INT201 Client-side

Document Object Modeling (DOM)

Document Object Model (DOM) ไม่ได้ผูกติดกับ JS ไม่ใช่ภาษา แต่เป็น model ที่คล้าย tree

- Understanding the Document Object Model in JavaScript
- Hierarchy Of Nodes
- Node Types and Relationships
- Traversing Nodes เข้าถึงแต่ละ node
- Manipulating Nodes insert เพิ่ม an node
- 4 Key Node Types (Document, Element, Text, Attribute)

Document Object Modeling (DOM)

- One of the most important objects in client side JavaScript programming is the Document object-which comprise the structure and content of a document on the web.
- The Document Object Model (DOM) represents the page so that programs can change the document structure, style, and content.
- A web page is a document that can be either displayed in the browser window or as the HTML source. In both cases, it is the same document, but the Document Object Model (DOM) representation allows it to be manipulated.
- As an object oriented representation of the web page, it can be modified with a scripting language such as JavaScript.
- The DOM is an application programming interface (API) for HTML and XML documents.
- The DOM represents a document as a **hierarchical tree of nodes**, allowing developers to add, remove, and modify individual parts of the page.
- The DOM is now a truly cross-platform, language-independent way of representing and manipulating pages for markup.
- The DOM is not a programming language, but without it, the JavaScript language wouldn't have any model or notion of web pages, HTML documents, XML documents, and their component parts.

<script> Tag Placement

- Including all JavaScript files in the <head> of a document means that all of the JavaScript code must be downloaded, parsed, and interpreted before the page begins rendering
- For pages that **require a lot of JavaScript code**, this can cause a noticeable delay in page rendering, during which time the browser will be completely blank. รอให้ render ให้เสร็จ
- For this reason, modern web applications typically include all JavaScript references in the <body> element, after the page content.

```
<html>
  <head>
      <title>Example HTML Page</title>
      <script src="script.js"></script>
  </head>
  <body>
      <!-- content here -->
  </body>
</html>
```

```
<html>
  <head>
      <title>Example HTML Page</title>
  </head>
  <body>
      <!-- content here -->
      <script src="script.js"></script>
  </body>
</html>
```

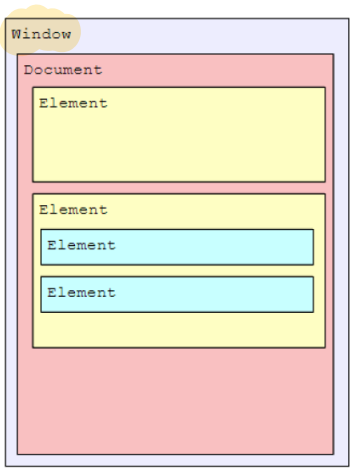
delay script

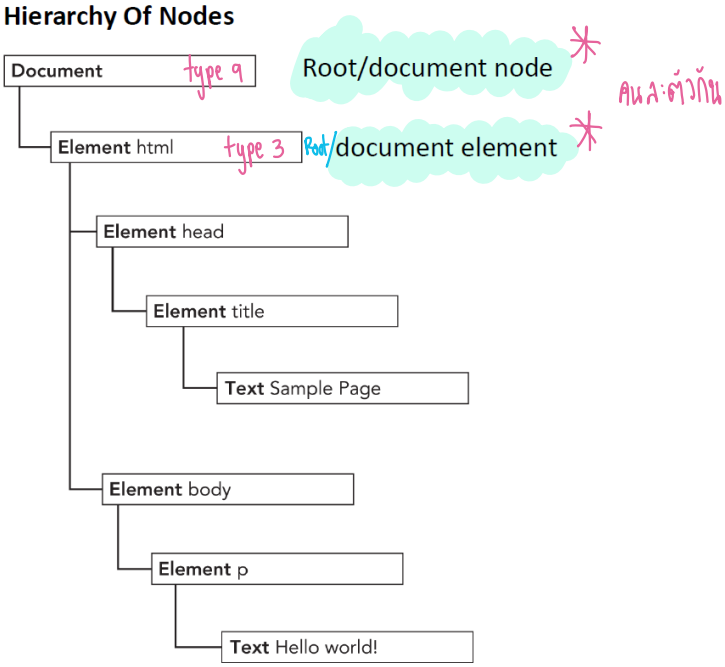
Document Object Modeling (DOM)

- When an HTML document is loaded into a web browser, it becomes a **document object,** everything is a **node**
- The document object is the root node of the HTML document and all other nodes: element nodes, text nodes, attribute nodes, and comment nodes
- Each node type has different characteristics, data, and methods, and each may have relationships with other nodes.

