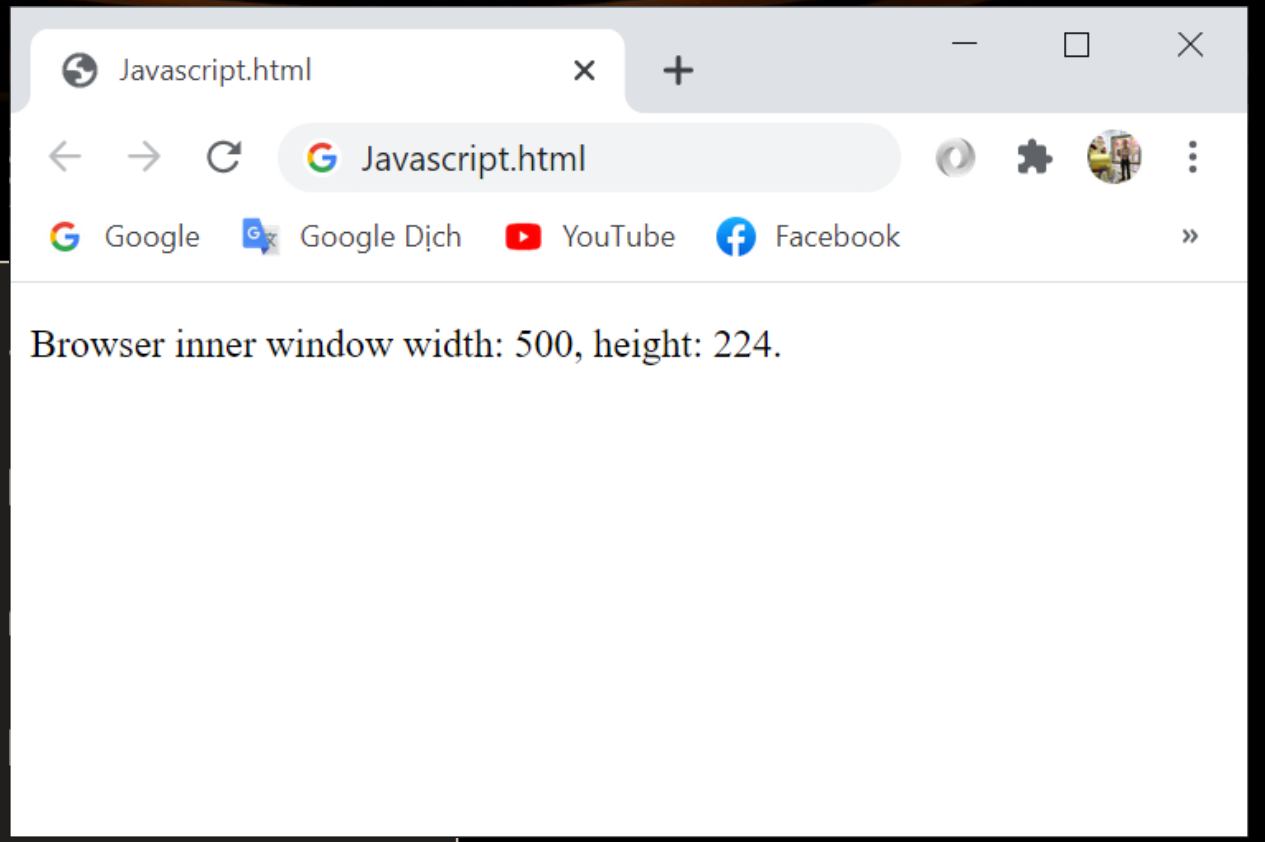
  - `location`
    - The window.location object can be used to get the current page address (URL) and to redirect the browser to a new page

```
window.location.(href/hostname/pathname/protocol/port/assign)
```

Window – Example

```
<p id="demo"></p>
```

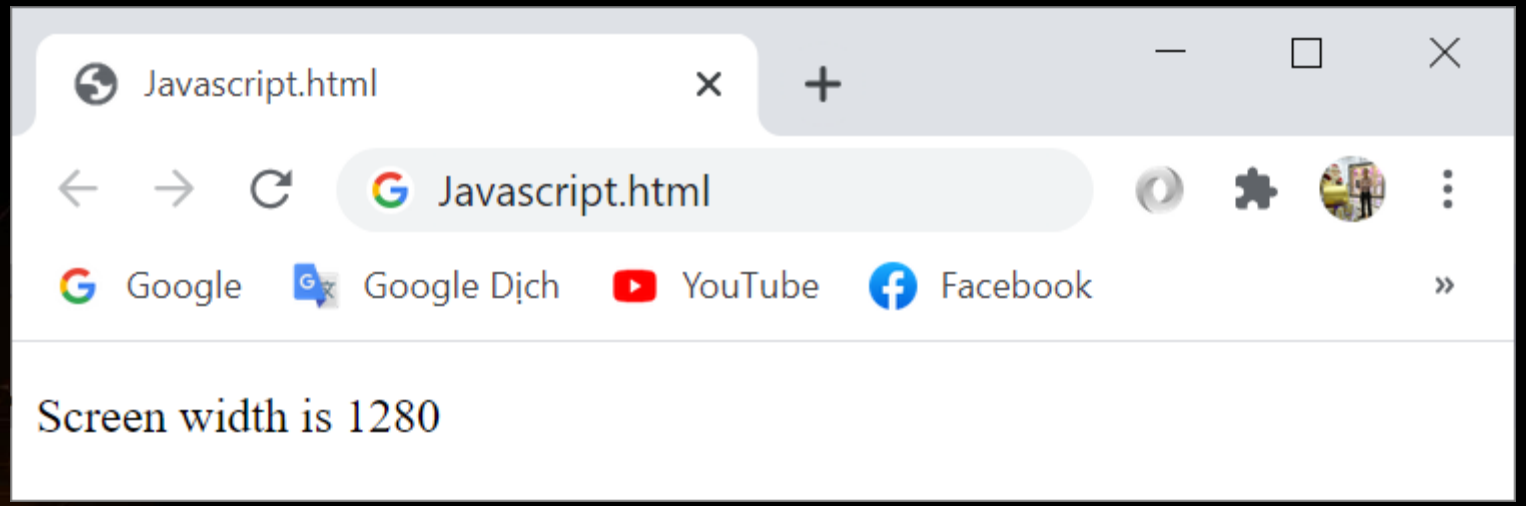
```
<script>
var w = window.innerWidth
|| document.documentElement.clientWidth
|| document.body.clientWidth;

var h = window.innerHeight
|| document.documentElement.clientHeight
|| document.body.clientHeight;

var x = document.getElementById("demo");
x.innerHTML = "Browser inner window width: " + w + ", height: " + h + ".";
</script>
```



Screen – Example

```
<p id="demo"></p>
```

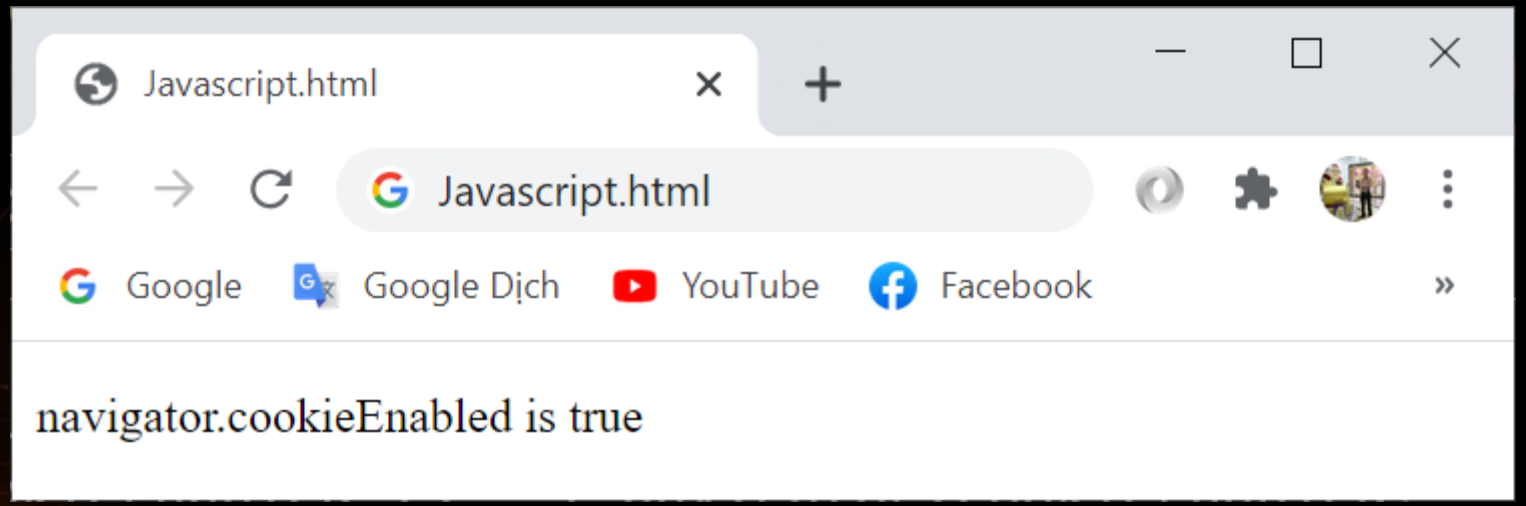
```
<script>
document.getElementById("demo").innerHTML =  "Screen
width is " + screen.width;
</script>
```



Navigator – Example

```
<p id="demo"></p>
```

```
<script>
document.getElementById("demo").innerHTML =
"navigator.cookieEnabled is " + navigator.cookieEnabled;
</script>
```



Document Object Model

- What is Document Object Model (DOM)?
  - A concept of representing a HTML document as a "DOM tree"
  - Consists of elements that have child elements
  - Elements have properties (attribute + value) and events
- DOM provides an API for traversing / modifying the DOM tree
  - Enables developers to modify the HTML content and the visual presentation of the currently loaded HTML document
  - E.g. load a table data (JSON) and show it as a HTML table

The DOM API

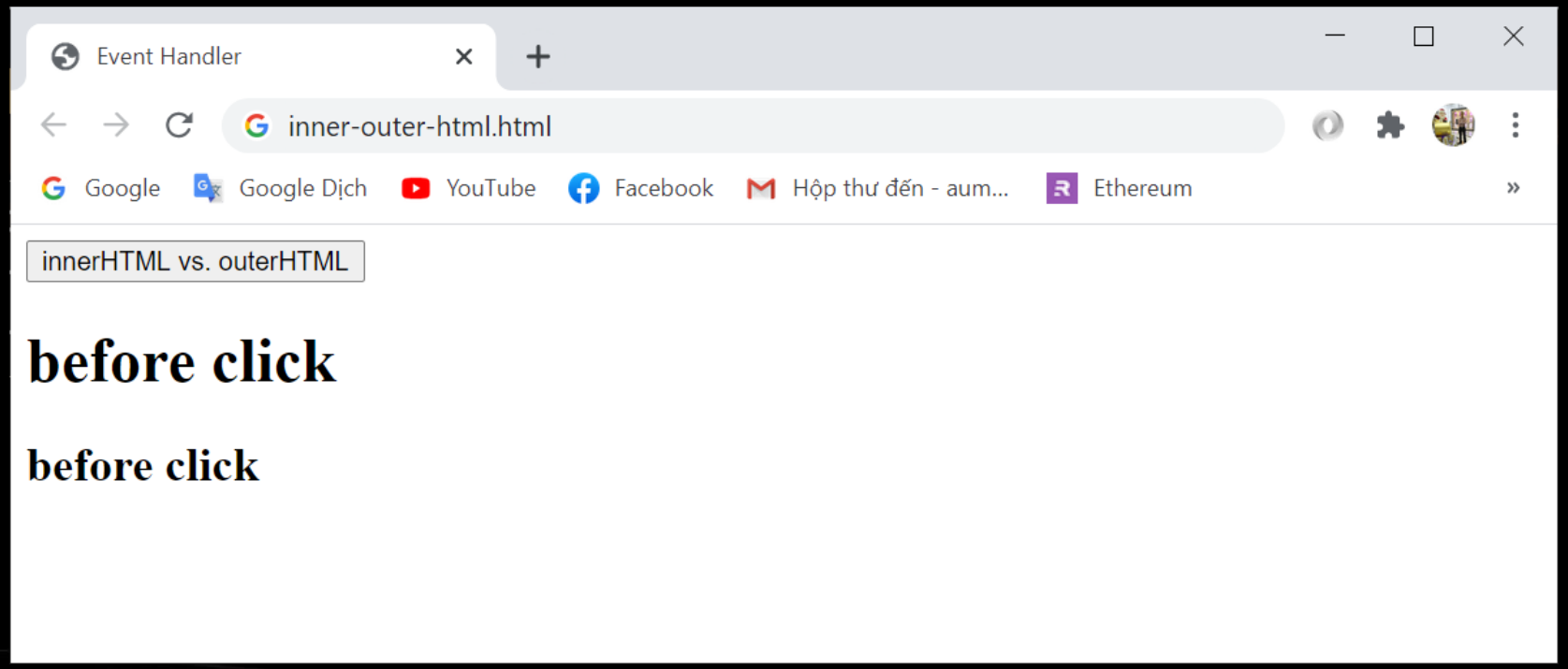
- Web browsers provide a DOM API
  - Consists of objects and methods to interact with the HTML page
  - Can add / modify / remove HTML elements
  - Can add / modify / remove HTML attributes
  - Can apply CSS styles dynamically
- HTML elements and their properties are mapped to JS objects
  - **`document.documentElement`** is the **`<html>`** element
  - **`document.body`** is the **`<body>`** element of the page

