Document Object Model (DOM)
The Document Object Model (DOM) is a W3C standard that is a platform- and language-neutral interface which allows programs and scripts to dynamically access and update the content, structure and presentation of documents. Web browsers use the model to render HTML documents. The DOM is also required by JavaScript scripts to interact with a web page dynamically.

The DOM is consisted of 3 different parts:

- Core DOM: the standard model for any structured document
- XML DOM: the standard model for XML documents
- HTML DOM: the standard model for HTML documents

The last one is the one we are most interested in right now.
The HTML DOM views an HTML document as a hierarchical tree structure.

The tree structure is called a **node tree**.

```html
<html>
  <head>
    <title> My title </title>
  </head>
  <body>
    <a href="./myFile.html"> My link </a>
    <h1> My header </h1>
  </body>
</html>
```
Node Trees and the document Object

In a node tree, HTML elements are embedded in other HTML elements, and the elements are the nodes in the tree. By traversing this tree, one could access, modify, or delete elements and their contents.

When JavaScript accesses an HTML document, it treats the elements of the document as objects that have both attributes (data or properties) and methods (operations or functions).

The JavaScript object document represents the entire page DOM, created from the document's HTML. Changes made to document are reflected in the browser presentation and/or behavior.
The window Object

JavaScript running in a web browser has access to the window object, which represents an open browser window. In a tabbed browser, each tab has its own window object.

The document object is a property of the window object, and can be accessed formally as window.document, or informally as just document.

Other useful properties of the window object include:

- window.location: information about the window's current URL. For example, window.location.hostname is the URL's hostname, and window.location.protocol is "http:" or "https:".

- window.innerHeight and window.innerWidth: the height and width of the window's content area, in pixels.
Some useful methods associated with the window object include:

- `window.alert(message)`: displays an alert dialog box. We've already seen `window.alert("Hello World!")`. (Like with document, we don't have to specify window.)

- `window.confirm(prompt)`: displays a confirmation dialog box with OK and Cancel buttons. `confirm("Are you sure?")` returns `true` if "OK" is pressed, and `false` if "Cancel" is pressed.

- `window.open(url,name,properties)`: opens a new browser window with the URL specified as an argument. (Note that name does not mean "title".)
Methods for finding nodes within the DOM

These methods all operate on the `document` object, such as `document.getElementById()`.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getElementById()</code></td>
<td>return the DOM node whose <code>id</code> attribute matches the parameter</td>
</tr>
<tr>
<td><code>getElementsByTagName()</code></td>
<td>return an array containing all of the DOM nodes whose tag type matches the parameter</td>
</tr>
<tr>
<td><code>getElementsByClassName()</code></td>
<td>return an array containing all of the DOM nodes whose <code>class</code> attribute matches the parameter</td>
</tr>
<tr>
<td><code>querySelector()</code></td>
<td>returns the FIRST element found in the DOM whose tag is one of the tags listed in the parameter (such as &quot;td th tr&quot;)</td>
</tr>
<tr>
<td><code>querySelectorAll()</code></td>
<td>return an array containing all of the DOM nodes whose tags are one of the tags listed in the parameter (such as &quot;td th tr&quot;)</td>
</tr>
</tbody>
</table>
## Properties for navigating from one node to another

These properties all operate on a specified node, such as "p".

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p.parentNode</td>
<td>the node p's parent element</td>
</tr>
<tr>
<td>p.childNodes</td>
<td>an array of the children of node p (including whitespace)</td>
</tr>
<tr>
<td>p.children</td>
<td>an array of the children of node p (not including the whitespace). This is the one you probably want to use.</td>
</tr>
<tr>
<td>p.nextSibling</td>
<td>the node with the same parent following the node p in the document</td>
</tr>
<tr>
<td>p.prevSibling</td>
<td>the node with the same parent preceding the node p in the document</td>
</tr>
</tbody>
</table>
Dynamic HTML
Dynamic HTML (DHTML) can be used to enhance the interactivity of a web page. The interactivity is at the client end, without any extra transactions to the server.

DHTML is a combination of four different technologies:

\[
\text{DHTML} = \text{HTML} + \text{CSS} + \text{JavaScript} + \text{DOM}
\]

In the normal sequence of events:

- The browser parses an HTML document and renders it.
- A user event causes a JavaScript function to be executed by the browser.
- The function changes the state of the DOM, and this causes an instantaneous change in the Web page rendition.
Dynamic HTML

The DOM can manipulate windows, HTML forms, and create rollover images.

Objects such as *forms* and *images* are found at predictable locations in the object tree.

Elements like *span* and *div* allow style specification through CSS.

The HTML DOM allows elements to be accessed through a unique ID. The method *getElementById()* returns a reference to the particular HTML element. By using the reference, the CSS style properties can be changed dynamically.
Modifying HTML element attributes

Think of an attribute as something that appears inside an HTML tag, such as `style` in:

```html
<p id="tinypg" style="font-size: 10">
```

You can change its value using an assignment statement:

```javascript
ptr = document.getElementById("tinypg")
ptr.style = "font-size:12"
```

or simply:

```javascript
document.getElementById("tinypg").style = "font-size:12"
```

To delete an attribute:

```javascript
ptr.removeAttribute("style")
```
CSSOM

Just like the DOM allows us to manipulate a document's HTML, there is a **CSS Object Model (CSSOM)** that allows us to manipulate a style sheet to dynamically alter the appearance of a web page using JavaScript.

Every element in the DOM has a *style* property that contains the inline styles for the element. There are three JavaScript functions that operate on the style:

- `get PropertyValue()`: returns the value of a specified CSS property.
- `setProperty()`: changes the value of a specified CSS property.
- `removeProperty()`: deletes a specified CSS property.
This enables you to say something like:

```javascript
elem.style.setProperty("color","blue")
```

Alternatively, you can bypass the method call and just access the properties using JavaScript property names:

```javascript
elem.style.color = "blue"
```

CSS property names that include hyphens are converted into JavaScript properties that use camelcasing:

```javascript
elem.style.setProperty("background-color","blue")
```

becomes

```javascript
elem.style.backgroundColor = "blue"
```