- These relationships create a hierarchy that allows markup to be represented as a tree, rooted at a particular node.

```
<html>
   <head>
      <title>Sample Page</title>
   </head>
   <body>
      <p>Hello World!</p>
   </body>
</html>
```





Node types are represented by one of the following 12 numeric constants on the Node types:

//07_Nodes

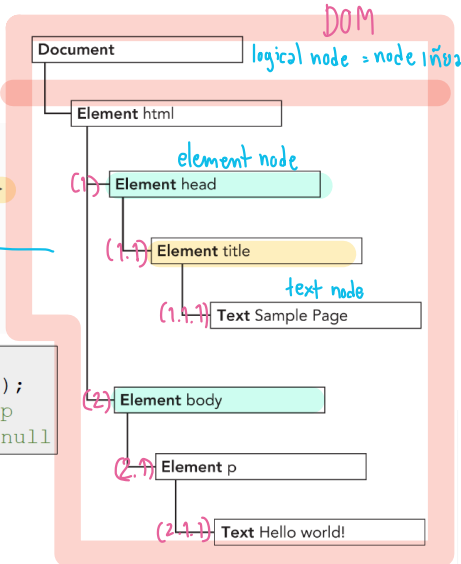
- ***Node.ELEMENT_NODE (1)***
- ***Node.ATTRIBUTE_NODE (2)***
- ***Node.TEXT_NODE (3)***
- Node.CDATA_SECTION_NODE (4)
- Node.ENTITY_REFERENCE_NODE (5)
- Node.ENTITY_NODE (6)
- Node.PROCESSING_INSTRUCTION_NODE (7)
- Node.COMMENT_NODE (8)
- ***Node.DOCUMENT_NODE (9)***
- Node.DOCUMENT_TYPE_NODE (10)
- Node.DOCUMENT_FRAGMENT_NODE (11)
- Node.NOTATION_NODE (12)

- The **Node** interface is implemented in JavaScript as the Node type, which is accessible in all browsers except Internet Explorer.
- All node types inherit from Node in JavaScript, so **all node types share the same basic properties and methods.**
- Every node has a **node Type property** that indicates the type of node that it is.

check node

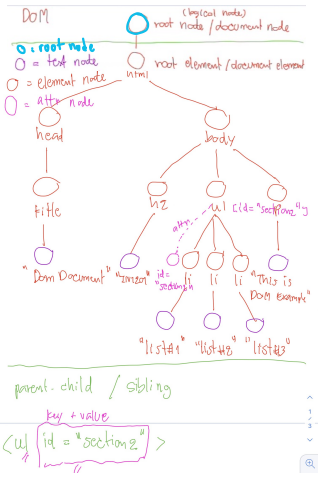
```
if (someNode.nodeType == Node.ELEMENT_NODE){
   alert("Node is an element.");
}
```

DOM




```
<html> root element
   <head>
      <title>Sample Page</title>
   </head>
   <body>
      <p>Hello World!</p>
   </body>
</html>
```

```
const paragraphs =
document.getElementsByTagName("p");
alert(paragraphs[0].nodeName); //p
alert(paragraphs[0].nodeValue);//null
```



L10.html

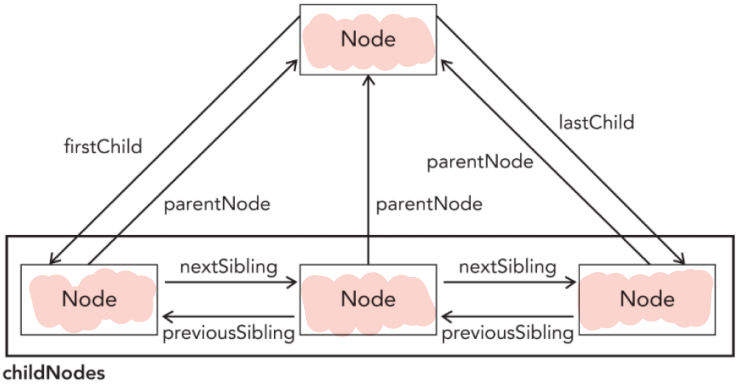
Node Relationships

- All nodes in a document have relationships to other nodes.

