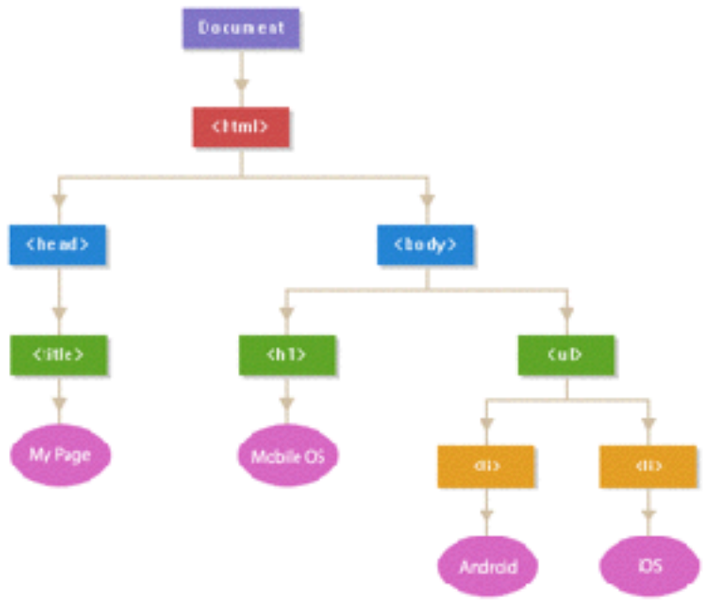
Document Object Model

felipe@ift3225 Hiver 2020

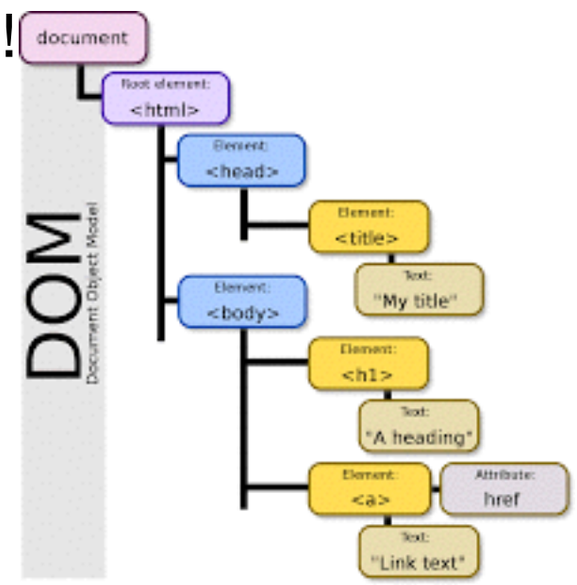


Faits

- DOM 0 - 1995 - guerre des navigateurs (procédés propriétaires)
- DOM 1 - 1998 - première norme (2 modules: Core et HTML)
- DOM 2 - 2000 - en 6 modules (Core, Views, Events, Style, Traversal and Range, and the DOM2 HTML) — introduction dans le Core de `getElementById()`
- **DOM 3** - 2004 - 5 modules (DOM3 Core, Load and Save, Validation, Events, and XPath)
- DOM 4 - 2015 - dernière modification: 24 janvier 2020 !

Qu'est-ce que c'est?

- Une interface (agnostique au langage) permettant de représenter ou modifier un document XML/HTML.
- Chaque document est représenté par un arbre *scriptable*.



https://software.hixie.ch/utilities/js/live-dom-viewer/

```html
<!DOCTYPE html>
<html lang="fr">
<head>
  <title>Bonjour le Monde</title>
  <meta charset="UTF-8">

  <script>
    var init = function() {
        //
    };
  </script>
</head>


<body onload="init();">

Bonjour le monde !
<ul id="malist" style="background-color:#d5f4e6;">
    <li>un</li>
    <li id="monitem">deux</li>
    <li>trois</li>
</ul>
</body>

</html>
```

