JavaScript HTML DOM

HTML DOM

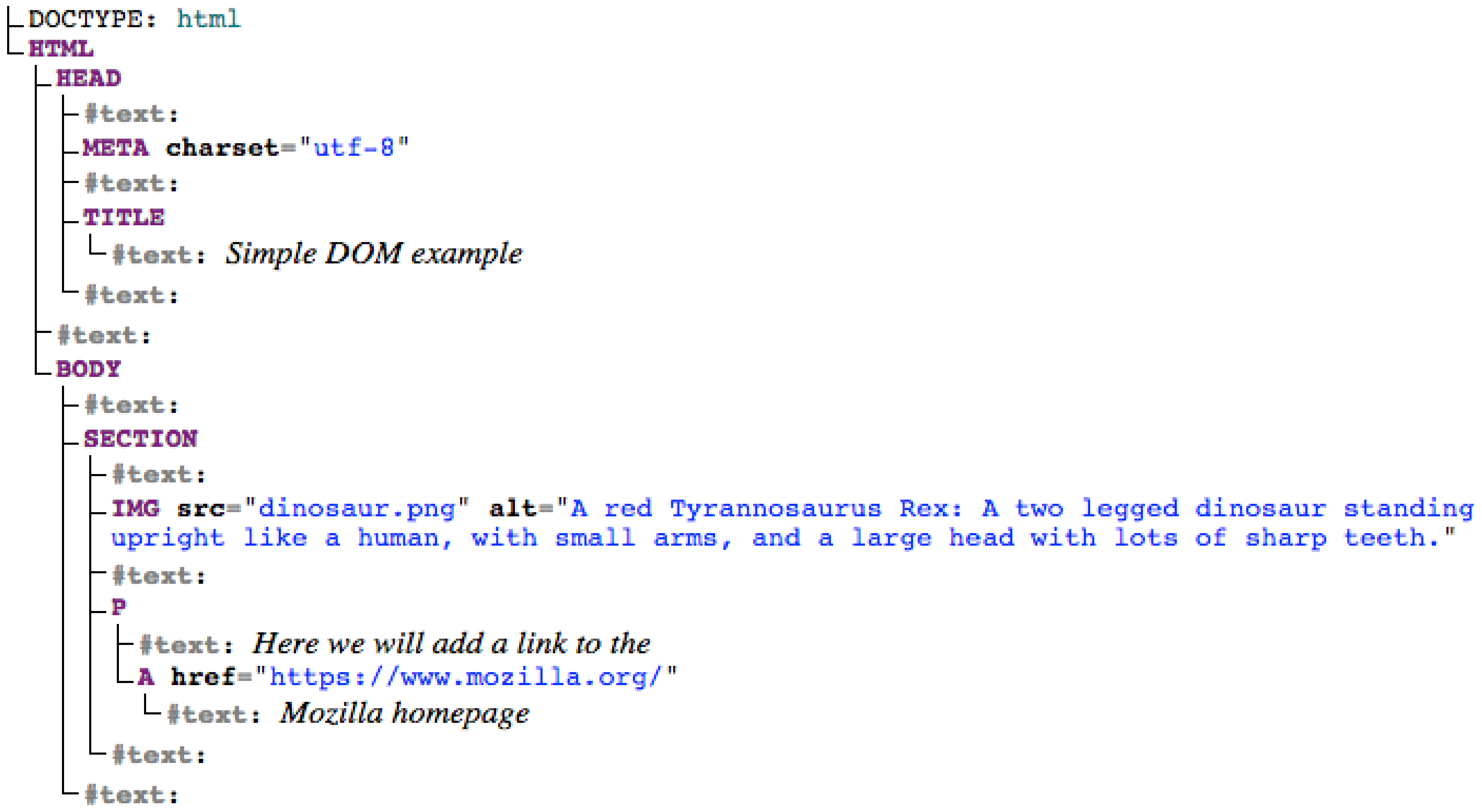
When a web page is loaded, the browser creates a Document Object Model of the page.