ขนาด document ต้องไม่ใหญ่ ถึงจะ generate ได้

- These relationships are described in terms of **traditional family relationships** as if the document tree were a family tree.
- In HTML, the `<body>` element is considered a child of the `<html>` element; likewise the `<html>` element is considered the parent of the `<body>` element.
- The `<head>` element is considered a sibling of the `<body>` element, because they both share the same immediate parent, the `<html>` element.

The value of `someNode.firstChild` is always equal to `someNode.childNodes[0]`, and the value of `someNode.lastChild` is always equal to `someNode.childNodes [someNode.childNodes.length - 1]`.



With all of these relationships, the **childNodes property** is really more of a convenience than a necessity because **it's possible to reach any node in a document tree** by simply using the relationship pointers.

Traversing Nodes

Element Traversal

The Element Traversal API adds five new properties to DOM elements:

- `childElementCount`—Returns the number of child elements (excludes text nodes and comments).
- `firstElementChild`—Points to the first child that is an element. Element-only version of `firstChild`.
- `lastElementChild`—Points to the last child that is an element. Element-only version of `lastChild`.
- `previousElementSibling`—Points to the previous sibling that is an element. Element-only version of `previousSibling`.
- `nextElementSibling`—Points to the next sibling that is an element. Element-only version of `nextSibling`.

childElement = 3

next child
next sibling

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Manipulating Elements</title>
  </head>
  <body>
    <h3>SIT@KMUTT Restaurant</h3>
    <div class="menu">
      <ul id="appetizer">
        <li class="vegan">Vegetable Rolls</li>
        <li class="meat">Chicken Wings</li>
        <li class="meat">Tuna Sandwich</li>
      </ul>
      <ul id="soup">
        <li class="meat">Spicy Bacon-Corn Soup</li>
        <li class="vegan">Vegetable Soup</li>
        <li class="meat">Beef Soup</li>
        <li class="vegan">Coconut Soup</li>
      </ul>
    </div>
    <p>***Enjoy Your Meal, Thank you***</p>
    <script src="script.js"></script>
  </body>
</html>
```

method document obj. ที่มาคือ DOMstring return first element
type 9

//07_TraversingNodes

```
const soupMenu = document.querySelector("#soup");
console.log(soupMenu.parentNode); //<div class="menu">
console.log(soupMenu.parentElement); //<div class="menu">
let currentChildNode=soupMenu.firstElementChild;
// <li class="meat">Spicy Bacon-Corn Soup</li>

while(currentChildNode!==soupMenu.lastChild){
    if(currentChildNode.nodeType===1){
        console.log(currentChildNode);
    }
    currentChildNode=currentChildNode.nextSibling
}
```

```
<li class="meat">Spicy Bacon-Corn Soup</li>
<li class="vegan">Vegetable Soup</li>
<li class="meat">Beef Soup</li>
<li class="vegan">Coconut Soup</li>
```

*The firstChild and lastChild return the first and last child of a node, which can be any node type including text node, comment node, and element node.

**The firstElementChild and lastElementChild return the first and last child Element node.

Selecting Elements

The Document type provides two methods to specific element or sets of elements to perform certain operations.

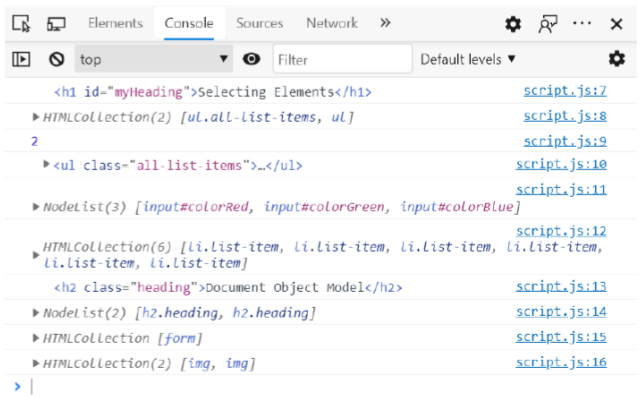
- getElementById() //returns the element if found, or null if an element with that ID doesn't exist.
- getElementsByTagName() //returns an HTMLCollection of elements with the given tag name.
- getElementsByName() //returns a NodeList, which returns all elements that have a given name attribute.
- getElementsByClassName() //returns a NodeList containing all elements that have all of the specified classes applied.
- querySelector() //a CSS query and returns the first descendant element that matches the pattern or null if there is no matching element.
- querySelectorAll() //the CSS query and returns all matching nodes instead of just one. This method returns a static instance of NodeList.

An HTMLCollection is a collection of HTML elements.
A NodeList is a collection of document nodes.
Both an HTMLCollection object and a NodeList object is an array-like (collection) of objects.

//07_SelectingElements

