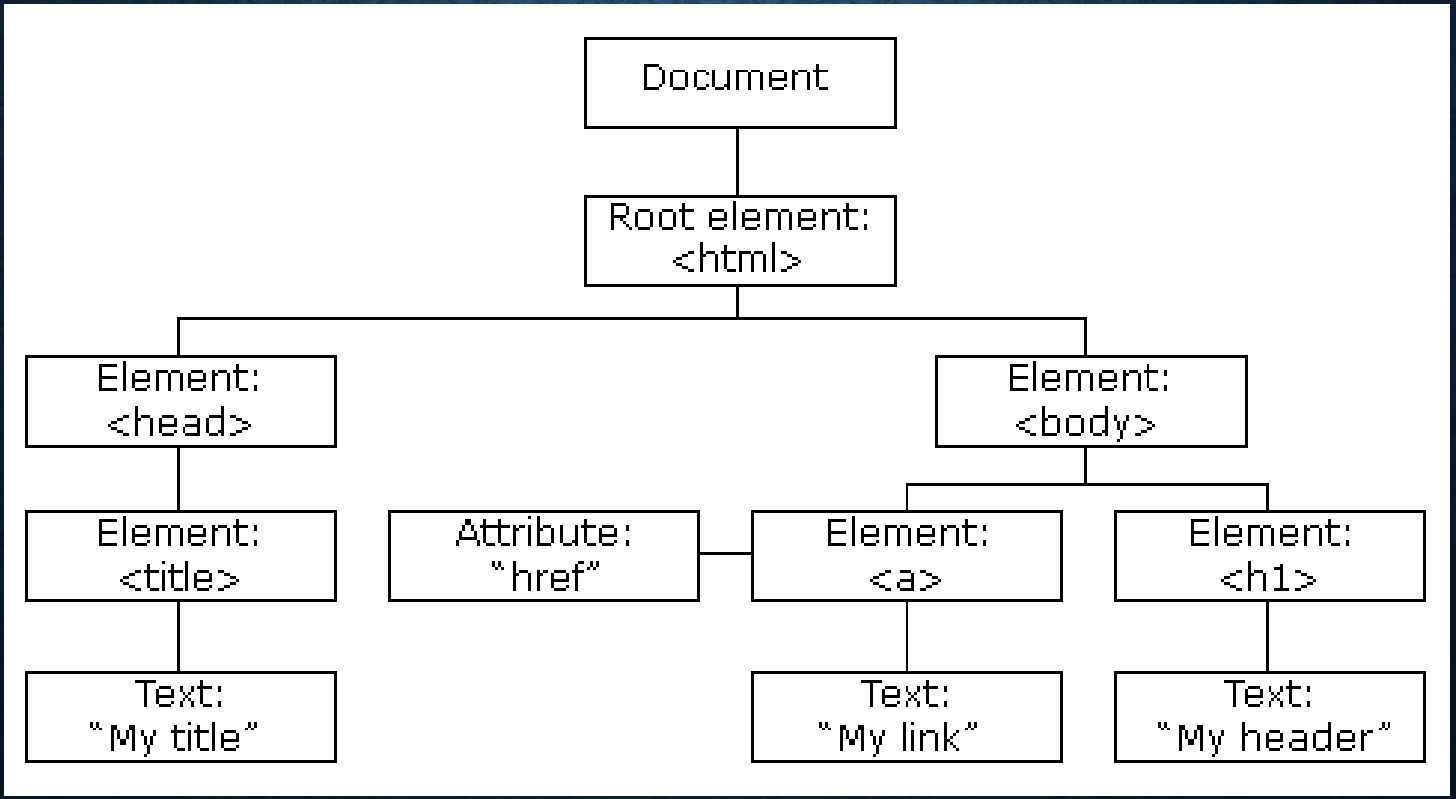
WEB TECHNOLOGY
LECTURE - 07

DOCUMENT OBJECT MODEL (DOM)

- When an HTML document is loaded into a web browser, it becomes a document object (global).
- It organize the whole HTML document in a tree structure where the document object is the root node of it.
- Therefore, anything inside an HTML document can be accessed, changed, deleted, or added by JS through the **document** object.