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

HTML DOM

```html
<!DOCTYPE html>
<html>
        <head>
                <meta charset="utf-8">
                <title>Simple DOM example</title>
        </head>
        <body>
                <section>
                <img src="dinosaur.png" alt="A red Tyrannosaurus">
                <p>Here link to the
                        <a href="https://www.mozilla.org/">
                        Mozilla homepage</a>
                </p>
                </section>
        </body>
</html>
```

HTML DOM



```
├DOCTYPE: html
└HTML
  ├HEAD
  │ ├#text:
  │ ├META charset="utf-8"
  │ ├#text:
  │ ├TITLE
  │ │ └#text: Simple DOM example
  │ └#text:
  ├#text:
  └BODY
    ├#text:
    ├SECTION
    │ ├#text:
    │ ├IMG src="dinosaur.png" alt="A red Tyrannosaurus Rex: A two legged dinosaur standing
    │ │ upright like a human, with small arms, and a large head with lots of sharp teeth."
    │ ├#text:
    │ ├P
    │ │ ├#text: Here we will add a link to the
    │ │ └A href="https://www.mozilla.org/"
    │ │   └#text: Mozilla homepage
    │ └#text:
    └#text:
```

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

JavaScript can

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

HTML DOM

```
 <html>
        <body>
                <p id="demo"></p>
                <script>
document.getElementById("demo").innerHTML
= "Hello World!";
                </script>
        </body>
</html>
```

getElementById is a method
innerHTML is a property.

Finding HTML Elements

Method	Description
document.getElementById(id)	Find an element by Id
document.getElementsByTagName (name)	Find elements by tag name
document.getElementsByClassName (name)	Find elements by class name
document.querySelectorAll(selector)	Find elements by CSS selectors
document.element[“id”]	Find elements by HTML object collections

Changing HTML Elements

Properties
element.innerHTML = new html content
element.attribute = new value
element.style.property = new style
Method
element.setAttribute(attribute, value)

Adding and Deleting Elements

Methods
document.**createElement**(element)
element.**removeChild**(element)
element.**appendChild**(element)
element.**replaceChild**(element)
document.**write**(text)

Adding Events Handlers

Method
document.getElementById(id).onclick = function(){code}

Finding HTML Objects

The first HTML DOM Level 1 (1998), defined 11 HTML objects, object collections, and properties. These are still valid in HTML5.

Later, in HTML DOM Level 3, more objects, collections, and properties were added.

Property	Description	DOM
document.anchors	Returns all <a> elements that have a name attribute	1
document.applets	Returns all <applet> elements (Deprecated in HTML5)	1
document.baseURI	Returns the absolute base URI of the document	3
document.body	Returns the <body> element	1
document.cookie	Returns the document's cookie	1
document.doctype	Returns the document's doctype	3
document.documentElement	Returns the <html> element	3
document.documentMode	Returns the mode used by the browser	3
document.documentURI	Returns the URI of the document	3
document.domain	Returns the domain name of the document server	1
document.domConfig	Obsolete. Returns the DOM configuration	3
document.embeds	Returns all <embed> elements	3
document.forms	Returns all <form> elements	1
document.head	Returns the <head> element	3

Property	Description	DOM
document.images	Returns all <img> elements	1
document.implementation	Returns the DOM implementation	3
document.inputEncoding	Returns the document's encoding (character set)	3
document.lastModified	Returns the date and time the document was updated	3
document.links	Returns all <area> and <a> elements that have a href attribute	1
document.readyState	Returns the (loading) status of the document	3
document.referrer	Returns the URI of the referrer (the linking document)	1
document.scripts	Returns all <script> elements	3
document.strictErrorChecking	Returns if error checking is enforced	3
document.title	Returns the <title> element	1
document.URL	Returns the complete URL of the document	1

Changing the Value of an Attribute

```
document.getElementById(id).attribute = new value
```

Changing HTML Style

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

Using Events

Execute code when an event occurs.

- 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

Assign Events Using the HTML DOM

The HTML DOM allows you to assign events to HTML elements using JavaScript:

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

The onload and onunload Events

- The onload and onunload events are triggered when the user enters or leaves the page
- The onload event can be used to check the visitor's browser type and browser version, and load the proper version of the web page based on the information
- The onload and onunload events can be used to deal with cookies.

The onchange Event

The onchange event is often used in combination with validation of input fields.

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

The **onmouseover** and **onmouseout** Events

The onmouseover and onmouseout events can be used to trigger a function when the user mouses over, or out of, an HTML element.

The **onmousedown**, **onmouseup** and **onclick** Events

- The onmousedown, onmouseup, and onclick events are all parts of a mouse-click

1) First when a mouse-button is clicked, the onmousedown event is triggered, then,
2) when the mouse-button is released, the onmouseup event is triggered, finally,
3) when the mouse-click is completed, the onclick event is triggered.

The addEventListener() method

- The addEventListener() method attaches an event handler to the specified element without overwriting existing event handlers

- You can add many event handlers to one element

- You can add many event handlers of the same type to one element, i.e two "click" events

- You can add event listeners to any DOM object not only HTML elements. i.e the window object

The addEventListener() method

It easier to control how the event reacts to bubbling

the JavaScript is separated from the HTML markup, for better readability

Allows adding event listeners even when you do not control the HTML markup

remove an event listener by using the removeEventListener() method

Event Bubbling or Event Capturing?

