

Lecture 13

Not Your Father's TV: Video ...

[1]

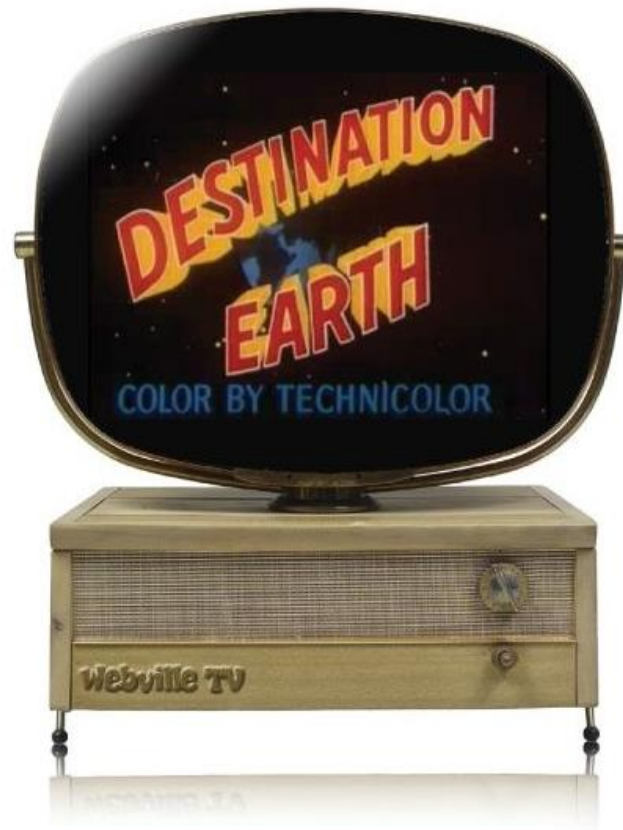
Samkeun Kim <skim@hknu.ac.kr>

<http://cyber.hknu.ac.kr/>

Meet Webville TV

We have to build **Webville TV** if we want it to be a reality. Over the next few pages we're going to build Webville TV from the ground up using **HTML5 markup**, the video element and a little JavaScript here and there.

Webville TV
built with 100%
HTML5 technology.



The HTML, let's get it done...

Let's jump right in and create some HTML:

```
<!doctype html>
<html lang="en">
<head>
  <title>Webville TV</title>
  <meta charset="utf-8">
  <link rel="stylesheet" href="webvilletv.css">
</head>
<body>
<div id="tv">
  <div id="tvConsole">
    <div id="highlight">
      
    </div>
    <div id="videoDiv">
      <video controls autoplay src="video/preroll.mp4" width="480" height="360"
        poster="images/prerollposter.jpg" id="video">
      </video>
    </div>
  </div>
</div>
</body>
</html>
```

Pretty standard HTML5.

Don't forget the CSS file to make it all look nice.

<http://ksamkeun.dothome.co.kr/wp/hfhtml5/ch8/webvilletv.css>

Just a little image to help make it look like a television set.

And here's our <video> element for playing our video. We'll take a closer look in a sec...

실습과제 13-1

You need to make sure of a few things here: first, make sure you've got the code above typed into a file named **webvilletv-1.html**; second, make sure you've downloaded the CSS file, and finally, make sure you've also downloaded the video files and placed them in a directory named video. After all that, load the page and sit back and watch.



=> 과제 제출 시 비디오 파일 제외하고 제출!!

<file:///E:/practice/wp/hfhtml5/ch8/webvilletv.html>

images.zip:

<https://drive.google.com/open?id=10f37EnA46jbgcib13-rW6vov1spTYc5N>

video.zip

<https://drive.google.com/open?id=1k9wnoEOi6LNO51TdjZG5zheb-zMCapMp>

I'm not seeing any video. I've triple checked the code and I have the video in the right directory. Any ideas?



Yes, it's probably the video format.

브라우저 메이커들이 HTML5에서 **<video>** 엘리먼트와 **API**가 어떤 모양을 가져야 하는 지에 대해서는 의견 일치를 보았다.

반면에, 비디오 파일 자체의 **실제 포맷**(format)에 대해서는 의견 일치를 이루지 못했다.

- ✓ Safari는 H.264 포맷을 선호, Chrome은 WebM을 선호

3가지 종류의 비디오 타입 (3가지 다른 확장자를 가진):
".mp4", ".ogv", ".webm"

Safari / IE9+

⇒ **.mp4** 사용 (src="video/preroll.mp4") (which contains H.264)

