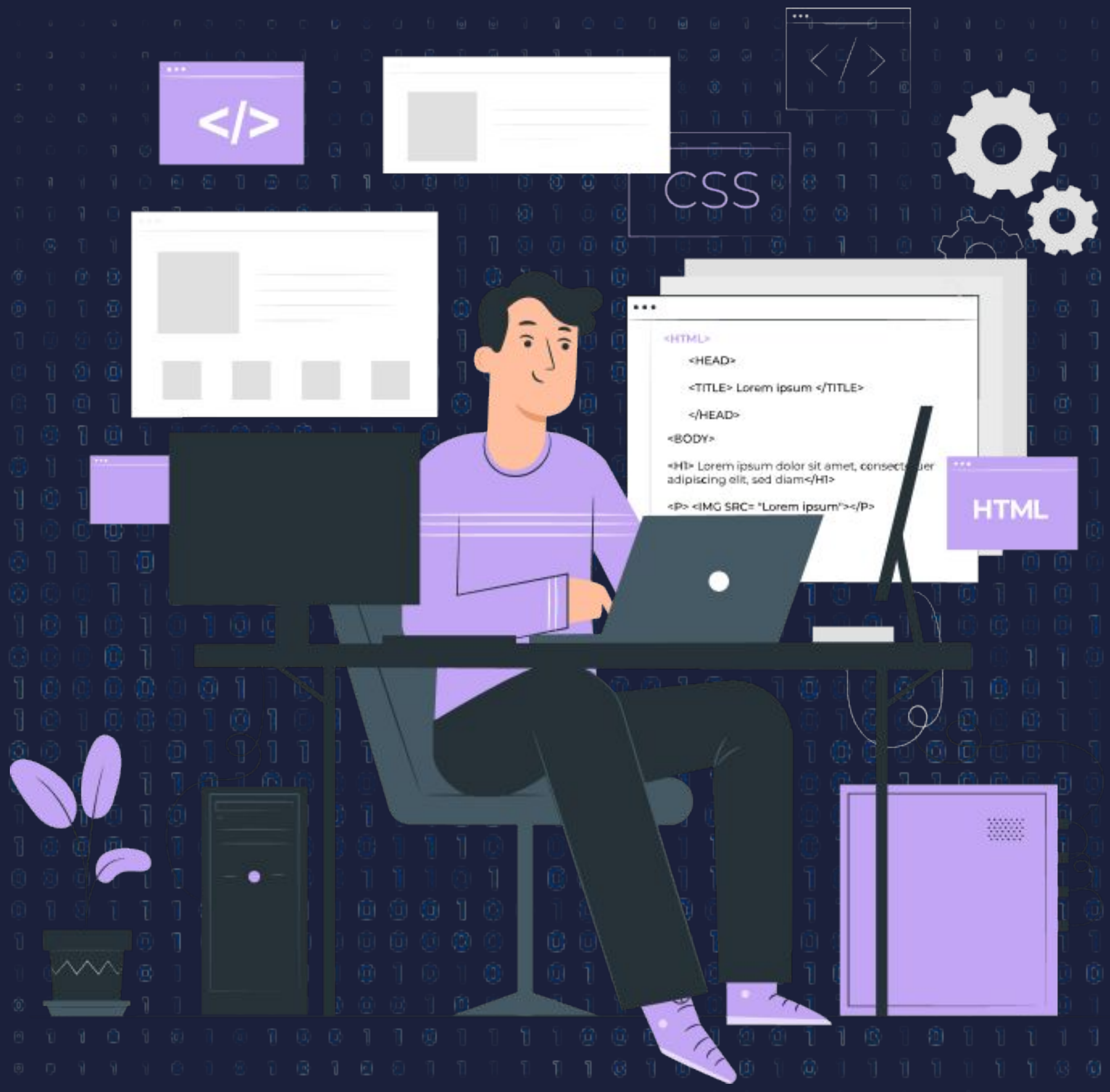
SKILLS

# Methods of DOM - Part 3



</>
CSS
<HTML>
<HEAD>
<TITLE> Lorem ipsum </TITLE>
</HEAD>
<BODY>
<H1> Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam</H1>
<P> <IMG SRC= "Lorem ipsum"></P>
HTML

## Topics

- createElement()
- appendChild()
- removeChild()
- classList
- Some useful properties
- Event Listener

## createElement()

This method takes a string as its argument, which specifies the type of element to be created.

**syntex:**

```
document.createElement('tagname')
```

**script.js**

```
const paragraph = document.getElementById('myParagraph');
paragraph.innerText = 'This is the new text.';
```

# appendChild()

The appendChild() method is a JavaScript DOM method that allows you to add a new child element to an existing parent element

**syntex:**

```
parentElement.appendChild(childElement);
```

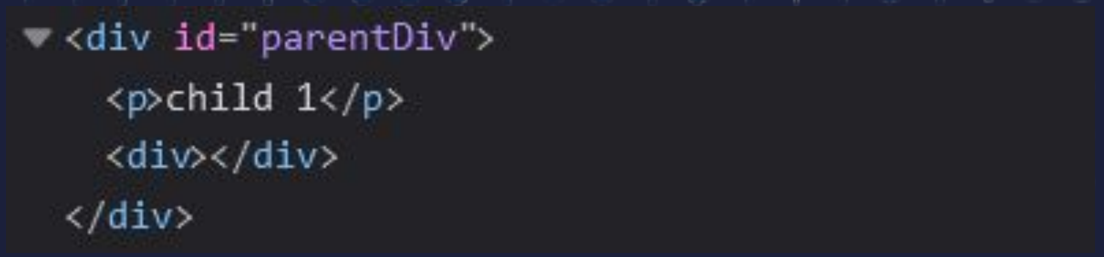
**index.html**

```
<div id="parentDiv">
<p>child 1<p>
</div>
```

**script.js**

```
const parentDiv = document.getElementById('#parentDiv');
const newDiv = document.createElement('div');
parentDiv.appendChild(newDiv);
```

**html**



```
▼<div id="parentDiv">
   <p>child 1</p>
   <div></div>
 </div>
```

# removeChild()

The **removeChild()** method in JavaScript is a DOM method that allows you to remove an existing child element from its parent element.

