Accessing HTML elements

- **Selector API**
- **DOM API**

Selector API

They introduce a way to use CSS selectors (including CSS3 selectors) for requesting the DOM, like jQuery introduced ages ago.

Any **CSS selector** can be passed as a parameter for these methods.

- **querySelector(css_selector)** will return the **first element in the DOM that matches the selector** (and you will be able to work with it directly).
- **querySelectorAll(css_selector)** returns a **collection of HTML elements corresponding to all elements matching the selector**. To process the results, it will be necessary to loop over each of the elements in the collection.

Application

Looking for an element in the whole document (the whole HTML page): call the querySelector method (or querySelectorAll) on the document object that corresponds to the whole DOM tree of your web page:

```
<!--
we have two buttons that will call a JavaScript function where
we will manipulate the DOM), and we have four images, the first
one with an id equal to "img1".
-->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width">
  <title>querySelector and querySelectorAll</title>
</head>
<body>
  <button onclick="addBorderToFirstImage();">
    Add a border to the first image
  </button>
  <br>
  <button onclick="resizeAllImages();">
    Resize all images
  </button>
  <br>
```

```
  <p>Click one of the buttons above!</p>
<img src="https://mainline.i3s.unice.fr/mooc/Ntvj5rq.png"
     id="img1"
     width=200 alt="image #1">
<img src="https://mainline.i3s.unice.fr/mooc/yiU59oi.gif"
     width=200 alt="image #2">
<img src="https://mainline.i3s.unice.fr/mooc/6FstYbc.jpg"
     width=200 alt="image #3">
<img src="https://mainline.i3s.unice.fr/mooc/L97CyS4.png"
     width=200 alt="image #4">

<script type="text/javascript">

  window.onload = init; // run 'init' once the page is loaded
   // the 'init' function is executed as soon as
  // the page is loaded (and the DOM is ready)

  // this function runs once the page is loaded
  function init() {
    // we're sure that the DOM is ready
    // before querying it add a shadow to all images
    // select all images
    var listImages = document.querySelectorAll("img");

    // change all their width to 100px
    listImages.forEach(function(img) {
      // img = current image
      img.style.boxShadow = "5px 5px 15px 5px grey";
      img.style.margin = "10px";
    });
  }

  function addBorderToFirstImage() {
    // select the first image with id = img1
    var img1 = document.querySelector('#img1');

    // Add a red border, 3px wide
    img1.style.border = '3px solid red';
  }

  function resizeAllImages() {
    // select all images
    var listImages = document.querySelectorAll("img");

    // change all their width to 100px
    listImages.forEach(function(img) {
      // img = current image
      img.width = 100;});
  }
</script></body></html>
```

DOM API

To access the elements of the page we can use the methods from the DOM API and can all be replaced by the **querySelector** and **querySelectorAll** methods. They are still used in many JavaScript applications, and are very simple to understand.

From the document we can access the elements composing our web page in a few ways:

- `document.getElementById(identifier)` returns the element which has the id “identifier”.
- `document.getElementsByTagName(tagName)` returns a list of elements which are named “tagName”.
- `document.getElementsByClassName(className)` returns a list of elements which have the class “className”.

Notice that `identifier`, `tagName` and `className` must be of type String.

Selector API vs DOM API

DOM API	Selector API
`document.getElementById(identifier)`	`document.querySelector('#identifier')`
Example: `var elm = document.getElementById('myDiv');` is equivalent to `document.querySelector('#myDiv');`	
`document.getElementsByTagName(tagName)`	`document.querySelectorAll('tagName')`
Example: `var list = document.getElementsByTagName('img');` is equivalent to `document.querySelectorAll('img');`	
`document.getElementsByClassName(className)`	`document.querySelectorAll('.className')`
Example: `var list = document.getElementsByClassName('important');` is equivalent to `document.querySelector('.important');`	

Other examples that use more complex selectors

```
// all elements li in ul elements in an element of id=nav
var el = document.querySelector('#nav ul li');

// all li in a ul, but only even elements
var els = document.querySelectorAll('ul li:nth-child(even)');


// all td directly in tr in a form of class test
var els = document.querySelectorAll('form.test > tr > td');

// all paragraphs of class warning or error
querySelectorAll("p.warning, p.error");

// first element of id=foo or id=bar
querySelector("#foo, #bar");

// first p in a div
var div = document.getElementById("bar");
var p = div.querySelector("p");
```

Changing the **style** of selected HTML elements

The **style** attribute: how to modify an HTML element's CSS properties from JavaScript

The most common way to modify the CSS style of one of several elements you selected using the DOM or Selector API, is to use the style attribute.

