What you will learn in Module 3

- **JavaScript arrays and strings:** in this module, we continue to study fundamental JavaScript concepts, and this time we look a bit deeper into JavaScript arrays and strings.

- **New HTML5 APIs - geolocation and video/audio APIs**: in module 2, we looked at some HTML5 APIs already: the selector and the DOM APIs, respectively for selecting and manipulating HTML elements dynamically. We also had a taste of the HTML5 canvas API for drawing and animating. This time, we will look at the audio and video elements APIs, and the geolocation API.

- **We will also add background Music and sound effects to the small game** we started writing during module 2.

Arrays (part 2): iterators, Arrays

JavaScript arrays: In JavaScript, arrays represent a collection of "things", which may be **strings**, **integer** values, **decimal** values, **boolean** values, or **any sort of JavaScript object**.

```
var myarr = ['red', 'blue', 'yellow', 'purple'];
myarr;
Myarr[0]; //output 'red'
Myarr[3]; //output 'purple'
typeof Myarr; //output 'object'
```

Each element of an array has a **key/index** and a **value**. Here are the keys/indexes and values from the above example:

Key	Value
0	red
1	blue
2	yellow
3	purple

Arrays (part 2): iterators, Array methods

JavaScript arrays are objects as you can see when we typed **`typeof Myarr`** , here we mention some of its useful properties and methods.

length	Return the number of the elements
sort()	Sort the array in the ascending order
reverse ()	Reverse the elements of the array
pop()	Remove the last element of the array
shift()	Remove the first element of the array
push(element)	Add **element** to the end of the array
unshift(element)	Add **element** to the start of the array
indexOf(element)	Return the index of **element**

Arrays (part 2): iterators, Array methods

You can access/call them using the "." operator. Here is an example.

```
var a = [1, 3, 2, 5, 7];
a.length; // returns 5
a.sort(); // a now has [1, 2, 3, 5, 7]
a.reverse() // a now has [ 7, 5, 3, 2, 1 ]
a.push(8); // a now has [ 7, 5, 3, 2, 1, 8]
a.pop() // returns 8, and a now has [ 7, 5, 3, 2, 1 ]
a.indexOf(7) // returns 0
```

Arrays (part 2): iterators, Array methods:

sort

By default, the **sort()** method sorts elements **alphabetically** if they are strings, or **from lowest to highest if they are numeric**. If you want to sort objects like `{firstName:'michel', lastName:'Buffa', age:51}`, you will need to use another method passed as an argument to the sort method, for example to indicate the property you want to use for sorting (i.e., sort by age);

```
var persons = [
   {givenName : 'Michel',
    familyName: 'Buffa',
           age:51},
   {givenName: 'Pig',
   familyName: 'Bodine',
          age:20},
   {givenName: 'Pirate',
   familyName: 'Prentice',
          age:32}
   ];
```

```
function compareByAge(a,b) {
   // comparison function, a and b are persons
      if (a.age < b.age)          // compare by age
      return -1;
      if (a.age > b.age)
      return 1;
      return 0;
   }

   persons.sort(compareByAge); // this will call
automatically compareByAge passing all persons from the
array, compare them by age and sort the array.
```

Arrays (part 2): iterators, Array methods:

sort(trick)

We can use a smart implementation of the sort function that is has **less computation** and **short syntax**:

```
persons.sort((personA, personB) =>{
    if(personA.age > personB.age) return 1;
})
```

Arrays (part 2): iterators, Array methods: indexOf

For primitive values indexOf works fine, how about objects ???

- It **won't work** if you create a new object:

```
person =  {givenName: 'Michel', familyName: 'Buffa', age:51}
persons.indexOf(person) // return -1
```

- It **works** if person refer to an object inside that array:

```
person = persons[0];
persons.indexOf(person) //returns 0
```

Detail explanation:

- For the first case you can find that object by looping on the array and comparing element by element of that object.
- In the second case JavaScript doesn't compare object by object, it the just references, because Javascript stores variable reference not by memory address(pointer)

Arrays (part 2): iterators, Array methods:

concat

Join two arrays,

Syntax:

arrayA.**concat**(arrayB)

Example:

```
a = [1, 2, 3, 4, 5 ]
b = [6, 7, 8, 9, 10]
C = a.concat(b)
//c = [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ]
```

Arrays (part 2): iterators, Array methods: slice

Copy a chunk of an array,

Syntax:

array.slice(startIndex, endIndex) : the startIndex is included, but the endIndex is excluded.

Example:

```
var a = [1, 3, 2, 5, 7];
a.slice(2,4)// returns [2,5] that are the values of
index  2, and 3
```

Arrays (part 2): iterators, Array methods:

splice

Removing elements from an array, The recommended method is to use the splice method:

Syntax:

```
array.splice(start)
array.splice(start, deleteCount)
```

- **start:** index at which to start changing the array (with origin 0).
- **deleteCount (Optional):** an integer indicating the number of old array elements to remove. If deleteCount is greater than the number of elements left in the array starting at start, then all of the elements through the end of the array will be deleted. If deleteCount is omitted, deleteCount will be equal to (array.length - start), i.e., all of the elements beginning with start index on through the end of the array will be deleted.
- **Return value:** an array containing the deleted elements. If only one element is removed, an array of one element is returned. If no elements are removed, an empty array is returned.

