## What is Document Object Model (DOM)

The Document Object Model (DOM) is an application programming interface (API) for manipulating HTML documents.

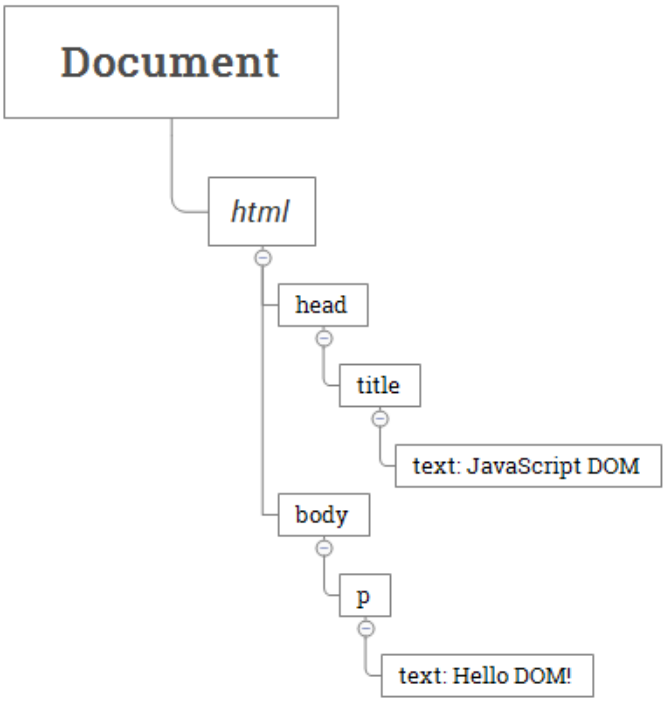
The DOM represents an HTML document as a tree of nodes. The DOM provides functions that allow you to add, remove, and modify parts of the document effectively.

Note that the DOM is cross-platform and language-independent ways of manipulating HTML and XML documents.

### A document as a hierarchy of nodes

The DOM represents an HTML document as a hierarchy of nodes. Consider the following HTML document:

```
<html>
    <head>
        <title>JavaScript DOM</title>
    </head>
    <body>
        <p>Hello DOM!</p>
    </body>
</html>
```




In this DOM tree, the document is the root node. The root node has one

child node which is the <html> element. The <html> element is called the *document element*.

Each document can have only one document element. In an HTML document, the document element is the <html> element. Each markup can be represented by a node in the tree.

## Node Types

Each node in the DOM tree is identified by a node type. JavaScript uses integer numbers to determine the node types. The following table illustrates the node type constants:

| Constant | Value | Description |
|---|---|---|
| `Node.ELEMENT_NODE` | 1 | An `Element` node like <p> or . |
| `Node.TEXT_NODE` | 3 | The actual `Text` inside an `Element` or `Attr`. |
| `Node.CDATA_SECTION_NODE` | 4 | A `CDATASection`, such as `<!CDATA[[ … ]]>`. |
| `Node.PROCESSING_INSTRUCTION_NODE` | 7 | A `ProcessingInstruction` of an XML document, such as `<?xml-stylesheet … ?>`. |
| `Node.COMMENT_NODE` | 8 | A `Comment` node, such as `<!-- … -->`. |
| `Node.DOCUMENT_NODE` | 9 | A `Document` node. |
| `Node.DOCUMENT_TYPE_NODE` | 10 | A `DocumentType` node, such as `<!DOCTYPE html>`. |
| `Node.DOCUMENT_FRAGMENT_NODE` | 11 | A `DocumentFragment` node. |

### The nodeName and nodeValue properties

A node has two important properties: nodeName and nodeValue that provide specific information about the node.

The values of these properties depend on the node type. For example, if the node type is the element node, the nodeName is always the same as the element's tag name and nodeValue is always null.

## Node and Element

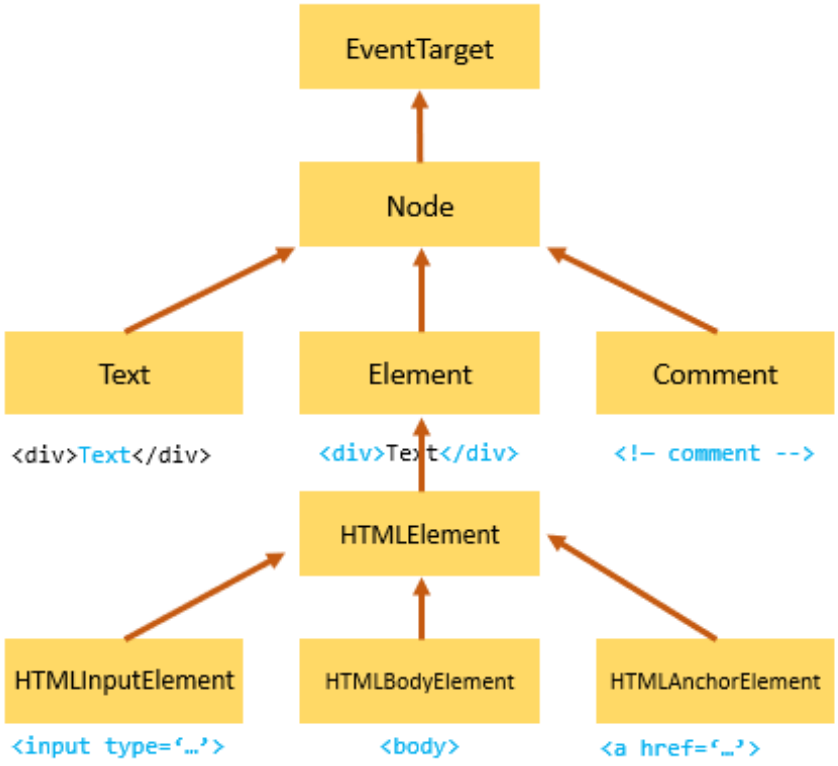
Sometimes it's easy to confuse between the Node and the Element.

A node is a generic name of any object in the DOM tree. It can be any built-in DOM element such as the document. Or it can be any HTML tag specified in the HTML document like  or <p>.

An element is a node with a specific node type Node.ELEMENT_NODE, which is equal to 1.

In other words, the node is the generic type of element. The element is a specific type of the node with the node type Node.ELEMENT_NODE.

The following picture illustrates the relationship between the Node and Element types:




Note that the getElementById() and querySelector() returns an object with the Element type while getElementsByTagName() or querySelectorAll() returns NodeList which is a collection of nodes.

Introduction to JavaScript getElementById() method

The document.getElementById() method returns an Element object that represents an HTML element with an id that matches a specified string.

If the document has no element with the specified id, the document.getElementById() returns null.

Because the id of an element is unique within an HTML document, the document.getElementById() is a quick way to access an element.

The following shows the syntax of the getElementById() method:

```
const element = document.getElementById(id);
```

The id is unique within an HTML document. However, HTML is a forgiving language. If the HTML document has multiple elements with the same id, the document.getElementById() method returns the first element it encounters.

```
<html>
    <head>
        <title>JavaScript getElementById() Method</title>
    </head>
    <body>
        <p id="message">A paragraph</p>
    </body>
</html>
```

The document contains a <p> element that has the id attribute with the value message:

```
const p = document.getElementById('message');
console.log(p);
Output:
<p id="message">A paragraph</p>
```

**Introduction to JavaScript getElementsByName() method**

Every element on an HTML document may have a name attribute:

```
<input type="radio" name="language" value=“JavaScript">
```

Unlike the id attribute, multiple HTML elements can share the same value of the name attribute like this:

```
<input type="radio" name="language" value="JavaScript">
```

<input type="radio" name="language" value="TypeScript">

To get all elements with a specified name, you use the getElementsByName() method of the document object:

```
let elements = document.getElementsByName(name);
```

The getElementsByName() accepts a name which is the value of the name attribute of elements and returns a live NodeList of elements.

JavaScript getElementsByName() example

The following example shows a radio group that consists of radio buttons that have the same name (rate).

