KINGSLAND UNIVERSITY

DOM

DOM

Document Object Model

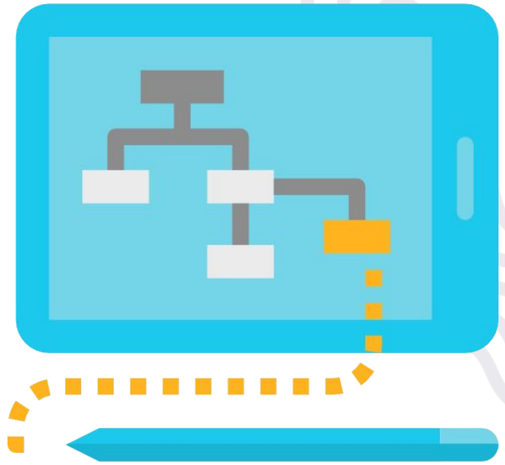
Table of Contents

Document with a Logical Tree

Document Object Model (DOM)

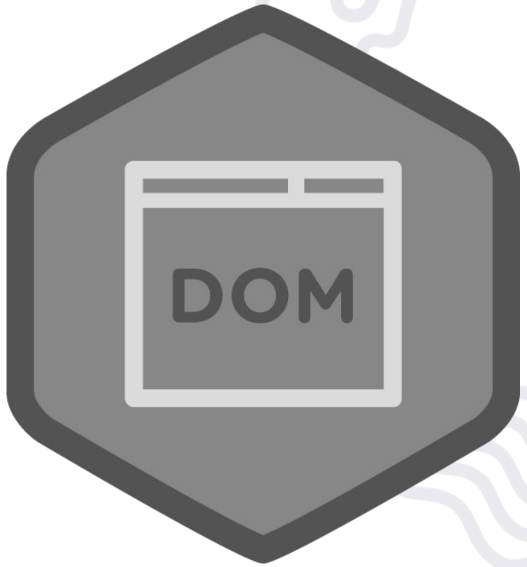
Document Object Model

- The **DOM** represents the document as **nodes** and **objects**
  - That way, the programming languages **can connect** to the page
- **DOM** is a **standard** of how to:
  - **Get** HTML element
  - **Change** HTML element
  - **Add** HTML element
  - **Delete** HTML element



HTML DOM

- The **HTML DOM** is an **Object Model** for **HTML**. It defines:
  - HTML elements as **objects**
  - **Properties** for all HTML elements
  - **Methods** for all HTML elements
  - **Events** for all HTML elements