JS SELECTORS

There exist multiple ways to identify an element in JS

1. **getElementById**
2. **getElementsByTagName**
3. **getElementsByClassName**
4. **getElementsByName**
5. **querySelector**
6. **querySelectorAll**

querySelector/querySelectorAll uses css selector style to access an element.

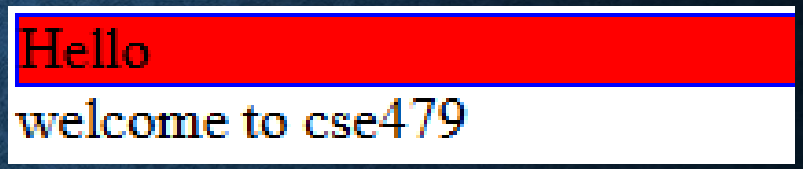
GETELEMENTBYID

- This method uses the id attribute to access an element.

Example (HTML):

```
<span id= "my_element" >Hello</span>
<span> Welcome to CSE479</span>
```

JS:




```
var myVariable = document.getElementById("my_element");
myVariable.style.display = "block";
myVariable.style.backgroundColor = "#f00";
myVariable.style.border = "solid 1px #00f";
```

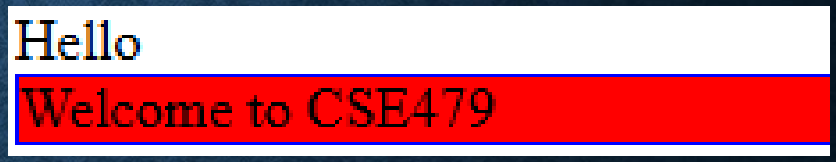
GETELEMENTSBYTAGNAME

- getElementsByTagName allows you looking for all the elements on your page with a specified tag name. It returns an HTMLCollection.
- **Example:**

Hello

** Welcome to CSE479**

JS:




```
var spanList = document.getElementsByTagName("span");
spanList[1].style.display = "block";
spanList[1].style.backgroundColor = "#f00";
spanList[1].style.border = "solid 1px #00f";
```

GETELEMENTSBYTAGNAME

- You can also use loop to iterate over the list
- For example, following code sets a class name "link_class" to all the anchors in a webpage.
- **Example**

```
var myLinkCollection = document.getElementsByTagName("a");
for (i = 0; i < myLinkCollection.length; i++) {
    myLinkCollection[i].className ="link_class“
}
```

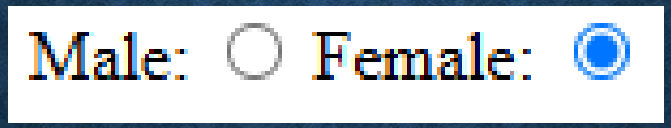
GETELEMENTSBYNAME

- This method uses the name attribute to access all the elements.

Example:

Male: <input name="gender" type= "radio" >

Female: <input name="gender" type= "radio" >



JS:

document.getElementsByName("gender")[1].checked = true

QUERYSELECTOR

- querySelector used to select single elements from the DOM using CSS selectors.
- Returns the first matching element in the DOM.
- If no matching element is found, it returns null.

```
<div id="container">
    <p>First paragraph</p>
    <p>Second paragraph</p>
</div>
```

Script

```
const firstPara = document.querySelector("#container p");
console.log(firstPara.innerText); // Output: "First paragraph“
```

Note : you do not always need the document, i.e., document can be replaced by another element/node in DOM, i.e., container.

Example:

```
const box = document.querySelector(".box")
box.querySelector("p“) // it will selects the <p> inside the box element
```

QUERYSELECTORALL

- Returns a NodeList of all matching elements.
- The returned NodeList is static

```
<div id="container">
    <p>First paragraph</p>
    <p>Second paragraph</p>
</div>
```

Script:

```
const allParas = document.querySelectorAll("#container p");
console.log(allParas.length); // Output: 2
```

* Note: unlike HTMLCollection (from getelementsByTagName) nodelist doesn't update if elements are later added/removed.