When you select a radio button and click the submit button, the page will show the selected value such as Very Poor, Poor, OK, Good, or Very Good:

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8">
    <title>JavaScript getElementsByName Demo</title>
</head>

<body>
    <p>Please rate the service:</p>
    <p>
        <label for="very-poor">
            <input type="radio" name="rate" value="Very poor" id="very-
poor"> Very poor
        </label>
        <label for="poor">
            <input type="radio" name="rate" value="Poor" id="poor"> Poor
        </label>
        <label for="ok">
            <input type="radio" name="rate" value="OK" id="ok"> OK
        </label>
        <label for="good">
            <input type="radio" name="rate" value="Good"> Good
```

```
        </label>
        <label for="very-good">
            <input type="radio" name="rate" value="Very Good" id="very-
good"> Very Good
        </label>
    </p>
    <p>
        <button id="btnRate">Submit</button>
    </p>
    <p id="output"></p>
    <script>
        let btn = document.getElementById('btnRate');
        let output = document.getElementById('output');

        btn.addEventListener('click', () => {
            let rates = document.getElementsByName('rate');
            rates.forEach((rate) => {
                if (rate.checked) {
                    output.innerText = `You selected: ${rate.value}`;
                }
            });

        });
    </script>
</body>

</html>
```

**How it works:**

- First, select the submit button by its id `btnRate` using the `getElementById()` method.
- Second, listen to the `click` event of the submit button.
- Third, get all the radio buttons using the `getElementsByName()` and show the selected value in the output element.

## Introduction to JavaScript getElementsByTagName() method

The `getElementsByTagName()` is a method of the `document` object or a specific DOM element.

The getElementsByTagName() method accepts a tag name and returns a live HTMLCollection of elements with the matching tag name in the order which they appear in the document.

The following illustrates the syntax of the getElementsByTagName():

```
let elements = document.getElementsByTagName(tagName);
```

The return collection of the getElementsByTagName() is live, meaning that it is automatically updated when elements with the matching tag name are added and/or removed from the document.

## JavaScript getElementsByTagName() example

The following example illustrates how to use the getElementsByTagName() method to get the number of H2 tags in the document.

When you click the **Count H2** button, the page shows the number of H2 tags:

```
<!DOCTYPE html>
<html>
<head>
    <title>JavaScript getElementsByTagName() Demo</title>
</head>
<body>
    <h1>JavaScript getElementsByTagName() Demo</h1>
    <h2>First heading</h2>
    <p>This is the first paragraph.</p>
    <h2>Second heading</h2>
    <p>This is the second paragraph.</p>
    <h2>Third heading</h2>
    <p>This is the third paragraph.</p>

    <button id="btnCount">Count H2</button>

    <script>
        let btn = document.getElementById('btnCount');
        btn.addEventListener('click', () => {
            let headings = document.getElementsByTagName('h2');
            alert(`The number of H2 tags: ${headings.length}`);
```

```
        });
    </script>
</body>

</html>
```

**How it works:**

- First, select the button **Count H2** by using the `getElementById()` method.
- Second, hook the click event of the button to an anonymous function.
- Third, in the anonymous function, use the `document.getElementsByTagName()` to get a list of `H2` tags.
- Finally, show the number of `H2` tags using the `alert()` function.

## JavaScript getElementsByClassName() method examples

Let's take some examples of using the `getElementsByClassName()` method.

Suppose that you have the following HTML document:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>JavaScript getElementsByClassName</title>
</head>
<body>
    <header>
        <nav>
            <ul id="menu">
                <li class="item">HTML</li>
                <li class="item">CSS</li>
                <li class="item highlight">JavaScript</li>
                <li class="item">TypeScript</li>
            </ul>
        </nav>
```

```
        <h1>getElementsByClassName Demo</h1>
    </header>
    <section>
        <article>
            <h2 class="secondary">Example 1</h2>
        </article>
        <article>
            <h2 class="secondary">Example 2</h2>
        </article>
    </section>
</body>
</html>
```

### 1) Calling JavaScript getElementsByClassName() on an element example

The following example illustrates how to use the `getElementsByClassName()` method to select the `<li>` items which are the descendants of the `<ul>` element:

```
let menu = document.getElementById('menu');
let items = menu.getElementsByClassName('item');

let data = [].map.call(items, item => item.textContent);

console.log(data);
```

Output:
['HTML', 'CSS', 'JavaScript', ‘TypeScript']

How it works:

- First, select the `<ul>` element with the class name `menu` using the `getElementById()` method.
- Then, select `<li>` elements, which are the descendants of the `<ul>` element, using the `getElementsByClassName()` method.
- Finally, create an array of the text content of `<li>` elements by borrowing the `map()` method of the `Array` object.

## Introduction to parentNode attribute

To get the parent node of a specified node in the DOM tree, you use the parentNode property:

```
let parent = node.parentNode;
```

The parentNode is read-only.

The Document and DocumentFragment nodes do not have a parent. Therefore, the parentNode will always be null.

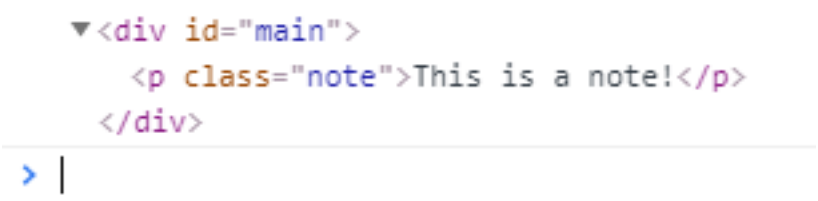
If you create a new node but haven't attached it to the DOM tree, the parentNode of that node will also be null.

### JavaScript parentNode example

See the following HTML document:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>JavaScript parentNode</title>
</head>
<body>
    <div id="main">
        <p class="note">This is a note!</p>
    </div>

    <script>
        let note = document.querySelector('.note');
        console.log(note.parentNode);
    </script>
</body>
</html>
```

The following picture shows the output in the Console:



How it works:

- First, select the element with the `.note` class by using the `querySelector()` method.
- Second, find the parent node of the element.

**Getting Child Elements of a Node in JavaScript**
**Summary**: in this tutorial, you will learn how to get the first child element, last child element, and all children of a specified element.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>JS Get Child Elements</title>
</head>
<body>
  <ul id="menu">
    <li class="first">Home</li>
    <li>Products</li>
    <li class="current">Customer Support</li>
    <li>Careers</li>
    <li>Investors</li>
    <li>News</li>
    <li class="last">About Us</li>
  </ul>
</body>
</html>
```

## Get the first child element

To get the first child element of a specified element, you use the `firstChild` property of the element:

```
let firstChild = parentElement.firstChild;
```

If the `parentElement` does not have any child element, the `firstChild` returns `null`. The `firstChild` property returns a child node which can be

any node type such as an element node, a text node, or a comment node. The following script shows the first child of the #menu element:

```
let content = document.getElementById('menu');
let firstChild = content.firstChild.nodeName;
console.log(firstChild);
```

Output:
#text

The Console window show #text because a text node is inserted to maintain the whitespace between the openning <ul> and <li> tags. This whitespace creates a #text node.

Note that any whitespace such as a single space, multiple spaces, returns, and tabs will create a #text node. To remove the #text node, you can remove the whitespaces as follows:

Or to get the first child with the Element node only, you can use the firstElementChild property:

```
let firstElementChild = parentElement.firstElementChild;
```

The following code returns the first list item which is the first child element of the menu:

```
let content = document.getElementById('menu');
console.log(content.firstElementChild);
```

Output:
<li class=“first">Home</li>

In this example:

- First, select the #menu element by using the getElementById() method.
- Second, get the first child element by using the firstElementChild property.