Arrays (part 2): iterators, Array methods:

splice

Example:

```
c = [ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ]
c.splice(3, 3) // returns [3,4,5] and c is equal to
               // to [ 0, 1, 2, 6, 7, 8, 9, 10 ]
c.splice(3)     // returns [6, 7, 8, 9, 10 ]
               // and c equal to  [ 0, 1, 2]
```

Arrays (part 2): iterators, Array methods:

splice

Assignments-3-1: Insert / delete elements inside the array, as I mentioned push/unshift and pop/shift could work just at the start and the end of the array, but if you wanna insert/delete more than one element or do these operation inside an array you need to use a combination of slice and concat.

- Turn [0, 1, 2, 3, 4, 5] to [0, 1, 2, 3, 4, 4.1, 4.2, 4.3, 5] using loops
- Turn [0, 1, 2, 3, 4, 5] to [0, 1, 2, 3, 4, 4.1, 4.2, 4.3, 5] using slice and concat
- Turn [0, 1, 2, 3, 4, 5] to [0, 1, 2, 3, 4, 4.1, 4.2, 4.3, 5] using splice

- Turn back [0, 1, 2, 3, 4, 4.1, 4.2, 4.3, 5] to [0, 1, 2, 3, 4, 5] using loops
- Turn back [0, 1, 2, 3, 4, 4.1, 4.2, 4.3, 5] to [0, 1, 2, 3, 4, 5] using slice and concat
- Turn back [0, 1, 2, 3, 4, 4.1, 4.2, 4.3, 5] to [0, 1, 2, 3, 4, 5] using splice

- Using [0, 1, 2, 3, 4, 5] replace push/unshift/pop/shift by splice

Dealine: 2019-December-15

Arrays (part 2): iterators, Array

methods:undefined

Elements can be of different types in the same array:

```
var a = [1,2,3];
a[2] = 'three'
a //output [1, 2, 'three']
```

When using indexes, be careful not to leave "holes" in the array:

```
a[7] = 'height';
a //output [ 1, 2, "three", <4 empty slots>, "height" ]
```

This array is valid, but having a [3] to a[6] equals to "undefined" is often prone to errors. Be careful when using absolute indexes for adding elements. We recommend using the **push** method instead.

Trap: the **delete** method is not good for removing an element from an array!

```
a.splice(3) // a equal to [1, 2, 'three']
delete a[1];// a equal to [1, <1 empty slots>, 'three']
// the element became undefined,
// but it's still in the array!
```

Arrays (part 2): iterators, Array

methods:array of arrays

It is possible for an array to be an element within an array! This example shows an array made of two arrays of three elements each. It's a 2x3 matrix with two rows and three columns!

```
var a = [[1,2,3], [4,5,6]]; // a is a matrix: 2 rows, 3 columns.
a[0]; // first row
a[1]; // second row
a[0][0]; // top left element
a[0][1]; // second element, first line
a[0][2]; // third element, first line
a[1][0]; // first element, second line
```

It is possible to have different arrays with different lengths and different types of element in an array:

```
var a = [];
a[0] = [1, 2, 3, 4, 5];
a[1] = ['michel', 'henri', 'francois']
["michel", "henri", "francois"]
```

Arrays (part 2): iterators, Strings are arrays of characters

Strings are arrays of characters, Yes, they do look like arrays!

JavaScript strings are "like" arrays of characters, but they have some limitations, and some dedicated properties and methods:

```
var s = 'Michel';
s[0];// output "M"
s[1]; // output "i"
s.length // output 6
```

Indeed, the string **s** behaves like an array, it has the length property like an array, and we can access individual characters using indexes that go from 0 to length-1, like arrays...

However...

Arrays (part 2): iterators, Strings are arrays of characters

But they are not quite the same as arrays!

You cannot add elements to strings using a non-existent index, you **cannot** use the **push/pop** methods for **adding/removing** characters at the end of the string:

```
s.push(' Buffa');
//output: ERROR: VM5748:1 Uncaught TypeError: s.push is not a function
//at <anonymous>:1:3 (anonymous) @ VM5748:1

s.splice(0, 3);// the same error as above

s[s.length] = 'B'; // add 'B' at the end?
//output: "B"

s[s.length] = 'u'; // add 'u' at the end?
//output: "u"
```

Arrays (part 2): iterators, Strings are arrays of characters

```
s[s.length] = 'f'; // add 'f' at the end?
//output: "f"

s; // s remained UNCHANGED!
//output: "Michel"

s[0] = "R"; // trying to change the 'M' into an 'R'
```

Arrays (part 2): iterators, String methods

We take this variable as an example `var str = "Please locate where 'locate' occurs!";`

Method/Property	Usage	Example
str.length	returns the length of a string	str.length
str.indexOf(element)	returns the index of (the position of) the first occurrence of a specified text in a string	str.indexOf(‘locate’)
str.lastIndexOf(element)	returns the index of the last occurrence of a specified text in a string	str.lastIndexOf(‘locate’)
str.search(element)	searches a string for a specified value and returns the position of the match	str.search(‘locate’)
str.slice(start, end) **str.substr(stard, length)**	extracting a part of a string	str.slice(7, 21) str.substr(7, 14)

