HTML5.1x Learn HTML5 from W3C

course notes

A Student

July 21, 2015

Contents

Part I

HTML

Chapter 1

What we're going to learn

- New and simplified HTML5 tags and microdata
- Play with the audio and video tags
- Draw and animate fun Web graphics
- Discover the newest HTML5 forms
- Understand why accessibility is important
- Test the basic APIs, such as Web storage and geolocation
- Practice coding techniques thanks to multiple interactive examples

Chapter 2

Syllabus

2.1 Live on monday 1 june 2015 - 13:30 UTC - Course introduction and practical information

- Course syllabus
- Getting around the course
- Grading and due dates
- Course tools
- Why accessibility is important
- Why internationalisation is important
- Welcome survey

2.1.1 Week 1: HTML basics

- 1.1 Week introduction - Week 1
- 1.2 From HTML1.0 to HTML5
- 1.3 New structural elements
- 1.4 Other elements and attributes
- 1.5 Microdata
- 1.6 Exercises - Week 1

2.2 Live on tuesday 9 june 2015 - 15:00 UTC

2.2.1 Week 2: HTML5 Multimedia

- 2.1 Video introduction - Week 2
- 2.2 Streaming multimedia content: the video and audio elements
- 2.3 Subtitles and closed captions
- 2.4 Enhanced HTML5 media players and frameworks
- 2.5 Webcam, microphone: the getUserMedia API
- 2.6 Exercises - Week 2

2.3 Live on tuesday 16 june 2015 - 15:00 UTC

2.3.1 Week 3: HTML5 Graphics

- 3.1 Video introduction - Week 3
- 3.2 Basics of HTML5 canvas
- 3.3 Immediate drawing mode: rectangles, text, images
- 3.4 Path drawing mode: lines, circles, arcs, curves and other path drawing methods
- 3.5 Colors, gradients, patterns, shadows, etc.
- 3.6 Exercises - Week 3

2.4 Live on tuesday 23 june 2015 - 15:00 UTC

2.4.1 Week 4: HTML5 Animations

- 4.1 Video introduction - Week 4
- 4.2 Basic animation techniques
- 4.3 Canvas and user interaction (keyboard, mouse)
- 4.4 A glimpse of advances canvas functionalities
- 4.5 Exercises - Week 4
- 4.6 Optional exercises: draw an animate monster!

2.5 Live on tuesday 30 june 2015 - 15:00 UTC

2.5.1 Week 5: HTML5 Forms

- 5.1 Video introduction - Week 5
- 5.2 Introduction to HTML5 Forms
- 5.3 Accessible forms
- 5.4 New <input> types
- 5.5 New forms attributes
- 5.6 New elements related to forms (output, datalist, etc.)
- 5.7 Form validation API
- 5.8 Examples of powerful HTML5 forms (use of CSS3, webcam, persistence)
- 5.9 Exercises - Week 5

2.6 Live on tuesday 7 july 2015 - 15:00 UTC

2.6.1 Week 6: HTML5 Basic APIs

- 6.1 Video introduction - Week 6
- 6.2 HTML5 APIs introduction
- 6.3 HTML5 Cache
- 6.4 The Web Storage API
- 6.5 The File API: reading files (metadata, content, previewing)
- 6.6 Geolocation API
- 6.7 Final exam: more exercises

Chapter 3

Lessons 1 - HTML Basics

3.1 Why accessibility is important

Always satisfy these aspects:

- Page title must be properly set, coincise, efficient. The page title is the first item that is played out by a browser in "help mode" for people with accessibility differences.
- Alt text must alwaye be present within <img> tags
- For other easy checks see the WAI Easy Checks page

Internationalization is the design and development of a product, application or document content that enables easy localization for target audiences that vary in culture, region or language. *Localization* refers to the adaptation of it to meet the language, cultural and other requirements of a specific target market.

3.2 Character encoding

```
<!DOCTYPE html>
<html lan="en">
<head>
<meta charset="UTF-8"/>
```

Microdata are pieces of data that will be looked at by search engines.

3.3 A minimal HTML5 document

```
<!DOCTYPE html>
```

```
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>Page Title</title>
        <link rel="stylesheet" href="style.css">
        <script src="script.js"></script>
    </head>
    <body>
        ... <!-- The rest is content -->
    </body>
</html>
```

3.4 Improvements towards simplicity

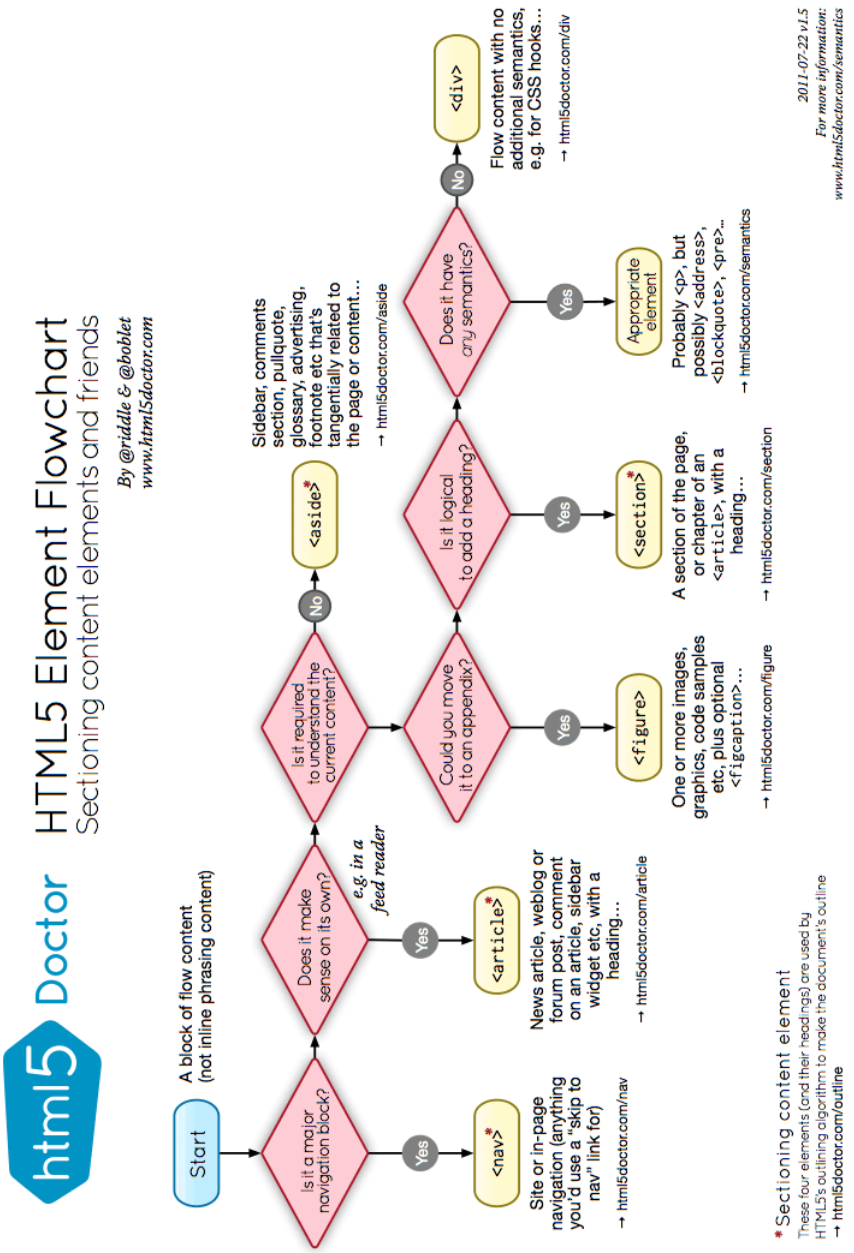
- No more closing tags required (unless for the ones that divide a particular area of the document).

- No more quotes for the values of attributes!

3.5 Basic structure

3.6 New tags

Take a look at *section* and *article*. They can each contain the other, to better present the content.



3.7 Best practices to use <section>, <article>, <nav>, <aside>

These are called *sectioning elements*, since they cut documents in sections. Headings start implied sections.