Get the last child element

To get the last child element of a node, you use the lastChild property:

```
let lastChild = parentElement.lastChild;
```

In case the parentElement does not have any child element, the lastChild returns null. Similar to the the firstChild property, the lastChild property returns the first element node, text node, or comment node. If you want to select only the last child element with the element node type, you use the lastElementChild property:

```
let lastChild = parentElement.lastElementChild;
```

The following code returns the list item which is the last child element of the menu:

```
let menu = document.getElementById('menu');
console.log(main.lastElementChild);
```

Output:
<li class="last">About Us</li>

## Get all child elements

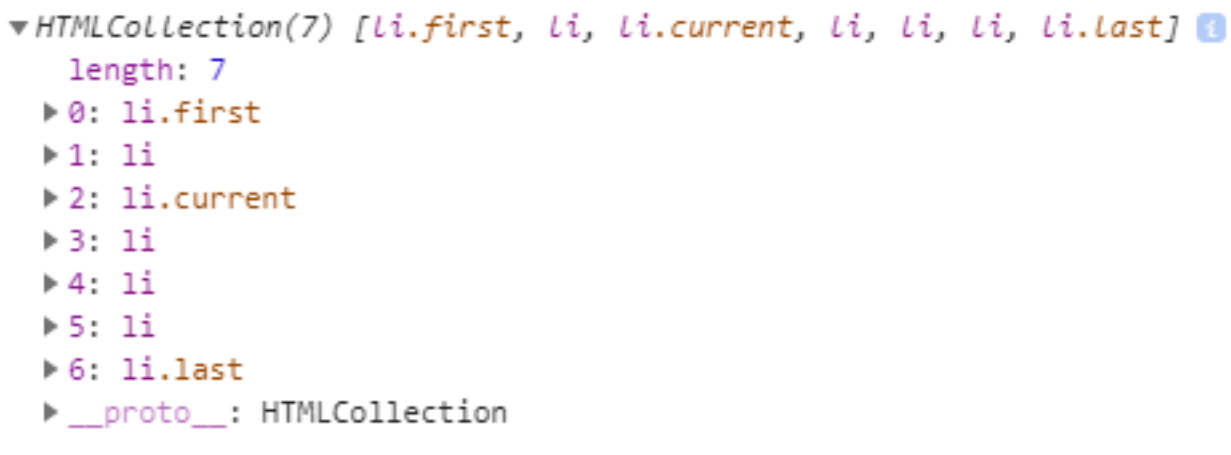
The childNodes property returns all child elements with any node type. To get the child element with only the element node type, you use the children property:

```
let children = parentElement.children;
```

The following example selects all child elements of the element with the Id main:

```
let menu = document.getElementById('menu');
let children = menu.children;
console.log(children);
```

Output:



### Summary

- The firstChild and lastChild return the first and last child of a node, which can be any node type including text node, comment node, and element node.
- The firstElementChild and lastElementChild return the first and last child Element node.
- The childNodes returns a live NodeList of all child nodes of any node type of a specified node. The children return all child Element nodes of a specified node.

### Introduction to JavaScript querySelector() and querySelectorAll() methods

The querySelector() is a method of the Element interface. The querySelector() method allows you to select the first element that matches one or more CSS selectors.

The following illustrates the syntax of the querySelector() method:

```
let element = parentNode.querySelector(selector);
```

In this syntax, the selector is a CSS selector or a group of CSS selectors to match the descendant elements of the parentNode.

If the selector is not valid CSS syntax, the method will raise a SyntaxError exception.

If no element matches the CSS selectors, the querySelector() returns null.

Besides the querySelector(), you can use the querySelectorAll() method to select all elements that match a CSS selector or a group of CSS selectors:

```
let elementList = parentNode.querySelectorAll(selector);
```

The querySelectorAll() method returns a static NodeList of elements that match the CSS selector. If no element matches, it returns an empty NodeList.

To convert the NodeList to an array, you use the Array.from() method like this:

```
let nodeList = document.querySelectorAll(selector);
let elements = Array.from(nodeList);
```

**Basic selectors**

Suppose that you have the following HTML document:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>querySelector() Demo</title>
</head>
<body>
    <header>
        <div id="logo">
            <img src="img/logo.jpg" alt="Logo" id="logo">
        </div>
        <nav class="primary-nav">
            <ul>
                <li class="menu-item current"><a href="#home">Home</
a></li>
                <li class="menu-item"><a href="#services">Services</a></
li>
                <li class="menu-item"><a href="#about">About</a></li>
                <li class="menu-item"><a href="#contact">Contact</a></li>
            </ul>
        </nav>
    </header>
    <main>
        <h1>Welcome to the JS Dev Agency</h1>

        <div class="container">
            <section class="section-a">
                <h2>UI/UX</h2>
                <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit.
Autem placeat, atque accusamus voluptas
                    laudantium facilis iure adipisci ab veritatis eos neque culpa
id nostrum tempora tempore minima.
                    Adipisci, obcaecati repellat.</p>
```

```
            <button>Read More</button>
        </section>
        <section class="section-b">
            <h2>PWA Development</h2>
            <p>Lorem ipsum dolor sit, amet consectetur adipisicing elit.
Magni fugiat similique illo nobis quibusdam
                commodi aspernatur, tempora doloribus quod, consectetur
deserunt, facilis natus optio. Iure
                provident labore nihil in earum.</p>
            <button>Read More</button>
        </section>
        <section class="section-c">
            <h2>Mobile App Dev</h2>
            <p>Lorem ipsum dolor sit amet consectetur adipisicing elit.
Animi eos culpa laudantium consequatur ea!
                Quibusdam, iure obcaecati. Adipisci deserunt, alias repellat
eligendi odit labore! Fugit iste sit
                laborum debitis eos?</p>
            <button>Read More</button>
        </section>
    </div>
</main>
<script src="js/main.js"></script>
</body>
</html>
```

## 1) Universal selector

The universal selector is denoted by `*` that matches all elements of any type:

```
*
```

The following example uses the `querySelector()` selects the first element in the document:

```
let element = document.querySelector('*');
```

And this select all elements in the document:

```
let elements = document.querySelectorAll('*');
```

## 2) Type selector

To select elements by node name, you use the type selector e.g., `a` selects all `<a>` elements:

The following example finds the first `h1` element in the document:

```
let firstHeading = document.querySelector('h1');
```

And the following example finds all `h2` elements:

```
let heading2 = document.querySelectorAll('h2');
```

### 3) Class selector

To find the element with a given CSS class, you use the class selector syntax:

The following example finds the first element with the `menu-item` class:

```
let note = document.querySelector('.menu-item');
```

And the following example finds all elements with the `menu` class:

```
let notes = document.querySelectorAll('.menu-item');
```

### 4) ID Selector

To select an element based on the value of its id, you use the id selector syntax:

```
#id
```

The following example finds the first element with the id `#logo`:

```
let logo = document.querySelector('#logo');
```

Since the `id` should be unique in the document, the `querySelectorAll()` is not relevant.

### 5) Attribute selector

To select all elements that have a given attribute, you use one of the following attribute selector syntaxes:

[attribute]
[attribute=value]
[attribute~=value]
[attribute|=value]
[attribute^=value]
[attribute$=value]
[attribute*$*=value]

The following example finds the first element with the attribute `[autoplay]` with any value:

```
let autoplay = document.querySelector(‘[autoplay]');
```

And the following example finds all elements that have `[autoplay]` attribute with any value:

```
let autoplays = document.querySelectorAll(‘[autoplay]');
```

## Grouping selectors

To group multiple selectors, you use the following syntax:

The following example finds all `<div>` and `<p>` elements:

```
let elements = document.querySelectorAll('div, p’);
```

## Combinators

### 1) descendant combinator

To find descendants of a node, you use the space ( ) descendant combinator syntax:

```
let links = document.querySelector('p a’);
```

### 2) Child combinator

The `>` child combinator finds all elements that are direct children of the first element:

```
let listItems = document.querySelectorAll('ul > li’);
```

To select all li elements that are directly inside a <ul> element with the class nav:

let listItems = document.querySelectorAll('ul.nav > li’);

3) General sibling combinator

The ~ combinator selects siblings that share the same parent:

let links = document.querySelectorAll('p ~ a’);

4) Adjacent sibling combinator

The + adjacent sibling combinator selects adjacent siblings:

let links = document.querySelector('h1 + a');

Summary

- The querySelector() finds the first element that matches a CSS selector or a group of CSS selectors.
- The querySelectorAll() finds all elements that match a CSS selector or a group of CSS selectors.

**Understanding Relationships Between HTML Attributes & DOM Object’s Properties**

When the web browser loads an HTML page, it generates the corresponding DOM objects based on the DOM nodes of the document.

For example, if a page contains the following input element:

<input type="text" id=“username">

The web browser will generate an HTMLInputElement object.

The input element has two attributes:

- The type attribute with the value text.
- The id attribute with the value username.

The generated HTMLInputElement object will have the corresponding properties:

- The input.type with the value text.
- The input.id with the value username.

In other words, the web browser will automatically convert attributes of HTML elements to properties of DOM objects.

However, the web browser only converts the *standard* attributes to the DOM object's properties. The standard attributes of an element are listed on the element's specification.

<input type="text" id="username" secured="true">

In this example, the secured is a non-standard attribute:

```
let input = document.querySelector('#username');

console.log(input.secured); // undefined
```

### element.attributes

The element.attributes property provides a live collection of attributes available on a specific element. For example:

```
let input = document.querySelector('#username');

for(let attr of input.attributes) {
    console.log(`${attr.name} = ${attr.value}` )
}
```

Output:
type = text
id = username
secure = true

### Attribute-property synchronization

When a standard attribute changes, the corresponding property is auto-updated with some exceptions and vice versa.

Suppose that you have the following input element:

```
<input type="text" id="username" tabindex=“1">
```

The following example illustrates the change of the tabindex attribute is propagated to the tabIndex property and vice versa:

```
let input = document.querySelector('#username');

// attribute -> property
input.setAttribute('tabindex', 2);
console.log(input.tabIndex);  // 2


// property -> attribute
input.tabIndex = 3;
console.log(input.getAttribute('tabIndex')); // 3
```

The following example shows when the value attribute changes, it reflects in the value property, but not the other way around:

```
let input = document.querySelector('#username');

// attribute -> property: OK
input.setAttribute('value','guest');
console.log(input.value);  // guest


// property -> attribute: doesn't change
input.value = 'admin';
console.log(input.getAttribute('value')); // guest
```

## DOM properties are typed

The value of an attribute is always a string. However, when the attribute is converted to the property of a DOM object, the property value can be a string, a boolean, an object, etc.

The following checkbox element has the checked attribute. When the checked attribute is converted to the property, it is a boolean value:

```
<input type="checkbox" id="chkAccept" checked> Accept

let checkbox = document.querySelector('#chkAccept');
```

```
console.log(checkbox.checked); // true
```

The following shows an input element with the style attribute:

```
<input type="password" id="password" style=“color:red;with:100%">
```

The style attribute is a string while the style property is an object:

```
let input = document.querySelector('#password');

let styleAttr = input.getAttribute('style');
console.log(styleAttr);

console.dir(input.style);
```

Output:

```
[object CSSStyleDeclaration]
```

## The data-* attributes

If you want to add a custom attribute to an element, you should prefix it with the data- e.g., data-secured because all attributes start with data- are reserved for the developer’s uses.

To access data-* attributes, you can use the dataset property. For example, we have the following div element with custom attributes:

```
<div id="main" data-progress="pending" data-value=“10%"></div>
```

The following shows how to access the data-* attributes via the dataset property:

```
let bar = document.querySelector('#main');
console.log(bar.dataset);
```

Output:

```
[object DOMStringMap] {
    progress: "pending",
    value: "10%"
}
```

## Introduction to the JavaScript setAttribute() method

To set a value of an attribute on a specified element, you use the setAttribute() method:

```
element.setAttribute(name, value);
```

### Parameters

The name specifies the attribute name whose value is set. It's automatically converted to lowercase if you call the setAttribute() on an HTML element.

The value specifies the value to assign to the attribute. It's automatically converted to a string if you pass a non-string value to the method.

### Return value

The setAttribute() returns undefined.

Note that if the attribute already exists on the element, the setAttribute() method updates the value; otherwise, it adds a new attribute with the specified name and value.

Typically, you use the querySelector() or getElementById() to select an element before calling the setAttribute() on the selected element.

To get the current value of an attribute, you use the getAttribute() method. To remove an attribute, you call the removeAttribute() method.

### JavaScript setAttribute() example

See the following example:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>JS setAttribute() Demo</title>
</head>
<body>
    <button id="btnSend">Send</button>
```

```
    <script>
        let btnSend = document.querySelector('#btnSend');
        if (btnSend) {
            btnSend.setAttribute('name', 'send');
            btnSend.setAttribute('disabled', '');
        }
    </script>
</body>
</html>
```

How it works:

- First, select the button with the id `btnSend` by using the `querySelector()` method.
- Second, set the value of the `name` attribute to `send` using the `setAttribute()` method.
- Third, set the value of the `disabled` attribute so that when users click the button, it will do nothing.

## Introduction to the JavaScript getAttribute() method

To get the value of an attribute on a specified element, you call the `getAttribute()` method of the element:

```
let value = element.getAttribute(name);
```

### Parameters

The `getAttribute()` accepts an argument which is the name of the attribute from which you want to return the value.

### JavaScript getAttribute() example

Consider the following example:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>JS getAttribute() Demo</title>
</head>
```

```
<body>

    <a href="https://www.javascripttutorial.net"
       target="_blank"
       id="js">JavaScript Tutorial
    </a>

    <script>
        let link = document.querySelector('#js');
        if (link) {
            let target = link.getAttribute('target');
            console.log(target);
        }
    </script>
</body>
</html>
```

Output:
_blank

How it works:

- First, select the link element with the id `js` using the `querySelector()` method.
- Second, get the target attribute of the link by calling the `getAttribute()` of the selected link element.
- Third, show the value of the target on the Console window.

The following example uses the `getAttribute()` method to get the value of the title attribute of the link element with the id `js`:

```
let link = document.querySelector('#js');
if (link) {
    let title = link.getAttribute('title');
    console.log(title);
}
```

Output:
null

## Introduction to JavaScript removeAttribute() method

The removeAttribute() removes an attribute with a specified name from an element:

```
element.removeAttribute(name);
```

### Parameters

The removeAttribute() accepts an argument which is the name of the attribute that you want to remove. If the attribute does not exist, the removeAttribute() method wil not raise an error.

For example, the values of the disabled attributes are true in the following cases:

```
<button disabled>Save Draft</button>
<button disabled="">Save</button>
<button disabled="disabled">Cancel</button>
```

Similarly, the values of the following readonly attributes are true:

```
<input type="text" readonly>
<textarea type="text" readonly="">
<textarea type="text" readonly="readonly">
```

### JavaScript removeAttribute() example

The following example uses the removeAttribute() method to remove the target attribute from the link element with the id js:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>JS removeAttribute() Demo</title>
</head>
<body>
    <a href="https://www.javascripttutorial.net"
       target="_blank"
       id="js">JavaScript Tutorial</a>
```

```
    <script>
        let link = document.querySelector('#js');
        if (link) {
            link.removeAttribute('target');
        }
    </script>
</body>
</html>
```

How it works:

- Select the link element with id `js` using the `querySelector()` method.
- Remove the `target` attribute by calling the `removeAttribute()` on the selected link element.

## Introduction to the JavaScript hasAttribute() method

To check an element has a specified attribute or not, you use the `hasAttribute()` method:

```
let result = element.hasAttribute(name);
```

### Parameters

The `hasAttribute()` method accepts an argument that specifies the name of the attribute that you want to check.

### JavaScript hasAttribute() example

See the following example:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>JS hasAttribute() Demo</title>
</head>
<body>

    <button id="btnSend" disabled>Send</button>
```

```
    <script>
        let btn = document.querySelector('#btnSend');
        if (btn) {
            let disabled = btn.hasAttribute('disabled');
            console.log(disabled);
        }
    </script>
</body>
</html>
```

Output:
true

How it works:

- Select the button with the id btnSend by using the querySelector() method.
- Check if the button has the disabled attribute by calling the hasAttribute() method on the button element.

**Setting inline styles**

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>JS Style Demo</title>
</head>
<body>
    <p id="content">JavaScript Setting Style Demo!</p>
    <script>
        let p = document.querySelector('#content');
        p.style.color = 'red';
        p.style.fontWeight = 'bold';
    </script>
</body>
</html>
```

How it works:

- First, select the paragraph element whose id is `content` by using the `querySelector()` method.
- Then, set the color and font-weight properties of the paragraph by setting the `color` and `fontWeight` properties of the `style` object.