Typical use:

```
// select the paragraph with id = "paragraph1"
var p = document.querySelector('#paragraph1');
// change its color
p.style.color = 'red';
```

The most useful CSS properties (we do recommend that you follow the W3Cx courses CSS basics, CSS and HTML5 fundamentals from W3Cx to learn more about CSS):

- **color:** changing the color of the text content of selected element(s).
- **background-color:** same but this time the background color.
- **margin** and **padding** properties (external and internal margins), including their variants: margin-left, margin-top, margin-right, margin-bottom, also padding-left, etc.
- **border** and **border-radius:** change the border, type (plain, dashed), color, thickness, rounded corners etc.
- **box-shadow** to add shadows to selected elements.

- **font, font-style:** font characters and style (italic, bold, plain).
- **text-align** (centered, justified...).

Using the ClassList interface to change more than one CSS property simultaneously

External resources:

- The W3C specification about the classList DOM interface
- An article from the Mozilla Developer's web site

Until now, to manipulate CSS classes of an HTML element was a bit complex, both for verifying the presence of a class name in an element, and for adding or removing classes associated with a given element.

The **ClassList API** simplifies it all by acting as a container object and by providing a set of methods to manipulate its content.

The classList property applies to an HTML element, and returns a collection of class names:

```
var elem= document.querySelector("#id1");

var allClasses = elem.classList;
```

The list of methods usable on a classList object are **add()**, **remove()**, **toggle()** and **contains().**

```
// By default, start without a class in the div: <div class=""/>

// Set "foo" as the class by adding it to the classList
div.classList.add('foo'); // now <div class="foo"/>

// Check that the classList contains the class "foo"
div.classList.contains('foo'); // returns true

// Remove the class "foo" from the list
div.classList.remove('foo'); // now <div class=""/>

// Check if classList contains the class "foo"
div.classList.contains('foo'); // returns false: "foo" is gone

// Check if class contains the class "foo",
// If it does, "foo" is removed, if it doesn't, it's added
div.classList.toggle('foo'); // class set to <div class="foo"/>
div.classList.toggle('foo'); // class set to <div class=""/>
```

Changing the **content** of HTML elements

Properties that can be used to change the value of a DOM node:

1. Using the **innerHTML** property

This property is useful when you want to change all the children of a given element. It can be used to modify the text content of an element, or to insert a whole set of HTML elements inside another one.

Typical use:

```
var elem = document.querySelector('#myElem');

elem.innerHTML = 'Hello '; // replace content by Hello

elem.innerHTML += '<b>Michel Buffa</b>', // append at the end
                                         // Michel Buffa in bold

elem.innerHTML = 'Welcome' + elem.innerHTML; // insert Welcome
                                             // at the beginning
elem.innerHTML = ''; // empty the elem
```

2. Using the **textContent** property

It's also possible, with selected nodes/elements that contain text, to use the `textContent` property to read the text content or to modify it.

Extract from the HTML code:

```
1. <p id="first">first paragraph</p>
2. <p id="second"><em>second</em> paragraph</p>
```

JavaScript code: the comments after lines that start with console.log correspond to what is printed in the devtool debug console. Notice the difference between the textNode value and the innerHTML property values at lines 13-14: while textContent returns only the text inside the second paragraph, innerHTML also returns the ... that surrounds it. However, when we modify the textContent value, it also replaces the text decoration (the is removed), this is done at lines 16-20.

```
1. window.onload = init;
2.
3. function init() {
4.     // DOM is ready
5.     var firstP = document.querySelector("#first");
6.     console.log(firstP.textContent); // "first paragraph"
7.     console.log(firstP.innerHTML);   // "first paragraph"
8.
9.     firstP.textContent = "Hello I'm the first paragraph";
```

```
10.     console.log(firstP.textContent); // "Hello I'm the first
   paragraph"
11.
12.     var secondP = document.querySelector("#second");
13.     console.log(secondP.textContent); // "second paragraph"
14.     console.log(secondP.innerHTML);   // "<em>second</em>
   paragraph"
15.
16.     secondP.textContent = "Hello I'm the second paragraph";
17.     console.log(secondP.textContent); // "Hello I'm the second
18.                                       // paragraph"
19.     console.log(secondP.innerHTML);   // "Hello I'm the second
20.                                       // paragraph"
21.  }
```