- Always add a heading in each sectioning element (section, article, nav, aside), but also after the body element (called a "sectioning root"). This is for accessibility reasons.

  In this case:

```
<section>
<header>
    <h1>Summer</h1>
    <p>Posted by Pvag</p>
</header>
...
</section>
```

  a screen reader will say *"Entering section with heading Summer"*

  In this other case:

```
<section>
<header>
    <p class="article title">Summer</p>
    <p>Posted by Pvag</p>
</header>
...
</section>
```

  the screen reader will just say *"Entering section"* AND THIS IS NOT GOOD!

  Note: body is a sectioning element too, so it is good practice to put a hx (x = 1:6) for it too! Actually, it's the root sectioning element. An alternative to an hx is a header element.

Recap: always put a hx (or header) after each sectioning element (body too).

Example:

```
<body>
    <h1>Example Blog</h1>
```

```
        <section>
            <header>
                <h2>Blog post of april 2015</h2>
                <p>Posted by Fruttariello</p>
            </header>
            <p>Content of the blog post</p>
        </section>
    </body>
```

Best practice example for many hxs: use sections!

```
<body>
    <h1>Apples</h1>
    <p>Apples are fruit.</p>
    <section>
        <h2>Taste</h2>
        <p>They taste lovely.</p>
        <section>
            <h3>Sweet</h3>
            <p>Red apples are sweeter than green ones.</p>
        </section>
    </section>
    <section>
        <h2>Color</h2>
        <p>Apples come in various colors.</p>
    </section>
</body>
```

Note that the x of the different hxs must be different in an internal hierarchy, in the sense that the automatic numbering for hxs inside a section, inside another section is not guaranteed by all browsers! So state it explicitly! Note: there are 2 h2 tags: this is permitted, since they are not related (i.e. they are not one within the other!).

Best practice: visualizing the table of content is good to spot missing hxs (headings) after sectioning content, debugging the structure of the page.

3.8 <main>

If you use <nav>, <header>, <footer> to structure the document, you can also use <main> to identify the main content of the document. Doing so provides a navigable document structure for assistive technology users as well as styling hooks for devs. <main> is subjected to some constraints:

- No more than one main are allowed in a document
- It must not be a **descendent** of an <article>, <aside>, <footer>, <header>, <nav> element.

Note: if you do not want to make the heading visible in the page use a **CSS offset**, instead of hiding it! This will be handy when properly adding headers to navs, for example and not wanting to show their text. If you don't want to display the heading content after a <nav> or an <aside> element.

3.9 <details> and <summary>

To display a *foldable* zone in an HTML document. The summary is the text beside the arrow icon; the details is the text displayed when we click the arrow and unfold the element.

Chapter 4

Lesson 2 - Multimedia

4.1 Streaming multimedia content: the video and audio elements

4.1.1 <video>

See someHTML/Video/video.html for and example. The **controls** attribute renders the play/stop/ volume/progress widgets; the **autoplay** realizes the auto play. The browser will use the first video format that it is able to play or it will choose it based upon its preference over the different formats. The <video> is an member of the DOM, so CSS styling and transformations can be applied, as well as manipulation using the DOM API. NOTE: youtube, Vimeo, DailyMotion and other websites offering videos and commercials on them, are not embeddable in video elements; use and iframe instead. The iframe will embed and rendere the HTML5 video, together with the advertising that come along with it. See ~/Coding/HTML5_edX/someHTML/Video/videoIframe.html .

Beware the *Codecs*: each browser has its favourite coded: as of 2015, the most supported is **H264/mp4**. Encode your videos in the most supported formats!

attributes

- controls : boolean value - true is present
- preload="none" (suggested value for mobile devices) It's a hint, so the browser may ignore it. If autoplay is present, preload is ignored.
  - none
  - metadata : length, format
  - auto : the browser will decide - depends on the wi-fi connection and implementation

- poster="previewImgURL" : watch out: IE puts the first frame as preview image - to change this i need JS
- loop
- autoplay : no good for mobile devices and for pages with $>$ 1 videos

Recap: for mobile devices set preload="none" and no autoplay (i.e. don't write autoplay, since it's a boolean attribute). Controls, loop and autoplay are **boolean attributes**.

4.1.2 <audio>

HTML5 is composed of several audio layers:

- the <audio> element: it's similar to the <video> element; it's useful to embed an audio player in a web page.
- the WebAudio API (JavaScript): it is designed for musical application and to add effects to games. It's a pure JavaScript API that supports manipulation of sound sample, music synthesis and sound generation. It also has a set of predefined sound processing modules (reverb, delay, etc.).

See an example at

/Coding/HTML5_edX/someHTML/Audio/audio.html

4.1.3 Styling <video> and <audio> with CSS3

Transitions, Animations and Transforms can be added, since video and audio are full-fledged html elements. See audio_transitions.html for transform and wsv.html for a cool example of window-wide video, together with some easy JS.

4.1.4 Controlling audio and video with JS

A simple example of video creation in JavaScript:

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

See this web page for a live example of the different methods and properties.

4.2 Subtitles and Closed Captions

4.2.1 The <track> element

There's a difference between subtitles (only words) and captions (description of sounds too). A web server to load the caption (.vtt) file is needed (http:// , file:// won't work!). Furthermore, the MIME type must be specified to text/vtt and charset to utf-8. Here's an example of code to add to an Apache web server:

```
<Files file_with_captions.vtt>
   ForceType text/vtt; charset=utf-8
   </Files>
```

Here's an example to add a track file (file with subtitles):

```
<track src="http://demo.jwplayer.com/html5-report/sintel-
    captions.vtt" kind="captions" label="Closed Captions"
    default>
```

Important attributes

- `crossorigin="anonymous"`

  (in the video element: some servers require it)

- `kind="captions|subtitles|descriptions|chapters|metadata"`

- `default`

  when i want this track to be displayed by default when playing the video, since multiple tracks can be included in a video element (see multipleTracks.html)

- `srclang="en|fr|de"`

  or other language of the subtitles|chapters|descriptions This must be a BCP 47 Language Tag. This must be present if the kind is set to subtitles.

- `label="italian"`

  This value will be displayed in the GUI control that is included in the HTML5 default video player.

Note: to insert a chapter vtt file, together with a subtitle/caption vtt file, add another track element inside the video element, with the value *chapter* for the attribute *kind*.

The WebVTT format for subtitles|captions|other

A **cue** is an id + a starting time + –> + an ending time + a string. Example of a cue:

```
Opening
00:00:00.000 --> 00:00:22.000
Welcome to our <i>nice film</i>
```

HTML tags to format the text are allowed. The id can be a string or a number; it is used in JS with the

```
getCueById(idName)
```

method of the TextTrack class. To make the subtitles WebVTT file, use a tool like `http://www.amara.org`. CSS attributes can be specified on the same line of the times to position the text. For example,

```
size : 33%
```

will make the size of each line to fill the 33% of the video width, automatically spanning on multiple lines, if needed.

Classes can be used to style the text, using the c HTML element, like this:

```
<c.myclass>Whatever you think you know of this man is irrelevant.</c>
```

The v tag can be used to style different voices:

```
01:00:10.000-->01:00:12.000
<v Tarzan>O-hohoh-ohoh
<v Jane>ok
```

And then for the CSS part (i guess inside the WebVTT too):

```
<style type="text/css">
   ::cue(.myclass) {color : red;}
   ::cue(v[voice="Tarzan"]) {color : blue;}
   ::cue(v[voice="Jane"]) {color : green;}
   ::cue(#Opening) {font-size : 150%;}
   </style>
```

- `::cue`

  is the pseudo-element selector to match cues in the WebVTT file;

- Opening was defined above as id of the overture of the movie;
- The voices are contained inside a kind of a voice array;
- use the . for classes, the # for ids.

Note: converters from other subtitle format (like .srt) are available; on-the-fly converters are too, like enhanced HTML5 video players and JS libraries (like JS_videosub).