## Introduction to JavaScript events

An event is an action that occurs in the web browser, which the web browser feedbacks to you so that you can respond to it.

For example, when users click a button on a webpage, you may want to respond to this `click` event by displaying a dialog box.

Each event may have an event handler which is a block of code that will execute when the event occurs.

An event handler is also known as an event listener. It listens to the event and executes when the event occurs.

```
let btn = document.querySelector('#btn');

function display() {
    alert('It was clicked!');
}

btn.addEventListener(‘click',display);
```

How it works.

- First, select the button with the id `btn` by using the `querySelector()` method.
- Then, define a function called `display()` as an event handler.
- Finally, register an event handler using the `addEventListener()` so that when users click the button, the `display()` function will be executed.

## Event flow

Assuming that you have the following HTML document:

```
<!DOCTYPE html>
<html>
<head>
    <title>JS Event Demo</title>
</head>
<body>
    <div id="container">
        <button id='btn'>Click Me!</button>
    </div>
</body>
```

When you click the button, you're clicking not only the button but also the button's container, the div, and the whole webpage.

Event flow explains the order in which events are received on the page from the element where the event occurs and propagated through the DOM tree.

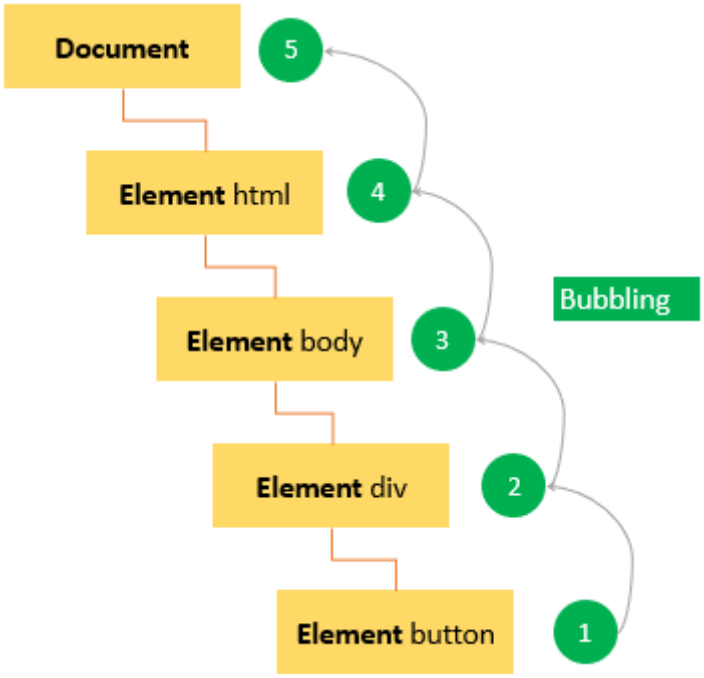
### Event bubbling

In the event bubbling model, an event starts at the most specific element and then flows upward toward the least specific element (the document or even window).

When you click the button, the click event occurs in the following order:

1. button
2. div with the id container
3. body
4. html
5. document

The click event first occurs on the button which is the element that was clicked. Then the click event goes up the DOM tree, firing on each node along its way until it reaches the document object.



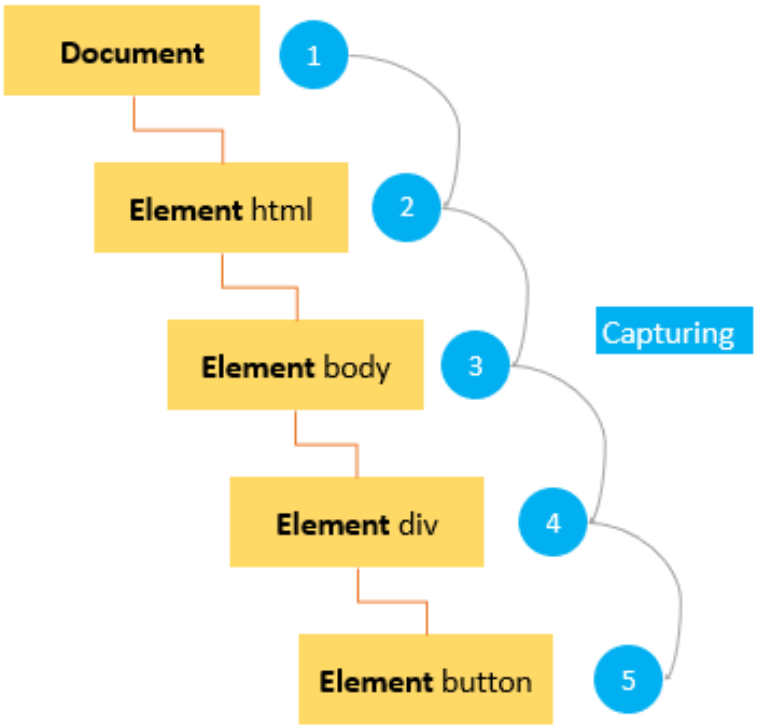

## Event capturing

In the event capturing model, an event starts at the least specific element and flows downward toward the most specific element.

When you click the button, the click event occurs in the following order:

1. document
2. html
3. body
4. div with the id container
5. button

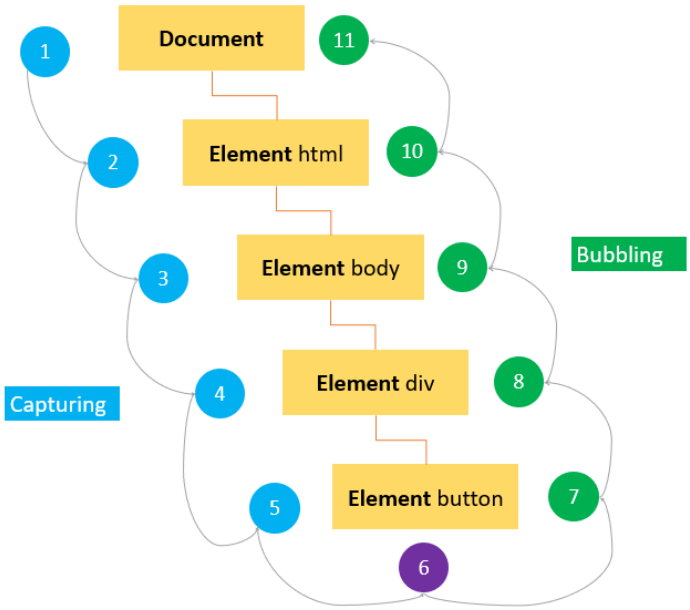
The following picture illustrates the event capturing effect:




## DOM Level 2 Event flow

DOM level 2 events specify that event flow has three phases:

- First, event capturing occurs, which provides the opportunity to intercept the event.
- Then, the actual target receives the event.
- Finally, event bubbling occurs, which allows a final response to the event.



### Event object

When the event occurs, the web browser passed an Event object to the event handler:

```
let btn = document.querySelector('#btn');

btn.addEventListener('click', function(event) {

    console.log(event.type);

});
```

Output:

‘click’

| Property / Method | Description |
| --- | --- |
| bubbles | true if the event bubbles |
| cancelable | true if the default behavior of the event can be canceled |
| currentTarget | the current element on which the event is firing |
| defaultPrevented | return true if the preventDefault() has been called. |
| detail | more informatio nabout the event |
| eventPhase | 1 for capturing phase, 2 for target, 3 for bubbling |
| preventDefault() | cancel the default behavior for the event. This method is only effective if the `cancelable` property is true |

| | |
|---|---|
| stopPropagation() | cancel any further event capturing or bubbling. This method only can be used if the `bubbles` property is true. |
| target | the target element of the event |
| type | the type of event that was fired |

**preventDefault()**

To prevent the default behavior of an event, you use the preventDefault() method.

