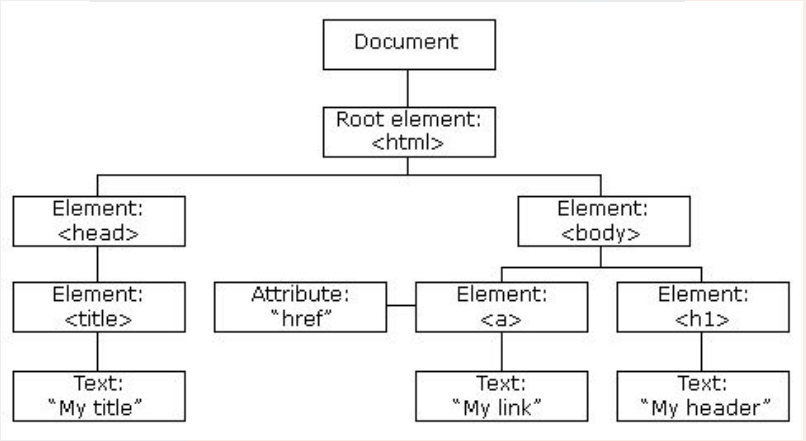
# HTML DOM (Document Object Model)

When a web page is loaded, the browser creates a Document Object Model of the page.

The HTML DOM model is constructed as a tree of Objects:

The HTML DOM Tree of Objects



# Javascript and DOM

With the object model, JavaScript gets all the power it needs to create dynamic HTML:

- JavaScript can change all the HTML elements in the page
- JavaScript can change all the HTML attributes in the page
- JavaScript can change all the CSS styles in the page
- JavaScript can remove existing HTML elements and attributes
- JavaScript can add new HTML elements and attributes
- JavaScript can react to all existing HTML events in the page
- JavaScript can create new HTML events in the page

# What you will learn?

- How to change the content of HTML elements
- How to change the style (CSS) of HTML elements
- How to react to HTML DOM events
- How to add and delete HTML elements

# What is the DOM

The DOM is a W3C (World Wide Web Consortium) standard.

The DOM defines a standard for accessing documents:

*"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."*

The W3C DOM standard is separated into 3 different parts:

1. Core DOM - standard model for all document types
2. XML DOM - standard model for XML documents
3. HTML DOM - standard model for HTML documents

# What is the HTML DOM?

The HTML DOM is a standard object model and programming interface for HTML. It defines:

- The HTML elements as objects
- The properties of all HTML elements
- The methods to access all HTML elements
- The events for all HTML elements

*The HTML DOM is a standard for how to get, change, add, or delete HTML elements.*

# HTML DOM Methods

**The DOM Programming Interface**
The HTML DOM can be accessed with JavaScript (and with other programming languages).

In the DOM, all HTML elements are defined as objects.

The programming interface is the properties and methods of each object.

A **property** is a value that you can get or set (like changing the content of an HTML element).

A **method** is an action you can do (like add or deleting an HTML element).

# Example

```
<html>
<body>

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

<script>
document.getElementById("demo").innerHTML = "Hello World!";
</script>

</body>
</html>
```





The `innerHTML` property can be used to get or change any HTML element, including `<html>` and `<body>`.

# DOM Document

The document object represents your web page.

If you want to access any element in an HTML page, you always start with accessing the document object.

Finding HTML Elements

| Method | Description |
| --- | --- |
| document.getElementById(*id*) | Find an element by element id |
| document.getElementsByTagName(*name*) | Find elements by tag name |
| document.getElementsByClassName(*name*) | Find elements by class name |

# DOM Document

## Changing HTML Elements

| **Property** | **Description** |
| --- | --- |
| *element*.innerHTML = *new html content* | Change the inner HTML of an element |
| *element.attribute = new value* | Change the attribute value of an HTML element |
| *element*.style.*property = new style* | Change the style of an HTML element |
| **Method** | **Description** |
| *element*.setAttribute*(attribute, value)* | Change the attribute value of an HTML element |

# DOM Document

Adding and Deleting Elements

| Method | Description |
| --- | --- |
| document.createElement(*element*) | Create an HTML element |
| document.removeChild(*element*) | Remove an HTML element |
| document.appendChild(*element*) | Add an HTML element |
| document.replaceChild(*new, old*) | Replace an HTML element |
| document.write(*text*) | Write into the HTML output stream |

