DOM API

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 selected HTML elements

Adding new elements to the DOM

Moving HTML elements in the DOM

Removing elements from the DOM