However, you can prevent this behavior by using the preventDefault() method of the event object:

```
let link = document.querySelector('a');

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

Note that the preventDefault() method does not stop the event from bubbling up the DOM. And an event can be canceled when its cancelable property is true.

stopPropagation()

The stopPropagation() method immediately stops the flow of an event through the DOM tree. However, it does not stop the browers default behavior.

```
let btn = document.querySelector('#btn');

btn.addEventListener('click', function(event) {
    console.log('The button was clicked!');
    event.stopPropagation();
});

document.body.addEventListener('click',function(event) {
```

```
    console.log('The body was clicked!');
});
```

Without the stopPropagation() method, you would see two messages on the Console window.

However, the click event never reaches the body because the stopPropagation() was called on the click event handler of the button.

### 1) HTML event handler attributes

Event handlers typically have names that begin with on, for example, the event handler for the click event is onclick.

```
<input type="button" value="Save" onclick=“alert('Clicked!')">
```

An event handler defined in the HTML can call a function defined in a script. For example:

```
<script>
    function showAlert() {
        alert('Clicked!');
    }
</script>
<input type="button" value="Save" onclick=“showAlert()">
```

In this example, the button calls the showAlert() function when it is clicked.

The showAlert() is a function defined in a separate <script> element, and could be placed in an external JavaScript file.

### Overview of JavaScript page load events

When you open a page, the following events occur in sequence:

- DOMContentLoaded – the browser fully loaded HTML and completed building the DOM tree. However, it hasn’t loaded external resources like stylesheets and images. In this event, you can start selecting DOM nodes or initialize the interface.
- load – the browser fully loaded the HTML and also external resources like images and stylesheets.

When you leave the page, the following events fire in sequence:

- beforeunload – fires before the page and resources are unloaded. You can use this event to show a confirmation dialog to confirm if you really want to leave the page. By doing this, you can prevent data loss in case you are filling out a form and accidentally click a link to navigate to another page.
- unload – fires when the page has completely unloaded. You can use this event to send the analytic data or to clean up resources.

```
<!DOCTYPE html>
<html>
<head>
    <title>JS Page Load Events</title>
</head>

<body>
    <script>
        addEventListener('DOMContentLoaded', (event) => {
            console.log('The DOM is fully loaded.');
        });

        addEventListener('load', (event) => {
            console.log('The page is fully loaded.');
        });

        addEventListener('beforeunload', (event) => {
            // show the confirmation dialog
            event.preventDefault();
            // Google Chrome requires returnValue to be set.
            event.returnValue = '';
        });

        addEventListener('unload', (event) => {
            // send analytic data
        });
    </script>
</body>
</html>
```

The window’s load event

For the window object, the load event is fired when the whole webpage (HTML) has loaded fully, including all dependent resources, including JavaScript files, CSS files, and images.

To handle the load event, you register an event listener using the addEventListener() method:

```
window.addEventListener('load', (event) => {
    console.log('The page has fully loaded');
});
```

Or use the onload property of the window object:

```
window.onload = (event) => {
    console.log('The page has fully loaded');
};
```

```
<!DOCTYPE html>
<html>
<head>
    <title>JS load Event Demo</title>
</head>
<body onload="console.log('Loaded!')">
</body>
</html>
```

## The image's load event

The load event also fires on images. To handle the load event on images, you use the addEventListener() method of the image elements.

```
<!DOCTYPE html>
<html>
<head>
    <title>Image load Event Demo</title>
</head>
<body>
    <img id="logo">
    <script>
        let logo = document.querySelector('#logo');

        logo.addEventListener('load', (event) => {
```

```
            console.log('Logo has been loaded!');
        });

        logo.src = "logo.png";
    </script>
</body>
</html>
```

You can assign an `onload` event handler directly using the `onload` attribute of the `<img>` element, like this:

If you create an image element dynamically, you can assign an `onload` event handler before setting the `src` property as follows:

```
window.addEventListener('load' () => {
    let logo = document.createElement('img');
    // assign and onload event handler
    logo.addEventListener('load', (event) => {
        console.log('The logo has been loaded');
    });
    // add logo to the document
    document.body.appendChild(logo);
    logo.src = 'logo.png';
});
```

How it works:

- First, create an image element after the document has been fully loaded by placing the code inside the event handler of the window's load event.
- Second, assign the `onload` event handler to the image.
- Third, add the image to the document.
- Finally, assign an image URL to the `src` attribute. The image will be downloaded to the element as soon as the `src` property is set.

**The script's load event**

The `<script>` element also supports the `load` event slightly different from the standard ways. The script's `load` event allows you to check if a JavaScript file has been completely loaded.

```
window.addEventListener('load', checkJSLoaded)

function checkJSLoaded() {
    // create the script element
    let script = document.createElement('script');

    // assign an onload event handler
    script.addEventListener('load', (event) => {
        console.log('app.js file has been loaded');
    });

    // load the script file
    script.src = 'app.js';
    document.body.appendChild(script);
}
```

**JavaScript DOMContentLoaded**

The DOMContentLoaded fires when the DOM content is loaded, without waiting for images and stylesheets to finish loading.

```
<!DOCTYPE html>
<html>

<head>
    <title>JS DOMContentLoaded Event</title>
    <script>
        let btn = document.getElementById('btn');
        btn.addEventListener('click', (e) => {
            // handle the click event
            console.log('clicked');
        });
    </script>
</head>

<body>
    <button id="btn">Click Me!</button>
</body>

</html>
```

In this example, we reference the button in the body from the JavaScript in the head.

Because the DOM has not been loaded when the JavaScript engine parses the JavaScript in the head, the button with the id btn does not exist.

To fix this, you place the code inside an DOMContentLoaded event handler, like this:

```
<!DOCTYPE html>
<html>

<head>
    <title>JS DOMContentLoaded Event</title>
    <script>
        document.addEventListener('DOMContentLoaded', () => {
            let btn = document.getElementById('btn');
            btn.addEventListener('click', () => {
                // handle the click event
                console.log('clicked');
            });
        });
    </script>
</head>

<body>
    <button id="btn">Click Me!</button>
</body>

</html>
```

When you place JavaScript in the header, it will cause bottlenecks and rendering delays, so it's better to move the script before the </body> tag. In this case, you don't need to place the code in the DOMContentLoaded event, like this:

```
<!DOCTYPE html>
<html>

<head>
    <title>JS DOMContentLoaded Event</title>
```

```
</head>

<body>
    <button id="btn">Click Me!</button>
    <script>
        document.addEventListener('DOMContentLoaded', () => {
            let btn = document.getElementById('btn');
            btn.addEventListener('click', () => {
                // handle the click event
                console.log('clicked');
            });
        });
    </script>
</body>
</html>
```

### Summary

- The DOMContentLoaded event fires when the DOM content is loaded, without waiting for images and stylesheets to finish loading.
- Only handle DOMContentLoaded event if you place the JavaScript code in the head, which references elements in the body section.

## Introduction to JavaScript mouse events

Mouse events fire when you use the mouse to interact with the elements on the page. DOM Level 3 events define nine mouse events.

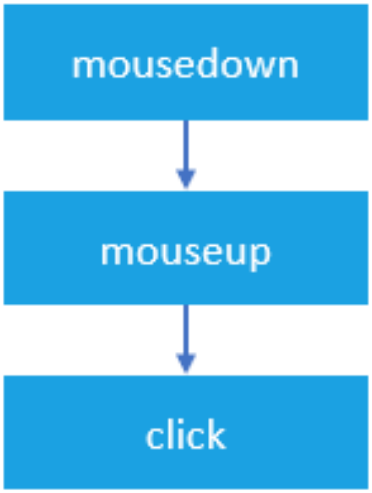
### mousedown, mouseup, and click

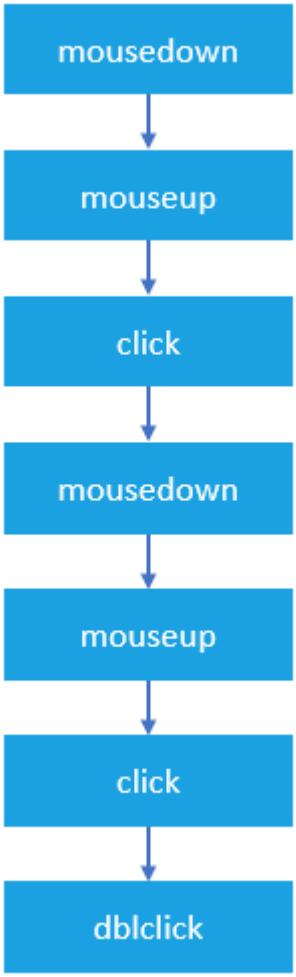
When you click an element, there are no less than three mouse events fire in the following sequence:

1. The mousedown fires when you depress the mouse button on the element.

2. The mouseup fires when you release the mouse button on the element.

3. The click fires when one mousedown and one mouseup detected on the element.




**mouseover / mouseout**

The mouseover fires when the mouse cursor is outside of the element and then move to inside the boundaries of the element.

The mouseout fires when the mouse cursor is over an element and then moves another element.

mouseenter / mouseleave

The mouseenter fires when the mouse cursor is outside of an element and then moves to inside the boundaries of the element.

The mouseleave fires when the mouse cursor is over an element and then moves to the outside of the element's boundaries.

## Registering mouse event handlers

To register a mouse event, you use these steps:

- First, select the element by using `querySelector()` or `getElementById()` method.
- Then, register the mouse event using the `addEventListener()` method.