DOM view (hide, refresh):

```
DOCTYPE: html
HTML lang="fr"
  HEAD
    #text:
    TITLE
      #text: Bonjour le Monde
    #text:
    META charset="UTF-8"
    #text:
    SCRIPT
      #text: var init = function() { // };
    #text:
  #text:
  BODY onload="init();"
    #text: Bonjour le monde !
    UL id="malist" style="background-color:#d5f4e6;"
      #text:
      LI
        #text: un
      #text:
      LI id="monitem"
        #text: deux
      #text:
      LI
        #text: trois
      #text:
    #text:
```

Tous les éléments HTML correspondent à des noeuds, pas l’inverse

DOM et javascript

- La DOM d'une page web est crée en mémoire par votre navigateur avant que la page ne soit visualisée
    - coûteux en mémoire
- L'interpréteur javascript embarqué dans votre navigateur implémente les interfaces de la DOM

- **Règle**: un script javascript dans une page HTML ne peut accéder aux éléments de la DOM qui le suivent
- **Recommandation**: il ne devrait pas y avoir de script javascript ailleurs que dans l'élément **<head>**
    - Attendre que la page (et donc la DOM) soit disponible avant de scripter
        - une solution:

```
<body onload= "init();">
    …
</body>
```

mix de javascript et de contenu…

DOM et javascript

- Une autre possibilité (préférable) consiste à enregistrer les gestionnaires d'événements directement dans le .js
    - plus de contamination dans le HTML
- Deux façons de le faire:
    - Standardisé W3C (DOM 2):

```
window.addEventListener("load",init,false);
window.addEventListener("keypress",doKey,false);
```

    - Certains éléments ont des propriétés pour lier un gestionnaire d'événement

```
document.querySelector('button').onclick = function(event) {
    …
};
```

      et plein d'autres: onclick, onblur, onfocus, etc.

8	**fgColor** Deprecated - A string that specifies the default text color for the document **Ex** – document.fgColor
9	**forms[ ]** An array of Form objects, one for each HTML form that appears in the document. **Ex** – document.forms[0], document.forms[1] and so on
10	**images[ ]** An array of Image objects, one for each image that is embedded in the document with the HTML <img> tag. **Ex** – document.images[0], document.images[1] and so on
11	**lastModified** A read-only string that specifies the date of the most recent change to the document **Ex** – document.lastModified
12	**linkColor** Deprecated – A string that specifies the color of unvisited links **Ex** – document.linkColor
13	**links[ ]** It is a document link array. **Ex** – document.links[0], document.links[1] and so on
14	**location** The URL of the document. Deprecated in favor of the URL property. **Ex** – document.location

DOM 0

```
function myFunc() {

    var ret = document.title;
    alert("Document Title : " + ret );

    var ret = document.URL;
    alert("Document URL : " + ret );

    var ret = document.forms[0];
    alert("Document First Form : " + ret );

    var ret = document.forms[0].elements[1];
    alert("Second element : " + ret );
}
```

DOM 2&3

```
interface Document : Node {
  // Modified in DOM Level 3:
  readonly attribute DocumentType    doctype;
  readonly attribute DOMImplementation implementation;
  readonly attribute Element         documentElement;
  Element            createElement(in DOMString tagName)
                                     raises(DOMException);
  DocumentFragment   createDocumentFragment();
  Text               createTextNode(in DOMString data);
  Comment            createComment(in DOMString data);
  CDATASection       createCDATASection(in DOMString data)
                                     raises(DOMException);
  ProcessingInstruction createProcessingInstruction(in DOMString target,
                                                     in DOMString data)
                                     raises(DOMException);
  Attr               createAttribute(in DOMString name)
                                     raises(DOMException);
  EntityReference    createEntityReference(in DOMString name)
                                     raises(DOMException);
  NodeList           getElementsByTagName(in DOMString tagname);
  // Introduced in DOM Level 2:
  Node               importNode(in Node importedNode,
                                in boolean deep)
                                     raises(DOMException);
  // Introduced in DOM Level 2:
  Element            createElementNS(in DOMString namespaceURI,
                                     in DOMString qualifiedName)
                                     raises(DOMException);
  // Introduced in DOM Level 2:
  Attr               createAttributeNS(in DOMString namespaceURI,
                                       in DOMString qualifiedName)
                                     raises(DOMException);
  // Introduced in DOM Level 2:
  NodeList           getElementsByTagNameNS(in DOMString namespaceURI,
                                            in DOMString localName);
  // Introduced in DOM Level 2:
  Element            getElementById(in DOMString elementId);
  // Introduced in DOM Level 3:
  readonly attribute DOMString       inputEncoding;
  // Introduced in DOM Level 3:
  readonly attribute DOMString       xmlEncoding;
  // Introduced in DOM Level 3:
           attribute boolean         xmlStandalone;
                                     // raises(DOMException) on setting
```