The track JS API

Using it i can do lots of cool tricks like:

- sync a tablature to a musical video
- sync Google Maps and/or Google Street View to a video: while the video plays, the JS code listens to the ontimeupdate event; using the currentTime property of the video i know where i am in the video; i read the data (maybe the tag? Or data can be put inside a cue in some other ways?) corresponding to that time: this data must contain the coordinates of the place that is viewable in that frame; i update Google Maps and/or Street View!
- dynamically building a navigation menu that shows the different chapters in the video
- making an app that creates on-the-fly subtitles/captions.

4.3 Enhanced HTML5 media players and frameworks

There are many enhanced video and audio players that can be easily customized through JS APIs. There are JS APIs to create my personal enhanced video and audio players.

4.4 Webcam, microphone: the getUserMedia API

The getUserMedia API is useful to control a webcam video stream. When dealing with video streams, it is always used in conjunction with the <video> element. It is used redirecting the video stream from the webcam to a video element.

Fundamental methods:

- navigator.getUserMedia - it looks like this is a property too
- window.URL.createObjectURL(localMediaStream); - this is not from the getUserMedia API

Working with the microphone

Use the `getUserMedia(audio:true, onSuccess, onError)` method. The W3C WebRTC - Web Real Time Communication - is another W3C specification for audio/video/data real time communication.

The microphone will be used for music web apps, from the WebAudio API.

Video constraints

```
var vgaConstraints = {
   video: {
   mandatory: {
   maxWidth = 640,
   maxHeight = 480
   }
   }
   };
```

and then use it inside the usual method:

```
navigator.getUserMedia(vgaConstraints, onSuccess, onError);
```

Chapter 5

Lesson 3 - HTML Graphics

5.1 Video Introduction

5.2 Basics of HTML5 Canvas

5.2.1 Pixel-based / Vector-based

The basic element is the `<canvas>`. It provides scripts with a resolution-dependent bitmap canvas, which can be used for rendering graphs, game graphics, or other visual images on the fly. The canvas has been designed for pixel-based graphics, while SVG (scalable Vector Graphics) is for vector-based graphics.

5.2.2 Canvas JS API

It supports many kinds of shapes: line, arc, rectangle, ellipse, curve, text, image. Some drawing styles need to be specified: these will affect the way shapes are drawn (color, drawing width, shadows, etc.). An alpha channel for drawing in transparent mode is also supported, as well as many advanced drawing modes and global filters (blur, etc.). The canvas is also use to display video at 60 fps (useful for games), to display videos with special effects, to display a webcam stream, etc. Note: most modern browsers support hardware acceleration, i.e. the browser's implementation of the canvas API has hardware acceleration support. 3D drawing using the WebGL API is also possible (BabylonJS, ThreeJS, others).

NOTE!!: It is very important NOT to access elements of the DOM before the page has been loaded entirely! A good way to do this is putting drawing code inside the `init()` method and specify the attribute and value `onload="init();"` for the `body` element. This ensures that the elements of the body have all been loaded; only then the function is executed. A good practice is getting canvas, context, width, height and other relevant properties inside the init method. A complete reference of canvas' properties and

methods can be found at this website

Fundamental Methods

- `canvasReference.getContext(“2d”);`
- `contextRef.fillRect(0, 0, 80, 100);`
- `strokeRect(x0, y0, width, height)`
- `clearRect(x0, y0, width, height)` : erases the specific rectangle; actually, it draws it in a color called “transparent black”.
- `fillText('theText', x0, y0)` : note: the coordinates refer to the origin of the base line.
- `strokeText('theText', x0, y0)` : it has a last optional parameter (maxwidth).
- `measureText('theText')` : gets all the parameters of theText and the context: the returned object has properties with the same name, like width, height, font, size, shadow, lineWidth, etc.
- `ctx.moveTo(x0, y0)`
- `ctx.lineTo(x1, y1)`
- `ctx.closePath()` : line from last point to first point of the path.
- `ctx.drawLine(x1, y1, x2, y2, color, width)` : this works in immediate drawing mode (useful when i need to draw just a line).
- `ctx.arc(xcenter, ycenter, radius, fromTheta1, toTheta2, drawComplementary)` : the angles are positive in the clockwise rotation; Math.PI/2 is bottom vertical.
- `ctx.arcTo(x1, y1, x2, y2, radius)` : the first tangent passes through the actual point and (x1, y1); the second tangent passes through (x1, y1) and (x2, y2); the radius is the last constraint to define the arc.
- `ctx.quadraticCurveTo(controlX, controlY, endX, endY)` : it works with 3 points: the context point (i.e. the actual point); the control point; the ending point.
- `ctx.bezierCurveTo(control1X, control1Y, control2X, control2Y, endX, endY)` : it works with 4 points: the context point (i.e. the actual point); the first control point; the second control point; the ending point. 2 tangents are considered: the first from the context point to the first control point: the second from the second control point to the

ending point. The midpoint (M1, M2) of each tangent is considered; the midpoint (M3) of the line that connects the 2 control points is considered; the midpoints (M4, M5) of the lines connecting the context point to M3 and M3 to the ending point are used as tangent points for the curve to the lines. So the curve passes through the context point, M4, M5 and the ending point.

- `ctx.textBaseline("value")` : vertical alignment.
- `ctx.textAlign("value")` : horizontal alignment (center, right, left, start, end).

Fundamental Properties

These affect the resulting canvas object; the value of these properties can be stored and retrieved.

- `fillStyle`: pattern, color or gradient. Any kind of "fill" drawing will use the value of this property. The default value is

  `black`

  . This and other *context properties* can be considered to be "global variables" of the context. NOTE: The color is expressed as a string, e.g. `''red''`.
- `strokeStyle`: defines how the shape's outline should be rendered. Used with

  `strokeRect`

  , for example.
- `lineWidth`
- `font` : e.g. `'italic 20pt Calibri'`

Transformations (context methods)

```
\item \texttt{transform(xOffset, yOffset}
\item \texttt{rotate(radians)} : positive clockwise.
\item \texttt{scale(xFactor, yFactor)}
```

Note: the order is important! Follow the order in which these three methods have been presented in the list above!

Transform ; Rotate ; Scale .

Saving and Restoring the Context

These are context's methods:

- `save()`
- `restore()`

What will be saved: fillStyle, strokeStyle, lineWidth, previous coordinate system, etc. - all properties affecting the drawing.

5.3 Immediate Drawing Mode: rectangles, text and images

The font property is CSS-compliant and accepts values in the form:

`[font style][font weight][font size][font face]`

Accepted values are:

- `font style: normal, italic, oblique, inherit`
- `font weight: normal, bold, bolder, lighter, auto, inherit, 100, 200, 300, 400, 500, 600, 700, 800, 900`
- `font size: a size in pixels or in points, such as 60pt, 20px, 36px`, etc.
- `font face: Arial, Calibri, Times, Courier`, etc. Some font faces may not work in all browsers.

Example of setting: `ctx.font = "italic bold 60pt Courier";`

Drawing an image from an Image object, in a canvas

- Create the Image object
- Load the image (an asynchronous request is sent in the background by the browser).
- Set the onload property of it
- At the end of the loading process - thanks to the function specified as value of onload will be called- the image is actually drawn in the canvas.

E.g.

```
var im = new Image();
im.src = "imageAddress" // this loads the image in the background; when it's finished, th
// in im.onload is called
im.onload = function () {
var cnv = getElementById("canvasId");
var ctx = cnv.getContext("2d");
ctx.drawImage(im);
}
```

The process is a little bit different when i must **load inside a canvas an image from an img element**: as usual, **i must ensure that the image has been completely loaded**! In order to do this, i must set the onload value of the window element to the function that draws the image inside the canvas using the `drawImage(im)` method. The onload method is not called back until the DOM implementation finished loading all content within a document, all frames within a FRAMESET, or an OBJECT element.