Changing the HTML

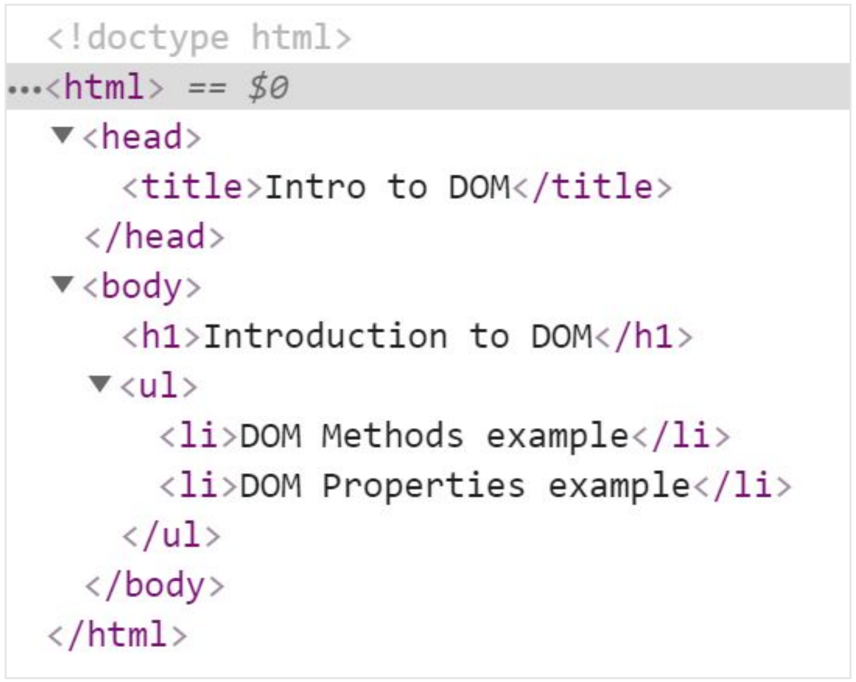
DOM Methods

DOM Methods

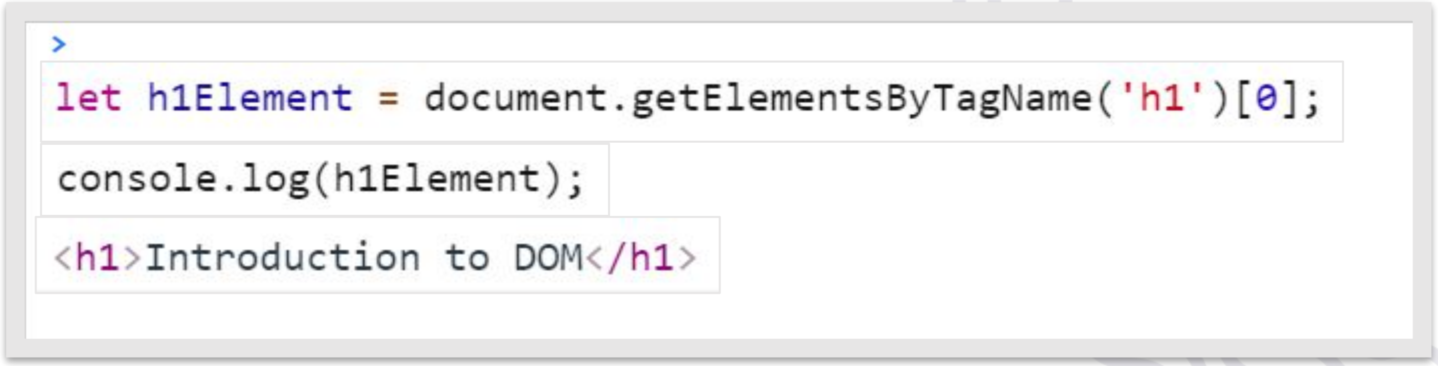
- **DOM Methods** - **actions** you can perform on HTML elements
- **DOM Properties** - values of HTML elements that you can **set** or **change**

Example: DOM Methods

- HTML DOM **method** is an action you can do (like **add** or **delete** an HTML element)



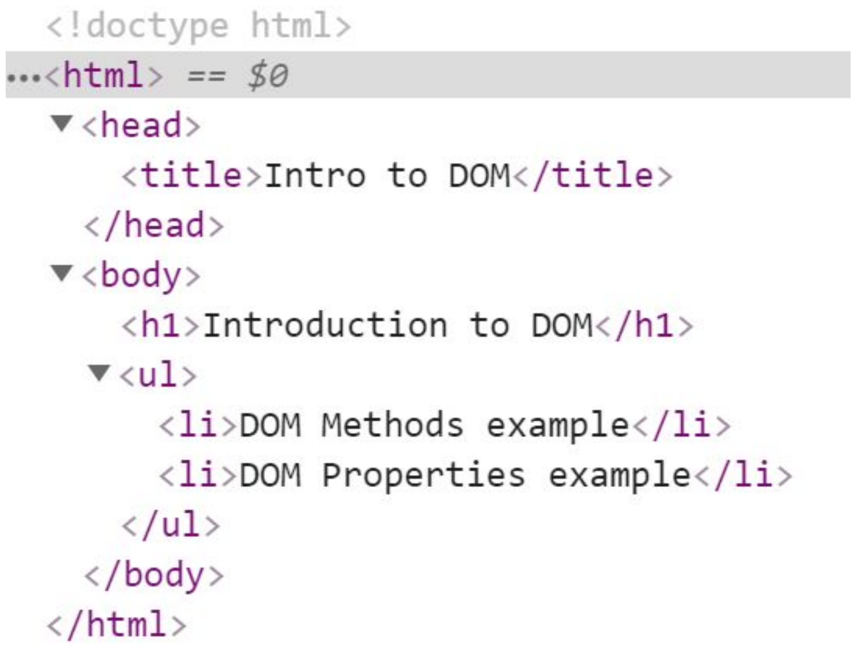
```
<!doctype html>
...<html> == $0
  ▼<head>
     <title>Intro to DOM</title>
   </head>
  ▼<body>
     <h1>Introduction to DOM</h1>
    ▼<ul>
       <li>DOM Methods example</li>
       <li>DOM Properties example</li>
     </ul>
   </body>
 </html>
```