**syntex:**

```
parentElement.removeChild(childElement)
```

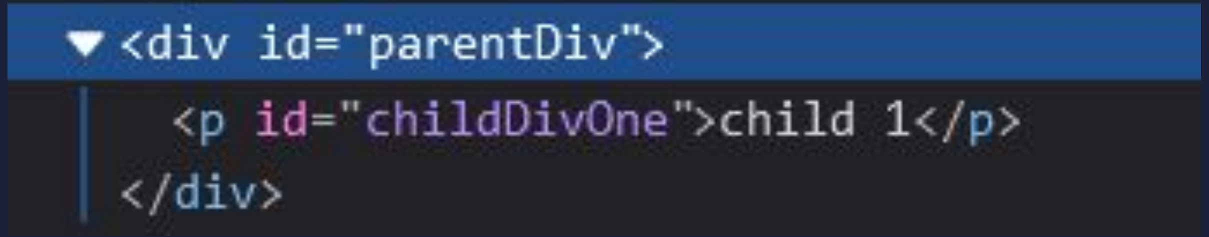
**index.html**

```
<div id="parentDiv">
<p id="childDiv">child 1<p>
</div>
```

**script.js**

```
const parentDiv = document.getElementById('parentDiv');
const childDiv = document.getElementById('childDiv');
parentDiv.removeChild(childDiv);
```

**HTML**



```
▼<div id="parentDiv">
   <p id="childDivOne">child 1</p>
 </div>
```

# classList

The **classList** property is a DOM property in JavaScript that allows you to **add, remove, toggle**, and check for the presence of classes on an element's list of classes.It returns a DOMTokenList object that represents the class attribute of an element as a space-separated list of strings.

**syntex:**

```
const elementClasses = element.classList;
```

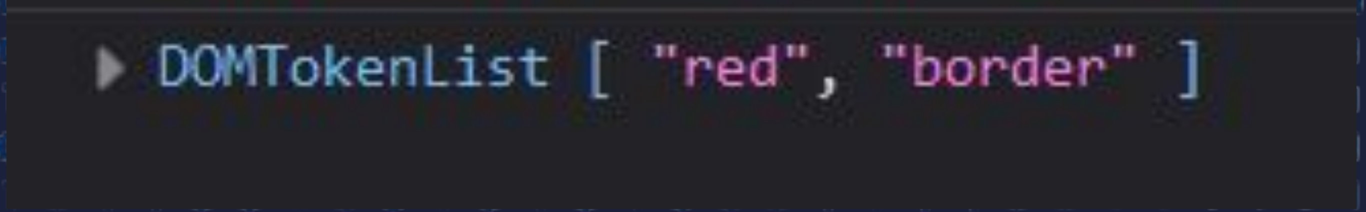
**index.html**

```
<div id="myDiv" class="red border">Some content
here</div>
```

**script.js**

```
const myDivClasses = myDiv.classList;
console.log(myDivClasses);
```

**console**



```
▶ DOMTokenList [ "red", "border" ]
```

# classList.add()

The **classList.add()** method is a JavaScript method that allows you to **add one or more classes** to an element's list of classes.

**syntex:**

```
element.classList.add(className1, className2, ...);
```

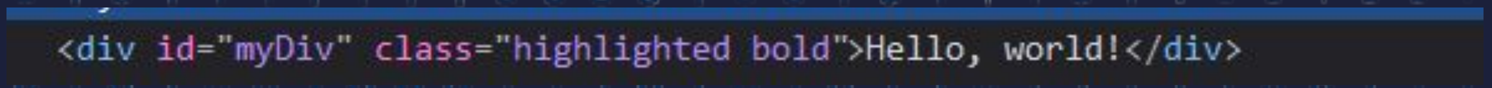
**index.html**

```
<div id="myDiv">Hello, world!</div>
```

**script.js**

```
const element = document.getElementById('myDiv');

// Adding the classes 'highlighted' and 'bold' to the
element
element.classList.add('highlighted', 'bold');
```

**HTML**



```
<div id="myDiv" class="highlighted bold">Hello, world!</div>
```

# classList.remove()

The **classList.remove()** method is a JavaScript method that allows you to **remove one or more classes** from an element's list of classes.

**syntex:**

```
element.classList.remove(className, className, ...);
```

**index.html**

```
<div id="myDiv">Hello, world!</div>
```

**script.js**

```
const element = document.getElementById('myDiv');

// Adding the classes 'highlighted' and 'bold' to the
element
element.classList.add('highlighted', 'bold');
```

# classList.toggle()

The **classList.toggle()** method allows you to toggle the presence of a class on an element.It adds the specified class to the element if it's not present, and removes it if it's already present.

**syntex:**

```
element.classList.toggle(className, force);
```

**index.html**

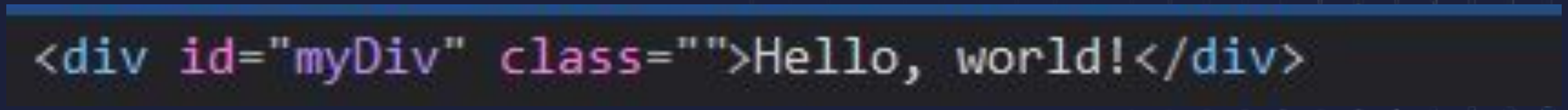
```
<div id="myDiv" class="highlighted">Hello, world!</div>
```

**script.js**

```
const element = document.getElementById('myDiv');
// Toggling the class 'highlighted' on the element
element.classList.toggle('highlighted');
```

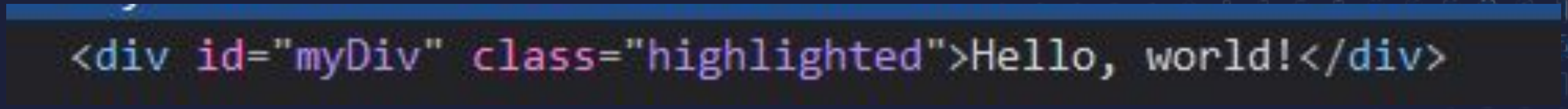
# classList.toggle()