HTML Elements – Common Properties

- All HTML elements have common properties
  - Corresponding to the their HTML attributes
  - `id`, `className`, `style`, `onclick`, etc.
  - `innerHTML`
    - Holds a string – the content of the element, without the element
  - `outerHTML`
    - Holds a string – the content of the element, with the element
  - `innerText` / `textContent`
    - Holds a string – the text content of the element, without the tags

HTML Elements – innerHTML vs outerHTML (1)

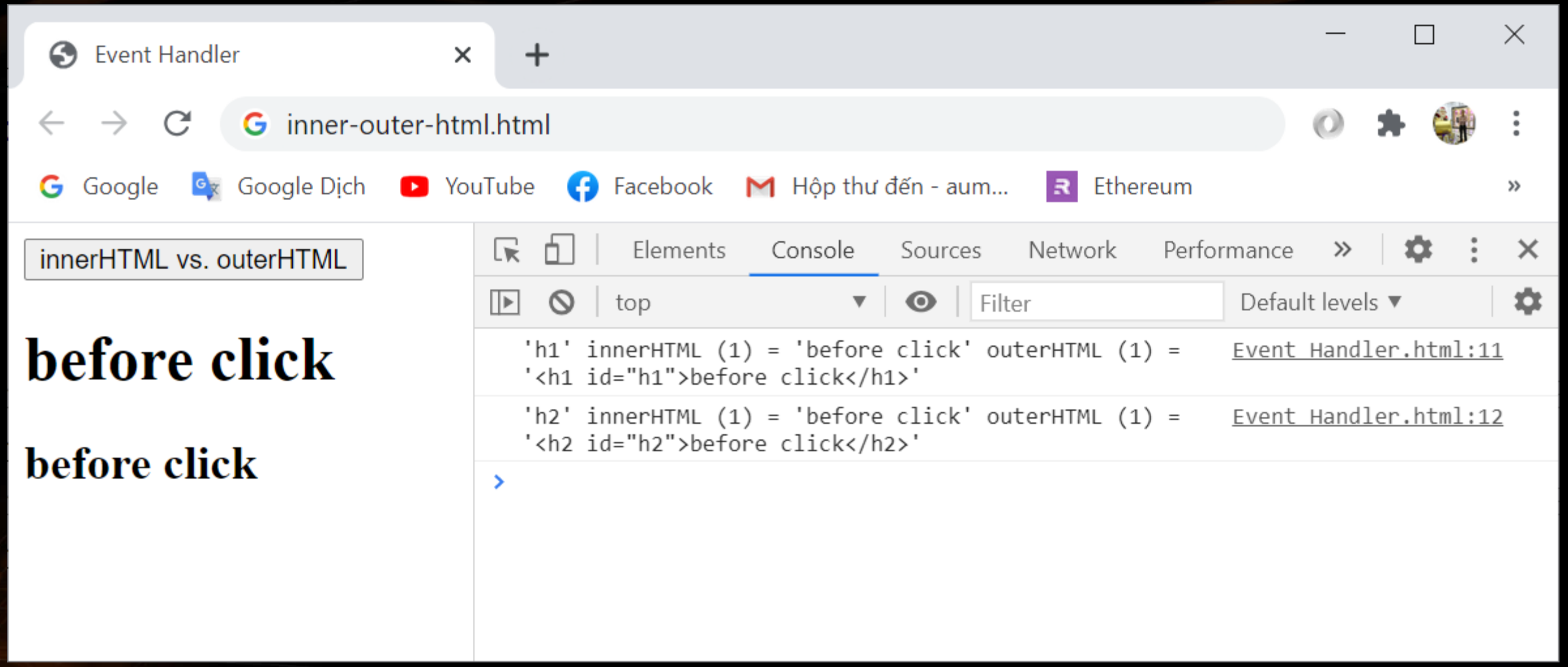
```
<button id="button"> innerHTML vs. outerHTML </button>
<h1 id="h1"> before click </h1>
<h2 id="h2"> before click </h2>
```



HTML Elements – innerHTML vs outerHTML (2)

```
<script>
var h1 = document.getElementById("h1"),
    h2 = document.getElementById("h2");
var ran = false;
console.log("'h1' innerHTML (1) =", "'" +
h1.innerHTML + "'", "outerHTML (1) =", "'" +
h1.outerHTML + "'");
console.log("'h2' innerHTML (1) =", "'" +
h2.innerHTML + "'", "outerHTML (1) =", "'" +
h2.outerHTML + "'");
</script>
```

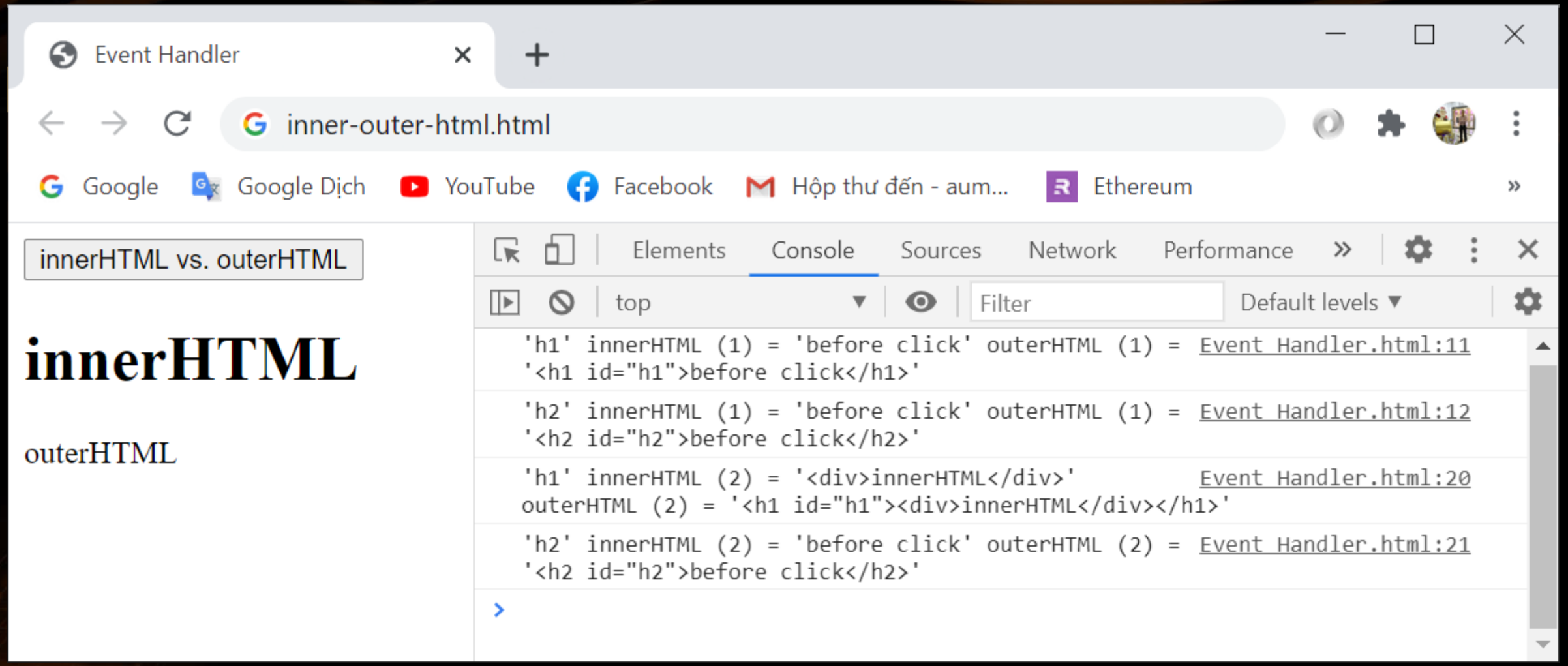
HTML Elements – innerHTML vs outerHTML (3)



HTML Elements – innerHTML vs outerHTML (4)

```
<script>
document.getElementById("button").onclick = evt => {
    if (ran) return false;

    h1.innerHTML = "<div>innerHTML</div>";
    h2.outerHTML = "<div>outerHTML</div>";

    console.log("'h1' innerHTML (2) =", "'" + h1.innerHTML
+ "'", "outerHTML (2) =", "'" + h1.outerHTML + "'");
    console.log("'h2' innerHTML (2) =", "'" + h2.innerHTML
+ "'", "outerHTML (2) =", "'" + h2.outerHTML + "'");

    ran = true
}
</script>
```

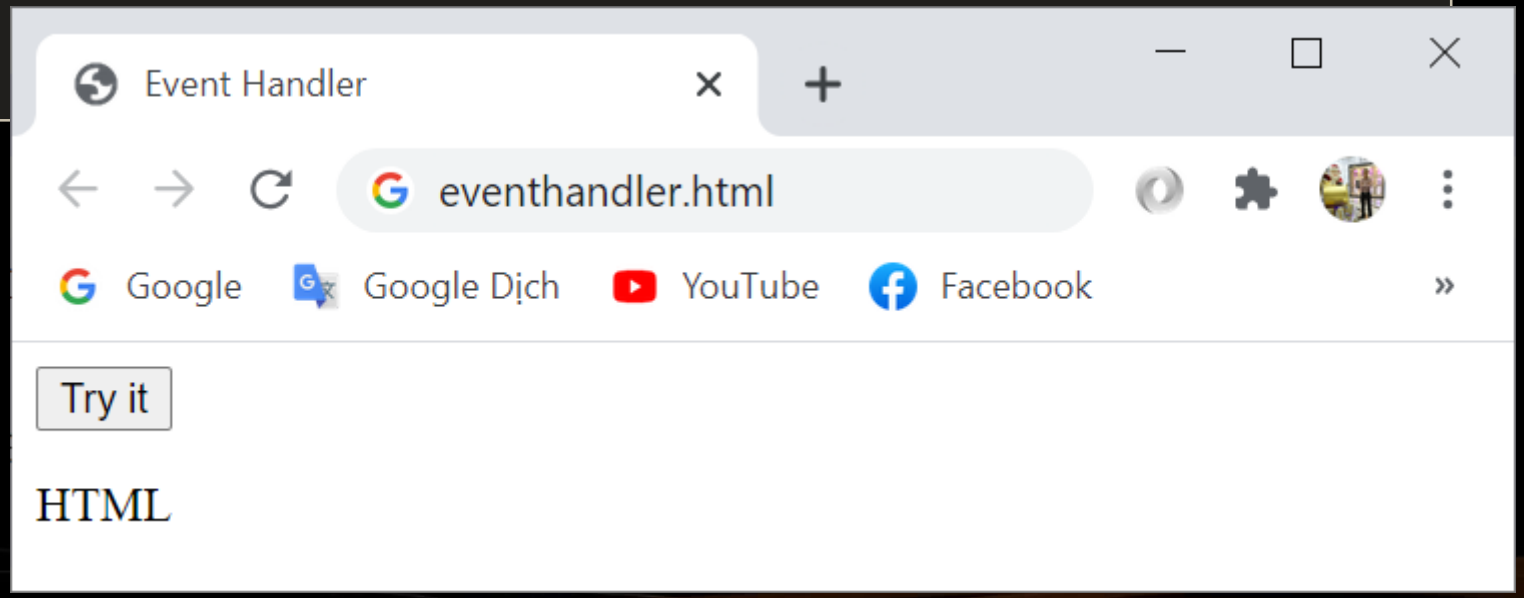
HTML Elements – innerHTML vs outerHTML (5)