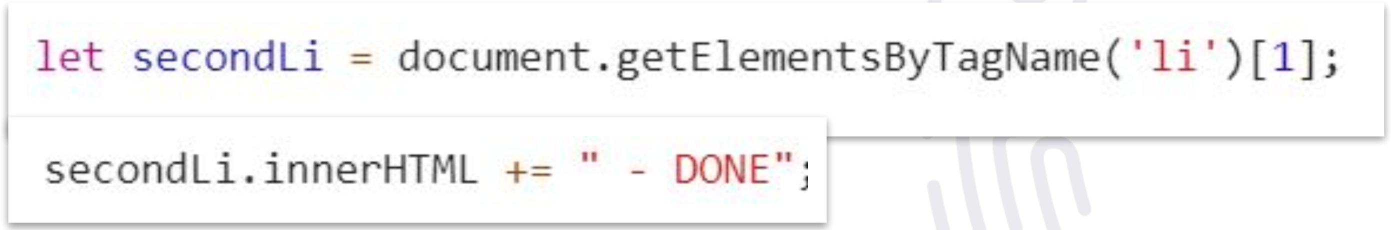
```
>
let h1Element = document.getElementsByTagName('h1')[0];
console.log(h1Element);
<h1>Introduction to DOM</h1>
```

Example: DOM Methods

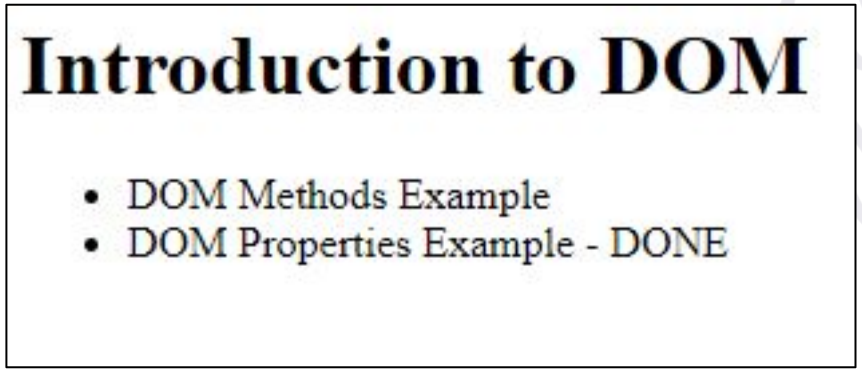
- HTML DOM **property** is a value that you can **get** or **set** (changing the content of an HTML element)



```
  <!doctype html>
...<html> == $0
  ▼<head>
      <title>Intro to DOM</title>
    </head>
  ▼<body>
      <h1>Introduction to DOM</h1>
    ▼<ul>
        <li>DOM Methods example</li>
        <li>DOM Properties example</li>
      </ul>
    </body>
  </html>
```



```
let secondLi = document.getElementsByTagName('li')[1];
secondLi.innerHTML += " - DONE";
```