```
<button id="btn">Click Me!</button>
```

To register a mouse click event handler, you use the following code:

```
let btn = document.querySelector('#btn');

btn.addEventListener('click',(event) => {
    console.log('clicked');
});
```

or you can assign a mouse event handler to the element's property:

```
let btn = document.querySelector('#btn');

btn.onclick = (event) => {
    console.log('clicked');
};
```

In legacy systems, you may find that the event handler is assigned in the HTML attribute of the element:

```
<button id="btn" onclick="console.log('clicked')">Click Me!</button>
```

## Detecting mouse buttons

The `event` object passed to the mouse event handler has a property called `button` that indicates which mouse button was pressed on the mouse to trigger the event.

The mouse button is represented by a number:

- 0: the main mouse button is pressed, usually the left button.
- 1: the auxiliary button is pressed, usually the middle button or the wheel button.

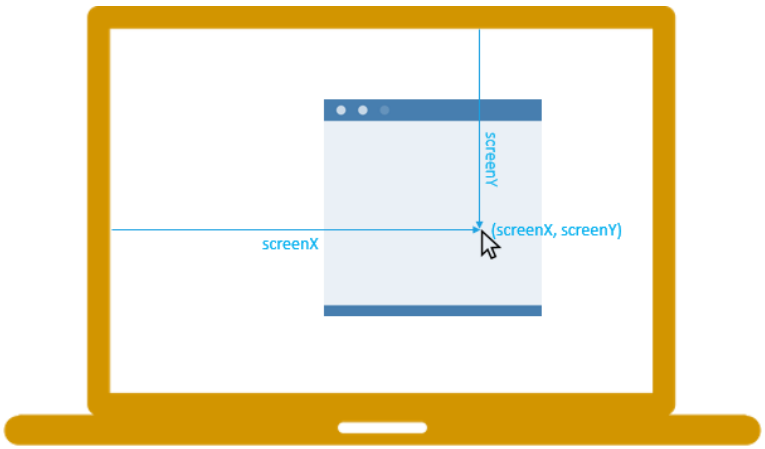
- 2: the secondary button is pressed, usually the right button.
- 3: the fourth button is pressed, usually the Browser Back button.
- 4: the fifth button is pressed, usually the *Browser Forward* button.


```
<!DOCTYPE html>
<html>
<head>
    <title>JS Mouse Events - Button Demo</title>
</head>
<body>
    <button id="btn">Click me with any mouse button: left, right, middle,
...</button>
    <p id="message"></p>
    <script>
        let btn = document.querySelector('#btn');

        // disable context menu when right-mouse clicked
        btn.addEventListener('contextmenu', (e) => {
            e.preventDefault();
        });

        // show the mouse event message
```

```
        btn.addEventListener('mouseup', (e) => {
            let msg = document.querySelector('#message');
            switch (e.button) {
                case 0:
                    msg.textContent = 'Left mouse button clicked.';
                    break;
                case 1:
                    msg.textContent = 'Middle mouse button clicked.';
                    break;
                case 2:
                    msg.textContent = 'Right mouse button clicked.';
                    break;
                default:
                    msg.textContent = `Unknown mouse button code: $
{event.button}`;
            }
        });
    </script>
</body>
</html>
```

## Getting Screen Coordinates

The `screenX` and `screenY` properties of the event passed to the mouse event handler return the screen coordinates of the location of the mouse in relation to the entire screen.




```
<!DOCTYPE html>
<html>
<head>
    <title>JS Mouse Location Demo</title>
```

```
    <style>
        #track {
            background-color: goldenrod;
            height: 200px;
            width: 400px;
        }
    </style>
</head>
<body>
    <p>Move your mouse to see its location.</p>
    <div id="track"></div>
    <p id="log"></p>

    <script>
        let track = document.querySelector('#track');
        track.addEventListener('mousemove', (e) => {
            let log = document.querySelector('#log');
            log.innerText = `
            Screen X/Y: (${e.screenX}, ${e.screenY})
            Client X/Y: (${e.clientX}, ${e.clientY})`
        });
    </script>
</body>
</html>
```

### Scrolling an element

The `scrollTop` property sets or gets the number of pixels that the element's content is vertically scrolled. The `scrollLeft` property gets and sets the number of pixels that an element's content is scrolled from its left edge.

The following example shows how to handle the `scroll` event of the `div` element with the id `scrollDemo`:

```
<!DOCTYPE html>
<html>
<head>
    <title>JS Scroll Events</title>
    <style>
        #scrollDemo {
            height: 200px;
```

```
            width: 200px;
            overflow: auto;
            background-color: #f0db4f
        }

        #scrollDemo p {
            /* show the scrollbar */
            height: 300px;
            width: 300px;
        }
    </style>
</head>
<body>
    <div id="scrollDemo">
        <p>JS Scroll Event Demo</p>
    </div>

    <div id="control">
        <button id="btnScrollLeft">Scroll Left</button>
        <button id="btnScrollTop">Scroll Top</button>
    </div>

    <script>
        let control = document.querySelector('#control');

        control.addEventListener('click', function (e) {
            // get the scrollDemo
            let div = document.getElementById('scrollDemo');
            // get the target
            let target = e.target;
            // handle each button's click
            switch (target.id) {
                case 'btnScrollLeft':
                    div.scrollLeft += 20;
                    break;

                case 'btnScrollTop':
                    div.scrollTop += 20;
                    break;
            }
        });
    </script>
```

```
</body>
</html>
```

## Introduction to JavaScript focus events

The focus events fire when an element receives or loses focus. These are the two main focus events:

- focus fires when an element has received focus.
- blur fires when an element has lost focus.

The focusin and focusout fire at the same time as focus and blur, however, they bubble while the focus and blur do not.

The following elements are focusable:

- The **window** gains focus when you bring it forward by using Alt+Tab or clicking on it and loses focus when you send it back.
- **Links** when you use a mouse or a keyboard.
- **Form fields** like input text when you use a keyboard or a mouse.
- Elements with **tabindex**, also when you use a keyboard or a mouse.