DOM API – document.documentElement

```
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
```

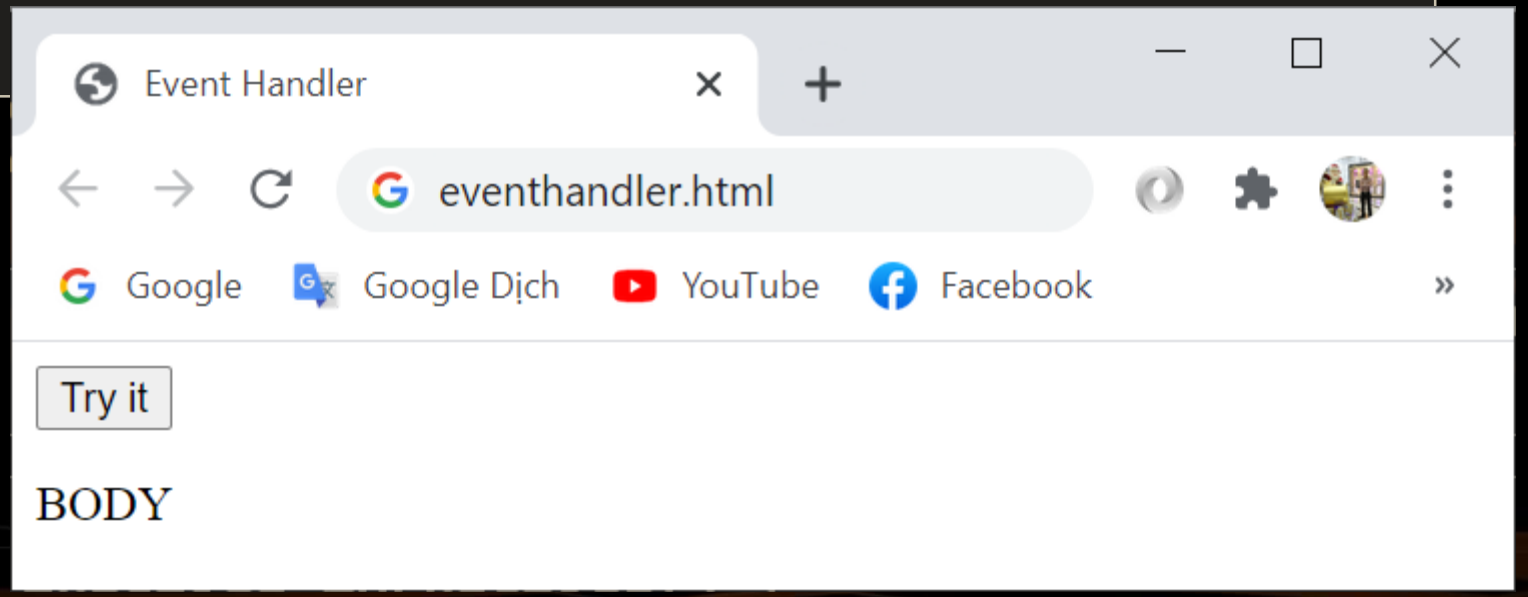
```
<script>
function myFunction() {
  var x = document.documentElement.nodeName;
  document.getElementById("demo").innerHTML = x;
}
</script>
```



DOM API – document.body

```
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
```

```
<script>
function myFunction() {
  var x = document.body.nodeName;
  document.getElementById("demo").innerHTML = x;
}
</script>
```



Selecting HTML Elements from DOM

- Select a single element → returns **HTMLElement**

```
var header = document.getElementById('header');
var nav = document.querySelector('#main-nav');
```

- Select a collection of elements → returns a collection

```
var inputs = document.getElementsByTagName('li');
var radiosGroup = document.getElementsByName('genders[]');
var header = document.querySelectorAll('#main-nav li');
```

- Access the predefined collections of elements

```
var links = document.links;
var forms = document.forms;
```

Selecing with document.getElementsBy…

- Select element by ID → returns **HTMLElement**

```
var header = document.getElementById('header');
```

- Select elements by CSS class → returns a collection

```
var posts = document.getElementsByClassName('post-item');
```

- Select elements tag name → returns a collection

```
var sidebars = document.getElementsByTagName('sidebar');
```

- Select element by name (in forms) → returns a collection

```
var gendersGroup = document.getElementsByName('genders[]');
```

Query Selectors

- CSS-like selectors for accessing the DOM tree
  - `querySelector(…)`
    - Returns the first element that matches the selector
  - `querySelectorAll(…)`
    - Returns a collection of all elements that match the selector

```
var header = document.querySelector('#header');
```

```
var tableCells = document.querySelectorAll('table tr td');
```

```
var selectedLi = document.querySelector('menu > li.selected');
```

```
var specialLinks = document.querySelectorAll('a.special');
```

Selecting Inner Elements

- HTML elements support select for their inner elements
  - Select all DIVs that are inside an element with id "**wrapper**"

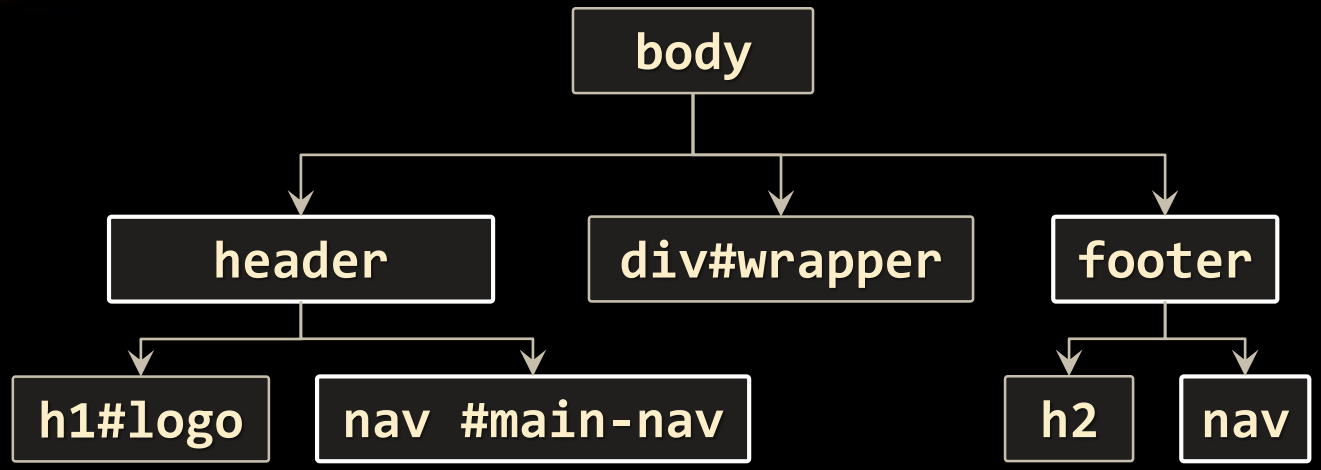
```
var wrapper = document.getElementById('wrapper');
var divsInWrapper = wrapper.getElementsByTagName('div');
```

- All methods can be used on HTML elements
  - Except **getElementById()**

NodeList

- A **NodeList** is a collection returned by the DOM selectors:
  - **getElementsByTagName(tagName)**
  - **getElementsByName(name)**
  - **getElementsByClassName(className)**
  - **querySelectorAll(selector)**
- https://developer.mozilla.org/en/docs/Web/API/NodeList

```
var divs = document.getElementsByTagName('div');
var queryDivs = document.querySelectorAll('div');
for (var i=0; i<divs.length; i++) {
   // Do something with divs[i] …
}
```



Traversing the DOM

Create, Remove, Alter and Append HTML Elements

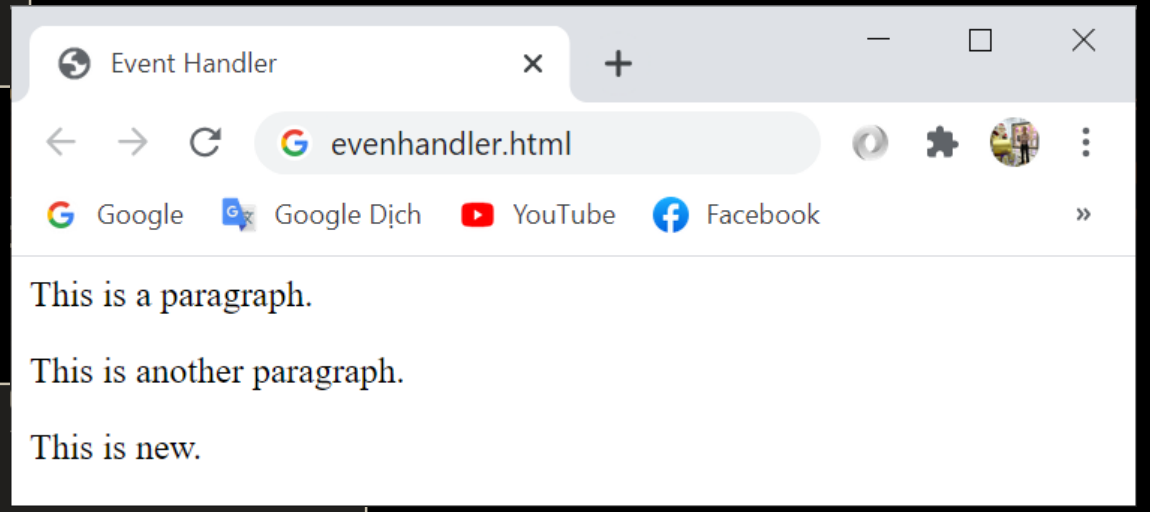
Creating HTML Elements

- The document object can create new HTML elements
  - `document.createElement(elementName)`
- Newly created elements are not in the DOM (the web page)
  - Must be appended to DOM manually

```
var studentsList = document.createElement("ul");
var studentLi = document.createElement("li");
studentLi.innerHTML = "Student: Pesho";
studentsList.appendChild(studentLi);
document.body.appendChild(studentsList);
```

Creating HTML Elements - Example

```
<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>
```



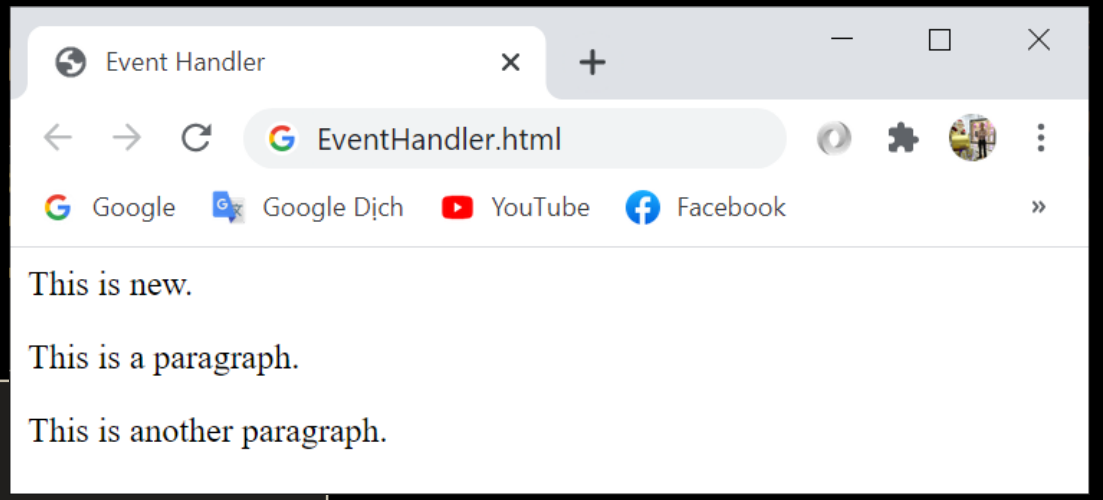

```
<script>
var para = document.createElement("p");
var node = document.createTextNode("This is new.");
para.appendChild(node);
var element = document.getElementById("div1");
element.appendChild(para);
</script>
```