# DOM Document

Adding Events Handlers

| Method | Description |
| --- | --- |
| document.getElementById(*id*).onclick = function(){*code*} | Adding event handler code to an onclick event |

# DOM Document

## Finding HTML Objects

| Property | Description |
|---|---|
| document.anchors | Returns all <a> elements that have a name attribute |
| document.applets | Deprecated |
| document.baseURI | Returns the absolute base URI of the document |
| document.body | Returns the <body> element |
| document.cookie | Returns the document's cookie |
| document.doctype | Returns the document's doctype |
| document.documentElement | Returns the <html> element |
| document.documentMode | Returns the mode used by the browser |
| document.documentURI | Returns the URI of the document |
| document.domain | Returns the domain name of the document server |
| document.embeds | Returns all <embed> elements |
| document.forms | Returns all <form> elements |
| document.head | Returns the <head> element |
| document.images | Returns all <img> elements |
| document.implementation | Returns the DOM implementation |
| document.inputEncoding | Returns the document's encoding (character set) |
| document.lastModified | Returns the date and time the document was updated |
| document.links | Returns all <area> and <a> elements that have a href attribute |
| document.readyState | Returns the (loading) status of the document |
| document.referrer | Returns the URI of the referrer (the linking document) |
| document.scripts | Returns all <script> elements |
| document.strictErrorChecking | Returns if error checking is enforced |
| document.title | Returns the <title> element |
| document.URL | Returns the complete URL of the document |

# DOM Elements

Finding HTML Elements

- Finding HTML elements by id

```
const element = document.getElementById("intro");
```

- Finding HTML elements by tag name

```
const element = document.getElementsByTagName("p");
```

- Finding HTML elements by class name

```
const x = document.getElementsByClassName("intro");
```

- Finding HTML elements by CSS selectors

```
const x = document.querySelectorAll("p.intro");
```

- Finding HTML elements by HTML object collections

```
const x = document.forms["frm1"];
let text = "";
for (let i = 0; i < x.length; i++) {
  text += x.elements[i].value + "<br>";
}
document.getElementById("demo").innerHTML = text;
```

# DOM HTML

Changing HTML Content

```
document.getElementById(id).innerHTML = new HTML
```

Changing the Value of an Attribute

```
document.getElementById("myImage").src = "landscape.jpg";
```

Dynamic HTML content

```
<script>
document.getElementById("demo").innerHTML = "Date : " + Date();
</script>
```

document.write()

# DOM Forms

JavaScript Form Validation
HTML form validation can be done by JavaScript.

If a form field (fname) is empty, this function alerts a message, and returns false, to prevent the form from being submitted:

```
function validateForm() {
  let x = document.forms["myForm"]["fname"].value;
  if (x == "") {
    alert("Name must be filled out");
    return false;
  }
}
```

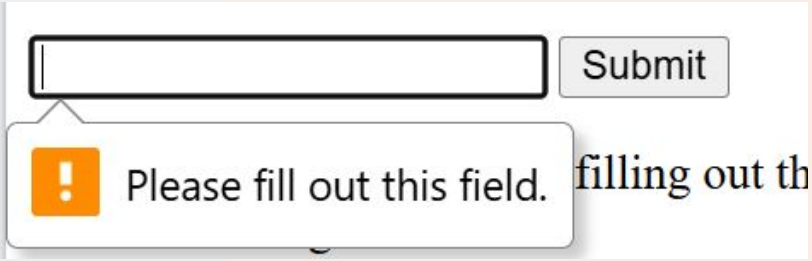
```
<form name="myForm" action="/action_page.php" onsubmit="return validateForm()" method="post">
Name: <input type="text" name="fname">
<input type="submit" value="Submit">
</form>
```

# DOM Forms

HTML form validation can be performed automatically by the browser:

If a form field (fname) is empty, the required attribute prevents this form from being submitted:

```
<form action="/action_page.php" method="post">
  <input type="text" name="fname" required>
  <input type="submit" value="Submit">
</form>
```



# DOM Form

Data validation is the process of ensuring that user input is clean, correct, and useful.