The highlight class is initially present on <div> element, when **"classList.toggle("highlight")"** is called, the class is removed from the element:



```
<div id="myDiv" class="">Hello, world!</div>
```

And if we call **"classList.toggle("highlight")"** again without providing the force parameter, the class "highlight" will be added back to the element.



```
<div id="myDiv" class="highlighted">Hello, world!</div>
```

**classList.toggle() second parameter**

```
// The class 'highlighted' is forcefully added
element.classList.toggle('highlighted', true);

// The class 'highlighted' is forcefully removed
element.classList.toggle('highlighted', false);
```

## classList.contains()

classList.contains() returns a Boolean value indicating whether the element has the specified class in its list of classes.

**syntex:**

```
element.classList.contains(className);
```

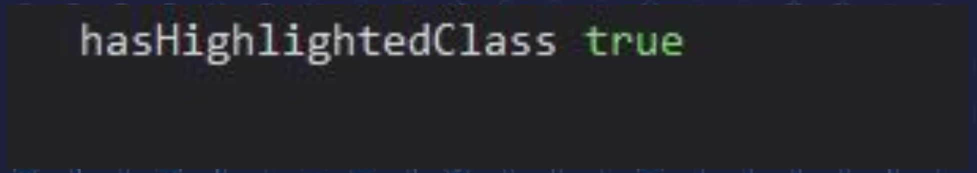
**index.html**

```
<div id="myDiv" class="highlighted">Hello, world!</div>
```

**script.js**

```
const element = document.getElementById('myDiv');

// Checking if the element has the class 'highlighted'
const hasHighlightedClass =
element.classList.contains('highlighted');
console.log("hasHighlightedClass", hasHighlightedClass);
```

**console**



```
hasHighlightedClass true
```

## Some useful properties

- parentNode
- previousSibling
- nextSibling
- firstChild
- lastChild

## parentNode

The **parentNode** property returns the parent node of the current node. If the current node does not have a parent, then parentNode will return **null**.

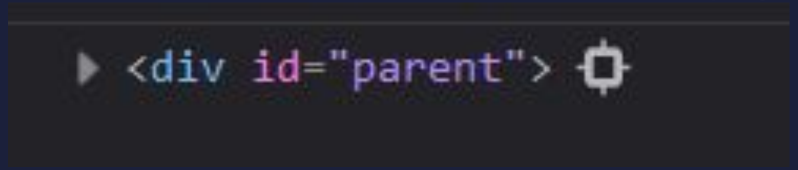
**index.html**

```html
<div id="parent">
  <p>This is a child element.</p>
</div>
```

**script.js**

```javascript
const childElement = document.querySelector('p');
const parentNode = childElement.parentNode;

console.log(parentNode);
```

**console**



```
▶ <div id="parent">
```

# previousSibling

The **previousSibling** property is used to access the previous sibling node of a specific DOM node. It returns the node immediately preceding the specified node within its parent's list of child nodes.

**Example :** series of elements directly adjacent to each other, with **no whitespace** between them

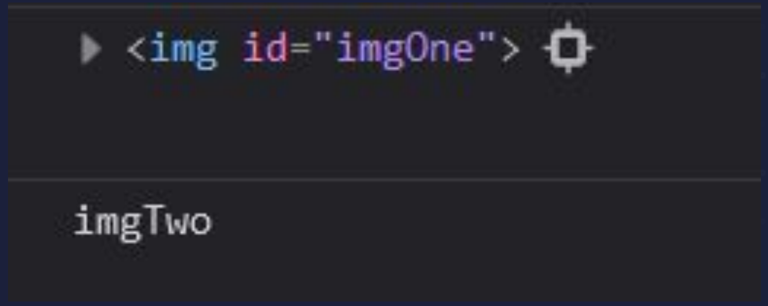
**index.html**

```html
<img id="imgOne" /><img id="imgTwo"
/><img id="imgThree" />
```

**script.js**

```javascript
const imgTwo =
document.getElementById("imgTwo").previousSibling;

const imgThreeId =
document.getElementById("imgThree").previousSibling.id;

console.log(imgTwo)
console.log(imgThreeId)
```

**console**



```
▸ <img id="imgOne">

imgTwo
```

# previousSibling

**Example**: whitespace text nodes (line breaks) between the elements.

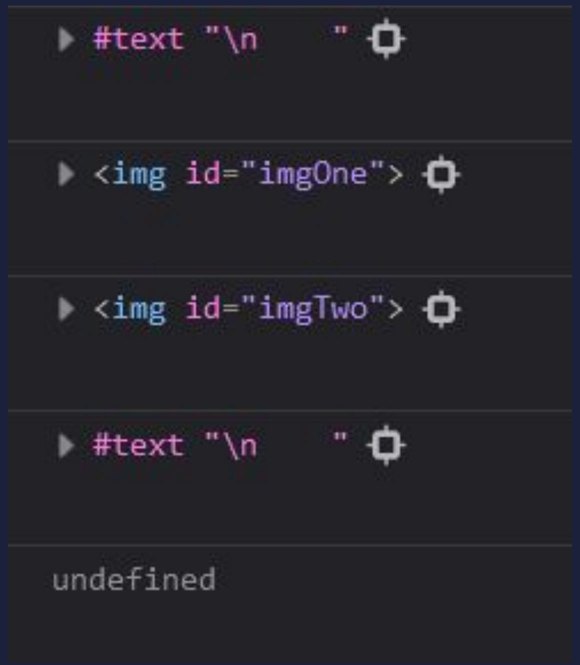
**index.html**

```html
<img id="imgOne"/>
<img id="imgTwo"/>
<img id="imgThree"/>
```

**script.js**

```javascript
const imgTwo = document.getElementById("imgTwo");
const imgThree = document.getElementById("imgThree");

imgTwo.previousSibling;
imgThree.previousSibling

console.log(imgTwo.previousSibling);
console.log(imgTwo.previousSibling.previousSibling);
console.log(imgThree.previousSibling.previousSibling);
console.log(imgThree.previousSibling);
console.log(imgThree.previousSibling.id);
```

**console**



```
▸ #text "\n    "
▸ <img id="imgOne">
▸ <img id="imgTwo">
▸ #text "\n    "
undefined
```

## nextSibling

**nextSibling** property is used to access the next sibling node of a specific DOM node. It returns the node immediately following the specified node within its parent's list of child nodes.