There are two ways of event propagation in the HTML DOM, **bubbling** and **capturing**.

Event propagation is a way of defining the element order when an event occurs. If you have a <p> element inside a <div> element, and the user clicks on the <p> element, which element's "click" event should be handled first?

Event Bubbling or Event Capturing?

In bubbling the inner most element's event is handled first and then the outer: the <p> element's click event is handled first, then the <div> element's click event.

In capturing the outer most element's event is handled first and then the inner: the <div> element's click event will be handled first, then the <p> element's click event.

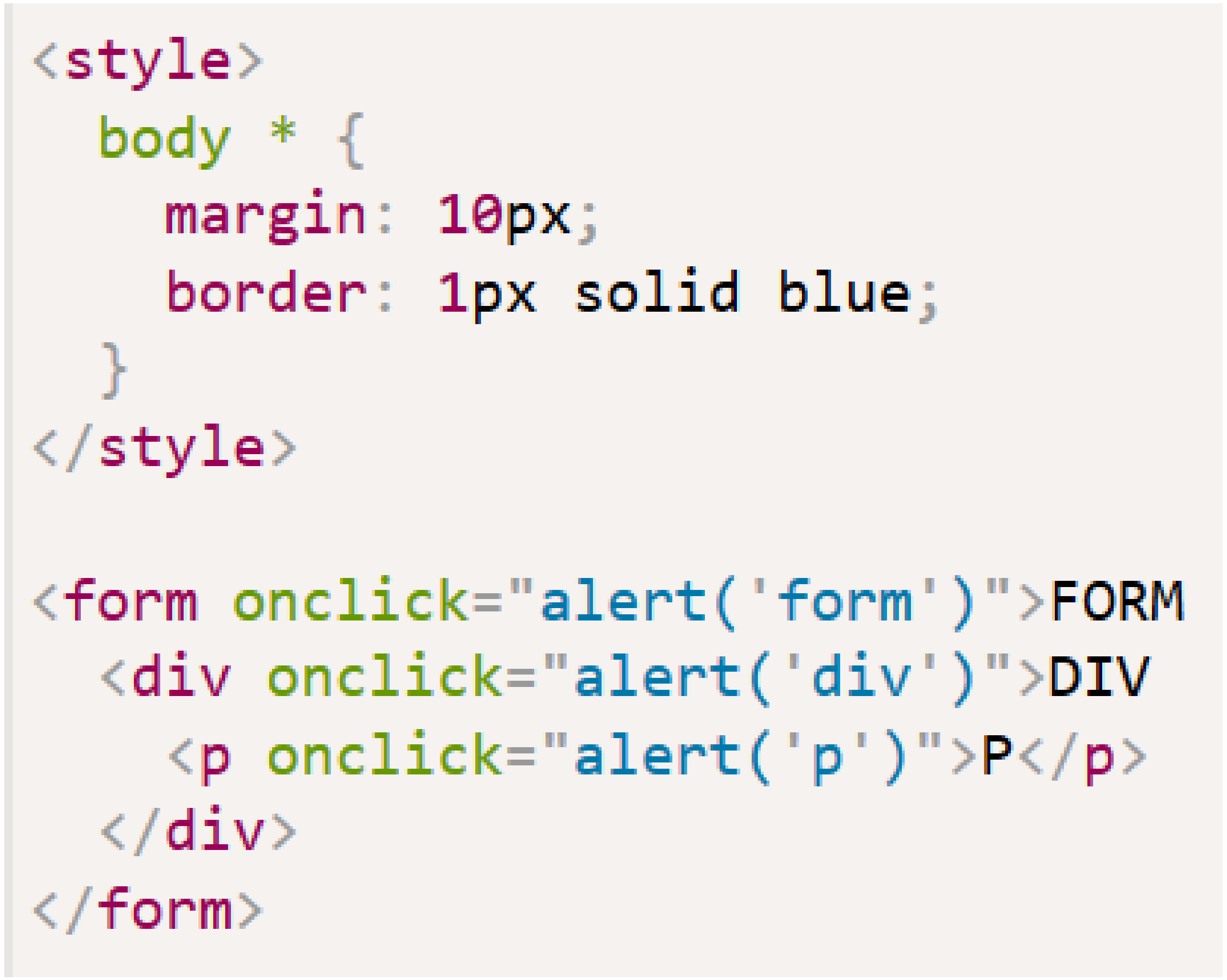
Event Bubbling or Event Capturing?