Exemple

```
<html>
<head>
…
<script>
var init = function() {
 console.log(
  "name: "     + document.doctype.name                 + "\n" +
  "tagName: "  + document.documentElement.tagName      + "\n"
  );
}
</script>
</head>

<body onload= "init();">
    …
</body>
</html>
```

The name of DTD; i.e., the name immediately following the `DOCTYPE` keyword.

name: html
tagname: HTML

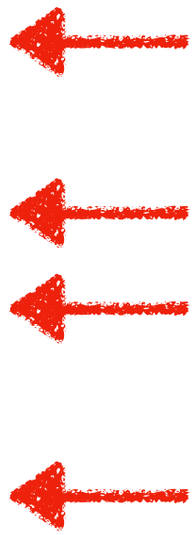
DOM 2&3

IDL Definition

```
interface Element : Node {
  readonly attribute DOMString        tagName;
  DOMString           getAttribute(in DOMString name);
  void                setAttribute(in DOMString name,
                                   in DOMString value)
                                     raises(DOMException);
  void                removeAttribute(in DOMString name)
                                     raises(DOMException);
  Attr                getAttributeNode(in DOMString name);
  Attr                setAttributeNode(in Attr newAttr)
                                     raises(DOMException);
  Attr                removeAttributeNode(in Attr oldAttr)
                                     raises(DOMException);
  NodeList            getElementsByTagName(in DOMString name);
  // Introduced in DOM Level 2:
  DOMString           getAttributeNS(in DOMString namespaceURI,
                                     in DOMString localName);
  // Introduced in DOM Level 2:
  void                setAttributeNS(in DOMString namespaceURI,
                                     in DOMString qualifiedName,
                                     in DOMString value)
                                       raises(DOMException);
  // Introduced in DOM Level 2:
  void                removeAttributeNS(in DOMString namespaceURI,
                                       in DOMString localName)
                                       raises(DOMException);
  // Introduced in DOM Level 2:
  Attr                getAttributeNodeNS(in DOMString namespaceURI,
                                       in DOMString localName);
  // Introduced in DOM Level 2:
  Attr                setAttributeNodeNS(in Attr newAttr)
                                       raises(DOMException);
  // Introduced in DOM Level 2:
  NodeList            getElementsByTagNameNS(in DOMString namespaceURI,
                                         in DOMString localName);
  // Introduced in DOM Level 2:
  boolean             hasAttribute(in DOMString name);
  // Introduced in DOM Level 2:
  boolean             hasAttributeNS(in DOMString namespaceURI,
                                     in DOMString localName);
};
```