```
<html lang="en">
  <head>
       <title>Document Object Model Sample Page</title>
  </head>
  <body>
    <h1 id="myHeading">Selecting Elements</h1>
    <h2 class="heading">Document Object Model</h2>
    <h2 class="heading">DOM</h2>
    <ul class="all-list-items">
      <li class=list-item>out of class</li>
      <li class="list-item">[1] Single Element by ID:</li>
      <li class="list-item">[2] Single Element by CSS selector:</li>
      <li class="list-item">[3] Multiple Element by CSS selector:</li>
      <li class="list-item">[4] Multiple Element by class (live):</li>
      <li class="list-item">[5] Multiple Element by tag (live):</li>
    </ul>
   <ul>
    <li>
       <input type="radio" value="red" name="color" id="colorRed">
       <label for="colorRed">Red</label>
    </li>
    <li>
       <input type="radio" value="green" name="color" id="colorGreen">
       <label for="colorGreen">Green</label>
    </li>
    <li>
       <input type="radio" value="blue" name="color" id="colorBlue">
       <label for="colorBlue">Blue</label>
    </li>
   </ul>
     <script src="script.js"></script>
  </body>
</html>
```

```
const domElementId = document.getElementById("myHeading");
const domByTagName = document.getElementsByTagName("ul");
const domByName =document.getElementsByName("color");
const domClassName = document.getElementsByClassName("list-item");
const domQuerySelector = document.querySelector(".heading");
const domQuerySelectorAll = document.querySelectorAll(".heading");
console.log(domElementId);//<h1 id="myHeading">Selecting Elements</h1>
console.log(domByTagName);//HTMLCollection(2) [ul.all-list-items,ul]
console.log(domByTagName.length); //2
console.log(domByTagName.item(0)); //<ul class="all-list-items">...</ul>
console.log(domByName);//NodeList(3) [input#colorRed, input#colorGreen, input#colorBlue]
console.log(domClassName);//HTMLCollection(6) [li.list-item, li.list-item,li.list-
item, li.list-item, li.list-item, li.list-item]
console.log(domQuerySelector);//<h2 class="heading">Document Object Model</h2>
console.log(domQuerySelectorAll);//NodeList(2) [h2.heading, h2.heading]
```



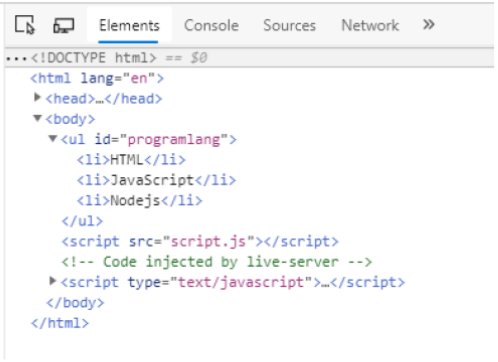
.textcontent = เพิ่ม text ธรรมดา

Manipulating Nodes

- appendChild() //adds a newly node to the end of the childNodes list
- createElement() //create a new HTML element
- insertBefore(newNode, referenceNode) //The node to insert becomes the previous sibling of the reference node
- replaceChild(newChild, oldChild) //replaces a child node within the given (parent) node
- removeChild(child) //removes a child node from the DOM and returns the removed node.

//07_ManipulatingElements

```
<html lang="en">
  <head>
    <title>Manipulating Elements</title>
  </head>
  <body>
    <ul id="programlang"></ul>
     <script src="script.js"></script>
  </body>
</html>
```



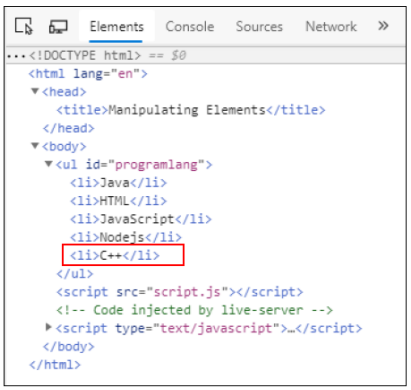
- HTML
- JavaScript
- Nodejs