**Example:** series of elements directly adjacent to each other, with **no whitespace** between them

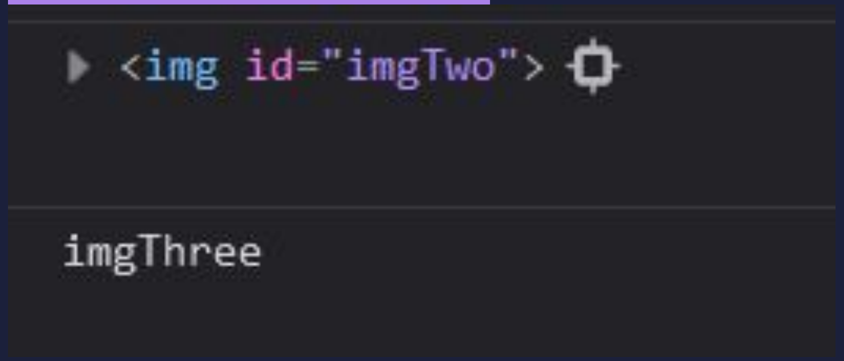
**index.html**

```html
<img id="imgOne" /><img id="imgTwo"
/><img id="imgThree" />
```

**script.js**

```javascript
const imgOne = document.getElementById("imgOne").nextSibling;
const imgTwo =
document.getElementById("imgTwo").nextSibling.id;

console.log(imgOne)
console.log(imgTwo)
```

**console**



```
▶ <img id="imgTwo">

imgThree
```

## nextSibling

**Example:** whitespace text nodes (line breaks) between the elements.

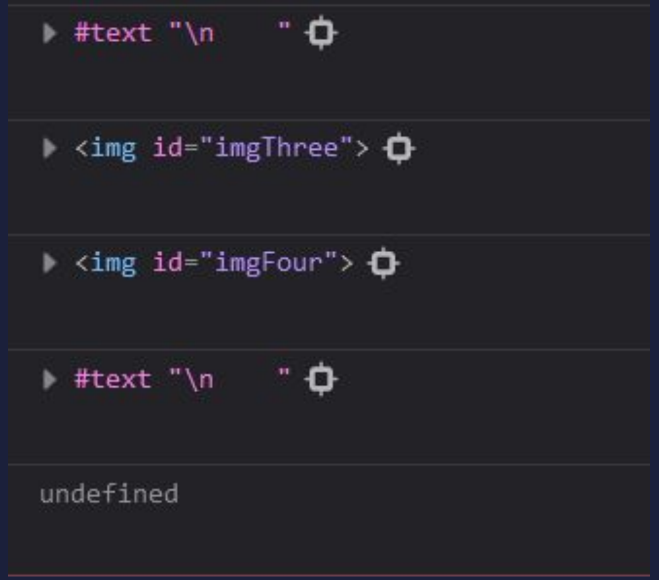
**index.html**

```html
<img id="imgOne" />
<img id="imgTwo" />
<img id="imgThree" />
<img id="imgFour" />
```

**script.js**

```javascript
const imgTwo = document.getElementById("imgTwo");
const imgThree = document.getElementById("imgThree");

console.log(imgTwo.nextSibling);
console.log(imgTwo.nextSibling.nextSibling);
console.log(imgThree.nextSibling.nextSibling);
console.log(imgThree.nextSibling);
console.log(imgThree.nextSibling.id);
```

**console**



```
▸ #text "\n    "
▸ <img id="imgThree">
▸ <img id="imgFour">
▸ #text "\n    "
undefined
```

## firstChild

firstChild property of the Node interface returns the node's first child in the tree, or null if the node has no children.

**Example: no whitespace** between parent and child

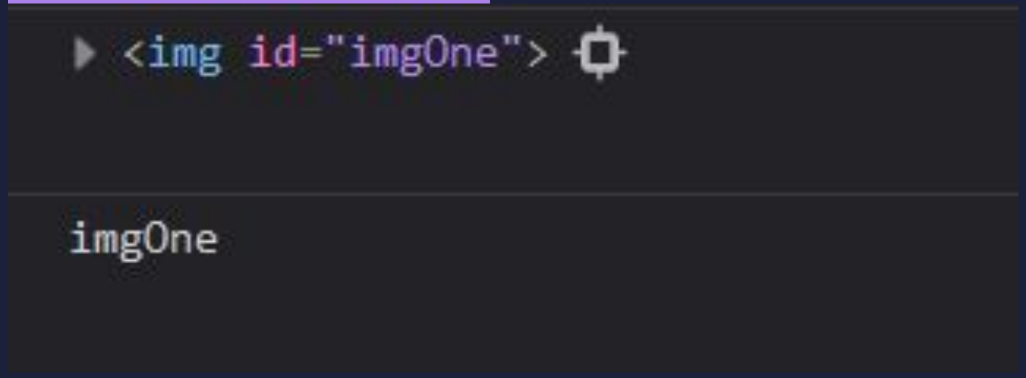
**index.html**

```
<div id="imgContainer"><img
id="imgOne" /><img id="imgTwo" /><img
id="imgThree" /></div>
```

**script.js**

```
const firstChild =
document.getElementById("imgContainer").firstChild;
const firstChildId =
document.getElementById("imgContainer").firstChild.id;
console.log(firstChild)
console.log(firstChildId)
```