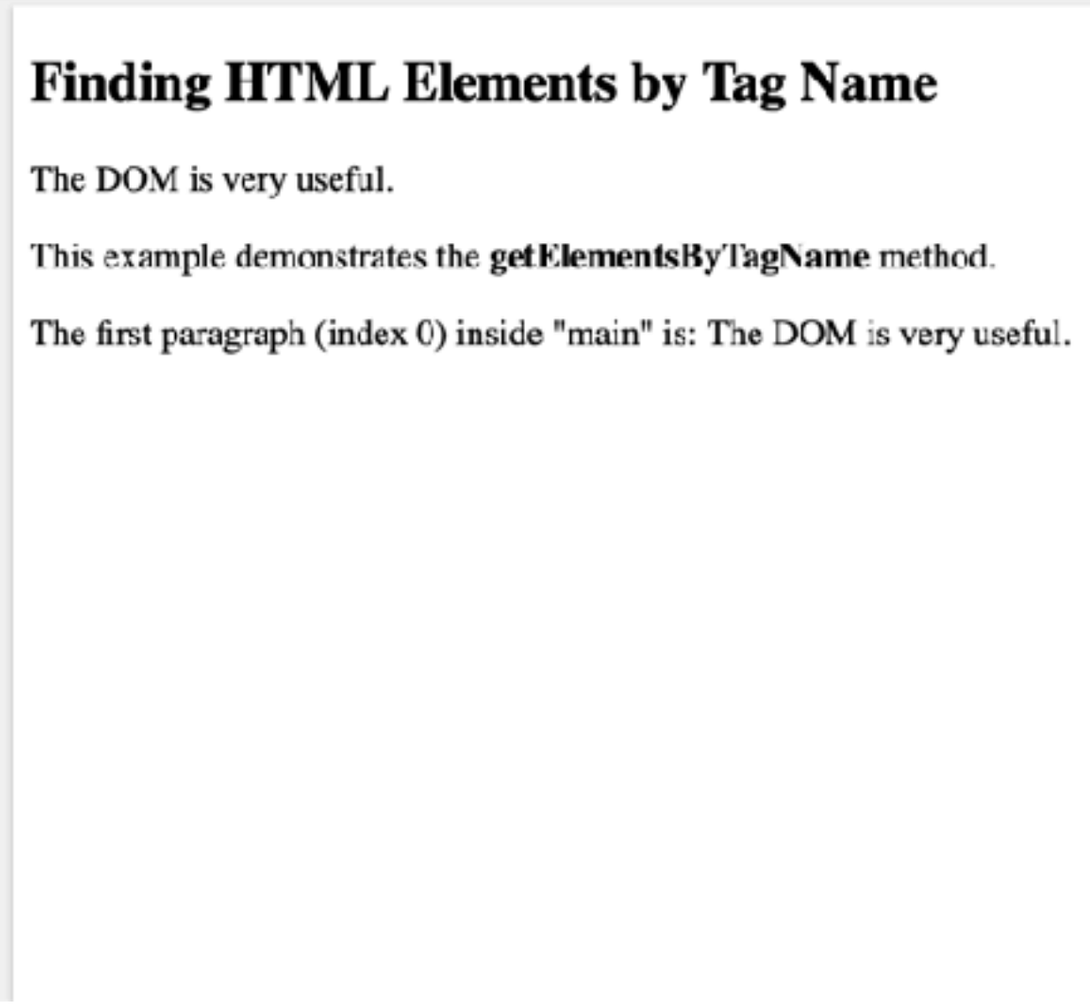
Interface Node

```
interface Node {

  // NodeType
  const unsigned short      ELEMENT_NODE                = 1;
  const unsigned short      ATTRIBUTE_NODE              = 2;
  const unsigned short      TEXT_NODE                   = 3;
  const unsigned short      CDATA_SECTION_NODE          = 4;
  const unsigned short      ENTITY_REFERENCE_NODE       = 5;
  const unsigned short      ENTITY_NODE                 = 6;
  const unsigned short      PROCESSING_INSTRUCTION_NODE = 7;
  const unsigned short      COMMENT_NODE                = 8;
  const unsigned short      DOCUMENT_NODE               = 9;
  const unsigned short      DOCUMENT_TYPE_NODE          = 10;
  const unsigned short      DOCUMENT_FRAGMENT_NODE      = 11;
  const unsigned short      NOTATION_NODE               = 12;

  readonly attribute DOMString       nodeName;
           attribute DOMString       nodeValue;
                                        // raises(DOMException) on setting
                                        // raises(DOMException) on retrieval

  readonly attribute unsigned short  nodeType;
  readonly attribute Node            parentNode;
  readonly attribute NodeList        childNodes;
  readonly attribute Node            firstChild;
  readonly attribute Node            lastChild;
  readonly attribute Node            previousSibling;
  readonly attribute Node            nextSibling;
  readonly attribute NamedNodeMap    attributes;
  // Modified in DOM Level 2:
  readonly attribute Document        ownerDocument;
  // Modified in DOM Level 3:
  Node               insertBefore(in Node newChild,
                                  in Node refChild)
                                     raises(DOMException);
  // Modified in DOM Level 3:
  Node               replaceChild(in Node newChild,
                                  in Node oldChild)
                                     raises(DOMException);
  // Modified in DOM Level 3:
  Node               removeChild(in Node oldChild)
                                     raises(DOMException);
```