```
let langs = ["HTML", "JavaScript", "Nodejs"];
const langElement = document.querySelector("#programlang");
for (const lang of langs) {
    let li=document.createElement("li");
    li.innerHTML=lang;
    langElement.appendChild(li);
}
```

```
const firstLangElement=langElement.firstElementChild;
const newLangNode=document.createElement("li");
newLangNode.innerHTML="Java";
langElement.insertBefore(newLangNode, firstLangElement);
```

- Java
- HTML
- JavaScript
- Nodejs

```
const newLastLangNode=document.createElement("li");
newLastLangNode.innerHTML="C++";
langElement.insertBefore(newLastLangNode, null);
```

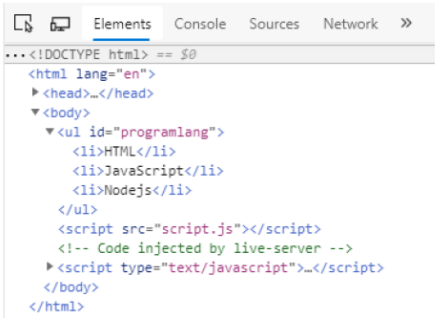


- Java
- HTML
- JavaScript
- Nodejs
- C++

```
langElement.replaceChild(newLastLangNode, newLangNode);
```

- C++
- HTML
- JavaScript
- Nodejs

```
langElement.removeChild(langElement.firstElementChild);
```



- ~~C++~~
- HTML
- JavaScript
- Nodejs

Nodes Types

Node.nodeType

Node.nodeType property is an integer that identifies what the node is. It distinguishes different kind of nodes from each other, such as elements, text and comments.

Node	Value of nodeValue	nodeName value
CDATASection	Content of the CDATA section	"#cdata-section"
Comment	Content of the comment	"#comment"
Document	null	"#document"
DocumentFragment	null	"#document-fragment"
DocumentType	null	The value of DocumentType.name
Element	null	The value of Element.tagName
ProcessingInstruction	Entire content excluding the target	The value of ProcessingInstruction.target
Text	Content of the text node	"#text"

The nodeName and nodeValue Properties

- Two properties, nodeName and nodeValue, give specific information about the node.
- The values of these properties are completely dependent on the node type.
- It s always best to test the node type before using one of these values, as the following code shows:

```
if (someNode.nodeType == 1){    //someNode.nodeType===Node.ELEMENT_NODE
    value = someNode.nodeName; // nodeName will be the element's tag name
                               // nodeValue is always null
}
```

Document Type

The Document Type

- JavaScript represents document nodes via the Document type.
- The Document type can represent HTML pages or other XML-based documents.
- The document object, as an instance of HTMLDocument, has several additional properties that standard Document objects do not have.
- The document object can be used to get information about the page and to manipulate both its appearance and the underlying structure.

A **Document node** has the following characteristics:

- nodeType is 9.
- nodeName is "#document".
- nodeValue is null.
- parentNode is null.
- child nodes may be Element (maximum of one), ProcessingInstruction, or Comment.
- appendChild(), removeChild(), and replaceChild() methods aren't used on document because the document type (if present) is read only and there can be only one element child node.

Document Children

- This example shows that the values of documentElement, firstChild, and childNodes[0] are all the same—all three point to the <html> element.
- As an instance of HTMLDocument, the document object also has a body property that points to the <body> element directly.

```
<html>
  <body>

  </body>
</html>
```

```
let html = document.documentElement;    // get reference to <html>
alert(html === document.childNodes[0]); // true
alert(html === document.firstChild);    // true
```

```
let body = document.body; // get reference to <body>
```

Document Information

- The first such property is title, which contains the text in the <title> element and is displayed in the title bar or tab of the browser window.