Introduction to DOM

- DOM Methods Example
- DOM Properties Example - DONE

Modify the DOM Tree

DOM Manipulations

Selection of Elements

- There are a few ways to **find** a certain **HTML element** in the **DOM**:
  - By id - **`getElementById()`**
  - By tag name - **`getElementsByTagName()`**
  - By class name - **`getElementsByClassName()`**
  - By CSS selector - **`querySelector()`**

CSS Selectors

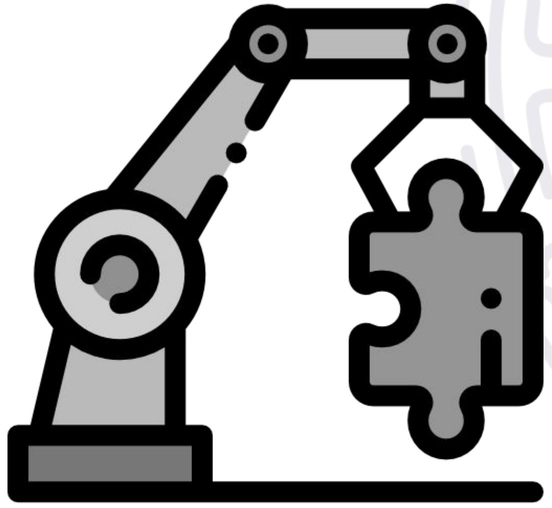
- CSS selectors are strings that follow CSS syntax for matching
- They allow very fast and powerful element matching, e.g.:
  - **"#main"** - returns the element with ID "main"
  - **"#content div"** - selects all **<div>**s inside **#content**
  - **".note, .alert"** - all elements with class "note" or "alert"
  - **"input[name='login']"** - **<input>** with name "login"

DOM Manipulations

- The **HTML DOM** allows JavaScript to change the content of **HTML elements**
  - **`innerHTML`**
  - **`attributes`**
  - **`setAttribute()`**
  - **`style.property`**

DOM Manipulations

- We can **create**, **append** and **remove** HTML elements dynamically
  - **`removeChild()`**
  - **`appendChild()`**
  - **`replaceChild()`**
  - **`document.write()`**



Creating DOM Elements

- Creating a new DOM element

```
let p = document.createElement("p");
let li = document.createElement("li");
```

Tag name

- Create a copy / cloning DOM element

```
let li = document.getElementById("my-list");
let newLi = li.cloneNode(true);
```

- The above code **creates a new elements**. But these elements **don't exist** anywhere except as values inside variables

Deleting DOM Elements

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

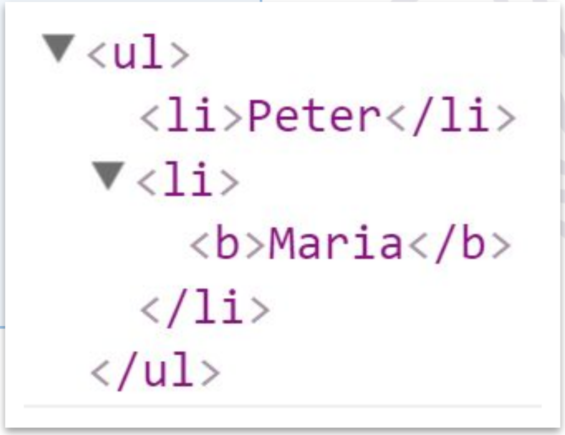
```
let parent = document.getElementById("div1");
let firstChild = document.getElementById("p1");
let secondChild = document.getElementById("p2");

firstChild.remove();
parent.removeChild(secondChild);
```

Directly deleting

Deleting by parent element

Creating DOM Elements

```
let list = document.createElement("ul");
let firstLi = document.createElement("li");
firstLi.textContent = "Peter";
list.appendChild(firstLi);
let secondLi = document.createElement("li");
secondLi.innerHTML = "<b>Maria</b>";
list.appendChild(secondLi);
document.body.appendChild(list);
```



```
▼<ul>
   <li>Peter</li>
 ▼<li>
     <b>Maria</b>
   </li>
 </ul>
```