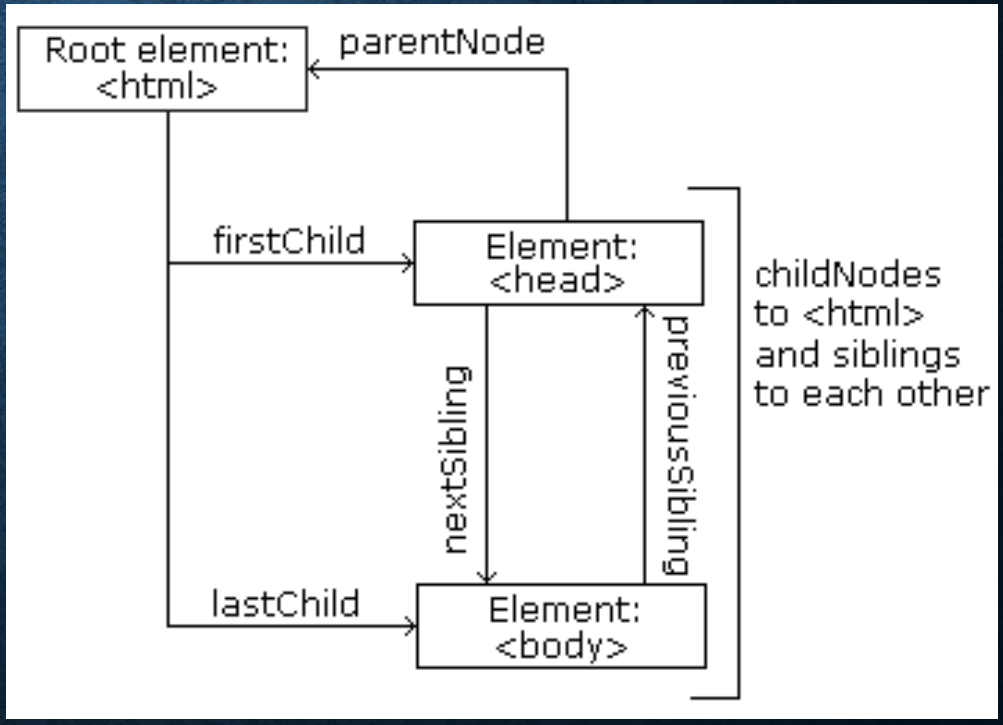
CSS SELECTORS

Selector	Meaning
`#title`	Selects an element by ID
`.info`	Selects elements with the class "info"
`button`	Selects the `<button>` element
`a[href='#']`	Selects an anchor with an attribute `href="#"`
`.box > p`	Selects direct child `<p>` inside `.box`
`div p`	Selects all descendant `<p>` inside `<div>`
`.info + p`	Selects the adjacent sibling `<p>` after `.info`
`.info ~ *`	Selects all siblings after `.info`
`ul li:first-child`	Selects the first child `<li>` inside `<ul>`
`ul li:last-child`	Selects the last child `<li>` inside `<ul>`
`ul li:nth-child(2)`	Selects the second `<li>` inside `<ul>`

NODE'S METHODS

NODE METHODS

- A "node" is essentially any element on your page within the DOM structure.
- The different node methods available for DOM navigation & manipulation are as follows:
  - **node.childNodes**
  - **node.firstChild**
  - **node.lastChild**
  - **node.parentNode**
  - **node.nextSibling**
  - **node.previousSibling**
  - **and more**



CHILDREN VS CHILDNODES

Node.children method:

- Returns only element nodes (ignores text nodes, comments, etc.).
- It is an HTMLCollection.

```
<div id="list">
    <div>Item 1</div>
    <div>Item 2</div>
</div>
```

Example:

```
const list = document.getElementById("list");
console.log(list.children); // Returns HTMLCollection
                            //   [div, div]
```

CHILDREN VS CHILDNODES

Node.childNodes method

- Returns all child nodes, including:
  1. Element nodes (, , etc.)
  2. Text nodes (whitespace, new lines)
  3. Comment nodes (<!-- comment -->)
- It is a NodeList.

```
<div id="list">
    <div>Item 1</div>
    <div>Item 2</div>
</div>
```