```
// get the document title
let originalTitle = document.title;

// set the document title
document.title = "New page title";

// get the complete URL
let url = document.URL;

// get the domain
let domain = document.domain;
```

Special Collections

- document.anchors—contains all <a>elements with a name attribute in the document.
- document.forms—contains all <form> elements in the document. The same as document.getElementsByTagName("form").
- document.images—contains all <img> elements in the document. The same as document.getElementsByTagName("img").

- document.links—contains all <a>elements with an href attribute in the document.

Element Type

- The **Element type** represents an XML or HTML element, providing access to information such as its tag name, children, and attributes.
- An Element node has the following characteristics:
  - nodeTypeis 1.
  - nodeNameis the element's tag name.
  - nodeValueis null.
  - parentNodemay be a Document or Element.
  - Child nodes may be Element, Text, Comment, ProcessingInstruction, CDATASection, or EntityReference.
- An element's tag name is accessed via the nodeName property or by using the tagName property; both properties return the same value.

```
<div id="myDiv"></div>
```

```
let div = document.getElementById("myDiv");
alert(div.tagName); // "DIV " When used with HTML, the tag name is always
represented in all uppercase;
alert(div.tagName == div.nodeName);  // true
```

```
if (element.tagName == "div") { // AVOID! Error prone!
 // do something here
}

if (element.tagName.toLowerCase() == "div"){ // Preferred - works in all
documents
// do something here
}
```

HTML Elements

- All HTML elements are represented by the HTMLElement type, either directly or through subtyping.
- The HTMLElement inherits directly from Element and adds several properties.
- Each property represents one of the following standard attributes that are available on every HTML element:
- id—A unique identifier for the element in the document.
- title—Additional information about the element, typically represented as a tooltip.
- lang—The language code for the contents of the element (rarely used).
- dir—The direction of the language, "ltr" (left-to-right) or "rtl" (right-to-left); also rarely used.
- className—The equivalent of the class attribute, which is used to specify CSS classes on an element. The property could not be named class because class is an ECMAScript reserved word.

```
<div id="myDiv" class="bd" title="Body text" lang="en" dir="ltr"></div>
```

```
let div = document.getElementById("myDiv");
alert(div.id); // "myDiv"
alert(div.className); // "bd"
alert(div.title); // "Body  text"
alert(div.lang); // "en"
alert(div.dir); // "ltr"
```

```
div.id = "someOtherId"; or div.setAttribute("id", "someOtherId");
div.className = "ft";
div.title = "Some other text";
div.lang = "fr";
div.dir ="rtl";
```

The three primary DOM methods for working with attributes are getAttribute(), setAttribute(), and removeAttribute().

```
let div = document.getElementById("myDiv");
alert(div.getAttribute("id"));   // "myDiv"
alert(div.getAttribute("class")); // "bd"
alert(div.getAttribute("title")); // "Body text"
alert(div.getAttribute("lang"));  // "en"
alert(div.getAttribute("dir"));  // "ltr"

div.removeAttribute("class");
```

Creating Elements

- New elements can be created by using the document.createElement() method.
- This method accepts a single argument, which is the tag name of the element to create.
- In HTML documents, the tag name is case-insensitive
- To create a element, the following code can be used:

```
let div = document.createElement("div");
```

- The element can be added to the document tree using appendChild(), insertBefore(), or replaceChild().
- The following code adds the newly created element to the document s <body> element:

```
document.body.appendChild(div);
```

- Once the element has been added to the document tree, the browser renders it immediately.
  Any changes to the element after this point are immediately reflected by the browser.

Text Type

- Text nodes are represented by the Text type and contain plain text that is interpreted literally and may contain escaped HTML characters but no HTML code.
- A Text node has the following characteristics:
  - nodeType is 3.
  - nodeName is "#text".
  - nodeValue is text contained in the node.
  - parentNode is an Element.
- child nodes are not supported.
- The text contained in a Text node may be accessed via either the nodeValue property or the data property, both of which contain the same value.
- Changes to either nodeValue or data are reflected in the other as well.

The following methods allow for manipulation of the text in the node:

- appendData(text) - Appends text to the end of the node.
- deleteData(offset, count) - Deletes count number of characters starting at position offset.
- insertData(offset, text) - Inserts text at position offset.
- replaceData(offset, count, text) - Replaces the text starting at offset through offset + count with text.
- splitText(offset) - Splits the text node into two text nodes separated at position offset.
- substringData(offset, count) - Extracts a string from the text beginning at position offset and continuing until offset + count.
- length property or nodeValue.length or data.length returns the number of characters in the node.