**console**



```
▸ <img id="imgOne">

imgOne
```

# firstChild

**Example:** whitespace text nodes, comments, and text between the parent and child.

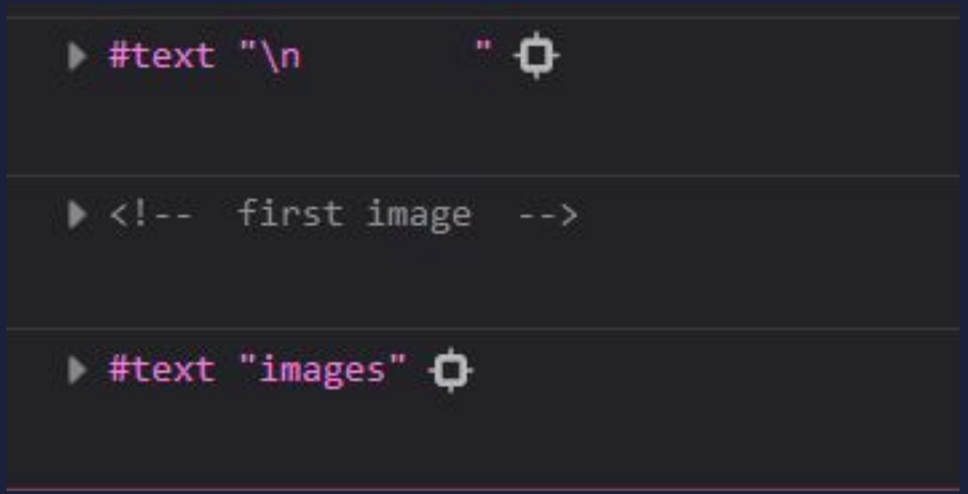
**index.html**

```
<div id="imgContainerOne">
      <img id="imgOne" />
  </div>
<div id="imgContainerTwo"><!-- first
image --><img id="imgOne" /></div>
<div
id="imgContainerThree">images<img
id="imgOne" /></div>
```

**script.js**

```
const firstChildContainerOne =
document.getElementById("imgContainerOne").firstChild;
const firstChildContainerTwo =
document.getElementById("imgContainerTwo").firstChild;
const firstChildContainerThree =
document.getElementById("imgContainerThree").firstChild;

console.log(firstChildContainerOne)
console.log(firstChildContainerTwo)
console.log(firstChildContainerThree)
```

**console**



```
▶ #text "\n        "

▶ <!-- first image -->

▶ #text "images"
```

# lastChild

**lastChild** property of the Node interface returns the node's last child in the tree, or null if the node has no children.

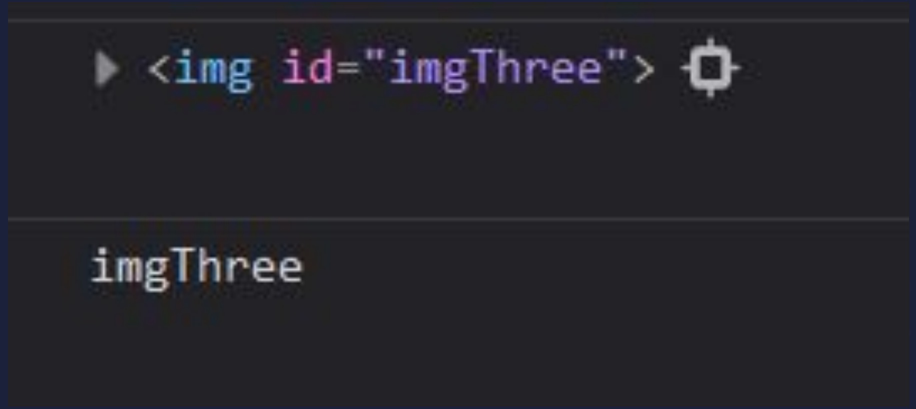
**index.html**

```
<div id="imgContainer"><img
id="imgOne" /><img id="imgTwo" /><img
id="imgThree" /></div>
```

**script.js**

```
const firstChild =
document.getElementById("imgContainer").firstChild;
const firstChildId =
document.getElementById("imgContainer").firstChild.id;
console.log(firstChild)
console.log(firstChildId)
```

**console**



```
▸ <img id="imgThree">

imgThree
```

## lastChild

**Example:** whitespace text nodes, comments, and text between the parent and child.

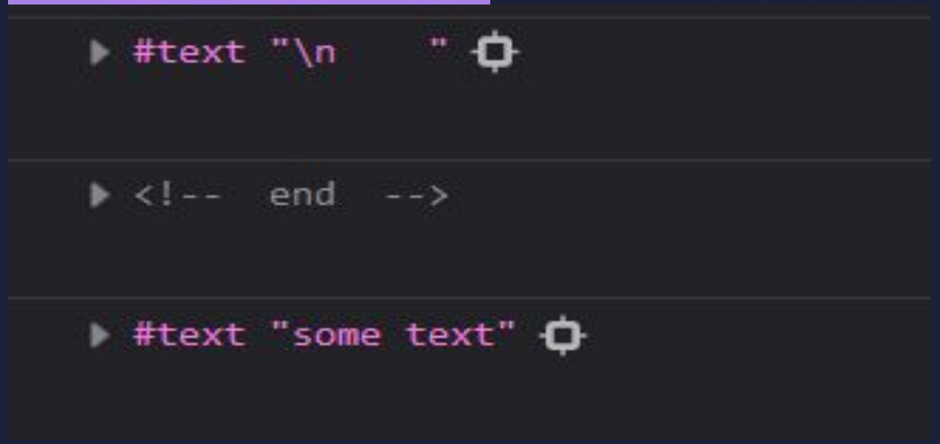
**index.html**

```html
<div id="imgContainerOne">
       <img id="imgOne" />
       <img id="imgTwo" />
</div>
<div id="imgContainerTwo"><!-- start --><img
id="imgOne" /><!-- end --></div>
<div id="imgContainerThree">image1<img
id="imgOne" />image2<img id="imgTwo" />some
text</div>
```

**script.js**

```javascript
const lastChildContainerOne =
document.getElementById("imgContainerOne").lastChild;
const lastChildContainerTwo =
document.getElementById("imgContainerTwo").lastChild;
const lastChildContainerThree =
document.getElementById("imgContainerThree").lastChild;

console.log(lastChildContainerOne)
console.log(lastChildContainerTwo)
console.log(lastChildContainerThree)
```

**console**



```
▶ #text "\n      "

▶ <!-- end -->

▶ #text "some text"
```

## Event Listener

- An event is an action that occurs on the web page, such as a user clicking a button, scrolling the page, or typing on the keyboard.
- Event listeners are functions that are called when a specific event occurs on an element.
- You can use the addEventListener method to attach event listeners to elements.