Typical validation tasks are:

- has the user filled in all required fields?
- has the user entered a valid date?
- has the user entered text in a numeric field?
- Most often, the purpose of data validation is to ensure correct user input.

**Server side validation** is performed by a web server, after input has been sent to the server.

**Client side validation** is performed by a web browser, before input is sent to a web server.

# DOM Forms

## Constraint Validation HTML Input Attributes

| Attribute | Description |
| --- | --- |
| disabled | Specifies that the input element should be disabled |
| max | Specifies the maximum value of an input element |
| min | Specifies the minimum value of an input element |
| pattern | Specifies the value pattern of an input element |
| required | Specifies that the input field requires an element |
| type | Specifies the type of an input element |

# DOM CSS

Changing HTML Style

```
document.getElementById(id).style.property = new style
```

```
document.getElementById("p2").style.color = "blue";
```

# DOM Events

Reacting to Events

Examples of HTML events:

- When a user clicks the mouse
- When a web page has loaded
- When an image has been loaded
- When the mouse moves over an element
- When an input field is changed
- When an HTML form is submitted
- When a user strokes a key

```
<h1 onclick="changeText(this)">Click on this text!</h1>

<script>
function changeText(id) {
  id.innerHTML = "Ooops!";
}
</script>
```

```
<script>
document.getElementById("myBtn").onclick = displayDate;
</script>
```

```
<body onload="checkCookies()">
```

```
<input type="text" id="fname" onchange="upperCase()">
```

```
<input type="text" id="fname" oninput="upperCase()">
```

# DOM EventListener

The addEventListener() method

- Attaches an event handler to the specified element
- You can add many event handlers to one element.
- Add event listeners to any DOM object not only HTML elements.
- JavaScript is separated from the HTML markup, for better readability and allows you to add event listeners even when you do not control the HTML markup
- Remove an event listener by using the removeEventListener() method

# DOM EventListener

```
element.addEventListener("mouseover", myFunction);
element.addEventListener("click", mySecondFunction);
element.addEventListener("mouseout", myThirdFunction);
```

```
window.addEventListener("resize", function(){
  document.getElementById("demo").innerHTML = sometext;
});
```

```
element.removeEventListener("mousemove", myFunction);
```

# DOM Navigation

With the HTML DOM, you can navigate the node tree using node relationships.

You can use the following node properties to navigate between nodes with JavaScript:

- parentNode
- childNodes[nodenumber]
- firstChild
- lastChild
- nextSibling
- previousSibling

```
myTitle = document.getElementById("demo").firstChild.nodeValue;
```

```
myTitle = document.getElementById("demo").childNodes[0].nodeValue;
```

# DOM Elements (nodes)

Create New HTML Elements(Nodes)

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

<script>
const para = document.createElement("p");
const node = document.createTextNode("This is new.");
para.appendChild(node);

const element = document.getElementById("div1");
element.appendChild(para);
</script>
```

This is a paragraph.

This is another paragraph.

This is new.

# DOM Elements (nodes)

Removing a Child Node

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

<script>
const parent = document.getElementById("div1");
const child = document.getElementById("p1");
parent.removeChild(child);
</script>
```

This is a paragraph.

This is another paragraph.

# DOM Elements (nodes)

## Replacing HTML Elements

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

<script>
const para = document.createElement("p");
const node = document.createTextNode("This is new.");
para.appendChild(node);

const parent = document.getElementById("div1");
const child = document.getElementById("p1");
parent.replaceChild(para, child);
</script>
```

# DOM Collections

The getElementsByTagName() method returns an **HTMLCollection** object.

An HTMLCollection is NOT an array!

An HTMLCollection may look like an array, but it is not.

You can loop through the list and refer to the elements with a number (just like an array).

However, you cannot use array methods like valueOf(), pop(), push(), or join() on an HTMLCollection.

The getElementsByClassName() and getElementsByTagName() methods return a live HTMLCollection.

# DOM Node List

A **NodeList** object is a list (collection) of nodes extracted from a document.

A **NodeList** object is almost the same as an **HTMLCollection** object.

The querySelectorAll() method returns a static NodeList.

The childNodes property returns a live NodeList.

```
const myNodeList = document.querySelectorAll("p");
```

# Home work

Comparing between NodeList and Node Collections