Arrays (part 2): iterators, String methods

Method/Property	Usage	Example
str.replace('old string', new string')	replaces a specified value with another value in a string	str.replace('locate','far')
str.toUpperCase()	converted to uppercase	str.toUpperCase()
str.toLowerCase()	converted to lowercase	str.toLowerCase()
str1.concat(str2)	joins two or more strings	str.concat('did you find your place')
str.trim()	removes whitespace from both sides of a string	str2 = " " + str + " "; str2.trim();
str.split(' ') str.split("\t")	convert a string to an array	str.split(' ')
str.charCodeAt()	returns the unicode of the index	str.harCodeAt(0)

As you can notice that all these methods mentioned in this page return a **new string**, cause we **can not change the string itself**. **String is an immutable object**.

Arrays (part 2): iterators, String methods

So: how do we add characters to a string, how can we modify a string? How can we delete elements in a string ? String is an immutable object

Here are more examples:

- Adding a string to the **beginning** of a string using the **+ operator**:

```
s = 'Azzeddine'
s = 'Rigat' + s;
s // equal to Rigat Azzeddine
```

- Adding a string to the **end** of another one with the **+ operator**:

```
s = 'Rigat';
s += ' Azzeddine';
s // equal to Rigat Azzeddine
```

- You can easily do the last two operation using cancat method.
- Removing chars from a string using the substring method:

```
s = 'Rigat'
newS = s.slice(0, s.length-1) // remove the last character
```

Arrays (part 2): iterators, String methods

Assignments-3-2:

- build a function that removes a given number of chars from a string, starting at a given index, you can use slice method as a helper.
- Write a script that find all occurrences of substring and replace them inside a string, using replace method.

Dealine: 2019-December-15

Arrays (part 2): iterators, Iterating on array elements(1)

1 - Iterating using the forEach method

The forEach method takes a single argument that is a function/callback that can have one, two or three parameters:

- The first parameter is the current element of the array,
- The second parameter (optional) is the index of the current element in the array,
- The third element is the array itself

Arrays (part 2): iterators, Iterating on array elements(1)

Example with one parameter (the current element):

```
var a = ['Monday', 'Tuesday', 'Wednesday'];

a.forEach(function(day) {
   // day is the current element
   document.body.innerHTML += day +
               "<br>"; // will display Monday, Tuesday, Wednesday
})
```

Arrays (part 2): iterators, Iterating on array elements(1)

Example with two parameters, we iterate on an array of person, and use two parameters in the callback function in order to get the index of the current element:

```
var persons = [
   {name:'Michel', age:51},
   {name:'Henri', age:20},
   {name:'Francois', age:29}
 ];
 persons.forEach(function(p, index) {
   document.body.innerHTML += p.name + ", age " + p.age +
                              ", at index " + index + " in the array<br>";
});
```

Arrays (part 2): iterators, Iterating on array elements(1)

Third example using three parameters, the last one being the array itself, to get the length of the array

```
var persons = [
   {name:'Michel', age:51},
   {name:'Henri', age:20},
   {name:'Francois', age:29}
];
persons.forEach(function(p, index, theArray) {
   document.body.innerHTML += p.name + ", age " + p.age +
                              ", at index " + index + " in the array of " +
                              theArray.length + " elements<br>";
});
```

Arrays (part 2): iterators, Iterating on array elements(2)

2 - Iterating on an array using regular loop statements, You can use any standard loop statement that we saw during in module 2. The most common way to iterate over an array is to use a for loop from 0 to length-1.

```
var persons = [
   {name:'Michel', age:51},
   {name:'Henri', age:20},
   {name:'Francois', age:29},
   {name:'John', age:69}
 ];
 for(var i = 0; i < persons.length; i+=2) {
   var p = persons[i]; // current element
   document.body.innerHTML += p.name + "<br>";
}
```

Arrays (part 2): iterators, Projects: build an album

Project:

- Run **example.lll.3.2.00.forEach.js** inside your dev.tool. Inside that script we will start with the array variable **myPicturesArray**. That is a an array of objects that contain stored links to images on the web. Actually, myPicturesArray is an array of pictures, each picture having a URL, a URL for a tiny version of the picture, called a thumbnail, a title, and the name of the album it belongs to, that can be used as a picture description (HTML alt attribute) but also for displaying it next to the picture.
- The last lines has a small example that iterates on the pictures and create <img> elements on the fly.
- **What you will have to do: ?**

Arrays (part 2): iterators, Projects: build an album

Project:

1. **Improve the display, by adding margins (CSS), shadows, border, and changing the URLs for real pictures**. Remember that you need to have a smaller versions of the pictures, i.e., thumbnails. You can use existing images (images.google.com is your friend) or images you upload somewhere.
2. **Use JavaScript for adding a click event listener on each image**, then when clicked, you will show a bigger version of each picture. For the moment, just change the value of the src attribute of the clicked image (set it to the URL of the full size image from the array).
3. **Try to make something nicer**: reserve a on the right of the document so to display the clicked image with a bigger size. In that case, you will need to create an image (only once, after the first click), to set it to the size of the div (use the width and height attributes of the img element), and to append it to the div.
4. **Try to add more images, and find a way to display them per album**. Create buttons entitled "album1", "album2", etc., and when clicked, you will only display images from the selected album.
5. **Add an option for deleting a picture**. It should be removed from the document and from the array too...
6. **Feel free to add any interesting feature you think about** ;-)