Inserting Elements Before / After Element

- The DOM API supports inserting a element before or after a specific element
  - `element.appendChild(child)`
    - Inserts the element always at the end of the DOM element
  - `parent.insertBefore(newNode, specificElement)`
    - Inserts the element before specific element
  - `parent.insertAfter(newNode, specificElement)`
    - JavaScript DOM hasn't supported

Inserting Before - Example

```
<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>
```




```
<script>
var para = document.createElement("p");
var node = document.createTextNode("This is new.");
para.appendChild(node);
var element = document.getElementById("div1");
var child = document.getElementById("p1");
element.insertBefore(para,child);
</script>
```

Removing Elements

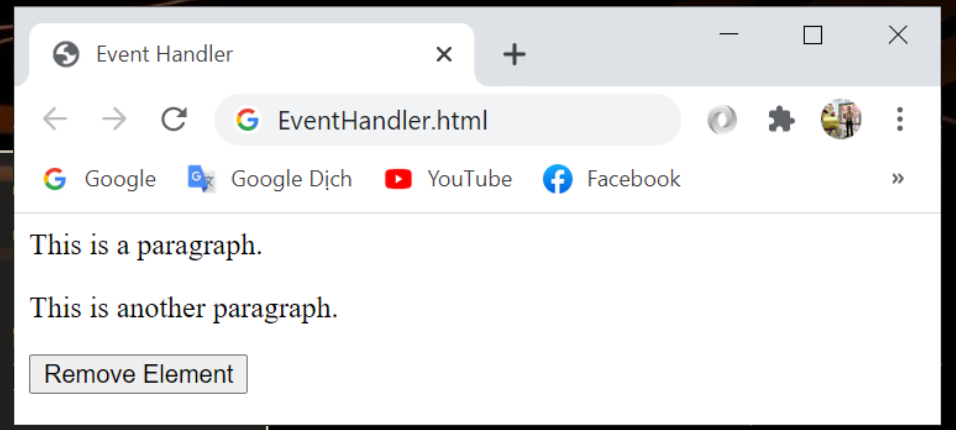
- Elements can be removed from the DOM
  - Using `element.removeChild(elToRemove)`
  - Pass the element-to-remove to their parent

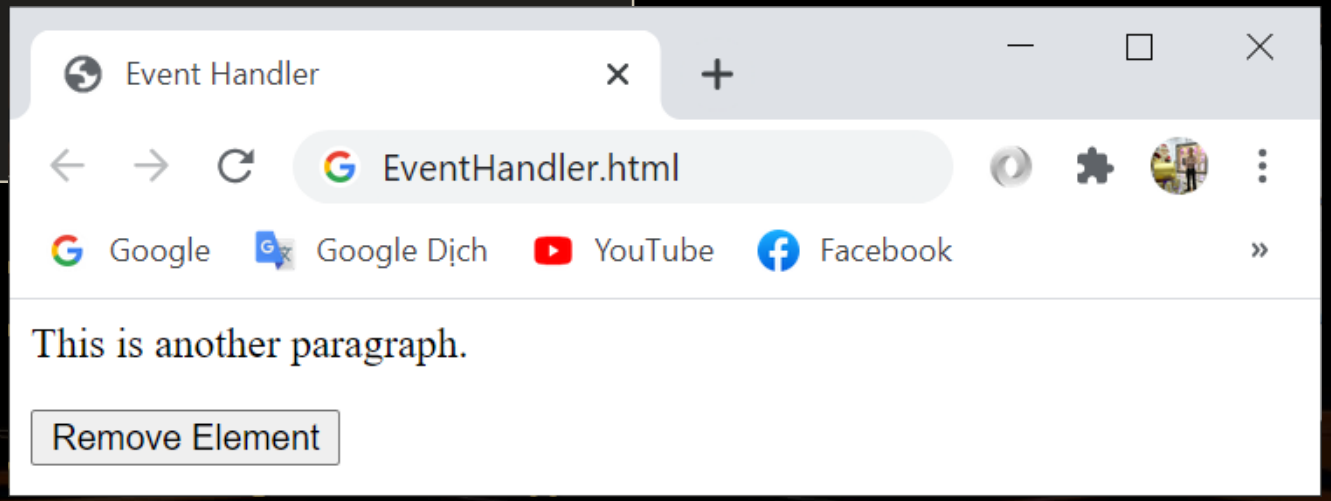
```
var trainers = document.getElementsByTagName("ul")[0];
var trainer = trainers.firstChild;
trainers.removeChild(trainer);

// Remove a selected element
var selectedElement = //select the element
selectedElement.parentNode.removeChild(selectedElement);
```

Removing Elements - Example

```
<div>
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>
<button onclick="myFunction()">Remove Element</button>
```

```
<script>
function myFunction() {
  var elmnt = document.getElementById("p1");
  elmnt.remove();
}
</script>
```



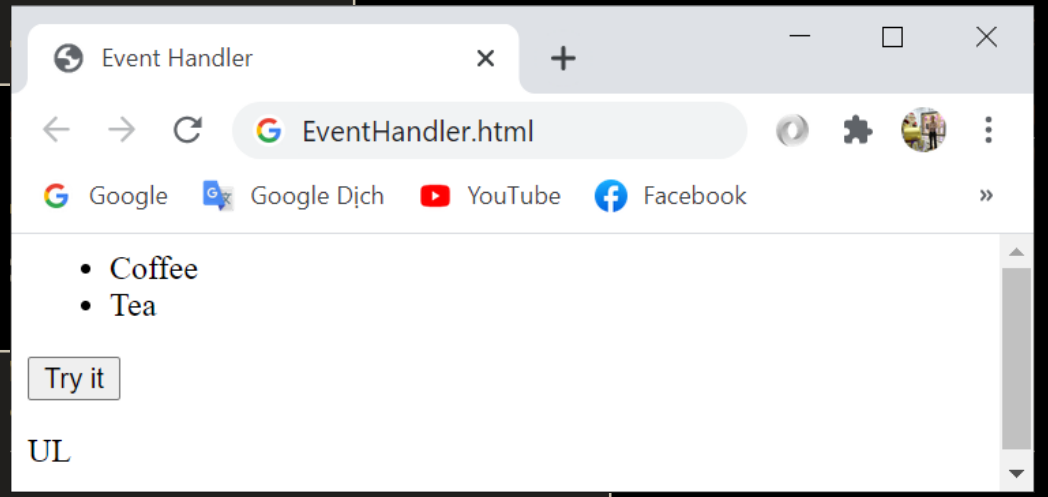


Traversing the DOM

- DOM elements know their position in the DOM tree
  - Parent: `element.parentNode`
    - Returns the direct parent of the element (null for the document)
  - Children: `element.childNodes`
    - Returns a `NodeList` of all the child nodes (including the text nodes)
    - First / last child – `element.firstElementChild` / `element.lastElementChild`
  - Siblings (elements around the element):
    - `element.nextElementSibling`
    - `element.previousElementSibling`

Elements - parentNode

```
<ul>
  <li id="myLI">Coffee</li>
  <li>Tea</li>
</ul>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
```

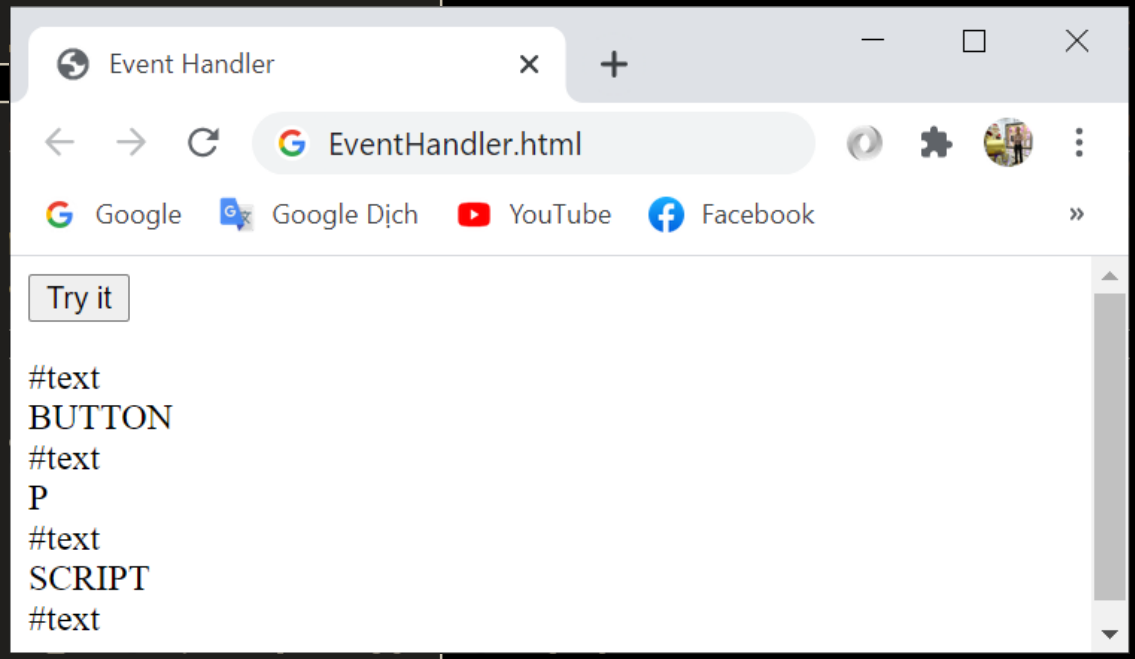



```
<script>
function myFunction() {
  var x = document.getElementById("myLI").parentNode.nodeName;
  document.getElementById("demo").innerHTML = x;
}
</script>
```

Elements - childNodes

```
<body>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
```

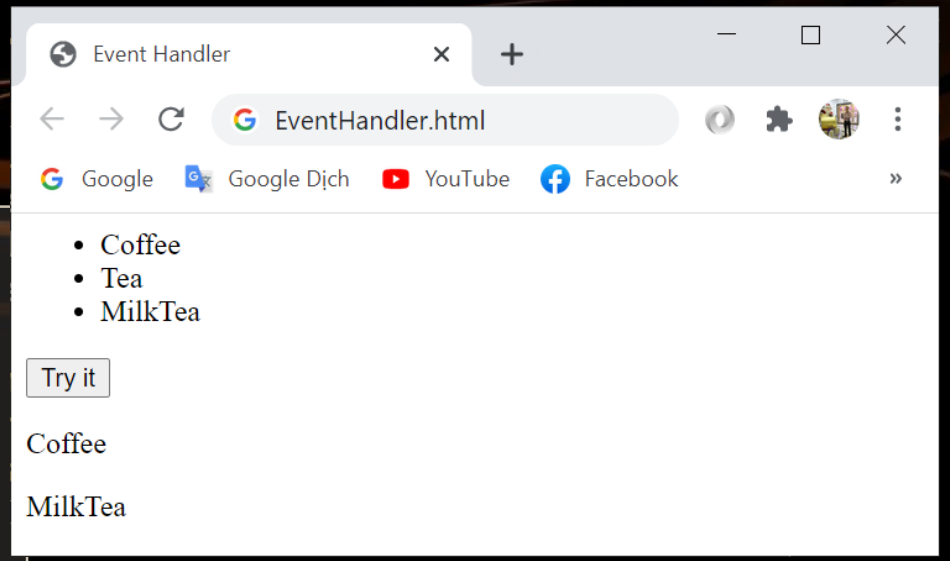
```
<script>
function myFunction() {
  var c = document.body.childNodes;
  var txt = "";
  var i;
  for (i = 0; i < c.length; i++) {
    txt = txt + c[i].nodeName + "<br>";
  }
  document.getElementById("demo").innerHTML = txt;
}
</script>
</body>
```



Elements - firstChild, lastChild

```
ul id="myList">
<li>Coffee</li>
<li>Tea</li>
<li>MilkTea</li>
</ul>
<button onclick="myFunction()">Try it</button>
<p id="first"></p>
<p id="last"></p>
```