With the addEventListener() method you can specify the propagation type by using the "useCapture" parameter:

```
addEventListener(event, function, useCapture);
```

The default value is false, which will use the bubbling propagation, when the value is set to true, the event uses the capturing propagation.

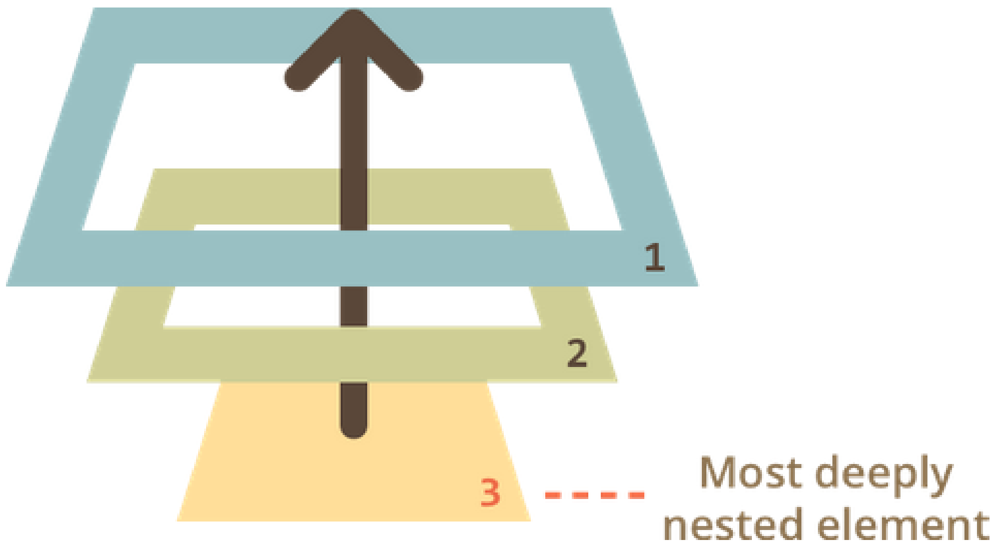
Event Bubbling and Capturing



```html
<style>
  body * {
    margin: 10px;
    border: 1px solid blue;
  }
</style>

<form onclick="alert('form')">FORM
  <div onclick="alert('div')">DIV
    <p onclick="alert('p')">P</p>
  </div>
</form>
```

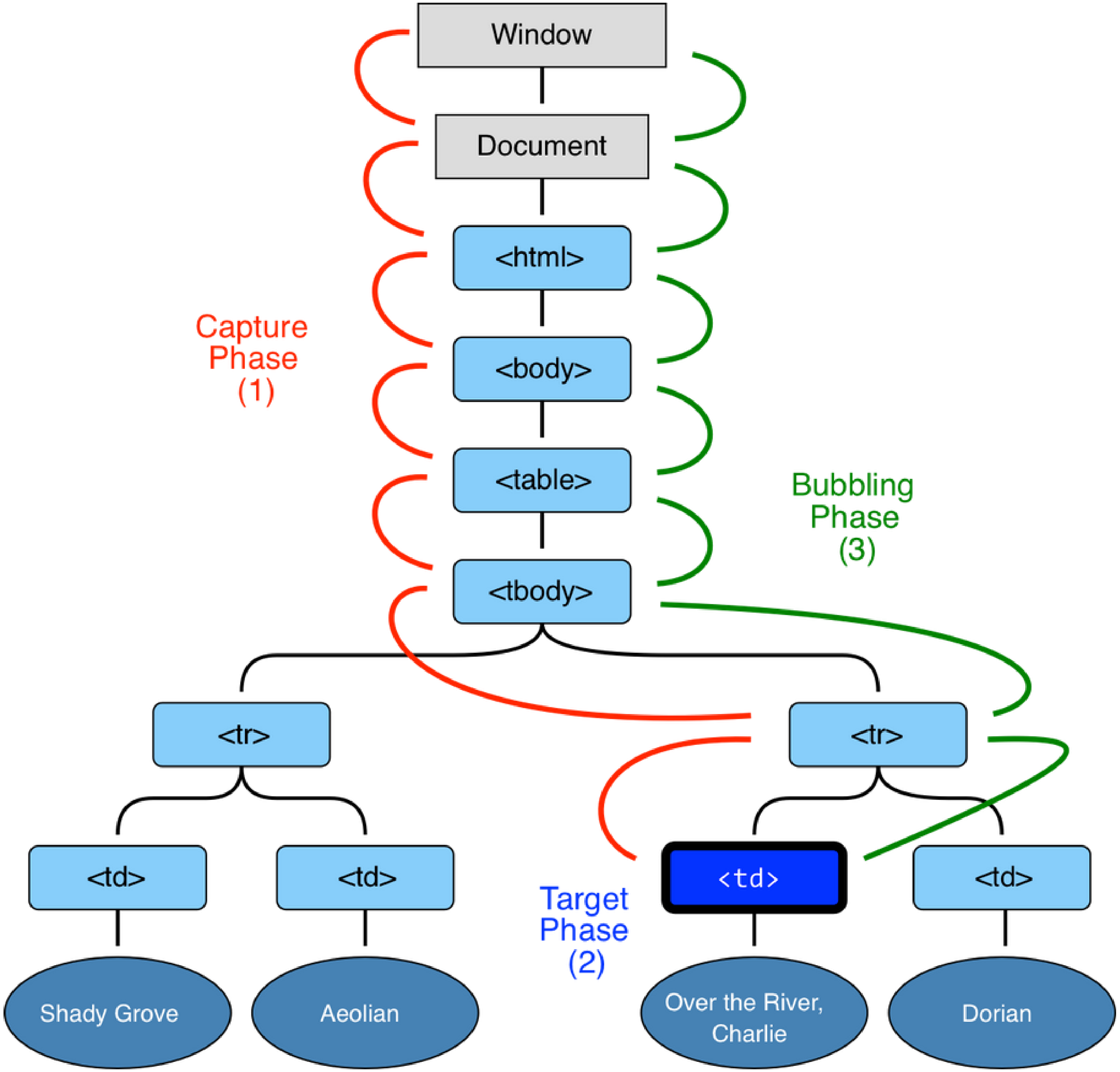
A click on the inner `<p>` first runs `onclick`:

1. On that `<p>`.
2. Then on the outer `<div>`.
3. Then on the outer `<form>`.
4. And so on upwards till the `document` object.



So if we click on `<p>`, then we'll see 3 alerts: `p` → `div` → `form`.

The process is called "bubbling", because events "bubble" from the inner element up through parents like a bubble in the water.


Window
Document
<html>
<body>
<table>
<tbody>
<tr>
<tr>
<td>
<td>
<td>
<td>
Shady Grove
Aeolian
Over the River, Charlie
Dorian
Capture Phase (1)
Target Phase (2)
Bubbling Phase (3)

The removeEventListener() method

The removeEventListener() method removes event handlers that have been attached with the addEventListener() method

Example:

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

HTML DOM Navigation

Everything in an HTML document is a node:

The entire document is a document node

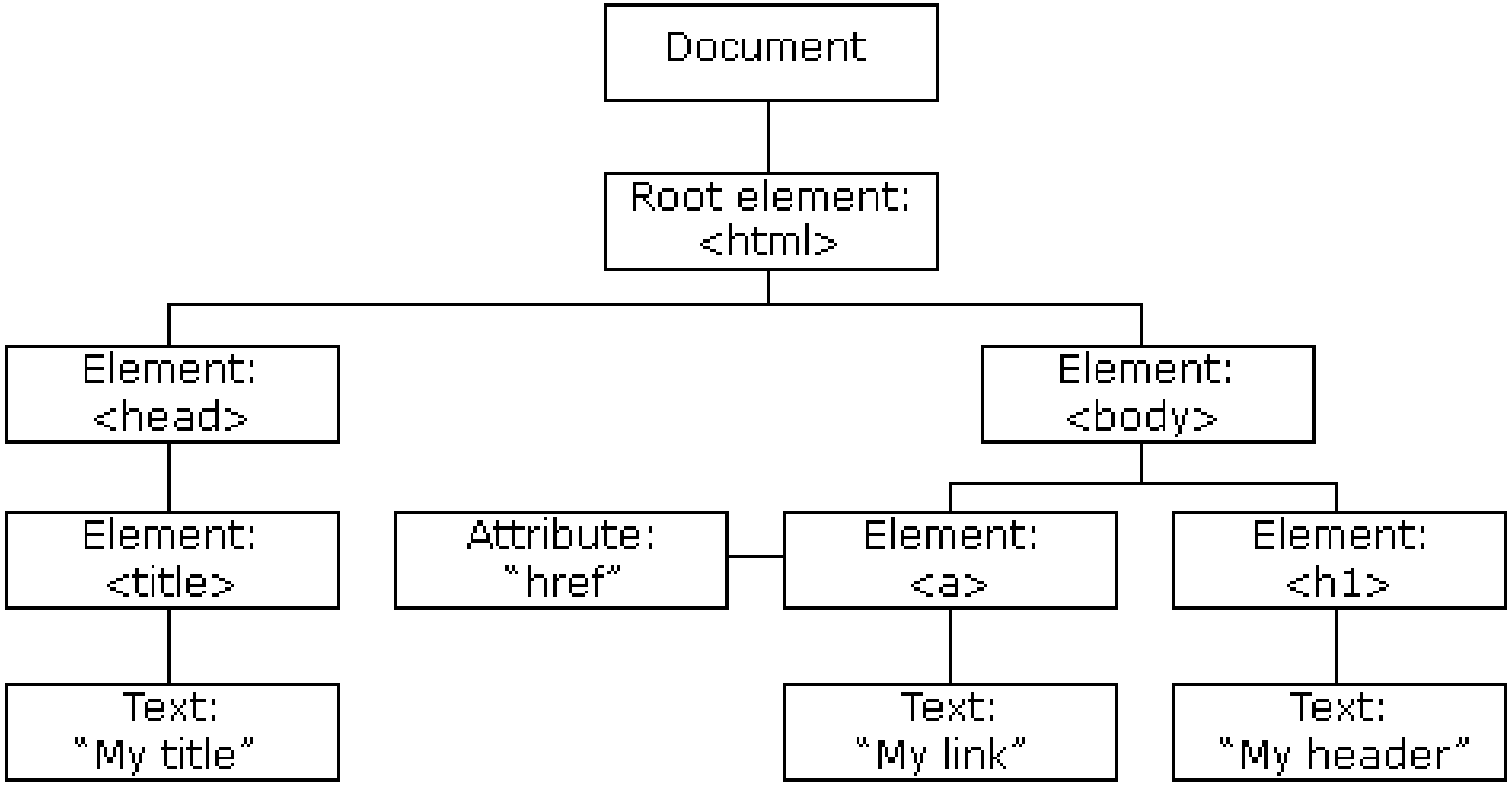
Every HTML element is an element node

The text inside HTML elements are text nodes

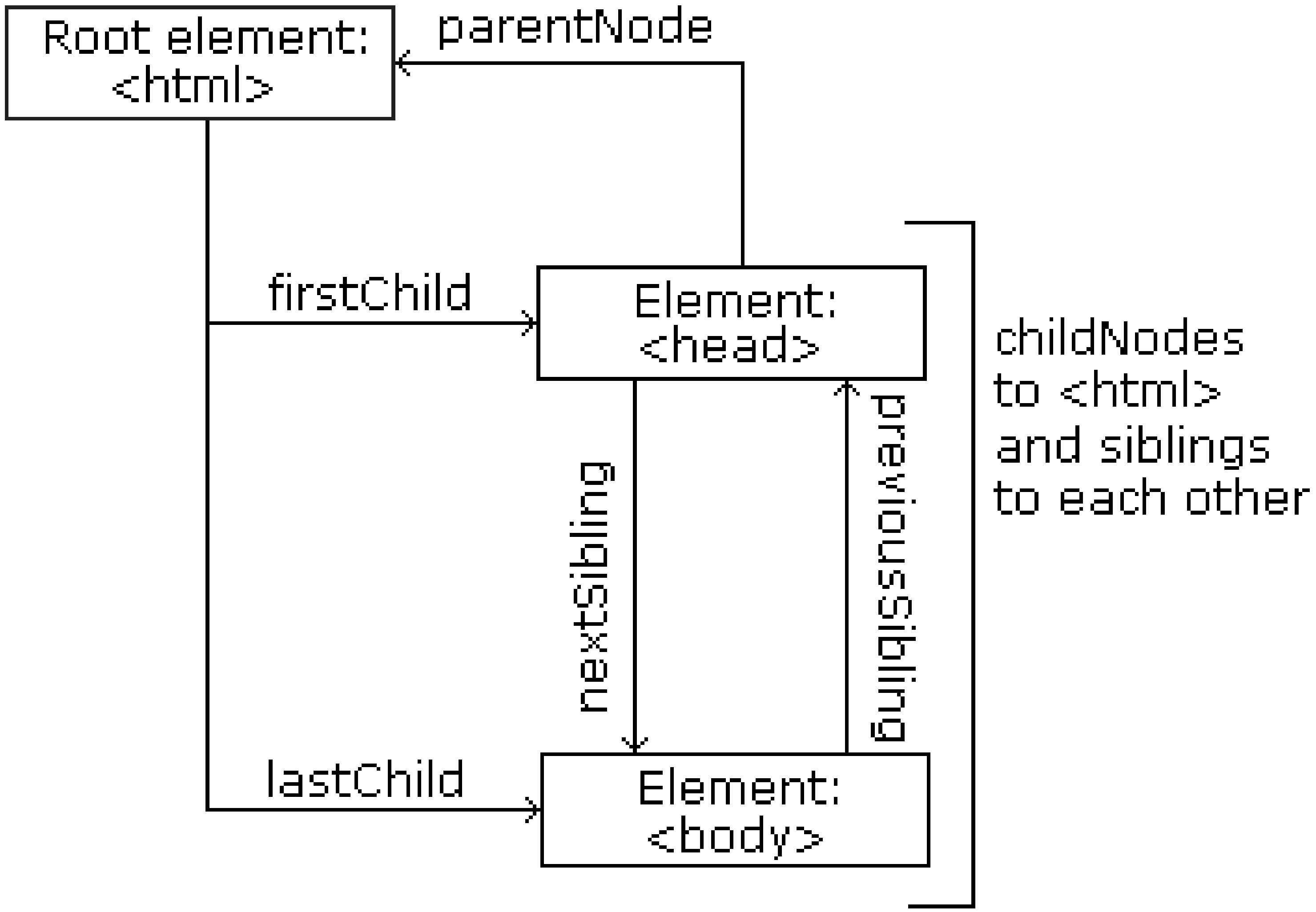
Every HTML attribute is an attribute node (deprecated)