5.4 Path Drawing Mode: lines, circles, arcs, curves and other path drawing methods

All the instructions are put into a buffer; the drawing methods fire a parallel execution of the instructions. It is called `buffered mode`: the orders are stored in the GPU. Modern GPUs can execute up to hundreds/thousands operations in parallel: it is better to use this mode, than stroking, for example using `strokeRect()`. Call the `ctx.rect(x, y, width, height)`, instead: this method works in buffered mode. This stores the rectangles in the buffer; `ctx.stroke()` and `ctx.fill()` actually render the rectangles. `ctx.beginPath()` will reset the buffer (empty its content). `ctx.moveTo(x0, y0); ctx.lineTo(x1, y1); ctx.moveTo(x2, y2); ctx.stroke()` . **Remember: call** `ctx.beginPath()` **to empty the drawing buffer!** Common mistake: draw something twice because you didn't remember to empty the drawing buffer. As usual, a good practice is to use `ctx.save()` and `ctx.restore()` inside any function that manipulates the context.

5.5 Colors, Gradients, Patterns, Shadows, etc.

Colors can be defined as

- `'red'`
- `'#00ff00'`
- `"rgb(0, 0, 255)"`
- `"rgb(0, 0, 255, 0.2)"`

In the last case, the opacity (**alpha channel**) is defined to be 0.2, in a scale from 0 to 1, meaning that the object will be transparent blue (very transparent, since 0.2 is close to 0 opacity). Remember: **Opacity**!

5.5.1 Gradients

NOTE: if the gradient will be used in different places, make it a global variable and load it in the init() method, after the page has loaded (onload).

Linear Gradients

4 steps:

- `grad = ctx.createLinearGradient(x1, y1, x2, y2)` : direction of the gradient and starting and ending points
- `grad.addColorStop(p, color)` : p can have values from 0 to 1: this is used to define the colors of the gradient
- `ctx.fillStyle = grad` : set the gradient in the fillStyle property of the context
- Draw some shape (e.g. `fillRect(xa, ya, xb, yb)`)

Notes: the drawn shape will have the colors defined in the gradient in the area of the gradient; outside of it the external colors of the gradient will be extended constantly (without change, i.e. without fading, i.e. without gradient).

Radial Gradients

Same procedure as for the Linear Gradients, but with a different method for setting the gradient:

```
ctx.createRadialGradient(x1, y1, r1, x2, y2, r2)
```

The radial gradient goes from the circle defined by the first 3 parameters, to the one defined by the last 3 parameters - these are 2 independent circles.

5.5.2 Patterns

3 steps:

- Define an image object (it could be useful to have this global)
- Put the drawing code inside the onload function of the image object - this is good practice, since it will only work on the pattern after the image has been completely loaded in background by the HTTP request!

```
var img = new Image;
   img.onload = function() {
   ctx.fillStyle = ctx.createPattern(img, "repeat");
   ctx.fillRect(x1, y1, x2, y2);
   };
```

- Specify the source for the image: `img.src = "url";`

Note: the second parameter of the `createPattern` method can have one of these values: (`repeat | repeat-x | repeat-y | no-repeat`).

Drawing multiple images in a pattern

Pay attention to the asynchronism of JavaScript! Use a manager to check trigger the drawing function only when each image has been completely loaded - use an array to store the images, for example.

5.6 Shadows

There are 4 properties to draw shadows:

```
\item \texttt{shadowColor}
\item \texttt{shadowBlur}
\item \texttt{shadowOffsetX}
\item \texttt{shadowOffsetY}
```

Note: unwanted shadows can be removed saving the context, then setting the shadow parameters and then restoring the context; what will be drawn next won't have shadows, for example, but many other different combinations are possible.

5.7 Styling Lines (context properties)

```
\item \texttt{lineWidth}
\item \texttt{lineCap = "butt" | "square" | "round"}
\item \texttt{lineJoin = "miter" | "bevel" | "round"}
\item \texttt{miterLimit} : useful when \texttt{lineJoin} is \texttt{miter}: renders the
was set to \texttt{bevel}.
```

5.8 Drawing Tips - Recap

- **Attention!** Drawing arcs or other shapes doesn't clear the buffer! This means that a line connecting the last point in buffer to the first point of the actual shape will be drawn! Use fill/stroke and then `ctx.beginPath()` if you don't want the connecting line!

Chapter 6

Lesson 4 - HTML5 Animations

Once, before HTML5, the basic functions were

- `setInterval(fun, ms)` : fun is repeatedly called every ms
- `setTimeout(fun, ms)` : fun is called after ms only once

In addition, CSS backgrounds inside `<div>` were used, together with `top left width height` properties of the divs to animate images on the screen.

6.1 Basic Animation Techniques

The logic of animating objects is:

- Clear the canvas (`ctx.clearRect(0, 0, canvasWidth, canvasHeight)`)
  NOTE clear all the canvas!
- Draw the ctx
- Move the graphic object(s)
- Loop

The two aforementioned methods are now completed by the `requestAnimationFrame` API.

Problem using `setInterval`: if the code takes more time than the interval, the function is called before its code has been completely executed. Other problem: the scheduling is not very accurate, i.e. the ms value may also be not always respected.

The updated style, available in HTML5 uses `requestAnimationFrame(drawingFunction)`, both inside the function that calls the drawing function (e.g. init()), and inside the drawing function itself (recursion). This results in much smoother animations - approaching the 60 fps.

Anyway, `setInterval` can still be used to change some state, like colors or other properties. Note: this function returns the unique ID of the action:

i can stop calling the function using the function `clearInterval(ID)`. Note: check for existence of ID, before clearing it, i.e. `if (ID) ....` Anyway, use this method only if the drawing code is trivial, since it can lead to rendering problems (partially completed), complex debugging and unexpected behaviour.

The cool difference with using `setTimeout` is that it will never call itself until the drawing code has been completely executed. It must be placed outside from the drawing function and also inside of it: this second call won't happen until the previous (drawing) code has been completely executed! Problem: the timing may be different than the one we set, since the function will not be called until all the previous code has been executed.

It is now clear that both `setInterval` and `setTimeout` do not guarantee smooth animations, with the exact timing that we ask to them. That's why we should use `requestAnimationFrame`! The advantages of using this method are:

- tt target 60 fps;
- it calls the function only once;
- the scheduling is more accurate;
- high resolution timer (good to make time-based animations)
- multiple animations are merged (animations happening at the same time are bundled into a single paint redraw, happening faster, with less CPU cycles);
- CPU/GPU optimization, leading to a less battery consumption: the animation will be executed, the objects will be moved, but they won't be drawn, if the tab is not visible or the screen of the device is black, or the app is in background (mobile devices).

Like the other functions, it returns an ID, that can be used as argument to `cancelAnimationFrame(ID)`.

6.2 Canvas and User Interaction (keyboard, mouse)

We treat the events made by the user as inputs and we manipulate the structure of the DOM as output. The events are called DOM events and we use the DOM JavaScript API to create event handlers.

The suggested way to handle events in the DOM is by calling `addEventListener(event)` on the elment that will have the interaction with the user that we want to listen. Inside event we'll find informations like the mouse button that was pressed, the key that was pressed, the position of the mouse, etc.

Example: adding a listener for keyboard related events (some of them are: `keydown, keyup, keypressed`), tipically to the `init` method:

```
window.addEventListener("keydown", function(event) {
   if (event.keyCode === 37) { // left arrow
   // code
   }
   }, false);
```

See fig. 6.1 for a list of key codes.

Note: the last parameter of `addEventListener` is used to specify if the ancestors of the element should be listening too.

Note: if i only want to listen to events in my canvas, i must:

- make the canvas focusable including the attribute `tabindex` and giving it a value (e.g. "1");
- force the focus on the canvas element calling the `focus()` method on it
- calling `addEventListener("event", function(event)  // code , false);`, this time on the canvas element!

A better way consists in setting the focus to the canvas when the cursor enters its area, and unset it when the cursor is out of it. I can do that using the `"mouseenter"` and `"mouseout"` events.

6.2.1 Events

- `keydown`
- `keyup`
- `mouseenter` : fires only once, coming from parent
- `mouseout`
- `mouseleave`
- `mouseover` : fires only once, coming from child or parent
- `mousedown`
- `mouseup`
- `mouseclick`
- `mousemove` : continously fires
- `resize` : good for smoothly resizing canvases inside divs (best practice). Note: there is no way to listen to a div's resize: listen to the window resize!
-

Note the different behaviour of `mouseenter`, `mouseover` and `mousemove`.