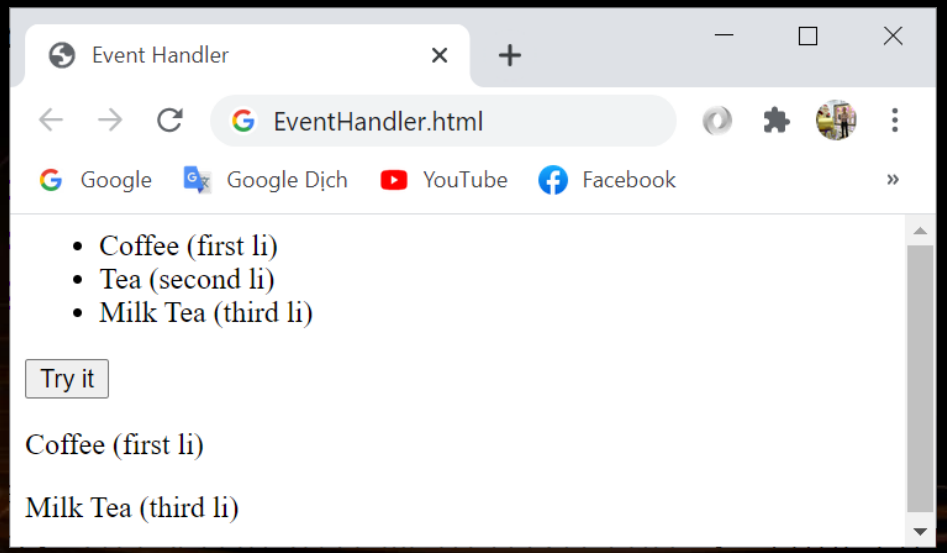
```
<script>
var list = document.getElementById("myList");
  document.getElementById("first").innerHTML =
list.firstElementChild.innerHTML;
  document.getElementById("last").innerHTML =
list.lastElementChild.innerHTML;
</script>
```



Elements – Sibling

```
<ul>
<li id="item1">Coffee (first li)</li>
<li id="item2">Tea (second li)</li>
<li id="item2">Milk Tea (third li)</li>
</ul>
<button onclick="myFunction()">Try
it</button>
<p id="previousElementSibling"></p>
<p id="nextElementSibling"></p>
```

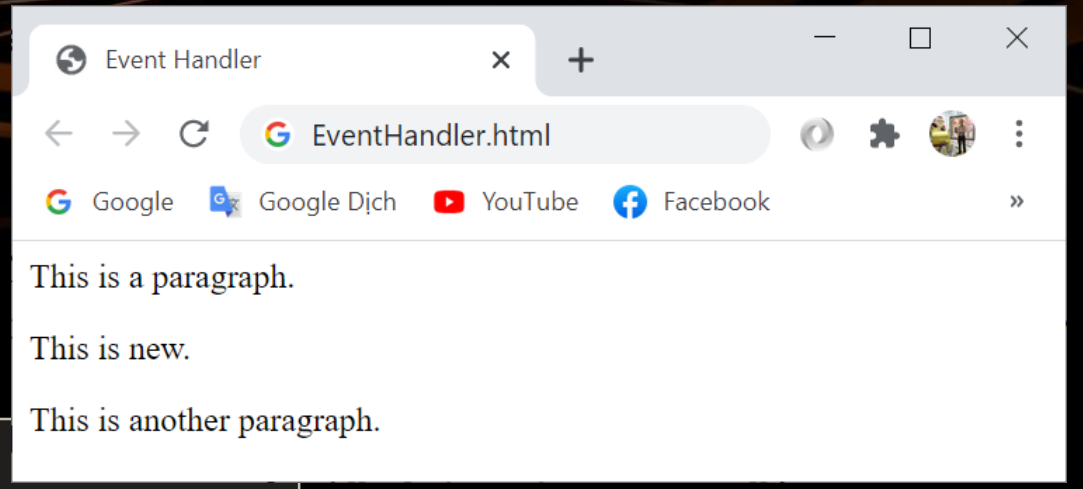
```
<script>
function myFunction() {
var x =
document.getElementById("item2");

document.getElementById("previousEl
ementSibling").innerHTML =
x.previousElementSibling.innerHTML;

document.getElementById("nextElemen
tSibling").innerHTML =
x.nextElementSibling.innerHTML;
}
</script>
```



Inserting After - Example

```
<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>
```

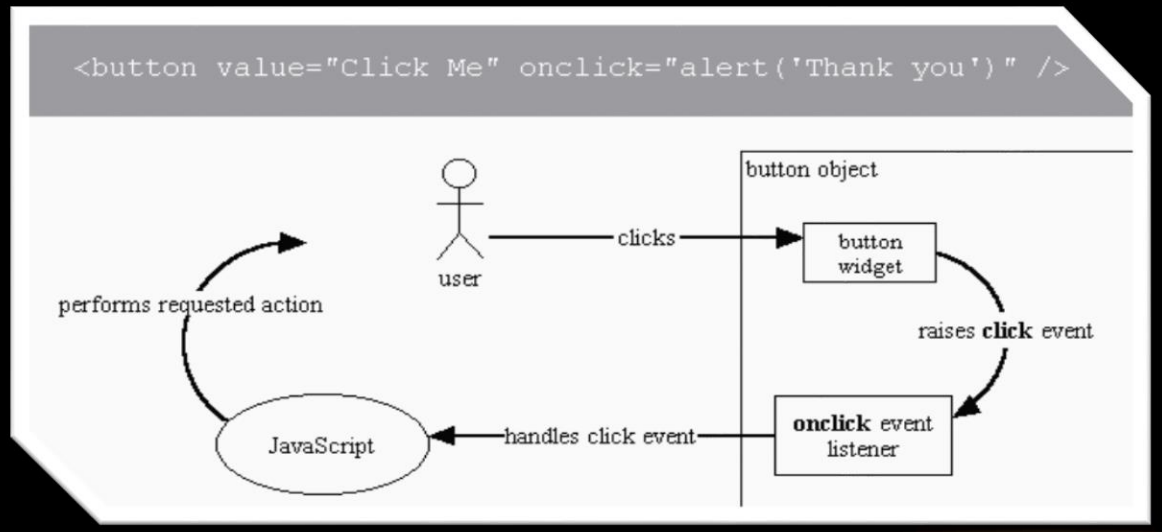
```
<script>
function insertAfter(newNode, existingNode) {
    existingNode.parentNode.insertBefore(newNode,
existingNode.nextSibling);
}
var para = document.createElement("p");
var node = document.createTextNode("This is new.");
para.appendChild(node);
var element = document.getElementById("div1");
var child = document.getElementById("p1");
insertAfter(para,child);
</script>
```



JavaScript Event Model

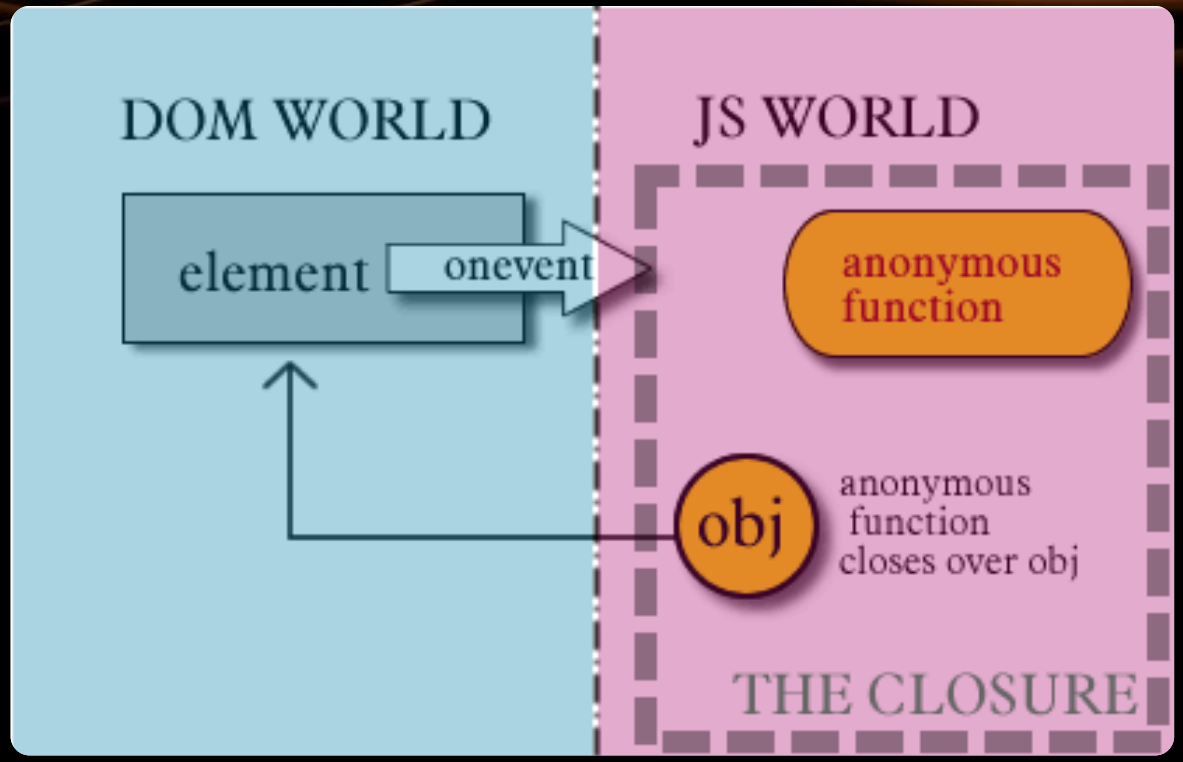
Event Model

- The DOM event model provides notifications for certain events
  - E.g. execute a JS function when a button is clicked
- The DOM event model consists of events and event listeners attached to the DOM objects
- Events Demo



Event Types

- DOM provides access to many events
  - Mouse events – mouse clicks, mouse moves, mouse over, …
  - Touch events – finger touch, touch start, end, move, …
  - Form events – field focus, value change, form submit, …
  - Keyboard events – key down, key up, key press, …
  - DOM / UI events – node insert, node remove, load, resize, …
- Full list of all DOM event types:
  - http://www.w3.org/TR/DOM-Level-3-Events/#event-types-list
- You may also define custom event types



Event Handler Registration

Define Event Handler in the HTML Code

- Event handling JavaScript code can be specified in the HTML attributes **onclick**, **onload**, **onmouseover**, **onresize**, …

```
<button onclick="buttonClickFunction()">Click Me!</button>
```

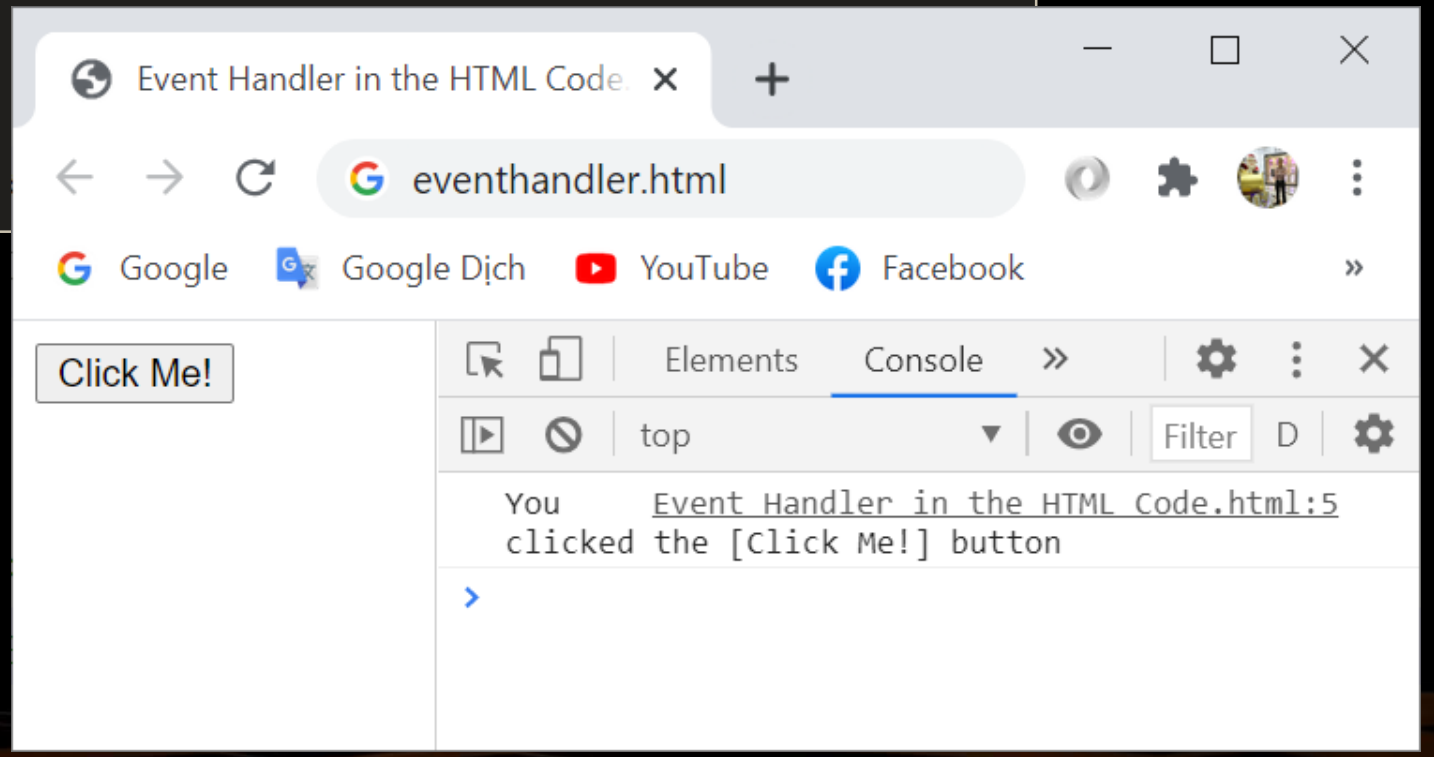
```
function buttonClickFunction() {
  console.log("You clicked the [Click Me!] button");
}
```

```
<button onclick="alert('OK clicked')">OK</button>
```