```
<!-- no content, so no text node -->
<div></div>
<!-- white space content, so one text node -->
<div> </div>
<!-- content, so one text node -->
<div>Hello World!</div>
```

The following code lets you access this node:

```
let textNode = div.firstChild; // or div.childNodes[0]
```

Once a reference to the text node is retrieved, it can be changed like this:

```
div.firstChild.nodeValue = "Some other message";
```

Creating Text Nodes

- New text nodes can be created using the document.createTextNode() method, which accepts a single argument the text to be inserted into the node.
- As with setting the value of an existing text node, the text will be HTML or XML encoded, as shown in this example:

```
let textNode = document.createTextNode("<strong>Hello</strong> world!");
```

- it does not appear in the browser window until it is added to a node in the document tree. The following code creates a new element and adds a message to it:

```
let element = document.createElement("div");
element.className = "message";

let textNode = document.createTextNode("Hello world!");
element.appendChild(textNode);

document.body.appendChild(element);
```

Attr Type

- Element attributes are represented by the Attr type in the DOM.
- The Attr type constructor and prototype are accessible in all browsers.
- Technically, attributes are nodes that exist in an element's attributes property.
- Attribute nodes have the following characteristics:
    - nodeType is 2.
    - nodeName is the attribute name.
    - nodeValue is the attribute value.
    - parentNode is null.
    - Child nodes are not supported in HTML.
    - Child nodes may be Text or EntityReference in XML.
- Even though they are nodes, attributes are not considered part of the DOM document tree.
- Attribute nodes are rarely referenced directly, with most developers favoring the use of getAttribute(), setAttribute(), and removeAttribute().

Creating Attribute Nodes

New attribute nodes can be created by using `document.createAttribute()` and passing in the name of the attribute. For example, to add an align attribute to an element, the following code can be used:

```
let attr = document.createAttribute("align");
attr.value = "left";
element.setAttributeNode(attr);

alert(element.attributes["align"].value); // "left"
alert(element.getAttributeNode("align").value); // "left"
alert(element.getAttribute("align")); // "left"
```

Note

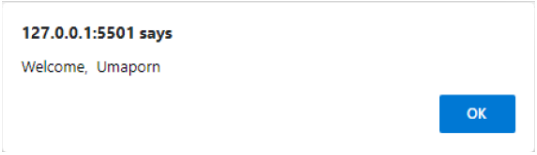
- The Attr node inherits from Node but is not considered a part of the document tree.
- Common Node attributes like parentNode, previousSibling, and nextSibling are null for an Attr node. You can, however, get the element to which the attribute belongs with the ownerElement property.

- Note that this is different from the W3C XML Document Object Model (DOM), which does not treat the element with an attribute as the parent of the attribute.

System Dialogs //07_SystemDialogs

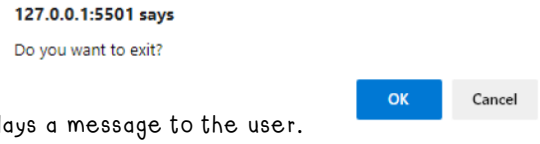
- The browser is capable of invoking system dialogs to display to the user through the **alert(), confirm(),** and **prompt()** methods.
- These dialogs are not related to the web page being displayed in the browser and do not contain HTML.
- Their appearance is determined by operating system and/or browser settings rather than CSS.
- Additionally, each of these dialogs is synchronous and modal, meaning code execution stops when a dialog is displayed, and resumes after it has been dismissed.

System Dialogs: alert()



- The alert() method simply accepts a string to display to the user.
- When alert() is called, a system message box displays the specified text to the user, followed by a single OK button.
- Alert dialogs are typically used when users must be made aware of something that they have no control over, such as an error.

System Dialogs: confirm()



- A confirm() method looks similar to an alert dialog in that it displays a message to the user.
- The main difference between the two is the **presence of a Cancel button along with the OK button**, which allows the user to indicate if a given action should be taken.
- The confirm() method returns true if the user clicked "OK", and false otherwise.

System Dialogs: prompt()

- The final type of dialog is displayed by calling prompt(), which along **with OK and Cancel buttons**, this dialog has a text box where the user may enter some data.
- The prompt() method accepts two arguments: the *text to display* to the user, and the *default value* for the text box (which can be an empty string).
- If the OK button is clicked, prompt() returns the value in the text box; if Cancel is clicked or the dialog is otherwise closed without clicking OK, the function returns null.