getElement...

```html
<!DOCTYPE html>
<html>
<body>

<h2>Finding HTML Elements by Tag Name</h2>

<div id="main">
<p>The DOM is very useful.</p>
<p>This example demonstrates the <b>getElementsByTagName</b> method.
</p>
</div>

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

<script>
var x = document.getElementById("main");
var y = x.getElementsByTagName("p");
document.getElementById("demo").innerHTML =
'The first paragraph (index 0) inside "main" is: ' + y[0].innerHTML;
</script>

</body>
</html>
```



Interface NodeList

```
interface NodeList {
  Node                item(in unsigned long index);
  readonly attribute unsigned long   length;
};
```

```
interface NamedNodeMap {
  Node                getNamedItem(in DOMString name);
  Node                setNamedItem(in Node arg)
                                   raises(DOMException);
  Node                removeNamedItem(in DOMString name)
                                   raises(DOMException);
  Node                item(in unsigned long index);
  readonly attribute unsigned long   length;
  // Introduced in DOM Level 2:
  Node                getNamedItemNS(in DOMString namespaceURI,
                                     in DOMString localName)
                                   raises(DOMException);
  // Introduced in DOM Level 2:
  Node                setNamedItemNS(in Node arg)
                                   raises(DOMException);
  // Introduced in DOM Level 2:
  Node                removeNamedItemNS(in DOMString namespaceURI,
                                        in DOMString localName)
                                   raises(DOMException);
};
```

Exemple

```
<script>
  var get_attrs = function(node) {
      var output = "node " + node.nodeName + " attributes :";
      var  atts = node.attributes;
      if (atts && atts.length)
          for (var i=0; i<atts.length; ++i)
              output += ( " [" + atts[i].name + "='" + atts[i].value +"'']");
      else
          output += " no attribute";

      return output;
  };

  var init = function() {
      var ul = document.getElementById("malist");
      var li = ul.childNodes;

      console.log(
        "ul info: " + ul.tagName + " " + ul.nodeName + " " + ul.nodeValue + "\n" +
        get_attrs(ul) + "\n" +
        get_attrs(li[0]) + "\n" +
        get_attrs(li[1]) + "\n" +
        get_attrs(li[2]) + "\n" +
        get_attrs(li[3]) + "\n"
      );
    };
  </script>
</head>


<body onload="init();">

Bonjour le monde !
<ul id="malist" style="background-color:#d5f4e6;">
    <li>un</li>
    <li id="monitem">deux</li>
    <li>trois</li>
</ul>
</body>
```

```
ul info: UL UL null
node UL attributes : [id='malist''] [style='background-color:#d5f4e6;'']
node #text attributes : no attribute
node LI attributes : no attribute
node #text attributes : no attribute
node LI attributes : [id='monitem'']
```

HTML DOM

IDL Definition

```
interface HTMLDocument : Document {
           attribute DOMString       title;
  readonly attribute DOMString       referrer;
  readonly attribute DOMString       domain;
  readonly attribute DOMString       URL;
           attribute HTMLElement     body;
  readonly attribute HTMLCollection  images;
  readonly attribute HTMLCollection  applets;
  readonly attribute HTMLCollection  links;
  readonly attribute HTMLCollection  forms;
  readonly attribute HTMLCollection  anchors;
           attribute DOMString       cookie;
                                        // raises(DOMException) on setting

  void               open();
  void               close();
  void               write(in DOMString text);
  void               writeln(in DOMString text);
  NodeList           getElementsByName(in DOMString elementName);
};
```

IDL Definition

```
interface HTMLElement : Element {
           attribute DOMString       id;
           attribute DOMString       title;
           attribute DOMString       lang;
           attribute DOMString       dir;
           attribute DOMString       className;
};
```