Example:

```
const list = document.getElementById("list");
console.log(list.childNodes); // Returns NodeList [text,
                              //div, text, div, text]
```

MORE EXAMPLE

```
<ul id="list">
    <li><a href="link1.html" class="link_one">Link Number
    One</a></li>
    <li><a href="link2.html">Link Number Two</a></li>
    <li><a href="link3.html">Link Number Three</a></li>
    <li><a href="link4.html">Link Number Four</a></li>
</ul>
```

JS CODE

```
var myLinkList =
document.getElementById("list");
var myFirstLink =
myLinkList.childNodes[0].childNodes[0];
alert(myFirstLink.className);
```

```
var myLinkList =
document.getElementById("list");
var myFirstLink =
myLinkList.firstChild.firstChild.nextSibling.
previousSibling;
alert(myFirstLink.className);
```

```
var myLinkList = document.getElementById("list");
var myFirstLink = myLinkList.firstChild.firstChild;
alert(myFirstLink.className);
```

CREATEELEMENT

- It creates an element and allows you to place that new element anywhere in the DOM structure.
- **Example**

```
var myNewListItem = document.createElement("li");
myNewListItem.innerText = "item1“;
```

- The above code creates a new <li> element.
- But these elements don't exist anywhere except as values inside variables.
- To see it’s existence you must append it.

```
document.getElementById("list").appendChild(myNewListItem);
```

REMOVECHILD

- In order to remove an element from DOM we use this method.
- **Example**

```
var myLinkList = document.getElementById("list");
var myRemovedLink = myLinkList.lastChild;
myLinkList.removeChild(myRemovedLink);
```

In jQuery simply we can use

```
$('.list').remove('selector')
```

GETATTRIBUTE

- The getAttribute method allows you to access the value of any attribute on any element on your page.
- **Example**

```
<ul id="list">
  <li><a href="link1.html" class="link_one">Link Number One</a></li>
  <li><a href="link2.html">Link Number Two</a></li>
  <li><a href="link3.html">Link Number Three</a></li>
  <li><a href="link4.html">Link Number Four</a></li>
  <li><a href="link5.html" id="link_5" rel="external">Link Number
  Five</a></li>
</ul>
```

```
var myLinkFive = document.getElementById("link_5");
var myLinkAttribute = myLinkFive.getAttribute("rel");
```

- In jQuery it is simply: $('. *link_5* ').attr('rel');

SETATTRIBUTE

- **Example**

```
<ul id="list">
  <li><a href="link1.html" class="link_one">Link Number One</a></li>
  <li><a href="link2.html">Link Number Two</a></li>
  <li><a href="link3.html">Link Number Three</a></li>
  <li><a href="link4.html">Link Number Four</a></li>
  <li><a href="link5.html" id="link_5" rel="external">Link Number
  Five</a></li>
</ul>
var myLinkFive = document.getElementById("link_5");
myLinkFive.setAttribute("href", “www.google.com");
```

MORE ABOUT SCRIPT

- It is recommended that you put JavaScript at the bottom of body.
- However, due to performance or other reasons you may require put script inside the head section. Consider the following code

```
<html>
<script src="myScript.js"></script>
<body>
<p id="demo"></p>
</body>
</html>
```

- And the script contains:
  document.getElementById("demo").innerHTML = "Hello World";
- Running the code will cause you error. Why?

DEFER AND ASYNC

The solution is to use the defer keyword.

```
<html>
<script src="myScript.js defer"></script>
<body>
<p id="demo"></p>
</body>
</html>
```

There is also async keyword. The differences are:

Attribute	HTML Parsing	Script Execution	Best Use Case
No Attribute	❌ Blocked until script loads	Executes immediately	Small inline scripts
`async`	✅ Continues parsing	Executes **ASAP** (before HTML is fully loaded)	Independent scripts (analytics, ads)
`defer`	✅ Continues parsing	Executes **after HTML is fully loaded**	Most scripts (DOM interactions)

TO DO LIST

Explore this by this TODO App as a real example of DOM manipulations.

THANK YOU