DOM Properties and HTML Attributes

Properties vs. Attributes

- Attributes are defined by **HTML**. Properties are defined by the **DOM**
- Attributes **initialize** DOM properties
  - **Property** values can **change**
  - **Attribute** values **can't**
- The HTML **attribute** and the DOM **property** are **not the same thing**, even when they have the same name

DOM Properties

- **`textContent`** - reads and writes

```
let text = Node.textContent;
Node.textContent = 'New text for element.';
```

- **`innerHTML`** - returns and writes the **HTML** of a given element

```
let html = myElement.innerHTML;
myElement.innerHTML = 'New text for element.';
```

- **`value`** - gets and sets

```
let theValue = theFormField.value;
theFormField.value = 'New value';
```

HTML Attributes and Methods

- **`getAttribute()`** - returns the value of attributes of specified HTML element

```
<input type="text" name="username"/>
<input type="password" name="password"/>
```

```
const inputEle = document.getElementByTagName('input')[0];
inputEle.getAttribute('type'); // text
inputEle.getAttribute('name'); // username
```

HTML Attributes and Methods

- **`setAttribute()`** - sets the value of an attribute on the specified HTML element

```
<input type="text" name="username"/>
<input type="password" />
```

```
const inputPassEle = document.getElementsByTagName('input')[1];
inputPassEle.setAttribute('name', 'password');
```

```
<input type="text" name="username"/>
<input type="password" name="password"/>
```

HTML Attributes and Methods

- **`removeAttribute()`** - removes the attribute with the specified name from an HTML element

```
<input type="text" name="username" placeholder="Username..."/>
<input type="password" name="password" placeholder="Password..."/>
```

```
const inputPassEle = document.getElementsByTagName('input')[1];
inputPassEle.removeAttribute('placeholder');
```

```
<input type="text" name="username" placeholder="Username..."/>
<input type="password" name="password"/>
```

HTML Attributes and Methods

- **`hasAttribute()` -** method returns true if the specified attribute exists, otherwise it returns false

```
<input type="text" name="username" placeholder="Username..."/>
<input type="password" name="password" id="password"/>
```

```
const passwordElement = document.getElementById(password');
passwordElement.hasAttribute('name'); // true
passwordElement.hasAttribute('placeholder'); // false
```

HTML Attributes and Methods

- **`classList`** - is a read-only property that returns a collection of the class attributes of specified element

```
<div class="container div root"></div>
```

```
const element = document.getElementById('myDiv').classList;
// DOMTokenList(3)
["container", "div", "root", value: "container div root"]
```

HTML Attributes and Methods

- `classList` **Methods**

```
<div class="container div root"></div>
```

  - **`add()` -** Adds the specified class values

```
document.getElementById('myDiv').classList.add('testClass');
```

  - **`remove()` -** Removes the specified class values

```
document.getElementById('myDiv').classList.remove('container');
```

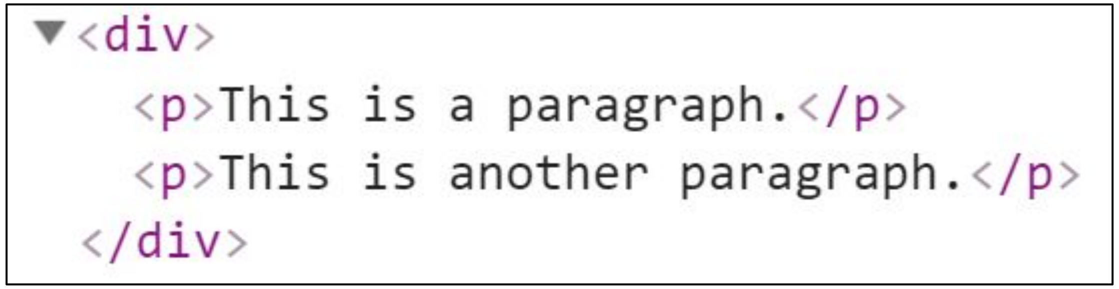
```
<div class="div root testClass"></div>
```