6.2.2 Getting the mouse coordinates relative to an element

The event parameter of the function passed to addEventListener contains many properties, like `clientX` and `clientY`, but these refer to the window; use `var rect = element.getBoundingRect()`, where `element` is a reference to the desired element and then the `rect.left` and `rect.top` properties.

6.2.3 Resizing a Canvas

Changing the size of a canvas using its `width` and/or `height` properties is not good, since it deletes its content and resets its context. Using % in the CSS `width` and `height` properties clearly doesn't change the image resolution, giving a blurry effect when the image gets too large. Note: if i declare `<canvas if=mycanvas width=100 height=100>` and then, in the CSS:

```
#mycanvas {
   width: 100%;
   }
```

the canvas will always have the same width of the window, also after resizing it, **but the content drawn in the canvas won't change its resolution**, probably leading to a blurry image!! So, never use CSS percentage on a canvas width or height!

The best practice for resizing, responsive and well rendered canvases is to use a **resize listener** and a **parent element**. The procedure is:

- embed the canvas into a div
- make a CSS rule with % for the width and/or height properties of the div
- add an event listener on `resize` to the window element
- (in the called back function) set the width and height of the canvas element to the size of the `parentDiv.clientWidth` and `parentDiv.clientHeight` - note: this also erases the content of the canvas!
- redraw stuff into the canvas taking into account the new canvas size.

Note: a common trick to avoid that the content goes out of the actual window when it is resized too small is to use the scale function.

Figure 6.1: A list of keycodes

Key	Code
backspace	8
tab	9
enter	13
shift	16
ctrl	17
alt	18
pause/break	19
caps lock	20
escape	27
(space)	32
page up	33
page down	34
end	35
home	36
left arrow	37
up arrow	38
right arrow	39
down arrow	40
insert	45
delete	46
0	48
1	49
2	50
3	51
4	52
5	53
6	54
7	55
8	56
9	57
a	65
b	66
c	67
d	68

Key	Code
e	69
f	70
g	71
h	72
i	73
j	74
k	75
l	76
m	77
n	78
o	79
p	80
q	81
r	82
s	83
t	84
u	85
v	86
w	87
x	88
y	89
z	90
left window key	91
right window key	92
select key	93
numpad 0	96
numpad 1	97
numpad 2	98
numpad 3	99
numpad 4	100
numpad 5	101
numpad 6	102
numpad 7	103

Key	Code
numpad 8	104
numpad 9	105
multiply	106
add	107
subtract	109
decimal point	110
divide	111
f1	112
f2	113
f3	114
f4	115
f5	116
f6	117
f7	118
f8	119
f9	120
f10	121
f11	122
f12	123
num lock	144
scroll lock	145
semi-colon	186
equal sign	187
comma	188
dash	189
period	190
forward slash	191
grave accent	192
open bracket	219
back slash	220
close braket	221
single quote	222

Chapter 7

Lesson 5 - HTML5 Forms

Some years ago, many frameworks provided the GUI elements and the input validation APIs to the various types of inputs. In the last years, HTML5 has added many elements, in order to take into also account mobile devices:

- input validation JS API
  - CSS subclasses to change the aspect of an input element depending upon its validity
- visual feedback
- contextualized keyboards (depending upon the type of the input field in the form)
- 13 new `<input type=...>` fields:
  - email
  - tel
  - color
  - url
  - date
  - datetime
  - datetime-local
  - month
  - week
  - time
  - range
  - number
  - search

Other elements:

- the option to set an input field out of a form;
- new elements, such as:
  - `<datalist>` for auto-completion;
  - `<output>` for feedback
  - others.

7.1 Introduction to HTML5 Forms

We'll see examples of some of the different input types. Some tips:

- I can *callback* methods using the `oninput` attribute of the input element.
- A good practice is to add a `label` for the input element: to make this label appear next to the input element, set the `for` attribute of the label to the same value of the `id` attribute of its relative input element. This is also good for *accessibility*.

7.1.1 `range`

It renders a slider with the round handle that the user can drag from left to right. Attributes:

- `min`
- `max`
- `value` - the default value for the slider.

If i add a `label` to the `range`, pressing the left/right arrow keys after selecting the label changes the value of its associated input element. Another thing that can be set on this set of elements (label + range) is the `output` field. Set the `id` attribute of the `output` element in order to be able to refer to it via code using the `getElementById()` method.

To get the value that the user has set using the slider:

```
size = parseInt(event.target.value);
```

where `event` is the argument of the callback function (i must specify the event parameter also in the value of the `oninput` attribute of the range element!).

To set the output value in the output element:

```
document.getElementById("output").innerHTML = size;
```

7.1.2 color

Renders a color chooser that opens the OS color chooser. Remember to associate a `label` using the `for` attribute of the label and also to set a callback method using the `oninput` attribute of the color element, as seen for the range element.

It's good to set the default value for the color (hexadecimal) using its own `value` attribute. Note: the value this time is an integer, so i don't need to parse the int from value as i did for the range.

7.1.3 Dynamically creating GUI elements

We have seen an example where an unordered list is specified in the HTML code and then all of the element items are dynamically added via JavaScript code. We use the DOM API to accomplish this task.

The DOM method used to create an element is

```
document.createElement("element");
```

where "element" can be a `label`, a `li`, an `input` or other HTML5 elements.

Once i have the element, i set its attribute using the

```
element.setAttribute("attribute", "value");
```

The label has the `textContent` property to set its text value; the input has the `value` property to set its default value; the list item and the list have the `list.appendChild(listElement)` method to append elements to them - note that the content of a list item must be appended to it using the previous method on the list item; then the list item itself must be appended to the list:

```
li.appendChild(label);
   li.appendChild(input);
   list.appendChild(li);
```

7.1.4 Forms

They have 2 main attributes:

- `action` the url of the server-side component that will process the data entered (usually a php, python, ruby, Java, etc. file)
- `method` the way data will be sent to the server: can be `POST` or `GET`

Good practice: group elements inside a `<fieldset></fieldset>`; use a `<legend></legend>` to add a title to the fieldset. There's a common technique that is used to align text input fields using CSS:

```
input {
   float:right;
   margin-right:70px;
   width:150px;
   }
```

Use the `border-radius` CSS attribute of the fieldset to make it look nicer.

An input field, with type `text` can have the `required` attribute, that will make it *invalid* if it's empty. I can style this state using the super class inherited by the input elements (new feature, introduced in HTML5), like this:

```
input:invalid {
   /* style.. */
   }
```

This is an example of input-validation system. A bubble with an alert will automatically pop up if a required field is left empty. The language of the message in the alert is the same as that of the OS.

Other features of input-validation exist for the input with type `email`: the input must look like an e-mail addres.

For the input with type `number` the attributes `min, max, step, required` can be specified.

The required attribute can also specified for the `date` type of the input element.

7.2 Accessible Forms

There are 2 main guidelines to make accessible forums:

- add a *descriptive* `label` item for each field of the form and use the `<label>` element to identify each form control;
- for larger or complex forms, use the `<fieldset>` and `<legend>` elements to *group* and *associate* related form controls.

A very important thing to implement is the `for` attribute for the label, setting it the same value as the id of the related element (i.e. an input element). Using the `for` attribute makes the label clickable. In the case of an input (text) element, clicking on its label puts the focus on the input element. This also happens in the case of a textarea. In the case of an input (checkbox), clicking the label will toggle the selection on the input. This also happens for input (radio) and input (checkbox). This behaviour is very good for visually impaired and people with difficulties in movements.

Labels and inputs can also be nested: this may lead to an easier CSS styling and produce better results with screen readers.

WAI-ARIA lets you create associated controls using the `role` attribute (i.e. specify it on a div element).

7.3 New Input Types

Letting the user choose a color is much easier than before, using the `color` element. A window will pop up with a color chooser. To restrict the basic color icon on the page to a selection of few colors, put the color values as `option`s inside a `datalist`; the datalist's `id` must have the same value of the color's `list` attribute. Color values must be specified using the hexadecimal CSS notation. As usual, take a look at the compatibility of these new elements, before taking their perfect behaviour for granted on each browser.