All comments are comment nodes

HTML DOM Navigation



Node Relationships



Navigating Between Nodes

Following node properties can be used to navigate between nodes with JavaScript:

- ParentNode
- childNodes[nodenumber]
- FirstChild
- LastChild
- NextSibling
- previousSibling

Child Nodes and Node Values

```
<title id="demo">DOM Tutorial</title>
```

```
document.getElementById("demo").innerHTML;
                    ≈
document.getElementById("demo").firstChild.nodeValue;
                    ≈
document.getElementById("demo").childNodes[0].nodeValue;
```

DOM Root Nodes

There are two special properties that allow access to the full document:

document.body	- The body of the document
document.documentElement	- The full document

The **nodeName** Property

The nodeName property specifies the name of a node.

- ➢ nodeName is read-only
- ➢ nodeName of an element node is the same as the tag name
- ➢ nodeName of an attribute node is the attribute name
- ➢ nodeName of a text node is always #text
- ➢ nodeName of the document node is always #document

The **nodeValue** Property

The nodeValue property specifies the value of a node.

- ➢ nodeValue for element nodes is undefined
- ➢ nodeValue for text nodes is the text itself
- ➢ nodeValue for attribute nodes is the attribute value

The **nodeType** Property

The nodeType property is read only. It returns the type of a node

```
 <h1 id="id01">My First Page</h1>
<p id="id02"></p>

<script>
document.getElementById("id02").innerHTML =
document.getElementById("id01").nodeType;
</script>
```

The **nodeType** Property

Node	**Type**	**Example**
ELEMENT_NODE	1	<h1 class="heading">W3Schools</h1>
ATTRIBUTE_NODE	2	class = "heading" (deprecated)
TEXT_NODE	3	W3Schools
COMMENT_NODE	8	<!-- This is a comment -->
DOCUMENT_NODE	9	The HTML document itself (the parent of <html>)
DOCUMENT_TYPE_NODE	10	<!Doctype html>

Type 2 is deprecated in the HTML DOM (but works). It is not deprecated in the XML DOM.

Creating New HTML Elements (appendChild)

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

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

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

Creating New HTML Elements (insertBefore)

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

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

var element = document.getElementById("div1");
var child = document.getElementById("p1");
element.insertBefore(para, child);
</script>
```

Removing Existing HTML Elements

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

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

Replacing HTML Elements

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

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

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

The HTMLCollection Object

The getElementsByTagName() method returns an HTMLCollection object.

An HTMLCollection object is an array-like list (collection) of HTML elements

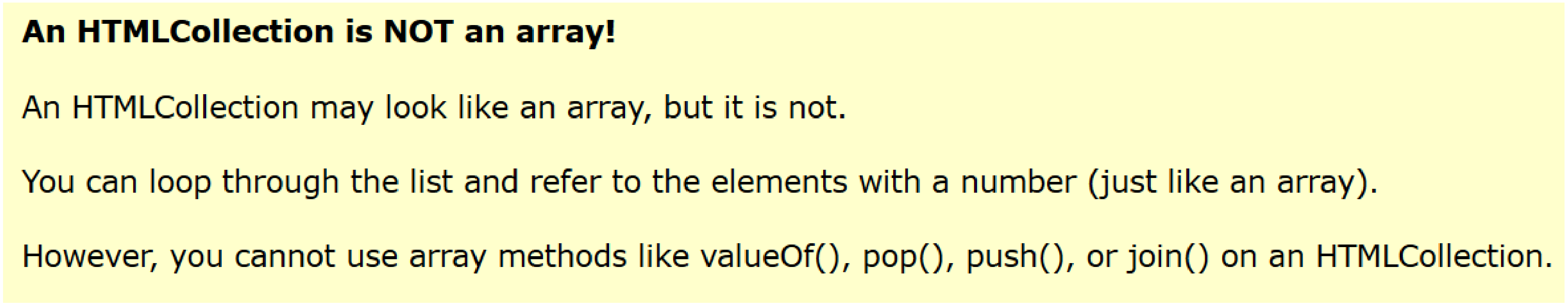
The HTMLCollection Object