HTML DOM

```
interface HTMLBodyElement : HTMLElement {
           attribute DOMString       aLink;
           attribute DOMString       background;
           attribute DOMString       bgColor;
           attribute DOMString       link;
           attribute DOMString       text;
           attribute DOMString       vLink;
};
```

```
interface HTMLFormElement : HTMLElement {
  readonly attribute HTMLCollection  elements;
  readonly attribute long            length;
           attribute DOMString       name;
           attribute DOMString       acceptCharset;
           attribute DOMString       action;
           attribute DOMString       enctype;
           attribute DOMString       method;
           attribute DOMString       target;
  void               submit();
  void               reset();
};
```

Chaque élément HTML possède son interface

```
interface HTMLTableElement : HTMLElement {
  // Modified in DOM Level 2:
           attribute HTMLTableCaptionElement caption;
                                              // raises(DOMException) on setting

  // Modified in DOM Level 2:
           attribute HTMLTableSectionElement tHead;
                                              // raises(DOMException) on setting

  // Modified in DOM Level 2:
           attribute HTMLTableSectionElement tFoot;
                                              // raises(DOMException) on setting

  readonly attribute HTMLCollection  rows;
  readonly attribute HTMLCollection  tBodies;
           attribute DOMString       align;
           attribute DOMString       bgColor;
           attribute DOMString       border;
           attribute DOMString       cellPadding;
           attribute DOMString       cellSpacing;
           attribute DOMString       frame;
           attribute DOMString       rules;
           attribute DOMString       summary;
           attribute DOMString       width;
  HTMLElement        createTHead();
  void               deleteTHead();
  HTMLElement        createTFoot();
  void               deleteTFoot();
  HTMLElement        createCaption();
  void               deleteCaption();
  // Modified in DOM Level 2:
  HTMLElement        insertRow(in long index)
                                        raises(DOMException);
  // Modified in DOM Level 2:
  void               deleteRow(in long index)
                                        raises(DOMException);
};
```

HTML DOM

```
<!DOCTYPE html>
<html>
<body>

<img id="image" src="smiley.gif" width="160" height="120">

<script>
document.getElementById("image").src = "landscape.jpg";
</script>

<p>The original image was smiley.gif, but the script changed it to
landscape.jpg</p>

</body>
</html>
```

The original image was smiley.gif, but the script changed it to landscape.jpg

Selector API Level 2

This is an example table written in HTML5.

```
<table id="score">
  <thead>
    <tr>
      <th>Test
      <th>Result
  <tfoot>
    <tr>
      <th>Average
      <td>82%
  <tbody>
    <tr>
      <td>A
      <td>87%
    <tr>
      <td>B
      <td>78%
    <tr>
      <td>C
      <td>81%
</table>
```

In order to obtain the cells containing the results in the table, which might be done, for example, to plot the values on a graph, there are at least two approaches that may be taken. Using only the APIs from DOM Level 2, it requires a script like the following that iterates through each `tr` within each `tbody` in the `table` to find the second cell of each row.

```
var table = document.getElementById("score");
var groups = table.tBodies;
var rows = null;
var cells = [];

for (var i = 0; i < groups.length; i++) {
  rows = groups[i].rows;
  for (var j = 0; j < rows.length; j++) {
    cells.push(rows[j].cells[1]);
  }
}
```

Alternatively, using the `querySelectorAll()` method, that script becomes much more concise.

```
var cells = document.querySelectorAll("#score>tbody>tr>td:nth-of-type(2)");
```

Selector API

```
partial interface Document {
  Element?  querySelector(DOMString selectors);
  NodeList  querySelectorAll(DOMString selectors);

  Element?  find(DOMString selectors, optional (Element or sequence<Node>)? refNodes);
  NodeList  findAll(DOMString selectors, optional (Element or sequence<Node>)? refNodes);
};

partial interface DocumentFragment {
  Element?  querySelector(DOMString selectors);
  NodeList  querySelectorAll(DOMString selectors);

  Element?  find(DOMString selectors, optional (Element or sequence<Node>)? refNodes);
  NodeList  findAll(DOMString selectors, optional (Element or sequence<Node>)? refNodes);
};

partial interface Element {
  Element?  querySelector(DOMString selectors);
  NodeList  querySelectorAll(DOMString selectors);

  Element?  find(DOMString selectors);
  NodeList  findAll(DOMString selectors);

  boolean   matches(DOMString selectors, optional (Element or sequence<Node>)? refNodes);
};
```