**syntex:**

```
element.addEventListener(eventType,
listenerFunction, useCapture);
```

# Event Listener

- **element** is the HTML element to which the event listener will be attached.
- **eventType** is the type of event to listen for. For example, click, keydown, or scroll.
- **listenerFunction** is the function that will be executed when the event occurs.
- **useCapture** is a boolean value that specifies whether the event listener should be attached in the capturing phase or the bubbling phase.

**addEventListener example**

```
const myButton = document.querySelector('#my-button');

myButton.addEventListener('click', function() {
  alert('Button clicked!');
});
```

## Remove Event listener

```
const myButton = document.querySelector('#my-button');

function onClick() {
  alert('Button clicked!');
}

myButton.addEventListener('click', onClick);

// Remove the event listener
myButton.removeEventListener('click', onClick);
```

## Event types

- **Mouse events:** These events are triggered by the user interacting with the mouse.
- **Keyboard events:** These events are triggered by the user interacting with the keyboard.
- **Form events:** These events are triggered by user actions within a form element.
- **Document events:** These events are triggered by changes to the document or the window.
- **Touch events:** These events are triggered by touch screens and mobile devices
- **Drag and drop events:** These events are triggered by dragging and dropping elements on the page.

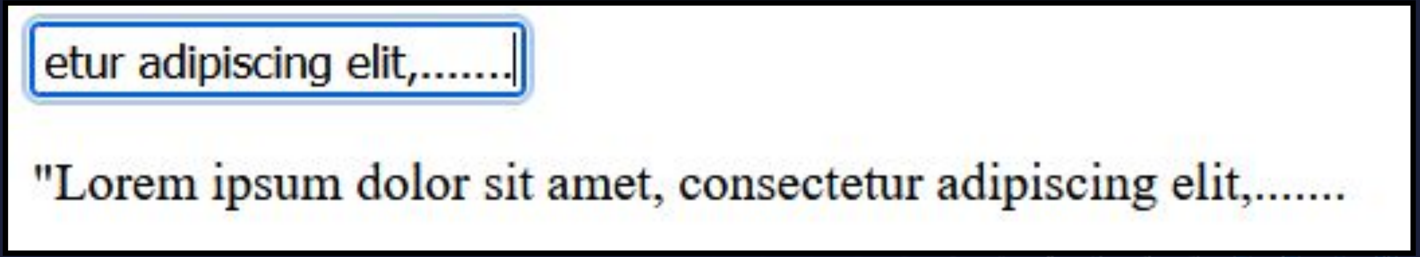
## Example

The input event can be used to detect changes in input fields. This can be useful for creating more complex applications that need to keep track of the changes that users make to input fields.

**index.html**

```
<input type="text" id="myInput">
<p id="output"></p>
```

**browser**



etur adipiscing elit,.......

"Lorem ipsum dolor sit amet, consectetur adipiscing elit,.......

**script.js**

```
// addEventListener() is used to add an event listener to an element
document.getElementById("myInput").addEventListener("input", function(event) {
// The value property of the event object contains the new value of the input field
const value = event.target.value;
// The output element is updated with the new value of the input field
document.getElementById("output").innerHTML = value;
});
```

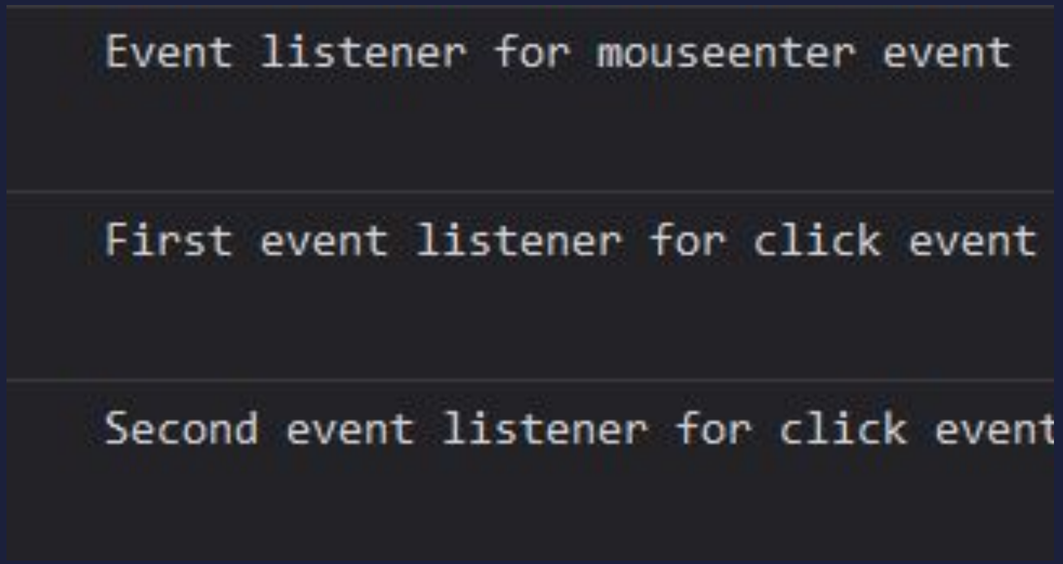
## Adding multiple events on same element

You can add multiple event listeners to a single element. Each event listener will be triggered when the event it is associated with occurs on the element.