One critic to complex widgets like the one rendered by the color and date elements is that they are rendered differently by different browsers/OSs and they are little or no customisable. One solution to this is using *Web Components*, a new approach to design HTML5 widgets.

7.3.1 `input type=date`

The dates are string in the format yyyy/mm/dd. Use these attributes to tweak:

- `value`: actual value
- `min`
- `max`
- `step`: set this to 7 and only the same day of the week as the one set in value will be selectable.

Use the `list` attribute to specify dates: the same value of list must be set for the id of the `datalist` element that will contain the specified dates. The dates must be the content of `option` tags.

I can retrieve the date chosen in JS setting the `oninput` attribute to a function (with a parameter for the event) and then use `this.value` inside of it. Note: this value is a string! If i want it to be a date object, i must use `this.valueAsDate` inside the callback function of the `oninput` attribute of the date element.

I can check if the date is in the past or in the future, comparing it to a `new Date()` object, that represents the date of today. Note: i must use the `this.valueAsDate` to get the date as a valid date object. This value can be obtained inside the callback function of the `oninput` attribute of the date

element. Note that the `this` keyword inside a callback represents the object that triggered the callback.

The different types of input elements automatically render different keyboards. The CSS classes `input:valid` and `input:invalid` are automatically inherited when the user starts interacting with the input element.

The `placeholder` (default, greyed out value) and `pattern` (specify which pattern - regex - the users' input should satisfy) attributes of input elements may be useful.

The `input type=number` can store integers or floats. Use text with regexp for zip codes (important for validation!). Some useful attributes are: `value, min, step, max`. It gets automatic CSS validation, inheriting the `input:valid|invalid` class. NOTE: step defaults to 1, so, in order to accept floating point values, it must be set to `any` or to a decimal value. This is not needed for integer values.

The `input type=range` renders a slider. The same useful attributes seen for the number type are valid for this element too. Furthermore ticks can be added using a datalist element with the values specified in option elements. The datalist's id value must be equal to the input's list value. Sliders can be styled using CSS.

Remember to use the `value` property to get values from input elements!

7.4 New Form Attributes

They are[1]:

- `form` : this element is useful to make element that are outside a form (i.e. external to a fieldset) part of the form too;
- `readonly` :
- `autocomplete`
- `autofocus` : puts the focus to the input element (if not present, it automatically defaults to the first)
- `list` : works together with datalist; datalist provide a list of items for the list using `option` elements. Note: the `list` attribute of the `item` element, and the value of the `id` attribute of the `datalist` element must match!
- `pattern` : must be a valid JS regex (see html5pattern for a list of ready-to-use useful patterns).
- `required` : makes empty fields invalid

[1] for the boolean attribute, writing the name of the attribute means that you want that feature

- `placeholder`
- `multiple`
- `list`
- `min`
- `max`
- `step`
- `formaction`
- `formenctype`
- `formmethod`
- `formtarget`
- `formnovalidate`

7.5 New Elements Related To Forms

These elements are:

- `datalist` : useful for linking a list of choices to an input element
- `output` : example:

```
<form oninput="o.value=a.value*b.value">
        <input type="number" name="a" id="a" value="2"> x
        <input type="number" name="b" id="b" value="3"> =
        <output for="a b" name="o">6</output>
   </form>
```

  Note that the for label of the output must contained a space separated list with the value of the ids that it will receive its data from; also, the value of the name of the output can be used in the `oninput` attribute to put values in it. Use the `valueAsNumber` attribute when dealing with range and number, to be sure to get a number, instead of a string (value).

- `meter` : represents graphically numeric values. Do not use it to show the progress of an activity. Example:

```
<p>Grades: <meter id="meter2" value="75" min="0" max="100" low="20" high
   <input min="0" max="100" value="75" id="meter2range"
          oninput="effect('meter2', 'meter2range')" type="range">
     <output id="meter2val" for="meter2range"></output></p>
     <script>
     function effect(meter, meterrange) {
        var currVal = document.getElementById(meterrange).value;
        document.getElementById(meter).value = currVal;
        document.getElementById(meter+ "val").innerHTML = currVal;
     }
     </script>
```

- `progress` : define it using its `id, value, min, max` attributes; set its value using its `value` attribute.
- `keygen`

7.6 Form Validation API

The built-in validation system that comes with HTML5 automatically add a pseudo class to every input fields and forms. Valid/invalid input in fields make that field automatically inherit the `input:valid/input:invalid` CSS pseudo classes. Other pseudo classes are `input:optional, input:required`. NOTE: always providing CSS styling for these subclasses is a good practice. Another good practice is using the `title` attribute to show messages for suggestions in case of invalid input.

The validation can also be done through the JS Validity API: there is a method, for example, that `input type=password` element can respond to: it is `setCustomValidity('msg')`. If msg is empty, it means no errors occurred; if it is not null, it means that the input is invalid and the string is used as error message. Each input element has a `validity` JS property that holds a validity state object: this is used to better understand the reason of invalidity. This reason can be one of:

- `valueMissing`
- `typeMismatch`
- `patternMismatch`
- `tooLong`
- `rangeUnderflow`
- `rangeOverflow`
- `stepMismatch`

- `valid`
- `customError`

Another way to get informations about the invalidity is using the CSS `validationMessage` property of input fields.

Chapter 8

Lesson 6 - HTML5 Basic APIs

8.1 Introduction to HTML5 Apis

We will take a look at:

- The HTML5 Cache API for making websites and web application work offline.
- The "Web Storage API" for storing pairs of key/values client-side. Useful for making web sites able to save/restore their state or for writing serverless applications.
- The File API, that makes web applications able to work with local files. E.g. listen to the music stored on your device, view pictures and document also stored on your device.
- The Geolocation API to get the longitude, latitude, altitude, speed, heading, etc. and getting able, for example, to auto-fill addresses in forms based on the current location!

8.2 HTML5 Cache

Caching webpages and webapps leads to the possibility of working offline and also to faster loading of the pages: this is especially important for frequently visited websites.

The Developer specify which resources should be cached in a manifest. What should and what should not be cached. Use the `manifest` attribute of the `html` tag and set it to a value (string) that ends with `.appcache`: this will tell the browser to cache that page, so i must do this for all the pages that i want to cache. Inside the `.appcache` file i must provide a list of all the files that need to be cached. Example, mycache.appcache :

```
CACHE MANIFEST
```

```
clock.html
clock.js
clock.css
jpeg files, documents, etc.
```

External resources must be cache too, like jQuery, for example, so that the website will continue to work properly offline. Use a tool, like ManifestR.

8.2.1 How to exclude certain files

Here's how:

- remove their name from the list of cached file;
- specify what not to cache; the `*` is used to not cache the files not listed in the `CACHE` section:

  ```
  NETWORK:
  *
  ```

- declare the fallback to use when the user chooses services that are not available when offline:

  ```
  FALLBACK:
  / offline.html
  ```

- The page that declares the manifest will automatically be cached, but adding it anyway is good practice. Include the manifest attribute in any page, if the web app must also work offline.
- It's better to use relative paths for the manifest attribute!
- The manifest file must be served with the correct MIME type: the HTTP server that serves the files of the website must be configure so that .appcache files are served with the MIME type: `text/cache-manifest`. With the Apache server, the `HTTP.conf` file must contain this line:

  ```
  AddType text/cache-manifest.appcache
  ```

- If a file is in cache, that will be used, even if online! See later how to update the cache.
- No partial update possible: if a file cannot be retrieved and cached, zero files will be updated in the cache.
- Comments can be added inside manifest files using `#` .
- Using a manifest file validator is a good practice!

- Partial URLs can be used in the NETWORK section (but not in the CACHE section, where each file must be explicitly declared with its full path), like:

  ```
  /images
  ```

  that means "do not cache the files in the /image directory".

- The lines:

  ```
  FALLBACK
  / offline.html
  ```

  tell the browser what to load when the user is trying to access a not cached resource: the `/` means "for all the resources that are not accessible" and the rest means "open offline.html".