Event Handler in the HTML Code – Example (1)

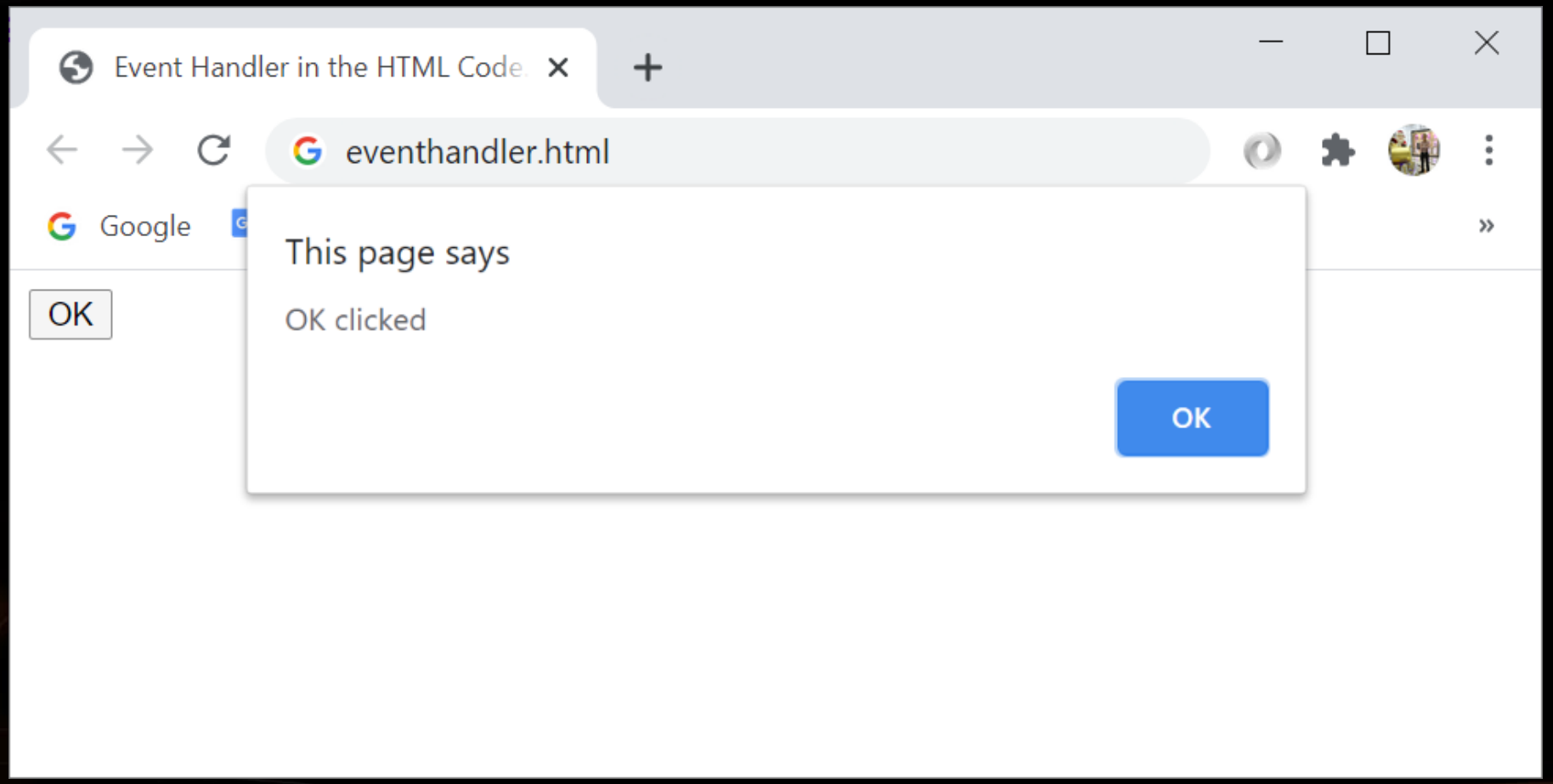
```
<button onclick="buttonClickFunction()">Click Me!</button>
```

```
<script> function buttonClickFunction() {
  console.log("You clicked the [Click Me!] button");
}
</script>
```



Event Handler in the HTML Code – Example (2)

```
<button onclick="alert('OK clicked')">OK</button>
```



Define Event Handler in the JS Code

- Event handling JavaScript code can be specified in the JS code through the properties **onclick**, **onresize**, …

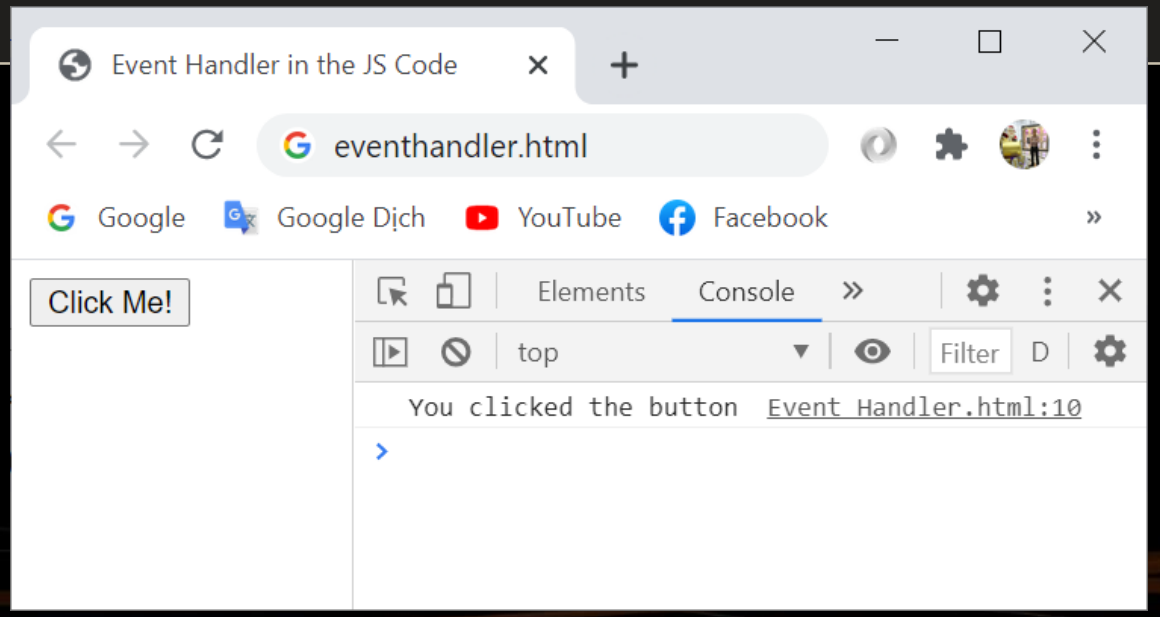
```
<button id="click-button">Click Me!</button>
```

```
var button = document.getElementById("click-button");
button.onclick = function onButtonClick() {
  console.log("You clicked the button");
}
```

Event Handler in the JS Code - Example

```
<button id="click-button">Click Me!</button>
```

```
<script>
var button = document.getElementById("click-button");
button.onclick = function onButtonClick() {
  console.log("You clicked the button");
} </script>
```



Using addEventListener(…)

- A more powerful way for attaching event handlers:

```
domElement.addEventListener(
  eventType, eventHandler, isCaptureEvent)
```

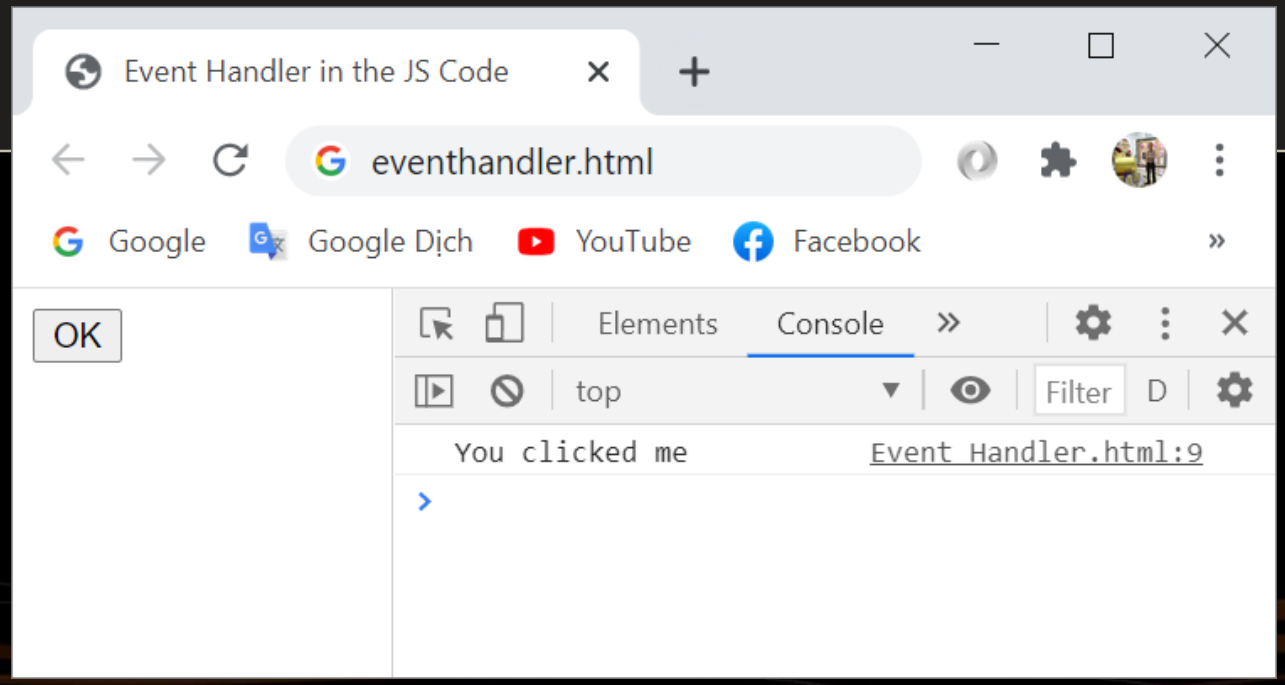
  - `isCaptureEvent`: catch the "capture" or "bubbling" phase
  - Can attach multiple events in a chain

```
var button = document.getElementById("buttonOK");
button.addEventListener("click", function() {
  console.log("You clicked me");
}, false);
```

Using addEventListener - Example

```
<button id="buttonOK">OK</button>
```

```
<script>
var button = document.getElementById("buttonOK");

button.addEventListener("click", function() {
  console.log("You clicked me");
}, false);
</script>
```



Just take it

The "event" Object

The "event" Object

- The "**event**" object holds information about the event
  - Passed as parameter to the event handling function

```
btn.onclick = function(event) { alert(event); }
```

- The **event** object contains information about:
  - The type of the event (e.g. 'click', 'resize', …)
  - The target of the event (e.g. the button clicked)
  - The key pressed for keyboard events
  - The mouse button / cursor position for mouse events