```
<!DOCTYPE html>
<html>
<head>
    <title>JavaScript Focus Events</title>
</head>
<body>
    <p>Move focus to the password field to see the effect:</p>

    <form id="form">
        <input type="text" placeholder="username">
        <input type="password" placeholder="password">
    </form>

    <script>
        const pwd = document.querySelector('input[type="password"]');

        pwd.addEventListener('focus', (e) => {
            e.target.style.backgroundColor = 'yellow';
```

```
        });

        pwd.addEventListener('blur', (e) => {
            e.target.style.backgroundColor = '';
        });
    </script>
</body>
</html>
```

**How to Toggle Password Visibility**

To make the password visible, you follow these steps:

- First, create an `<input>` element with the type of `password` and an icon that allows users to click it to toggle the visibility of the password.
- Second, bind an event handler to the click event of the icon. If the icon is clicked, toggle the `type` attribute of the password field between `text` and `password`. The input with the type `text` will show the actual password.
- Third, change the icon to make it more user-friendly. This step is optional.

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-
scale=1.0" />
    <title>Toggle Password Visibility</title>
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-
icons@1.3.0/font/bootstrap-icons.css" />
    <link rel="stylesheet" href="css/style.css" />
</head>

<body>
    <div class="container">
        <h1>Sign In</h1>
        <form method="post">
            <p>
```

```
            <label for="username">Username:</label>
            <input type="text" name="username" id="username">
        </p>
        <p>
            <label for="password">Password:</label>
            <input type="password" name="password" id="password" />
            <i class="bi bi-eye-slash" id="togglePassword"></i>
        </p>
        <button type="submit" id="submit" class="submit">Log In</
button>
    </form>
  </div>
  <script>
    const togglePassword =
document.querySelector("#togglePassword");
    const password = document.querySelector("#password");

    togglePassword.addEventListener("click", function () {
        // toggle the type attribute
        const type = password.getAttribute("type") === "password" ?
"text" : "password";
        password.setAttribute("type", type);

        // toggle the icon
        this.classList.toggle("bi-eye");
    });

    // prevent form submit
    const form = document.querySelector("form");
    form.addEventListener('submit', function (e) {
        e.preventDefault();
    });
  </script>
</body>

</html>
```

**In the JavaScript:**

First, select the toggle password icon and the password input field using the `querySelector()` method:

```
const togglePassword = document.querySelector('#togglePassword');

const password = document.querySelector('#password');
```

Then, attach an event listener to the togglePassword icon and toggle the type attribute of the password field as well as the class of the icon:

```
togglePassword.addEventListener('click', function (e) {
    // toggle the type attribute
    const type = password.getAttribute('type') === 'password' ? 'text' :
'password';
    password.setAttribute('type', type);
    // toggle the eye / eye slash icon
    this.classList.toggle('bi-eye');
});
```

### Username And Password Validation In JavaScript Code

```
<!doctype html>
<html>
<head>
   <title> JavaScript Validation </title>
</head>
<body>
<h1>
Talkers code – JavaScript Validation
</h1>
<h2>
Username and Password validation in JavaScript
</h2>
<form action="#" onsubmit="return validation(argu)" method="post" >
<label>
Enter Username:
</label>
   <input type="text" name="username "/><br/>
<label>
Enter Password:
</label>
<input type="password" name="password"/><br/>
<label>
Confirm Password:
```

```
</label>
<input type="password" name="cpassword"/><br/>
<input type="submit" value="login"/>
</form>
// JavaScript code here
<script type="text/javascript">
  function validation(form_val)
  {
    var username = form_val.username.value;
    var password = form_val.password.value;
    var cpassword = form_val.cpassword.value;
    if(username == "" )
    {
      alert("please enter your username first.");
      form_val.username.focus();
      return false;
    }
    if(username.length<6)
    {
      alert("username must be at least of 6 characters.");
      form_val.username.focus();
      return false;
    }
    if(password == " ")
    {
      alert("please enter your password.");
      form_val.password.focus();
      return false;
    }
    if(password.length<8)
    {
      alert("password must contain at least of 8 digits");
      form_val.password.focus();
      return false;
    }
    if(cpassword == "")
    {
      alert("please enter confirm password field.");
      form_val.cpassword.focus();
      return false;
```

```
    }
    if(password.length >= 8 && cpassword.length >= 8 )
    {
      if(password != cpassword)
      {
            alert("Password and Confirm Password does not matched. Try
again later. ");
            return false;
      }
    }
    return true;
  }
  </script>
</body>
</html>
```

**Password validation in JavaScript**

1. A password should be made up of alphanumeric characters.
2. The password's first letter should be in the capital.
3. A special character (@, $,! &, etc.) must be included in the password.
4. The length of the password must be higher than 8 characters.
5. One of the most essential things to remember is that the password fields should never be left blank.

There are two different approaches mentioned here for a valid password:

1. Validation form with username

2. Validation form with Confirm password

**Approach 1 - Validation Form with username**

```
<html>
<head>
<title> Verificatiom </title>
</head>
<script>
function verifyPassword() {
  var pw = document.getElementById("pswd").value;
```

```
  if(pw == "") {
     document.getElementById("message").innerHTML = "**Fill the
password!";
     return false;
  }
  if(pw.length < 8) {
     document.getElementById("message").innerHTML = "** length must
be atleast 8 characters";
     return false;
  }
  if(pw.length > 15) {
     document.getElementById("message").innerHTML = "** length must
not exceed 15 characters";
     return false;
  } else {
     alert("Password is correct");
  }
}
</script>
<body>
<center>
<h1 style="color:Orange">StudyTonight</h1>
<h3> Validate password </h3>
<form onsubmit ="return verifyPassword()">
<td> Enter Username</td>
<input type = "username" id = "usr" value = ""><br><br>
<td> Enter Password </td>
<input type = "password" id = "pswd" value = "">
<span id = "message" style="color:orange"> </span> <br><br>
<input type = "submit" value = "Submit" style="border-color:orange">
<button type = "reset" value = "Reset" style="border-
color:orange">Reset</button>
</form>
</center>
</body>
</html>
```

**Approach 2 - Validation Form with Confirm password**

```
<html>
<head>
```

```
<center>
<title> Password Validation</title>
</head>
<script>
function validateForm() {

    var pw1 = document.getElementById("pswd1").value;
    var pw2 = document.getElementById("pswd2").value;
    var name1 = document.getElementById("fname").value;

    if(name1 == "") {
      document.getElementById("blankMsg").innerHTML = "**Fill the first
name";
      return false;
    }


    if(!isNaN(name1)){
      document.getElementById("blankMsg").innerHTML = "**Only
characters are allowed";
      return false;
    }


    if(pw1 == "") {
      document.getElementById("message1").innerHTML = "**Fill the
password please!";
      return false;
    }

    if(pw2 == "") {
      document.getElementById("message2").innerHTML = "**Enter the
password please!";
      return false;
    }


    if(pw1.length < 8) {
      document.getElementById("message1").innerHTML = "**length must
be atleast 8 characters";
```

```
    return false;
   }

   if(pw1.length > 15) {
    document.getElementById("message1").innerHTML = "**length must
not exceed 15 characters";
    return false;
   }

   if(pw1 != pw2) {
    document.getElementById("message2").innerHTML = "**Passwords
are not same! Retry with new Password";
    return false;
   } else {
    alert ("password created successfully");
    document.write("form has been submitted successfully");
   }
 }
</script>

<body>
<h1 style="color:orange">StudyTonight</h1>
<h3>  Please Fill the form  </h3>

<form onsubmit ="return validateForm()">


<td> Full Name* </td>
<input type = "text" id = "fname" value = "">
<span id = "blankMsg" style="color:orange"> </span> <br><br>


<td> Create Password* </td>
<input type = "password" id = "pswd1" value = "">
<span id = "message1" style="color:orange"> </span> <br><br>

<td> Confirm Password* </td>
<input type = "password" id = "pswd2" value = "">
<span id = "message2" style="color:orange"> </span> <br><br>
```

```
<input type = "submit" value = "Submit">

<button type = "reset" value = "Reset" >Reset</button>
</form>
</body>
</center>
</html>
```

Approach 3: Username and Password value checking

```
<form>
    <input type="text" placeholder="Username" id="text1" /><br />
    <input type="password" placeholder="Password" id="text2" /><br />
    <input type="button" value="Login"  onclick="javascript:validate()" />
</form>
<script type="text/javascript">
function validate()
{
    if(   document.getElementById("text1").value == "workshop"
       && document.getElementById("text2").value == "workshop" )
    {
        alert( "validation succeeded" );
        location.href="run.html";
    }
    else
    {
        alert( "validation failed" );
        location.href="fail.html";
    }
}
</script>
```