Adding new elements to the DOM

The DOM API comes with a set of methods you can use on DOM elements.

In general, to **add new nodes to the DOM** we follow these steps:

1. Create a new element by calling the **createElement()** method, using a syntax like:

   ```
   var elm = document.createElement(name_of_the_element).
   ```

   Examples:

   ```
   var li = document.createElement('li');

   var img = document.createElement('img'); etc.
   ```

2. Set some attributes / values / styles for this element.

   Examples:

   ```
   li.innerHTML = '<b>This is a new list item in bold!</b>';
   // can add HTML in it

   li.textContent = 'Another new list item';

   li.style.color = 'green'; // green text

   img.src = "http://..../myImage.jpg"; // url of the image

   img.width = 200;
   ```

3. Add the newly created element to another element in the DOM, using append(), appendChild(), insertBefore() or the innerHTML property

   Examples:

```
var ul = document.querySelector('#myList');

ul.append(li); // insert at the end, appendChild() could
also be used (old)

ul.prepend(li); // insert at the beginning

ul.insertBefore(li, another_element_child_of_ul);//
insert in the middle

document.body.append(img); // adds the image at the end
of the document
```

A warning about **append** vs **appendChild**, prepend, etc...

- The DOM specification recently added some jQuery like methods that are similar to the ones proposed by the jQuery library (that was designed a long time ago to simplify DOM manipulations).
- For a long time, developers used **document.appendChild** to add an element to the DOM. You can also use **document.append,** along with some other methods such as document.prepend etc. See this table for compatibility.
- If you are looking for maximum compatibility, you can either use **document.appendChild** instead of document.append.

Use of the **createElement()**, append() methods and of the **textContent** attribute

HTML code extract: we use an `<input type="number">` for entering a number (line 2). Then if one clicks on the "Add to the list" button, the `add()` JavaScript function is called (line 3), this will add the typed number to the empty list at line 7. If one presses the "reset" button, it will empty this same list by calling the `reset()` JavaScript function.

```
1. <label for="newNumber">Please enter a number</label>
2. <input type="number" id="newNumber" value=0>
3. <button onclick="add();">Add to the list</button>
4. <br>
5. <button onclick="reset();">Reset list</button>
6.
7. <p>You entered:</p>
8. <ul id="numbers"></ul>
```

JavaScript code extract: notice at line 25 the use of the `innerHTML` property for resetting the content of the `<ul>` list. innerHTML corresponds to all the sub DOM contained inside the <ul>.... InnerHTML can be used for adding/deleting/modifying a DOM node's content.

```
1. function add() {
2.   // get the current value of the input field
3.   var val = document.querySelector('#newNumber').value;
4.
5.   if((val !== undefined) && (val !== "")) {
6.     // val exists and non empty
7.
8.     // get the list of numbers. It's a <ul>
9.     var ul = document.querySelector("#numbers");
10.
11.         // add it to the list as a new <li>
12.         var newNumber = document.createElement("li");
13.         newNumber.textContent = val;
14.         // or newNumber.innerHTML = val
15.
16.         ul.append(newNumber);
17.       }
18.     }
19.
20.
21.     function reset() {
22.     // get the list of numbers. It's a <ul>
23.     var ul = document.querySelector("#numbers");
24.     // reset it: no children
25.     ul.innerHTML = "";
26.     }
```

Using the innerHTML property to add new elements:

This is the same example, but in an abbreviated form, using the innerHTML property:

```
<body>
  <label for="newNumber">Please enter a number</label>
  <input type="number" id="newNumber" value=0>
  <button onclick="add()">Add to the list</button>
  <br>
  <button onclick="reset();">Reset list</button>

<p>You entered:</p>
<ul id="numbers"></ul>

<script>
function add() {
  // get the current value of the input field
  var val = document.querySelector('#newNumber').value;

  if((val !== undefined) && (val !== "")) {
    // val exists and non empty

    // get the list of numbers. It's a <ul>
```

```
    var ul = document.querySelector("#numbers");

    ul.innerHTML += "<li>" + val + "</li>";
  }
}

function reset() {
  document.querySelector("#numbers").innerHTML = "";
}
</script></body>
```

Moving HTML elements in the DOM

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Moving elements using appendChild()</title>
  <style>
    .box {
      border: silver solid;
      width: 256px;
      height: 128px;
      margin: 10px;
      padding: 5px;
      float: left;
    }
 </style>
</head>
<body>
  <p>Click on a browser image to move it to the list of cool
browsers:</p><br/>
  <img src="https://mainline.i3s.unice.fr/mooc//ABiBCwZ.png" id="cr"
       onclick="move(this)" alt="Logo Chrome">
  <img src="https://mainline.i3s.unice.fr/mooc//n7xo93U.png" id="ff"
       onclick="move(this)" alt="Logo Firefox">
  <img src="https://mainline.i3s.unice.fr/mooc//ugUmuGQ.png" id="ie"
       onclick="move(this)" alt="Logo IE">
  <img src="https://mainline.i3s.unice.fr/mooc//jfrNErz.png" id="op"
       onclick="move(this)" alt="Logo Opera">
  <img src="https://mainline.i3s.unice.fr/mooc//gDJCG0l.png" id="sf"
       onclick="move(this)" alt="Logo Safari"><br/>
  <div class="box" id="coolBrowsers">
      <p>Cool Web browsers</p>
  </div>

  <script>
     function move(elem) {
        var targetList = document.querySelector('#coolBrowsers');
        targetList.append(elem);
        // trick to remove the click listener once
        // the image has been moved into the list
        elem.onclick = null;
```

```
    } </script>
</body>
</html>
```

Removing elements from the DOM

Removing elements using the **removeChild()** method:

JavaScript code extract: we need to get the <ul> that contains all the <input type="checkbox"> elements (*line 3*). This is the element we will use for calling removeChild(...). The loop on the checked element (*lines 5-12*) iterates on a list of checked input elements. In order to make both the text (Apples, Oranges, etc.) AND the checkbox disappear, we need to access the different <li> elements that contain the selected checkboxes. This is done in *line 10*. Then, we can call ul.removeChild(li) on the <ul> for removing the <li> that contains the selected element (line 11).

```
1.  function removeSelected() {
2.    var list = document.querySelectorAll("#fruits input:checked");
3.    var ul = document.querySelector("#fruits");

4.    list.forEach(function(elm) {
5.    // elm is an <input type="checkbox">, its parent is a li
6.    // we want to remove from the <ul> list
7.    // when we remove the <li>, the <input> will also
8.    // be removed, as it's a child of the <li>
9.
10.       var li = elm.parentNode;
11.       ul.removeChild(li);
12.       });
13.       }
```

Remove all children of an element using the innerHTML property

In the same example, if you look at the reset() JavaScript function, we use the ul's innerHTML property both for emptying the list (*lines 3-4*) and for appending to it all the initial HTML code (*lines 6-17*):

```
1. function reset() {
2.   var ul = document.querySelector("#fruits");
3.   // Empty the <ul>
4.   ul.innerHTML = "";
5.
6.   // Adds each list item to the <ul> using innerHTML += ...
7.   ul.innerHTML += "<li><input type='checkbox' name='fruit'
                     value='apples'>Apples</li>";
8.
9.   ul.innerHTML += "<input type='checkbox' name='fruit'
10.                        value='oranges'>Oranges</li><br>";
11.
12.       ul.innerHTML += "<input type='checkbox' name='fruit'
13.                        value='bananas'>Bananas</li><br>";
14.
```

```
15.       ul.innerHTML += "<input type='checkbox' name='fruit'
16.                         value='grapes'>Grapes</li>";
17.     }
```