Event Object

- The "**event**" object is the only argument of the event handler

```
function onButtonClick(event) {
  console.log(event.target);
  console.log(event.type);
  console.log("(" + event.clientX + ", " + event.clientY + ")");
}
button.addEventListener("click", onButtonClick, false);
```

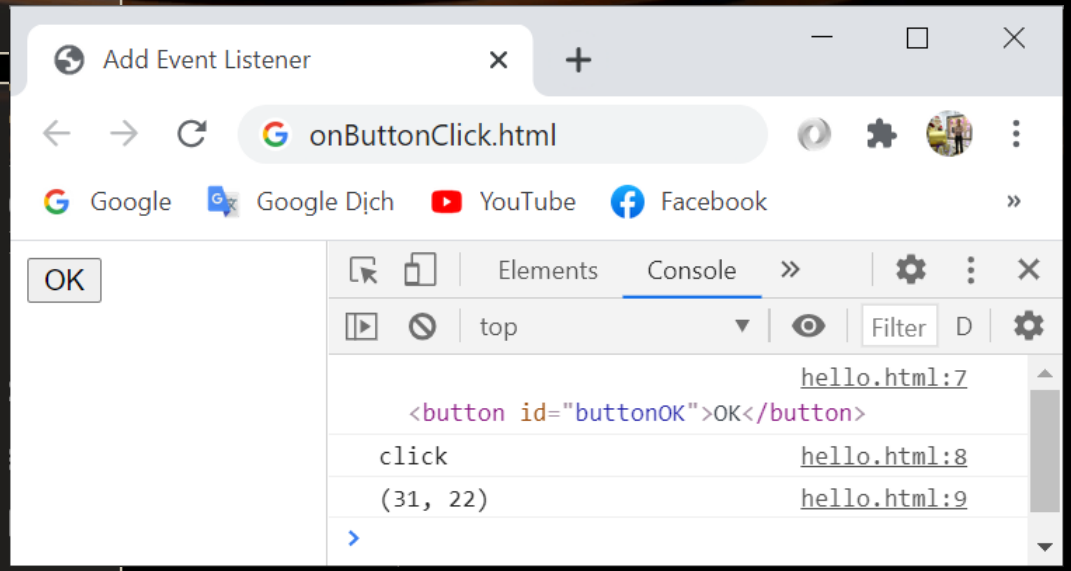
- Old IE versions pass the event in **window.event**

```
function onButtonClick(event) {
  if (!event) event = window.event;
  // Your event handling code …
}
```

Event Object - Example

```
<button id="buttonOK">OK</button>
```

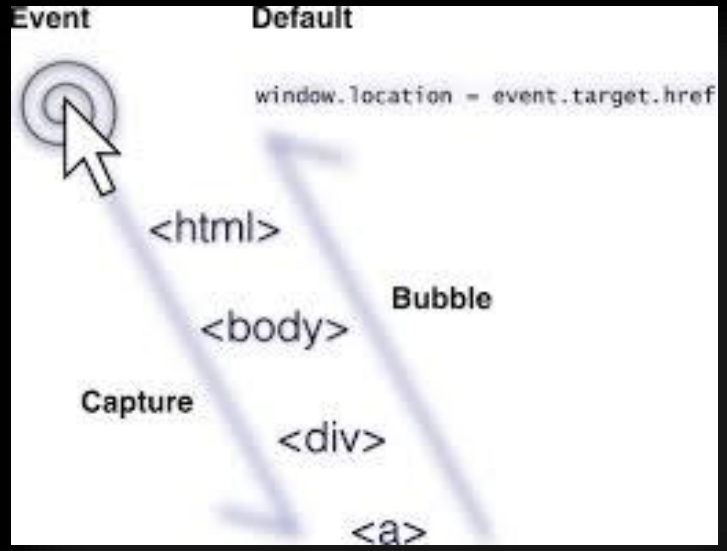
```
<script>
var button =
document.getElementById("buttonOK");
function onButtonClick(event) {
  console.log(event.target);
  console.log(event.type);
  console.log("(" + event.clientX + ", "
+ event.clientY + ")");
}
button.addEventListener("click",
onButtonClick, false);
</script>
```



Capturing and Bubbling Events

Browser Events Chain

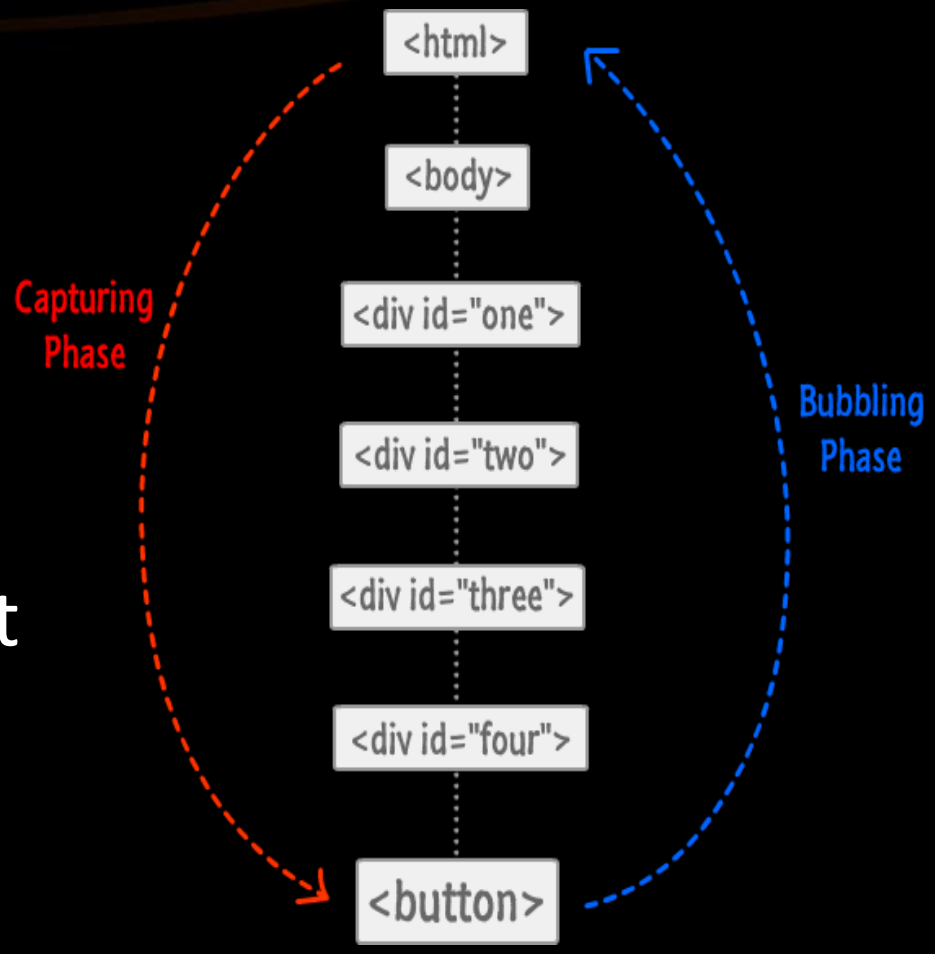
- When the user clicks on an HTML element
  - E.g. on a button in the page
  - The event is also fired on all of its parents
- The button is still the **`target`**
  - But the **`click`** event is fired on all of its parents
  - The **`click`** event is fired on all elements in the chain



Event Chains: Types

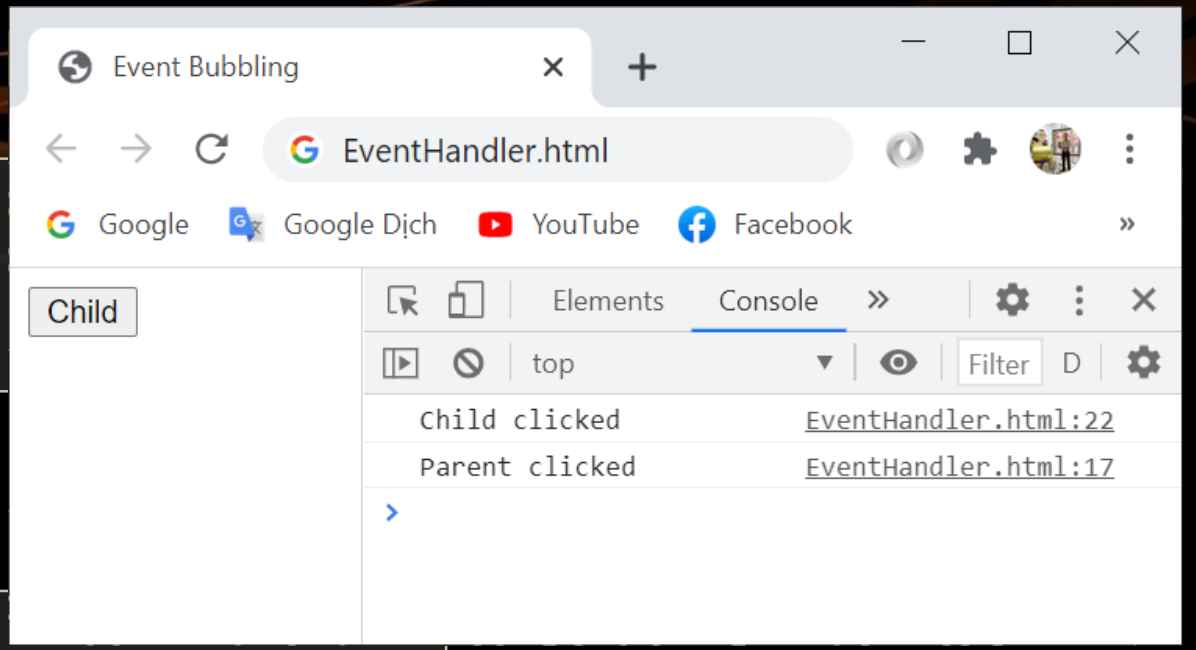
```
domElement.addEventListener(eventType,
  eventHandler, isCaptureEvent)
```

- Capturing handlers go down the chain
  - The first executed handler is on the parent
  - The last executed handler is on the target
- Bubbling handlers bubble up to the parent
  - The first executed handler is on the target
  - Then its parent's, and its parent's, etc.
- Capturing and Bubbling Explained



Event Chains: Bubbling

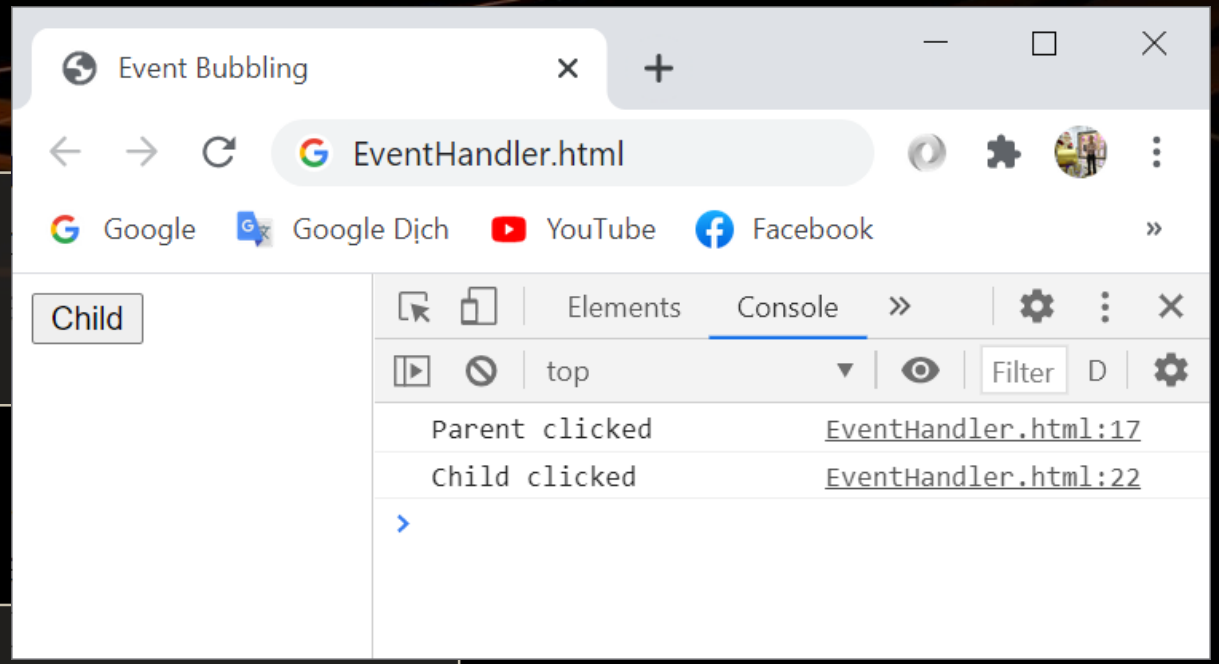
```
<div id="parent">
    <button id="child">Child</button>
</div>
```




```
<script>
var parent = document.querySelector('#parent');
    parent.addEventListener('click', function(){
        console.log("Parent clicked");
    });
    var child = document.querySelector('#child');
    child.addEventListener('click', function(){
        console.log("Child clicked");
    });
</script>
```