- Partial URLs are available in the FALLBACK section too:

  ```
  FALLBACK
  /images/ /images/missing.jpg
  ```

  meaning that all the files in the `/images/` directory - if not available in the cache - should be replaced by the `missing.jpg` image, from the same directory, when offline.

If my code imports other resources, using CSS `@import` or images, technologies (i.e. jQuery), CSS/JS addons or other, it could be hard to specify all of those resources in the manifest file; this is why some tools exist just for that purpose. There's a *same origin policy* about website accessible only through `https` that says that only resources from the same domain should be cached, but this is usually not contemplated, since it is too restrictive.

8.2.2 Updating the cache

Caution! *If a website is in cache, the browser will show that version, instead of the online version!* Best practice: after modifying the website, developers should add a comment with the modification date in the manifest! The browser looks at the manifest's modification date to understand if they must update the cache! If the date is more recent than the date of the cached manifest, the browser will update all of the files in cache the have a less recent date than the ones on the server.

Developers clean the cache manually: this will also update the cache, clearly.

The DOM Api has a property called `navigator.onLine` that is true if the computer is connected to the internet. NOTE: This doesn't guarantee that the browser will be able to retrieve any website, since it doesn't refer to DNS

server problems or remote server (the one that hosts the website/service) i am currently requesting) problems.

8.3 The Web Storage API

It features 2 related mechanism, similar to HTTP session cookies, to store structured data on the client side. It provides 2 interfaces, sessionStorage and localStorage: the first holds the data until it is deleted, the second does that only until the tab/browser is closed. This specification defines an API for persistent data storage of key-value pair data *in web clients*.

Both of them work as a simple key-value store, *one per domain*. The key-value pairs are strings. There is only 1 store per domain. The content of the store is global. The data is located in a store attached to the origin of the page.

Cookies are also a popular way to store key-value pairs of data, but they are limited in size and they generate HTTP traffic for each additional request. Objects managed by Web Storage are no longer carried on the network and HTTP and are easily accessible (read, change and delete) from JavaScript, using the Web Storage API.

Values can be set this way:

```
localStorage.key = "value";
```

The same applies to `sessionStorage` data.

For security reasons Web Storage only works through `http | https` and not with `file`.

An example of usage of this technology is restoring data in a form that was previously been input, after a reload of the page - very useful feature! E.g. add `onload="localStorage.firstName=this.value;"` inside each input text element (similar for other kinds of input)! Then, retrieve the data using,

```
window.onload = restoreFormContent();
   function restoreFormContent() {
   if (localStorage.firstName !== "undefined") {
   document.getElementById("firstName").value = localStorage.value;
   }
   }
```

Some methods:

```
localStorage.setItem(key, value);
localStorage.getItem(key);
localStorage.removeItem(key);
localStorage.clear(); // remove all keys
```

Using the accessors i can use more complex values for the keys (including spaces): I can't do this using a property name!

8.3.1 Iterating a localStorage

Example:

```
for (var i=0; i<localStorage.length; i++) {
   var k = localStorage.key(i);
   console.log(localStorage[k]);
   }
```

Notice the `length` property and the `key` method.

Instead of using cookies, use `sessionStorage`: it's more convienent, it can store more data and also it doesn't leak data through different tabs (the scoping is only inside the tab)!!

An alternative to basic key-value objects are JSON objects. Methods: `stringify`, to create one from a string; `parse`, to get a key-value object from a JSON object.

8.4 The File API

Designed to work with files on server-side in JavaScript. See `FilesAPI/files.html` inside the HTML5 coding directory for an example that uses this api to load images on the HTML5 document from images on the hard disk. Notes about this code:

- The files are input via a `input type="file"`; this input element has the `multiple` attribute, that lets the user choose more than one file from the hard drive (a window will appear after clicking the button). Selected files are referenced inside this input element with `this.files`.
- The different files are accessed from the array of files that has been passed as input to the `readImagesAndPreview` function set as value of the `onchange` attribute of `input type=file`.
- An empty div is used in order to get the container for the pictures; it gets populated via code using the `appendChild(img)` method on the reference to the div (obtained using `document.getElementById("divName")`).
- The img element is created using the `document.createElement("img")` method.
- The src of the img is set with `img.src = e.target.result;`, where `e` is the argument to the `reader.onload` function.

- `reader` is the file reader obtained with the constructor `reader = new FileReader();`.
- Each file is rendered as image using the reader: `reader.readAsDataURL(file)`. After this method finishes, the `onload` of the reader is automatically called, the data are read and the image is rendered inside the div (appended to it).

8.4.1 Metadata of files

The files selected using `input type=file id=finput` can be accessed through an array:

```
var file0 = document.getElementById("finput").files[0];
```

This file element has lots of meta-data properties: `name, type, size, lastModifiedDate`, etc.

Using the `accept` attribute of the `input type=file` element, i can let the user select only images: `accept="image/*"`.

8.4.2 New interfaces - File API

These are the new interfaces introduced by this API in HTML5:

- `FileList` : the `files` property is a `FileList`.
- `File` : one element of the `files` array is a `File`. They inherit their properties from the `Blob` class, and add 2 more: `name` and `lastModifiedDate`. Usually I will work with File objects, instead of Blob ones.
- `Blob` : chunk of raw data. A structure that represents *binary* data available as read-only. These objects have 2 properties: `size`, that store the size in byte and `type` that stores the MIME type of the data. There is also a method called `slice()` that is used to slice files.
- `FileReader` : to read file content

8.4.3 How to read the content of a File

Steps:

- create a `FileReader`;
- get the File (`<input type=file id=file ... ... onchange="readFileContent(this.files[0]);">`) object;
- use `readAsText | readAsArrayBuffer | readAsDataURL` ;

- set the `onload` callback to operate on the data just obtained used one of the previous methods.

Example:

```
var reader = new FileReader();

   function readFileContent(f) {
   // this will be called for second, after the
   // readAsText has finished loading the File
   reader.onload = function(e) {
   var content = e.target.result;
   // do what you need with the content
   console.log("File " + f.name + " content: " + content);
   };
   // this will be called first
   // start reading the file f asynchronously;
   // only calls the callback when the
   // content is completely loaded (i.e. the
   // file has been read completely)
   reader.readAsText(f);
   }
```

Notice that the operation of reading the File object, with one of the 3 possible methods, is asynchronous, since it may take some time; this is why it gets executed in background. Also notice that the File object `f` has been passed by the function specified in the `onchange` attribute of the `<input type=file..` element! Of course, if i want the entire array of files (if i specify the `multiple` attribute for the input element, then i will drop the `[0]` after `this.files` in the `onchange` attribute.

8.4.4 readAsArrayBuffer()

This is usually used to load audio to play later using the WebAudio API or to load texture that will be used with WebGL for 3D animations. Note: the WebAudio API works only with local files. I can create an MP3 player web app without the need of a server.

8.4.5 readAsDataURL()

A *dataURL* is a URL that contains both type and content at the same time. It's becoming quite popular, especially for mobile devices, since it reduces the internet traffic and increases the loading speed. It's getting popular for inlining images: it can be used as argument to the `src` attribute of `img` elements and also as url. There are many online tools that generate this kind of data. Any type of files can be encoded as dataURL, but it is usually

used with media files, like images, audio, video. Remember: this method passes a variable, e.g. `e` as argument to the function called back on the `reader.onload`, where the reader is the FileReader.

8.5 The Geolocation API

A simple example:

```
if (navigator.geolocation) {
   navigator.geolocation.getCurrentPosition(showPosition, error);
   } else {
   // inform the user that her browser does not support geolocation
   }

   function showPosition(position) {
   console.log("Position: latitude " + position.coords.latitude +
   " longitude: " + position.coords.longitude);
   }

   function error(e) {
   console.log("Error: ...");
   }
```

Note: when a current position is available, the callback function will be called asynchronously and the position parameter of this callback function will be the current position.

The error parameter of `getCurrentPosition()`

This parameter has a property called `code` that can have one of the following values:

- `error.TIMEOUT`
- `error.PERMISSION_DENIED`
- `error.POSITION_UNAVAILABLE`
- `error.UNKNOWN_ERROR`

8.5.1 Properties of the coords object

Note: not all browsers support this: a null object is returned, in that case.