**index.html**

```html
<button id="button">click here</button>
```

**console**



```
Event listener for mouseenter event

First event listener for click event

Second event listener for click event
```

**script.js**

```javascript
const buttonEle = document.getElementById("button")

// First event listener for 'click' event
buttonEle.addEventListener("click", function() {
   console.log('First event listener for click event');
})
// Second event listener for 'click' event
buttonEle.addEventListener("click", function() {
   console.log('Second event listener for click event');
})
// Event listener for 'mouseenter' event
buttonEle.addEventListener("mouseenter", function() {
   console.log('Event listener for mouseenter event');
})
```

## Event bubbling

- Event bubbling is a mechanism in JavaScript where an event triggered on a specific element propagates or "bubbles" up through its parent elements in the DOM tree.
- This means that when an event occurs on an element, such as a click event, it will not only trigger event listeners on that element but also trigger event listeners on its ancestor elements.
- The event is first handled by the element on which it occurred.
- If the element does not handle the event, it is passed to its parent element.
- This process continues until the event reaches the top-level document object.
- Each element has the opportunity to handle the event or pass it on to its parent.

## Event bubbling example

index.html

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

console

```
Button clicked

Child clicked

Parent clicked
```

script.js

```javascript
const parent = document.getElementById('parent');
const child = document.getElementById('child');
const button = document.getElementById('button');

parent.addEventListener('click', function() {
  console.log('Parent clicked');
});

child.addEventListener('click', function() {
  console.log('Child clicked');
});

button.addEventListener('click', function() {
  console.log('Button clicked');
});
```

## Event capturing

- Event capturing is the opposite of event bubbling in the event propagation process within the Document Object Model (DOM) in JavaScript.
- While event bubbling starts from the target element and moves up the DOM tree, event capturing starts from the top-level element (usually the document) and moves down to the target element.
- In event capturing, the event first triggers the event listeners attached to the highest-level ancestor element and then propagates down the DOM tree until it reaches the target element. Once the event reaches the target element, it triggers the event listener attached to that element.
- To specify event capturing instead of event bubbling, you can use the addEventListener() method with the third parameter set to true. By default, the third parameter is false, which represents event bubbling.

## Event capturing example

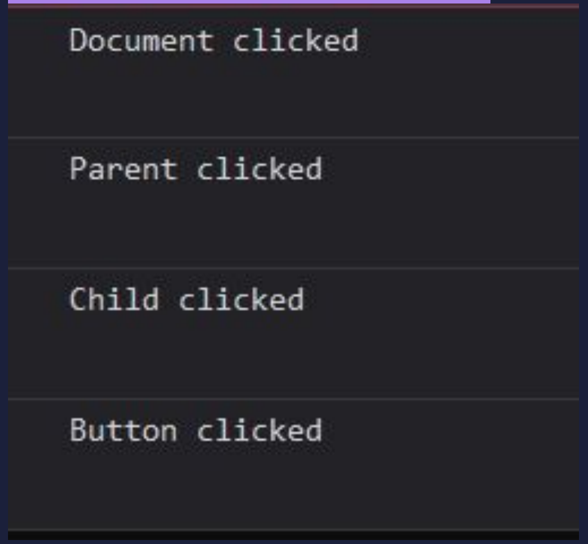
index.html

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

script.js

```javascript
const parent = document.getElementById('parent');
const child = document.getElementById('child');
const button = document.getElementById('button');

document.addEventListener('click', function() {
  console.log('Document clicked');
}, true);

parent.addEventListener('click', function() {
  console.log('Parent clicked');
}, true);

child.addEventListener('click', function() {
  console.log('Child clicked');
}, true);

button.addEventListener('click', function() {
  console.log('Button clicked');
}, true);
```

console



```
Document clicked

Parent clicked

Child clicked

Button clicked
```

## Event Methods

There are some very useful methods available for working with events.

- event.preventDefault()
- event.stopPropagation()
- event.stopImmediatePropagation()

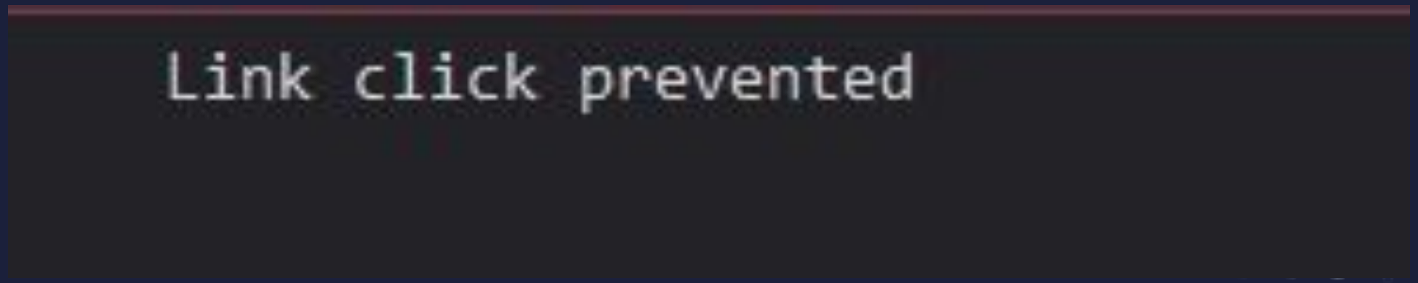
## preventDefault()

- preventDefault() is a method used in JavaScript to prevent the default action associated with an event from occurring.
- The default action is the action that would normally happen when an event occurs, such as submitting a form or following a link.
- By calling preventDefault(), you can prevent the default action from happening and perform your own custom action instead.