Parents and Child Elements

Parents and Child Elements

- Every DOM Elements has a **parent**

- Parents can be accessed by keywords **.parent** or **.parentNode**

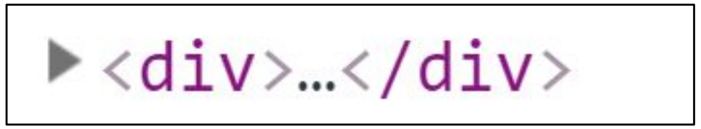


```
▼<div>
    <p>This is a paragraph.</p>
    <p>This is another paragraph.</p>
  </div>
```

```
let firstP = document.getElementsByTagName('p')[0];
console.log(firstP.parent);
```

Accessing the first child

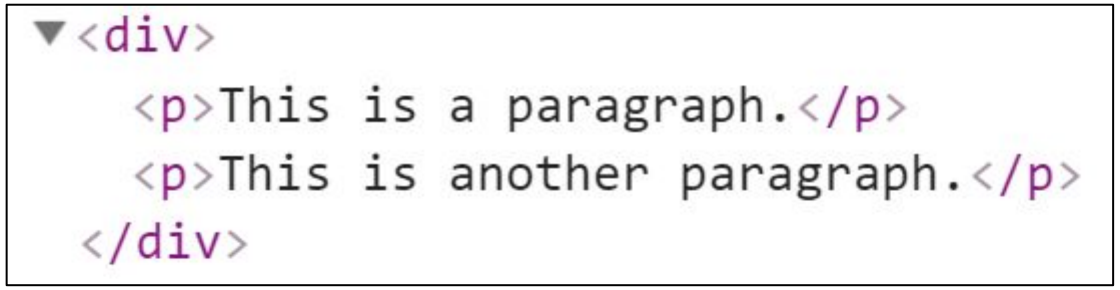
Accessing the child parent



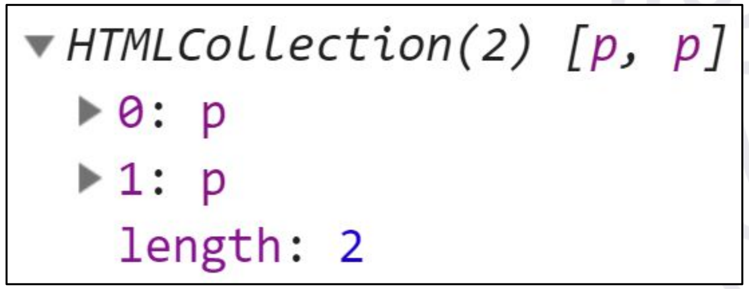
```
▶<div>…</div>
```

Parents and Child Elements

- When some element contains other elements, that means he is **parent** of this elements
- Also this elements is **children** to the **parent**. They can be accessed by keyword **`.children`**



```
▼<div>
    <p>This is a paragraph.</p>
    <p>This is another paragraph.</p>
  </div>
```



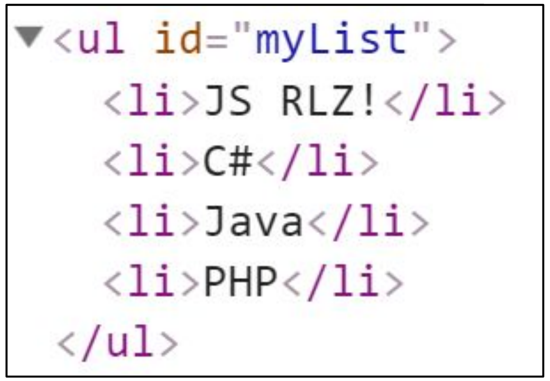
```
▼HTMLCollection(2) [p, p]
  ▶0: p
  ▶1: p
   length: 2
```

```
let pElements = document.getElementsByTagName('div')[0].children;
```