HTML5 multimedia and JavaScript API, video stream intro

The **<video>** element of HTML5 is one of the two "Flash killers" (the other being the <canvas> element). It was designed to replace horrible things like embedded Flash objects that we used to encounter not so long ago. Check **example.III.3.3.00.vidoTag**, The code source of this example is:

```
<video width="750" height="670" controls="controls">
      <source src="videos/testVideo.mp4">
      Your browser does not support the <video> element.
</video>
```

Please:

- **Modify the attributes values**
- **Add the attributes mentioned in the comments**
- **Omit and add attributes to sense the role of each one**

HTML5 multimedia and JavaScript API, video attributes

You may notice that:

Attribute	Value	Description
autoplay	true/false	Specifies that the video will start playing as soon as it is ready
controls	controls	Specifies that video controls should be displayed (such as a play/pause/progress/volume widgets etc).
height	pixels	Sets the height of the video player
width	pixels	Sets the width of the video player
loop	true/false	Specifies that the video will start over again, every time it is finished
muted	true/false	Specifies that the audio output of the video should be muted
poster	URL	Specifies an image to be shown while the video is downloading, or until the user hits the play button
src	URL	Specifies the URL of the video file
Preload	auto metadata none	Specifies if and how the author thinks the video should be loaded when the page loads

HTML5 multimedia and JavaScript API, video tag manipulation

You may notice as well that:

- Usually the browser will use the first format it recognizes (in this case, the browser checks whether mp4 is supported, and if not, it will check for the ogg format, and so on). Some browsers may use a different heuristic and choose a "preferred" format.
- The <video> element is a DOM member, so CSS styling can be applied, as well as manipulation using the DOM API.

HTML5 multimedia and JavaScript API, embedding external video inside you page

RESTRICTION: You cannot embed a YouTube or a Daily Motion video using the **<video>** element

Help! <video src="my youtube video URL"></video> **does not work!**

Instead you need another tag <iframe>, here is an example **example.III.3.3.01.iframe**:

```
<iframe width="560" height="315" src="https://www.youtube.com/embed/ZH1XOsv8Oyo" frameborder="0" allowfullscreen></iframe>
```

HTML5 multimedia and JavaScript API, audio intro

HTML5 audio is composed of several layers:

- The **<audio> element** is useful for embedding an audio player into a Web page. It is dedicated for streamed audio. It is very similar to the <video> element, both in its use and in its API.
- The "**Web Audio API**" is designed for musical applications and for adding sound effects to games. This pure JavaScript API supports manipulation of sound samples (loops, etc.), music synthesis and sound generation (oscillators, etc.). It also comes with a set of predefined sound processing modules (reverb, delay, etc.).

This course will focus on the **<audio> element**,The attributes, event set and JavaScript API of the <audio> element are just a "reduced" version of the ones from the <video> element, and here we will only address their differences and peculiarities. Check **example.III.3.3.02.audioTag** a simple basic example.

HTML5 multimedia and JavaScript API, audio intro

As you can see, the code is very similar to the basic <video> element usage:

```
<audio  controls='controls' >
<source src='sound/neighOfHorse.mp3'>
    Your browser does not support the audio element.
</audio>
```

HTML5 multimedia and JavaScript API, Audio and video player JavaScript API

Control <audio> and <video> elements from JavaScript

The <video> element has **methods**, **properties/attributes** and **events** that can be manipulated with JavaScript. Using the DOM API it's possible to manipulate an audio or video element as a **JavaScript object** that has:

- **Methods** for controlling its behavior, such as play(), pause(), etc.;
- **Properties** (duration, current position, etc.), either in read/write mode (such as volume), or in read-only mode (such as encoding, duration, etc.);
- **Events** generated during the life cycle of the element that can be processed using JavaScript callbacks. It is also possible to send events to control the video player.

Like any HTML element, the <video> element can be manipulated/created using the DOM JavaScript API. Here is an example of programmatically creating a <video> element:

HTML5 multimedia and JavaScript API, Audio and video player JavaScript API

```
var video = document.createElement('video');
video.src = 'video.mp4';
video.controls = true;
document.body.appendChild(video);
```

This will create a complete video player for the file "video.mp4", with control buttons, and will add it to the <body> element of the page.

JavaScript API of the <audio> and <video> elements: Methods, properties and events

The JavaScript API gives you powerful tools to manipulate the <video> element, as the video object provides many properties, methods and events.

HTML5 multimedia and JavaScript API, Audio and video player JavaScript API

Here is link for the best practice to embedded content W3C specifiication, and another link for a video API tutorial.

The complete list of properties can be found at the W3C HTML5 Video Events and API page. This page is interesting for Web developers because it shows an interactive view of the different values and events changing over time while the video is playing within the page. Click the picture to see it running online, then play with the different buttons and look at the table of events and properties that will change in real time. The displayed names show the properties, events, and methods from the API.