Event Chains: Capturing

```
<div id="parent">
    <button id="child">Child</button>
</div>
```




```
<script>
var parent = document.querySelector('#parent');
    parent.addEventListener('click', function(){
        console.log("Parent clicked");
    }, true);
    var child = document.querySelector('#child');
    child.addEventListener('click', function(){
        console.log("Child clicked");
    });
</script>
```

Common Event Types

- Mouse events

```
click
hover
mouseup
mousedown
mouseover
mouseout
```

- DOM / UI events

```
load
abort
select
resize
change
```

- Touch events

```
touchstart
touchend
touchcancel
touchleave
touchmove
```

- Keyboard events

```
keydown
keypress
keyup
```

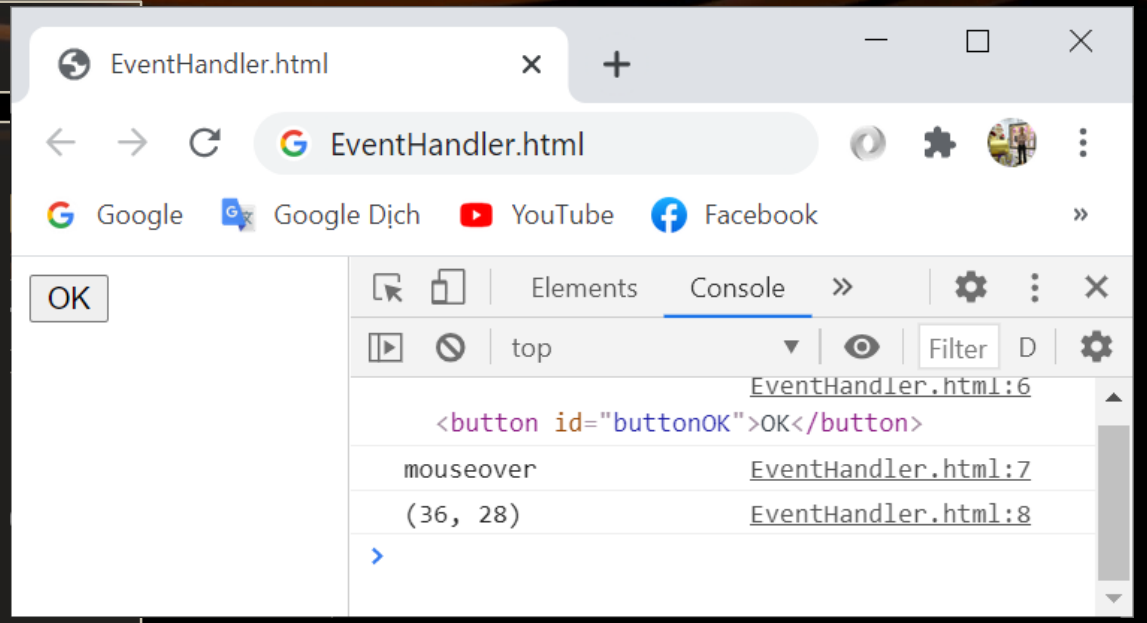
- Focus events

```
focus
blur
focusin
focusout
```

Common Event - Mouse events

```
<button id="buttonOK">OK</button>
```

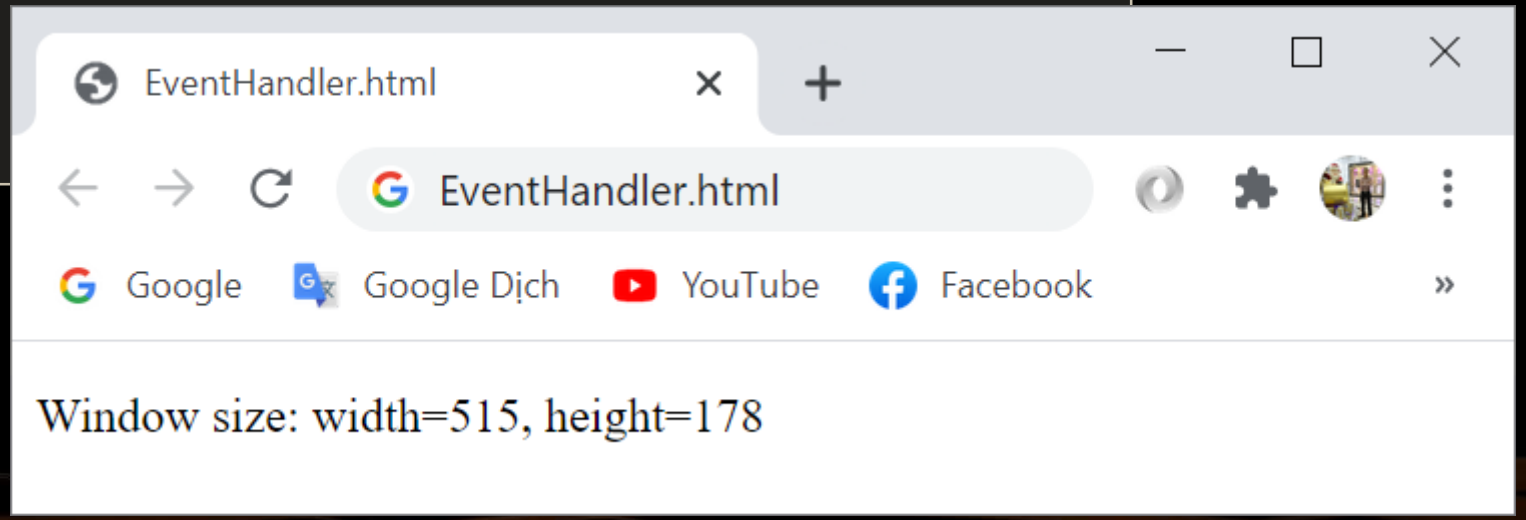
```
<script>
var button =
document.getElementById("buttonOK");
function onMouseOver(event) {
  console.log(event.target);
  console.log(event.type);
  console.log("(" + event.clientX + ", "
+ event.clientY + ")");
}
button.addEventListener("mouseover",
onMouseOver, false);
</script>
```



Common Event - DOM / UI events

```
<body onresize="myFunction()">
<p id="demo"></p>
```

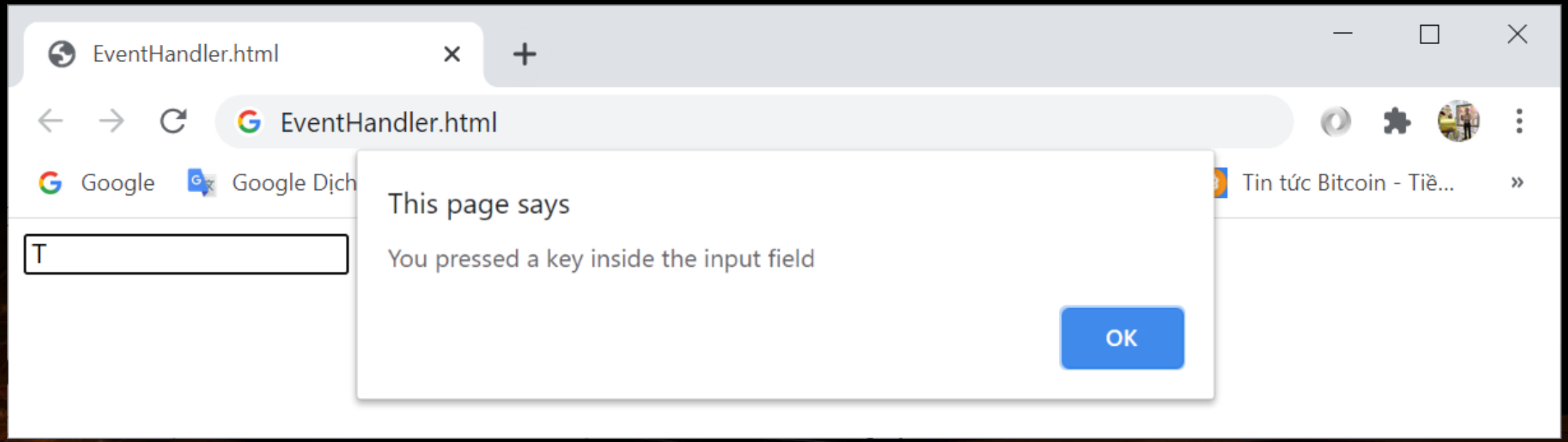
```
<script>
function myFunction() {
  var w = window.outerWidth;
  var h = window.outerHeight;
  var txt = "Window size: width=" + w + ", height=" + h;
document.getElementById("demo").innerHTML = txt;
}
</script>
</body>
```



Common Event - Keyboard events

```
<input type="text" onkeypress="myFunction()">
```

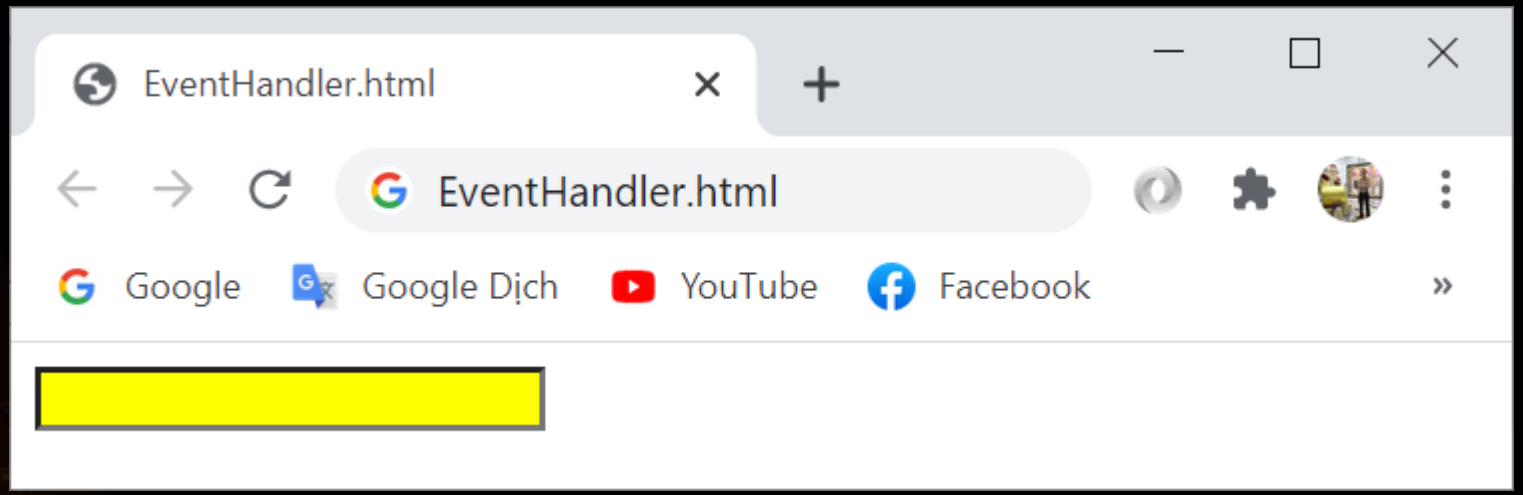
```
<script>
function myFunction() {
  alert("You pressed a key inside the input field");
}
</script>
```



Common Event - Focus events

```
<input type="text" onfocus="myFunction(this)">
```

```
<script>
function myFunction(x) {
  x.style.background = "yellow";
}
</script>
```



Summary

1. DOM
2. Selecting HTML Elements
3. Traversing the DOM
4. JavaScript Event Model
5. Event Handler Registration
6. The “event” object
7. Capturing and bubbling events
   - Event chaining
8. Common events

JavaScript DOM and Events

Questions?