Returns HTML Collection

Parents and Child Elements

- **`firstElementChild`** - Returns the **first** child node of an element
- **`lastElementChild`** - Returns the **last** child node of an element



```
▼<ul id="myList">
   <li>JS RLZ!</li>
   <li>C#</li>
   <li>Java</li>
   <li>PHP</li>
 </ul>
```

```
let list = document.getElementById('myList');
```

```
list.firstElementChild;
list.lastElementChild;
```

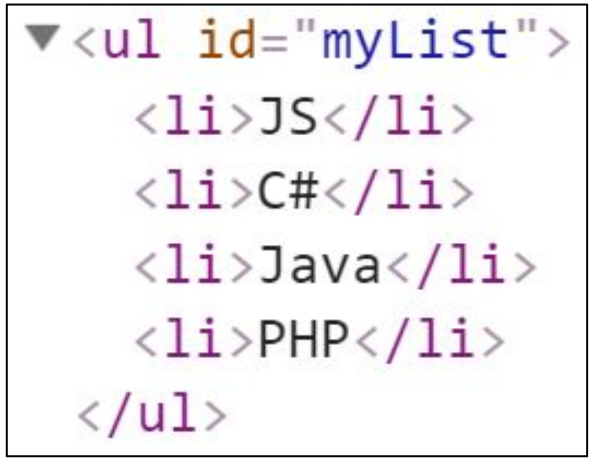
```
<li>JS</li>
```

```
<li>PHP</li>
```

```
list.firstElementChild.textContent += " RLZ!";
```

Parents and Child Elements

- **`nextElementSibling`** - Returns the **next** node at the same node tree level
- **`previousElementSibling`** - Returns the **previous** node at the same node tree level



```
▼<ul id="myList">
    <li>JS</li>
    <li>C#</li>
    <li>Java</li>
    <li>PHP</li>
  </ul>
```

```
let ul = document.getElementById('myList');
let next = ul.children[0].nextElementSibling;
console.log(next.textContent); // C#
let prev = next.previousElementSibling;
console.log(prev.textContent); // JS
```

Parents and Child Elements

- **`appendChild` -** Adds a new child, as the **last child**

```
let p = document.createElement("p");
let li = document.createElement("li");
li.appendChild(p);
```

- **`prepend` -** Adds a new child, as the **first child**

```
let ul = document.getElementById("my-list");
let li = document.createElement("li");
ul.prepend(li);
```

NodeList vs. HTMLCollection

- Both interfaces are **collections** of **DOM nodes**
- **NodeList** can contain **any** node type
- **HTMLCollection** is supposed to **only** contain **Element nodes**
- An **HTMLCollection** provides the **same methods** as a NodeList
  and **additionally** a method called **namedItem**

Handling DOM Events

DOM Events

DOM Events

- Events are **actions** or **occurrences**
- They allow JavaScript to register different **event handlers** on elements
- Events are normally used in combination with **functions**, and the function will not be executed before the event occurs

```
htmlRef.addEventListener( 'click' , handler );
```