- `latitude`
- `longitude`

- `altitude`
- `accuracy` (meters)
- `altitudeAccuracy`
- `heading` : orientation, relative to North
- `speed`

8.5.2 The `watchPosition()` method

It calls the callback each time the position changes (`getCurrentPosition()` only does that once). The parameters are the same as for the previous method. It returns an ID, so I can stop the current tracking using `navigator.geolocation.clearWatch(id`

A third parameter can be added to the last 2 methods introduced, that will hold one or all of these options:

- `enableHighAccuracy` : more or less it means "use the GPS!" (true|false)
- `maximumAge` : store data in cache, allowing for battery/bandwidth saving (millis)
- `timeout` : time before returning an error (millis)

These values must specified inside a structure, like this:

```
{enableHighAccuracy:true, maximumAge:10000, timeout:1000}
```

See an example at: `/home/pvag/Coding/HTML5_edX/someHTML/...`
`...Add variable as string using innerhtml/imgsrc_string_test.html`

Chapter 9

Tags

9.1 <time>

e.g.

```
<p>Appointment:<time datetime="2015-05-20">Monday 20/05/2015</time></p>
```

The datetime parameter is used to specify a date or time; for a period of time, use datetime="P4D" (period of time of 4 days).

9.2 <mark>

It can be used for:

- Display search results with searching words highlighted
- Highlight important parts of a text, like "quoting parts"
- Replace and when suitable.

9.3 <figure>

It allows inner <figcaption>. See

/Coding/HTML5_edX/someHTML/Audio/audio_transitions.html
and audio_transitions.css

9.4 <code>

Used to type code: fixed size font.

Chapter 10

Tricks

- `<body onload="init();">`

  calls the init() method

- `window.onload = function() { // code }`

Chapter 11

Tips

- When including videos, try to provide them in more than one codec, if possible. H264 is the most compatible, but webm allows much playing the video after a much shorter buffering time.

Chapter 12

Microdata

Contain content that is not visible by the user; it's just for the engines, to better understand the content of the pages, the topic they talk about. The main interest is Search Engine Optimization. Examples: events, a person's profile, the description of an organization, the details of a recipe, a prduct description, a geographical location, etc. It's HTML5's way to embed semantics into html documents. There are 3 attributes to consider:

- itemscope - starts a microdata section
- itemtype - defines the schema
- itemprop - defines the key name (aka property); then there's its value

Define a container element (e.g. a section) and insert the itemscope attribute inside it.

```
<section itemscope itemtype="http://schema.org/Person">...</person>
```

Now i have defined a global wrapper object, the Person: i can add elements inside it, to define the first name, last name, etc. HTML5 proposes semantic elements to represent sections, articles, header, etc., but it doesn't propose any specific element nor attribute for the elements of an Address, or a Person, and so on. I can use the specifications provided at the schema.org website, thanks to the itemtype attribute. I could also define my personal vocabulary to define the elements of an object like a person. Microdata works with properties defined as key/value pairs. This vocabulary is called a *schema*. I can also have an object as a value to a particular key. Inheritance is possible among schemas. See an example of microdata usage at /home/pvag/Coding/HTML5_edX/Microdata/microdata.html . It is possible to define the value of a property more than once. It is possible to nest schemas: the one in the inner section will lead. It is also possible to define more than one property at the same time with the same value(s). Note that

for particular html elements the value will be the value of another attirbute, as for example:

```
<img itemprop="image" src="site.com/im.png"
 alt="cool image description">
```

the value of the microdata image property will be the value of the src attribute.

Many tools exist to automatically generate microdata content.

Chapter 13

New Features

Chapter 14

Misc

14.1 Jargon

- **Polyfill**: is a new feature, not officially released yet, but that is implemented in JS.
- **DOM**: Document Object Model.
- User Agent: the browser

14.2 HTTP Request

In its header put

- Content-type:image/jpg - to open the resource in a new tab
- Content-Disposition: attachment; filename="myImage.png"; - to download the image, renaming it myImage.png

With the new <a> tag there's a new simple way to download the resource specified in href: use its download attribute set to a file name (the name i want the downloaded file to have after the download, even if different from its original name)..

```
<img src="theImage.png" download="imageNameAfterDownloading.png">
```

For security reasons, images should be located on the same domain of the HTML page containing the link.

14.3 Do Not Translate!

translate="no" (yes for translating) Useful for pages displaying stuff that translators, like Google Translator, shouldn't translate. Another state is possible: "inherit state". Children inherit this behaviour; the reverse is true.

14.4 Callback vs. Fallback

A callback is triggered by some element when something happens on it, i.e. it is clicked or selected (`oninput`); a fallback is started as an alternative when something is not found, i.e. polyfills are used if the browser doesn't provide native support to a specific element for which the browser does.

Part II

Beyond HTML

Chapter 15

JavaScript

JavaScript is an asynchronous language: it sends multiple requests and to perform different operations at the same time, in parallel. This means that i should pay attention when such a request is made, since its result may not be ready whenever i want to use it! That's why a good practice is using callback functions, tipically on the onload property.

15.1 Methods

- `querySelector` Example:

  ```
  document.querySelector(‘‘#id_of_element’’);
  ```

  Note: the selector-syntax (#) ! This method lets me get elements from the DOM. Note: `#id_of_element` could also be `element_name#id_of_element`, where `element_name` can be, for example, div.

  If theVideo is a video element:

  ```
  theVideo.addEventListener(‘‘ended’’, funcToCall, false);
  ```

  This calls back the function funcToCall when the video in theVideo object ends.

- `getElementById ; getElementsByTagName` Example:

  ```
  document.getElementById(‘‘theId’’);
  document.getElementsByTagName(‘‘theTag’’)[n];
  ```

  Notice: the id is the value of the id attribute of the element; the tag is the name of an element, like `video | section | body`. Note: usually, there are many elements with a specific tag name: they are returned into an array. This is the reason behind the method name: it's `getElements`ByTagName().

- `innerHTML` lets me introduce content inside an HTML element. Example:

  ```
  var theP = document.getElementsByTagName(‘‘p’’)[0];
  theP.innerHTML = ‘‘<h1>Cool p</h1>’’;
  ```

- `addEventListener` Applied for example to a div element containing many buttons, lets me execute some code depending upon which button was clicked. Example:

  ```
  controls.addEventListener(‘‘click’’, functionToCall());
  ```

- `setTimeout` Example:

  ```
  setTimeOut(function, timeoutmillis);
  ```

- `onClick` Is the method used in JS to set the callback function to a button when clicked.

- `setInterval(method, millis)` : often used inside init() to trigger methods at given time

- `console.time(nameOfTimer)`

- `console.endTime(nameOfTimer)` : prints the string contained in nameOfTimer and the elapsed time in ms.

- `output.insertBefore(span, null)` : a method of an output element; e.g. insert an image in the span and then the span in the output element. The output element has children spans inside of it.

15.2 JS Misc

- JS code in <script> gets executed when the page loads.

- access the attributes of an element via the variable that refers to it - e.g.

  ```
  var vid = document.getElementById("idOfThevideo");
  vid.width = 100;
  vid.style.left = 0;
  ```

If i pass `this` to a function call (e.g. `onclick`) in a tag, the `this` keyword represents the tag element. Inside the JS code i can refer to it, for example to change it's html value or other attribute.

`Math.PI` renders π.

Chapter 16

CSS

16.1 How to reference elements, ids, classes - selectors

- element : element_name ...
- id : #id_name ...
- class : .class ... - can also be element.class ...
- universal : * ... - all the elements of an HTML document

It is possible to group together different selectors and set their properties to the same values:

```
h1, h2, h3 {background : #aaaaaa;}
```

More than one class can be specified for an element:

```
<p class="class1 class2 class3" id="p1">...</p>
<p class="class1 class3" id="p2">...</p>
```

The CSS code

```
p.class1.class2 {color : white;} ( 1 )
```

Will only apply to the p with id p1, since all the listed classes must be present in the element, for the CSS style specified by (1) to be applied.