HTML5 multimedia and JavaScript API, Audio and video player JavaScript API

HTML5 Video Events and API

This page demonstrates the new HTML5 video element, its media API, and the media events. Play, pause, and seek in the entire video, change the volume, mute, change the playback rate (including going into negative values). See the effect on the video and on the underlying events and properties.

load() play() pause()

currentTime+=10 currentTime-=10 currentTime=50

playbackRate++ playbackRate-- playbackRate+=0.1 playbackRate-=0.1

volume+=0.1 volume-=0.1 muted=true muted=false

Sintel teaser Bunny trailer Bunny movie Test movie

Media Events

loadstart	1	progress	28	suspend	3	abort	0	error	0		
emptied	0	stalled	0	loadedmetadata	1	loadeddata	1	canplay	7		
canplaythrough	7	playing	8	waiting	7	seeking	11	seeked	5		
ended	0	durationchange	1	timeupdate	268	play	8	pause	8		
ratechange	0	resize	1	volumechange	0						

Media Properties

error		src		srcObject	
currentSrc	https://media.w3.org/2010/05/sintel/trailer.mp4	crossOrigin		networkState	1
preload	none	buffered	[object TimeRanges]	readyState	4
seeking	false	currentTime	6.07752	duration	52.208333
paused	true	defaultPlaybackRate	1	playbackRate	1
played	[object TimeRanges]	seekable	[object TimeRanges]	ended	false
autoplay	false	loop	false	controls	true
volume	1	muted	false	defaultMuted	false
audioTracks		videoTracks		textTracks	[object TextTrackList]
width	0	height	0	videoWidth	854
videoHeight	480	poster	https://media.w3.org/2010/05/sintel/poster.png		

canPlayType

video/mp4	video/webm
"maybe"	"maybe"

Tracks

Audio	Video	Text
?	?	0

HTML5 multimedia and JavaScript API,

Audio and video player JavaScript API

Here is a table that shows the most interesting methods, properties and events provided by the <video> element API. We provide this as a quick reminder - keep in mind that the complete list is much longer!

Methods	Properties	Events
play()	currentSrc	play
pause()	currentTime	timeupdate
load()	videoWidth	pause
canPlayType()	videoHeight	progress
	More on read/write: error, muted, seeking, volume, height, width,	More events: error, ended, abort, empty, emptied, waiting, loadedmetadata
	Readonly: startTime, duration, ended, paused, seekable, played	

HTML5 multimedia and JavaScript API, Examples using the JavaScript API

Now let's take a look at a set of examples demonstrating how to use the most important of these properties, methods, and events…

Example 1, check **example.III.3.3.03.externalButtons** - **how to use external buttons to control a player's behavior**

This example shows the first steps towards writing a custom video player. It shows basic usage of the JavaScript API for adding custom buttons to play/pause the video or to go back to the beginning by setting the **currentTime** property to zero

HTML5 multimedia and JavaScript API, Examples using the JavaScript API

```
<div id='controls' style="display:block">
      <button onclick="playVideo();"  style='cursor: pointer;'>Play</button>
      <button onclick="pauseVideo();">Pause</button>
      <button onclick="stopVideo();">Stop</button>
      <input type="range" onchange="setVolume(this.value);" min= 0 max =1 step =.1 value=.1></button>
      </div>
```

- we add a click listener to each button, in order to call a JavaScript function when the button is clicked.

```
window.onload = init;
function init(){vid = document.querySelector('#myPlayer');}
function playVideo(){vid.play(); }
function pauseVideo(){vid.pause(); }
function stopVideo(){vid.currentTime = 0; }
function setVolume(value){vid.volume = value; }
```

- using the DOM API we get the JavaScript object that corresponds to the video element we inserted in the HTML document. This line will be executed when the page loads.
- we call methods from the API for playing/pausing the video.
- we modify the currentTime property in order to rewind the video. Note that vid.load() also rewinds the video, shows the poster image again, but also pauses the video. By using currentTime=0 the playback does not stop.

HTML5 multimedia and JavaScript API, Examples using the JavaScript API

Example 2 **-how to detect the end of a video and start another one**.

This example listens for the ended event, and calls a callback function when the video has finished.

```
function init(event){
    vid = document.querySelector('#myPlayer');
    vid.addEventListener('ended', playNextVideo,false);
}

function playNextVideo(event){
   // Whatever you want to do after the event (play another video,
    // for example), change the src attribute, of the video element, etc.
}
```

HTML5 multimedia and JavaScript API, Examples using the JavaScript API

Example 3, check **example.III.3.3.04.playList** **-application of the above technique - how to manage playlists**.

Detail explanation:

```
sources = ['videos/video.webm','videos/testVideo.mp4','videos/Sintel.mp4'];
```

The JavaScript array that contains the URLs of the videos in the playlist. In this example, we've only got three of them, but if the array is larger the example will still work.

```
window.onload = init;
```

When the page is loaded, an init() function is called.

HTML5 multimedia and JavaScript API, Examples using the JavaScript API

```
vid = document.querySelector('#myPlayer');
vid.addEventListener('ended',
loadAndPlayNextVideo,false);
loadNextVideo();
```

We used the DOM to get the JavaScript object corresponding to the video element, then defined a listener for the ended event.