```html
<html>
<body>

<h2>JavaScript HTML DOM</h2>

<p>Hello World!</p>

<p>Hello Norway!</p>

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

<script>
var myCollection = document.getElementsByTagName("p");
document.getElementById("demo").innerHTML =
"The innerHTML of the second paragraph is: " +
myCollection[1].innerHTML;
</script>

</body>
</html>
```

HTML HTMLCollection Length

```
<html>
<body>

<h2>JavaScript HTML DOM</h2>

<p>Hello World!</p>

<p>Hello Norway!</p>

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

<script>
var myCollection = document.getElementsByTagName("p");
document.getElementById("demo").innerHTML =
"This document contains " + myCollection.length + " paragraphs.";
</script>

</body>
</html>
```

HTMLCollection vs. Array



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.

HTML DOM NodeList Object

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

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

Some (older) browsers return a NodeList object instead of an HTMLCollection for methods like getElementsByClassName().

All browsers return a NodeList object for the

Property - childNodes

Method - querySelectorAll().

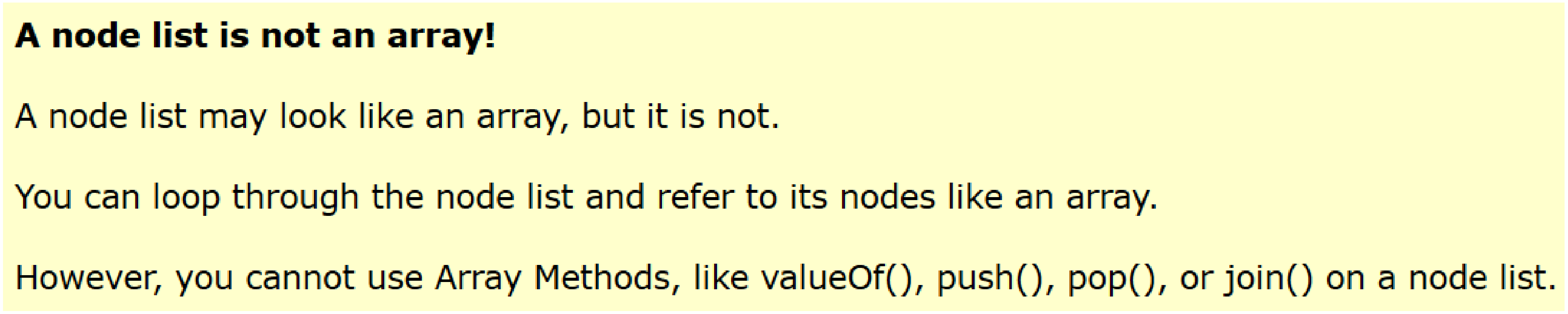
NodeList Object Example

```
<h2>JavaScript HTML DOM!</h2>

<p>Hello World!</p>

<p>Hello Norway!</p>

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

<script>
var myNodelist = document.querySelectorAll("p");
document.getElementById("demo").innerHTML =
"The innerHTML of the second paragraph is: " +
myNodelist[1].innerHTML;
</script>
```

HTMLCollection vs. NodeList

HTMLCollection	NodeList
collection of HTML elements	collection of document nodes
array-like list (collection) of objects	array-like list (collection) of objects
length property	length property
index (0, 1, 2, 3, 4, ...) to access each item	index (0, 1, 2, 3, 4, ...) to access each item
can be accessed by their name, id, or index number	can only be accessed by their index number
cannot contain attribute nodes and text nodes	can contain attribute nodes and text nodes

NodeList Object



A node list is not an array!

A node list may look like an array, but it is not.

You can loop through the node list and refer to its nodes like an array.

However, you cannot use Array Methods, like valueOf(), push(), pop(), or join() on a node list.

References

https://www.w3schools.com/js/js_htmldom.asp