**index.html**

```html
<a href="https://example.com" id="myLink">Click
me</a>
```

**console**



```
Link click prevented
```

**script.js**

```javascript
const link = document.getElementById('myLink');

link.addEventListener('click', function(event) {
  event.preventDefault();
  console.log('Link click prevented');
});
```

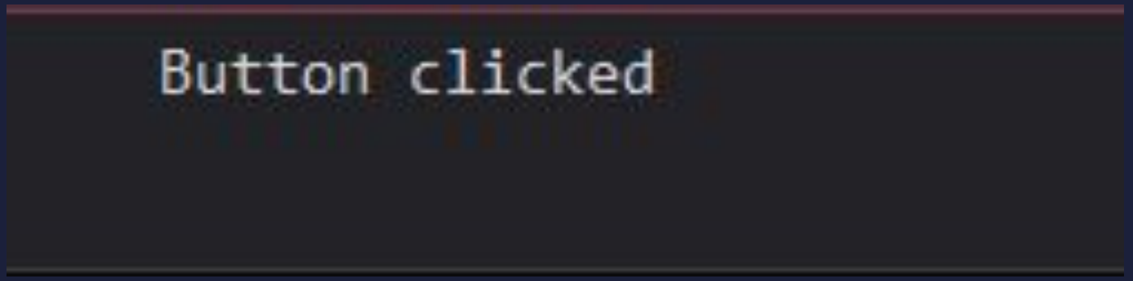
# stopPropagation()

The **stopPropagation()** method allows you to stop this event propagation. When called within an event listener, it prevents the event from triggering listeners on parent elements.

**index.html**

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

**console**



```
Button clicked
```

**script.js**

```javascript
const parent = document.getElementById('parent');
const child = document.getElementById('child');
const button = document.getElementById('button');

parent.addEventListener('click', function() {
  console.log('Parent clicked');
});

child.addEventListener('click', function() {
  console.log('Child clicked');
});

button.addEventListener('click', function(event) {
   // Stop the event from propagating further
  event.stopPropagation();
  console.log('Button clicked');
});
```

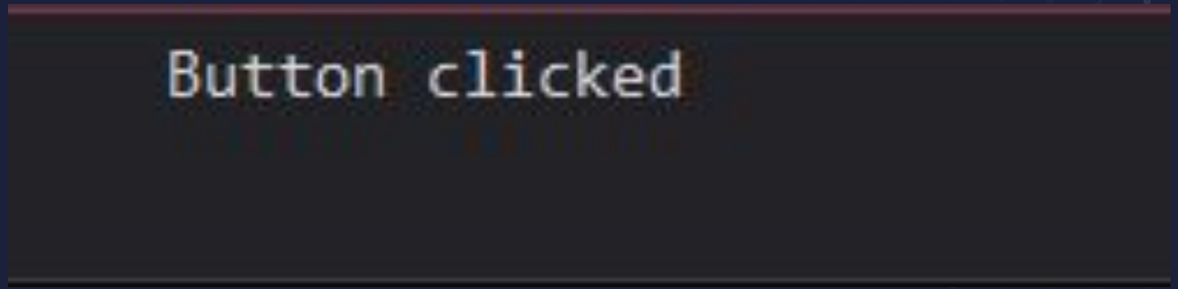
# stopImmediatePropagation()

**stopImmediatePropogation()** not only stops the propagation to parent elements but also prevents any other event listeners on the same element from being executed.

**index.html**

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

**console**



```
Button clicked
```

**script.js**

```javascript
const parent = document.getElementById('parent');
const child = document.getElementById('child');
const button = document.getElementById('button');

parent.addEventListener('click', function() {
  console.log('Parent clicked');
});

child.addEventListener('click', function() {
  console.log('Child clicked');
});

button.addEventListener('click', function(event) {
  // Stop further execution of event listeners on the same element
  event.stopImmediatePropagation();
  console.log('Button clicked');
});

button.addEventListener('click', function(event) {
  console.log("event attached on same element');
});
```

## Event Delegation

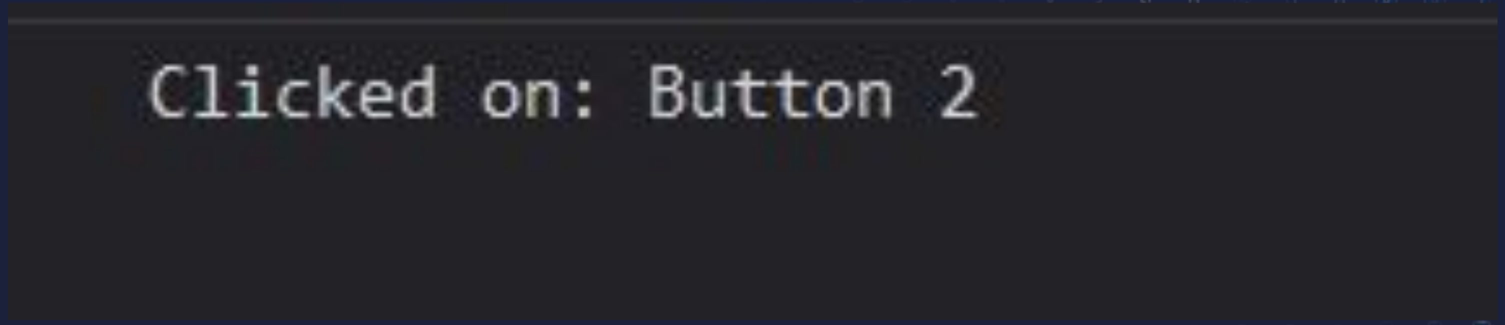
- Event delegation is a technique for handling events that occur on a large number of elements in a document.
- It works by attaching an event listener to a parent element, and then using the event.target property to determine which child element actually triggered the event.
- This can be useful for performance reasons, as it can reduce the number of event listeners that need to be attached to the document. It can also be useful for code organization, as it can make it easier to manage event-handling code

## Event delegation example

**index.html**

```html
<div id="buttonContainer">
  <button class="button">Button 1</button>
  <button class="button">Button 2</button>
  <button class="button">Button 3</button>
</div>
```

**console**



```
Clicked on: Button 2
```

**script.js**

```javascript
const parentList =
document.getElementById('parentList');

parentList.addEventListener('click',
function(event) {
  if (event.target.tagName === 'LI') {
    console.log('Clicked on:',
event.target.textContent);
  }
});
```

SKILLS

# THANK YOU