Each time a video ends, the loadAndPlayNextVideo() callback will be called.

As the video element has no src attribute by default, we also preload the first video (call to loadNextVideo()).

```
function loadNextVideo(event){
   vid.src = sources[currentVideoIndex
%(sources.length)];
   //vid.load();
   currentVideoIndex++;
}
```

The loadNextVideo() function uses a variable called currentVideo that corresponds to the index of the current video. By setting myVideo.src = sources [currentVideoIndex % sources.length], we set the src of the video element to sources[0], then to sources[1], and as we increment the currentVideo index each time. If it becomes greater than 2, the modulo (the "%" symbol is the modulo in JavaScript) will make it "loop" between 0 and the number of videos in the playlist. In other words, when the last video ends, it starts back at the first one.

HTML5 multimedia and JavaScript API, Using the Webcam

It's very easy to use the **getUserMedia** API for accessing the WebCam.

Here is a version that should work on any recent browser except Apple Safari (which still does not support this API). Note that for security reasons you must host your HTML/CSS/JS page on an HTTPS server for getUserMedia to work. Check **`example.III.3.3.05.webCam`**, which has two versions.

First version that uses callbacks (success/error, see the JS code):

```
navigator.mediaDevices.getUserMedia ({video: true}, success, error);
// successCallback
function success(localMediaStream) {
var video = document.querySelector('video');
video.srcObject = localMediaStream;
}
// errorCallback
function error(err) { console.log("The following error occured: " + err); }
```

HTML5 multimedia and JavaScript API, Using the Webcam and microphone

Second version that uses a new JavaScript syntax called "promises": This is another way of saying, "Please, browser, try to give me access to the WebCam, **THEN when the Webcam is ready**, please tell me so that I can display its stream in a <video> element".

```
navigator.mediaDevices.getUserMedia({video: true, audio: true})
.then(function (stream) {
      var video = document.querySelector('#video');
      video.srcObject = stream;
  })
.catch(function(err) {
    alert("something went wrong: " + err)
});
```

HTML5 multimedia and JavaScript API, Extended examples: CSS3 Filters

In this section, we provide some extended examples that use more JavaScript and more complex CSS manipulation.

Check **example.III.3.3.06.CssFilters**: **applying CSS3 filters to a video in real time**. Play the video and then move your mouse over the video while it's playing. This will change in real-time the CSS class of the video element. Each class uses the filter property with different values.

Note that CSS filters are not yet 100% supported by the major browsers.

HTML5 multimedia and JavaScript API, Extended examples: percentage of Buffering

check **example.III.3.3.07.percentageBuffering**: **how to display a percentage of buffering when using a slow connection.** Note that on mobile phones, the video does not start until the user presses the play control or clicks on the video picture. Using the "**canplaythrough**" event is a trick to call a function that starts the video player as soon as the page is loaded on desktop. This event is not supported by mobile devices, so if you try this example on a mobile, the video will not start automatically.

```
function processProgress(event){
   var endBuf = myplayer.buffered.end(0);
   var soFar  = parseInt((endBuf / myplayer.duration) * 100);
   document.querySelector('#loadStatus').innerHTML = soFar + "%";
  }
```

"The buffered property is a **TimeRanges object**: an array of start and stop times, not a single value. Consider what happens if the person watching the media uses the time scrubber to jump forward to a point in the movie that hasn't loaded yet--the movie stops loading and jumps forward to the new point in time, then starts buffering again from there. So the buffered property can contain an array of discontinuous ranges. The example simply seeks the end of the array and reads the last value, so it

HTML5 multimedia and JavaScript API, Projects: Sophisticated video player

Create a **quiz based on videos** - here is a proposed story telling:

- a video is playing, then it stops at a given time, and you display a question such as: "who is this actor?" followed by some radio buttons + a proposal (see what we do with quizzes in this course): "Leonardo Di Caprio" or "Harisson Ford"?
- Once the question is answered, you display "Correct" or "Incorrect"
- Then the video continues....
- When the video ends, please show the final score.

Displaying a map with the geolocation API, Introduction to the geolocation API

Here I presents the new Geolocation API and illustrate its use with several examples. The Geolocation HTML5 JavaScript API is implemented by most modern Web browsers, and uses different means to get the current location: GPS, GSM/3G triangulation, Wifi, IP address, etc.

It is possible to prompt the user to activate the GPS (this is what most GPS navigation software does on mobile phones), or ask for a particular mean among those available. It is also possible to track the current position when it changes. This is useful for writing a navigation application or for tracking in real time the position of different participants in the case of an application that involves several persons at the same time (using WebSockets, for example).

Typical example, open your browser dev tool, then type:

```
navigator.geolocation.getCurrentPosition(function(pos){

    document.body.innerHTML = "latitude is: " + pos.coords.latitude;
    document.body.innerHTML += "<br>longitude is: " + pos.coords.longitude;
    });
```

Displaying a map with the geolocation API, Practical examples using the Google Map API

Three practical examples: use the geolocation API together with Google Maps.