Créer un noeud

Principe: dès qu'un noeud est ajouté/modifié dans la DOM, la page web affiche à nouveau la page

```
var n = document.createElement('h4');
n.appendChild(document.createTextNode('mon titre'));

// l'élément n'est pas encore ajouté


document.getElementsByTagName('body').item(0).insertBefore(n,
document.getElementsByTagName('p').item(0));

// ajout d'un h4 devant le premier <p>
```

Testez vous !

https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Test_your_skills:_Events

In our first events-related task, you need to create a simple event handler that causes the text inside the button (`btn`) to change when it is clicked on, and change back when it is clicked again.The HTML should not be changed; just the JavaScript.

```
<html>
…
<body>

    <section class="preview">
    </section>

    <button class="off">Machine is off</button>

</body>

<script>
    let btn = document.querySelector(‘.off');
    …
</script>
</html>
```

Testez vous !

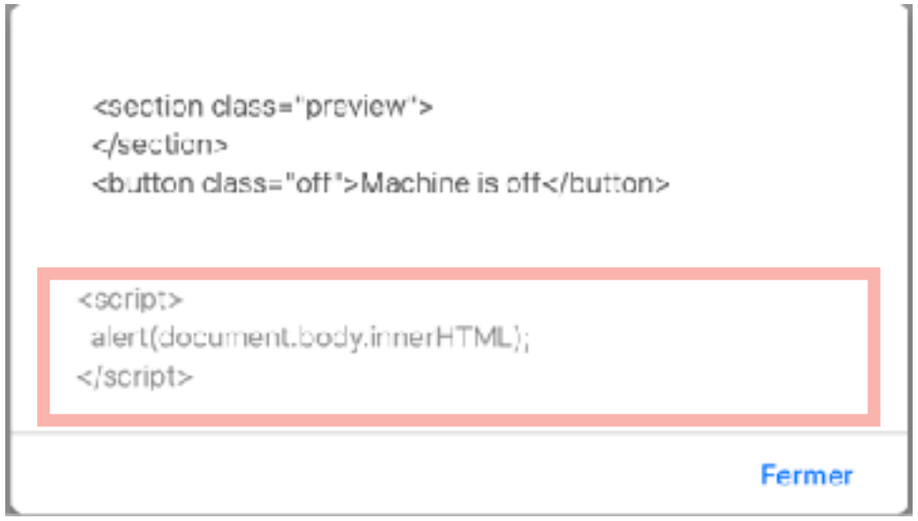
```
<script>
    let btn = document.querySelector('.off');
    let offText = btn.innerHTML;
    let onText = "Machine is on";

    // Add your code here
    btn.onclick = function() {
      let isOff = btn.getAttribute('class') === "off";
      btn.innerHTML = isOff? onText : offText;
      btn.setAttribute('class',isOff? "on" : "off");
    };

</script>
```

Préférez lui `textContent`

innerHTML

- Propriété de l'interface **Element**
- **Lecture =** Sérialisation (string) d'un élément de la DOM

```
<body>
   <section class="preview">
   </section>
   <button class="off">Machine is off</button>
</body>

<script>
   alert(document.body.innerHTML);
</script>
```



!

- **Écriture**
  - Le texte est parsé, résultant en un `DocumentFragment`
  - Le contenu de l'élément est remplacé par ce fragment

```
<body>
   <section class="preview">
   </section>
   <button class="off">Machine is off</button>
 </body>

 <script>
   document.body.innerHTML = « <b>bonjour</b>";
 </script>
```

InnerHTML

- Dangereux en écriture

```
<body>
    <section class="preview">
    </section>
    <button class="off">Machine is off</button>
  </body>

  <script>
     document.body.innerHTML = "<img src='x' onerror='alert(1)'>";
  </script>
```

- La propriété `textContent` n’implique pas de parsing

```
<body>
    <section class="preview">
    </section>
    <button class="off">Machine is off</button>
  </body>

  <script>
     document.body.textContent = "<img src='x' onerror='alert(1)'>";
  </script>
```