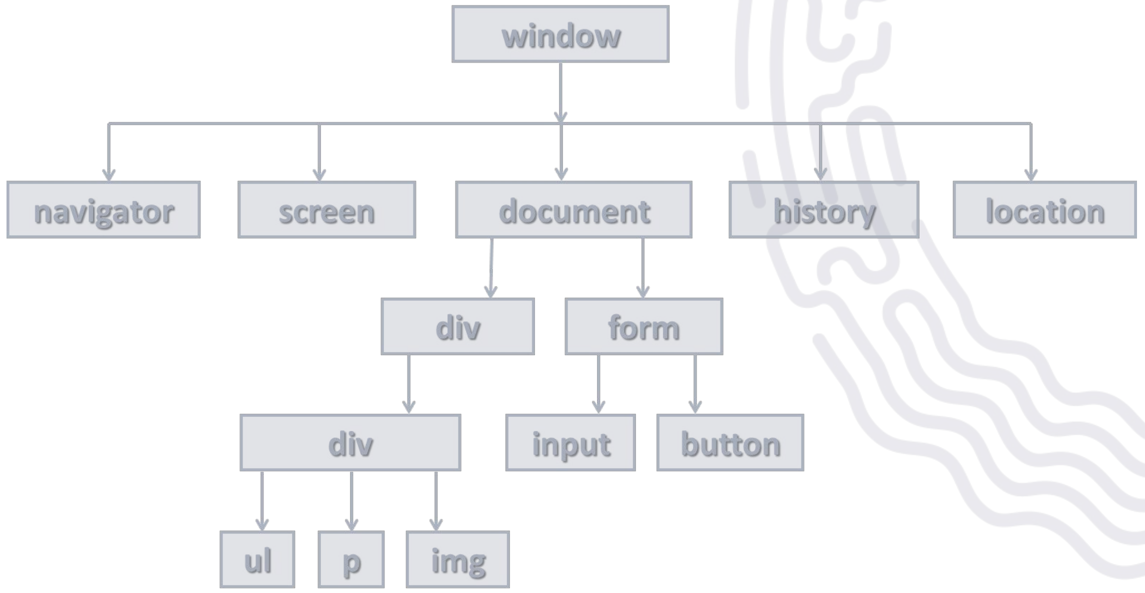
The Built-In Browser Objects

Browser Object Model (BOM)

Browser Object Model (BOM)

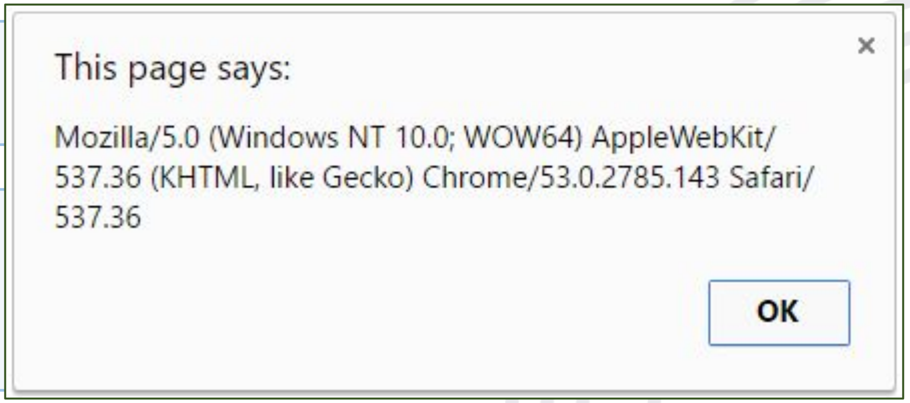
- Browsers expose some objects like **`window`**, **`screen`**, **`navigator`**, **`history`**, **`location`**, **`document`**, …

```
console.dir(window);
console.dir(navigator);
console.dir(screen);
console.dir(location);
console.dir(history);
console.dir(document);
```



Playing with BOM

```
alert(window.navigator.userAgent);
```


This page says:

Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/53.0.2785.143 Safari/537.36

```
console.log(navigator.language);
// en-US
```

```
console.log(screen.width + " x " + screen.height);
// 1920 x 1080
```

```
document.location = "https://kingslanduniversity.com";
```

```
history.back();
```

Summary

- DOM
  - DOM is a programming API for HTML and XML documents
  - DOM Methods and Properties
  - DOM Manipulations
- BOM

Questions?

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © Kingsland University – https://kingslanduniversity.com

KINGSLAND UNIVERSITY

THANK YOU