This section presents some examples of how to get a static map (**a picture**), using the Google Static Map API, how to display an **interactive map** using the Google Map JavaScript API and even how to get an estimation of a **physical address** from the longitude and latitude, using the Google Reverse Geocoding JavaScript API.

Displaying a map with the geolocation API, Practical examples using the Google Map API

WARNING : since 2018, Google requires an API key in order to use interactive maps in HTML pages. In most situations, using such a key is free of charge.

Please follow this YouTube tutorial on how to get a Google Map API key. In the examples of this JavaScript Introduction course, we use my own key that is restricted to jsbin.com. It means that you can clone my examples or write your own ones in jsbin.com and use my key to make them work. However, for applications hosted somewhere else, you will need a personal API key.

You can find equivalent examples using the free, open source, Open Street Map and Leaflet APIs.

Here are a few links that I have selected for you:

- Basic Map with leaflet and OSM : https://jsfiddle.net/user2314737/tvwkgo34/
- Get your location using the geolocation API and display it on an OSM: https://codepen.io/leemark/pen/vcbuf
- Another example that shows a map centered on your location: https://codepen.io/dangvanthanh/pen/tlpLG
- Google map (using iframe, no key is necessary for that) v.s. OSM/Leaflet : https://codepen.io/chris0stein/pen/qFKhg
- Reverse geocoding with OSM: https://services.gisgraphy.com/static/leaflet/index.html

Displaying a map with the geolocation API, 1-how to get a static image map centered on your longitude and latitude

First example, check **example.III.3.4.00.GeoImage**, that will generate an image by connecting to Google static map API.

```
function showPosition(position)
{
var latlon=position.coords.latitude+","+position.coords.longitude;

Var
img_url="https://maps.googleapis.com/maps/api/staticmap?key=AIzaSyAT5hmMkYSdpLVViDVv4p3Ybe5A1_Dwe68&cen
ter=" +latlon+"&zoom=14&size=500x500&sensor=false";

document.getElementById("mapholder").innerHTML="<img src='"+img_url+"' />";
}
```

In case, it doesn't work try it on this link.

Displaying a map with the geolocation API, 2-shows how to display an interactive Google map centered on the current position

Second example, check **example.III.3.4.01.GeoInteractive**, that will generate an interactive Google map centered on the current position. This example will work only on a jsbin.

```
// Init map object
var map = new google.maps.Map(document.getElementById("map"), optionsGmaps);
navigator.geolocation.getCurrentPosition(drawPosition);
```

Inside the success callback function(drawPosition)

```
 // Make new object LatLng for Google Maps
var latlng = new google.maps.LatLng(position.coords.latitude, position.coords.longitude);
// Add a marker at position
var marker = new google.maps.Marker({position: latlng, map: map,title:"You are here"});
// center map on longitude and latitude
map.panTo(latlng);
```

Displaying a map with the geolocation API, 3-how

to get a phythat obtains an address from longitude and latitudesical address from the longitude and latitude

Third example, check **example.III.3.4.02.GeoPysicalAddress**, that obtains an address from longitude and latitude. It uses the Google Reverse Geocoding JavaScript API. For those of you who are really interested to know how this API works, please read the Google documentation and tutorials. Without going into detail, the below example might be useful to copy/paste/adapt for trying to pre-fill a form where one is asked for an address. Geolocation is useful for guessing the country, city, zip code, street, etc.

```
geocoder = new google.maps.Geocoder();
geocoder.geocode({'latLng': latlng},reverseGeocoderSuccess);
function reverseGeocoderSuccess(results, status) {
 if (status == google.maps.GeocoderStatus.OK) {
    if (results[1]) {
       infowindow.setContent(results[1].formatted_address);
       infowindow.open(map, marker);
       // Display address as text in the page
       myAddress.innerHTML="Adress: " + results[0].formatted_address;
    }}} // end of reverseGeocoderSuccess
```

Displaying a map with the geolocation API, Projects

I am proposing a couple of projects:

- Compare the accuracy of Google Map, OpenStreetmap, and Baidu Map.
- Find other applications of geolocation, and implement the practical one
- Find a way to extract data from Google Map and Baidu Map and inject it into OpenStreetmap, or your own geolocation map.
- Find out the service that Google Map, OpenStreetmap and Baid Map APIs offer.

Playing sound samples and music, Background music (streamed)

WARNING ABOUT THE AUTOPLAY POLICY

Since 2018, most browsers have adopted the Autoplay Policy that prevents any Web page to start making music or playing sounds without a user interaction. For a user, it means that most examples from this course won't make sounds until you interact with the application (i.e. clicking on the canvas for the game example). For a developer, if you use libraries such as **Howler.js**, there are good chances that you won't have to change your code. If you are programming with the **WebAudio API**, then you'll need to resume the **AudioContext** after the first user interaction.

Background music (streamed); In a previous section we saw how we can add music to our Web page, using the **<audio></audio>** element. We can even hide its GUI and control the play/pause of the music from JavaScript. Streaming music is perfect for providing a background atmosphere in a video game.

Check **example.III.3.5.00.backgroundSound**, that is one simple example of background music controlled from JavaScript; we just remove control attribute then add the desired buttons and control them by JavaScript.

Playing sound samples and music, Sound effects using howler.js

Howler.js for using sound samples in memory

If you want to play short sounds that can occur very rapidly, streamed sound/music is not a good solution. This is where the WebAudio API, made by W3C and implemented by your browser, comes in handy. This API allows you to download and decode sound samples in memory, and play them on demand, using nearly zero CPU and with no delay when you play the sound (no buffering etc.). There are several JavaScript libraries that simplify the use of the WebAudio API. HowlerJS is one of these.

Check **example.III.3.5.01.howler**, that uses Howler.js to load a sound sample from a remote server/local hard disk, then decode it in memory, and play it.

Playing sound samples and music, Sound effects using howler.js

HTML code:

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/howler/1.1.28/howler.min.js"></script>
```

indicate that in our example we are using an external library.

```
<button onclick="playSound();" id="button1" disabled>Play sound sample 1</button>
```

declares a button, that is greyed by default and cannot be clicked. This is done by the **disabled=true attribute**.

JavaScript code:

```
sound = new Howl({
    urls: ['sound/plop.mp3'],
    onload: function () {
        console.log("Loaded asset ");
      button.disabled = false; // enable the play sound button  }});
```

Playing sound samples and music, Sound effects using howler.js

The Howler library is to be used like this: **sound = new Howl({...});** The part between the { and } is an object. The url's property is an **array** with at least one element: the URL of the sound we want to use, located on remote servers or even on local disk. The call to new Howl({...}); will start downloading the sound in background, then, once it has loaded, it will "decode it" (i.e., an mp3 file will use some cpu to be decoded on the fly and played, whereas a decoded sound will use nearly zero cpu, which makes it good for games!).

Finally, once the sound is decoded, the **onload callback** is executed. In other words, the function after onload: will be executed. In this callback, **we enable the button because the sound is ready to be played**.

The playSound function can only be called when the button is enabled (when the sound sample has been loaded and decoded). In order to play a sound loaded by Howler.JS, we just call the play() method.

Playing sound samples and music, Sound effects using howler.js

Assignment-3-3:

Use the old fashion with the <audio> tag to produce the same result as the last example **example.III.3.5.01.howler,** then try to find out if you could sense the difference between them, specially if you click many time rapidly on the button(the decoding part).

Dealine: 2019-December-15

Playing sound samples and music, Adding music and sound effects

Here is the last version in this course of the game from Module 2 with music and sound effects (when the player eats a ball), check **example.III.3.5.02.soundEffectsOnTheGame**.

Look at the HTML part: we added two <audio> players (invisible; we removed the controls attribute) for background music. In the JavaScript code, we start the background music as soon as the page is loaded.

Assignment-3-4:

Now you can see clearly that when you have many balls and while you are deleting them at once the decoding is very slow, which sounds like you are just deleting one ball instead of many balls.

You task is to implement the hawler player for the collision sound, then see the difference.

Dealine: 2019-December-15

Playing sound samples and music, [Advanced] a multiple image, sound and music loader

A utility background loader for images, music and sound samples

In video games, you very often need to load assets before starting the game:

- Images must be loaded (background image, game logo, sprite sheets, etc.)
- Sound samples must be loaded and decoded (the previous example used only one single sound sample, but with multiple samples it becomes more difficult to know when they are all ready to be used, as they come asynchronously over the network),
- For streamed music, you need an <audio> player element. If you use different pieces of music, you may use multiple audio elements, and you pause one and start another when you change the music. Alternatively you may use a single audio element, and change its src attribute.

So, we wrote a multiple "asset loader" to make all these tasks easy.

Playing sound samples and music, [Advanced] a multiple image, sound and music loader

Here is a small example **example.lll.3.5.03.multipleAssets.js,** that you may use if you like, which takes an array of "assets to be loaded", that can be either an image, a sound sample or streamed background music. You call the loadAssets(callback) function, passing as a parameter a single callback function of yours. When all assets are loaded, your callback will be executed, and will get a single parameter: the assets ready to be used!

Example (to hear the music and sound sample, there are two lines to uncomment in the startGame(...) function).

This example can explain in more advanced JavaScript courses.

Playing sound samples and music, projects

Suggested topics

- HowlerJS is an easy way to manipulate "the real API" that is named WebAudio. If you are curious, look at the webaudiodemos.appspot.com Web site, look on twitter with the hashtag #webaudio, or on YouTube. This API is really powerful!
- Do you know other libraries similar to HowlerJS, useful for manipulating audio (streamed or as sound samples)?
- And, what about synthesizing music instead of using files? I recommend this MOOC for the curious ones (after this MOOC, then you'll know JavaScript and it will be easier!).
- You can also use another funny library for synthesizing 8 bits sound effects, try this demo!

Playing sound samples and music, projects

Optional projects:

- Try to make nice audio player that will chain background musics, when one is finished the next one starts (use an "ended" event listener on the audio element, for example, add onended="....")
- Add some buttons/menu to the game so that we can choose between 2 or 3 different background musics, or turn the music off.
- Add a slider for adjusting the volume of the background music
- Use the multiple image/music/sound loader for adding multiple sound effects to your game, make different sounds depending on the color of balls that collide with the player
- try to think about a way to display a progress bar while the multiple image/sound/music loader is loading the files...