Firefox / Opera

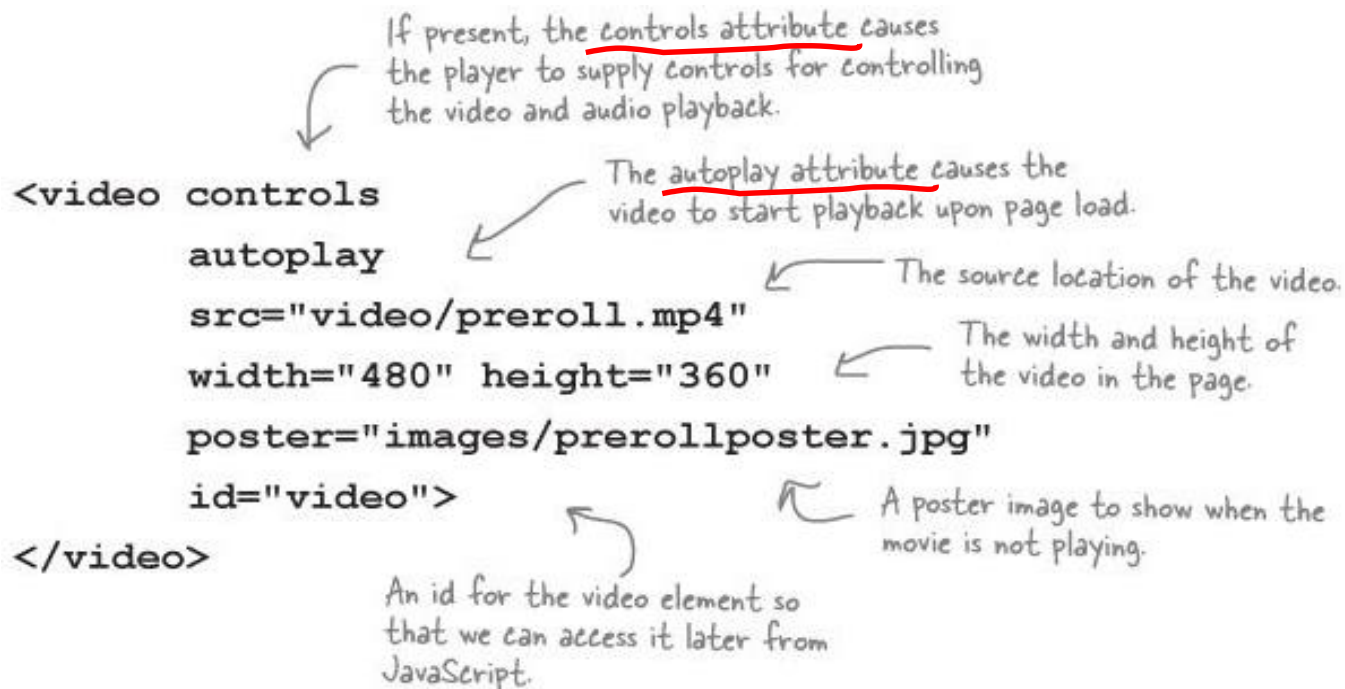
⇒ **.ogv** 포맷 사용 (src= "video/preroll.ogv")

Google Chrome

⇒ **.webm** 포맷 사용 (src="video/preroll.webm")

How does the video element work?

Look at that **video** element we used in our markup:



Closely inspecting the video attributes...

controls

The controls attribute is a **boolean** attribute. It's either there or it's not. If it is there, then the browser will add its built-in controls to the video display. This varies by browser, so check out each browser to see what they look like. Here's what they look like in Safari.

src is what video file is used here.

src

The src attribute is just like the `` element's src—it is a URL that tells the video element where to find the source file. In this case, the source is `video/preroll.mp4`. (If you downloaded the code for this chapter, you'll find this video and two others in the video directory).

preload

The boolean attribute preload is typically used for fine-grained control over how video loads for optimization purposes. Most of the time, the browser chooses how much video to load, based on things like whether autoplay is set and the user's bandwidth. You can override this by setting preload to none (none of the video is downloaded until the user "plays"), metadata (the video metadata is downloaded, but no video content), and auto to let the browser make the decision.

autoplay

The autoplay boolean attribute tells the browser to start playing the video as soon as it has enough data. For the videos we're demoing with, you'll probably see them start to play almost immediately.

poster

The browser will typically display one frame of the video as a "poster" image to represent the video. If you remove the autoplay attribute you'll see this image displayed before you click play. It's up to the browser to pick which frame to show; often, the browser will just show the first frame of the video... which is often black. If you want to show a specific image, then it's up to you to create an image to display, and specify it by using the poster attribute.

loop

Another boolean attribute, loop automatically restarts the video after it finishes playing.

width, height


The width and height attributes set the width and height of the video display area (also known as the "viewport"). If you specify a poster, the poster image will be scaled to the width and height you specify. The video will also be scaled, but will maintain its aspect ratio (e.g., 4:3 or 16:9) so if there's extra room on the sides, or the top and bottom, the video will be letter-boxed or pillar-boxed to fit into the display area size. You should try to match the native dimensions of the video if you want the best performance (so the browser doesn't have to scale in real time).

Pillar-boxing



Letter-boxing





The controls look different on every browser; at least with solutions like Flash I had consistent looking controls.

각 브라우저의 **controls**는 HTML video마다 다르다.

- ✓ Controls의 look and feel은 브라우저를 구현한 벤더에 의해 결정된다
- ✓ 즉 controls는 브라우저와 OS마다 다르게 보이는 경향이 있다

What you need to know about video formats

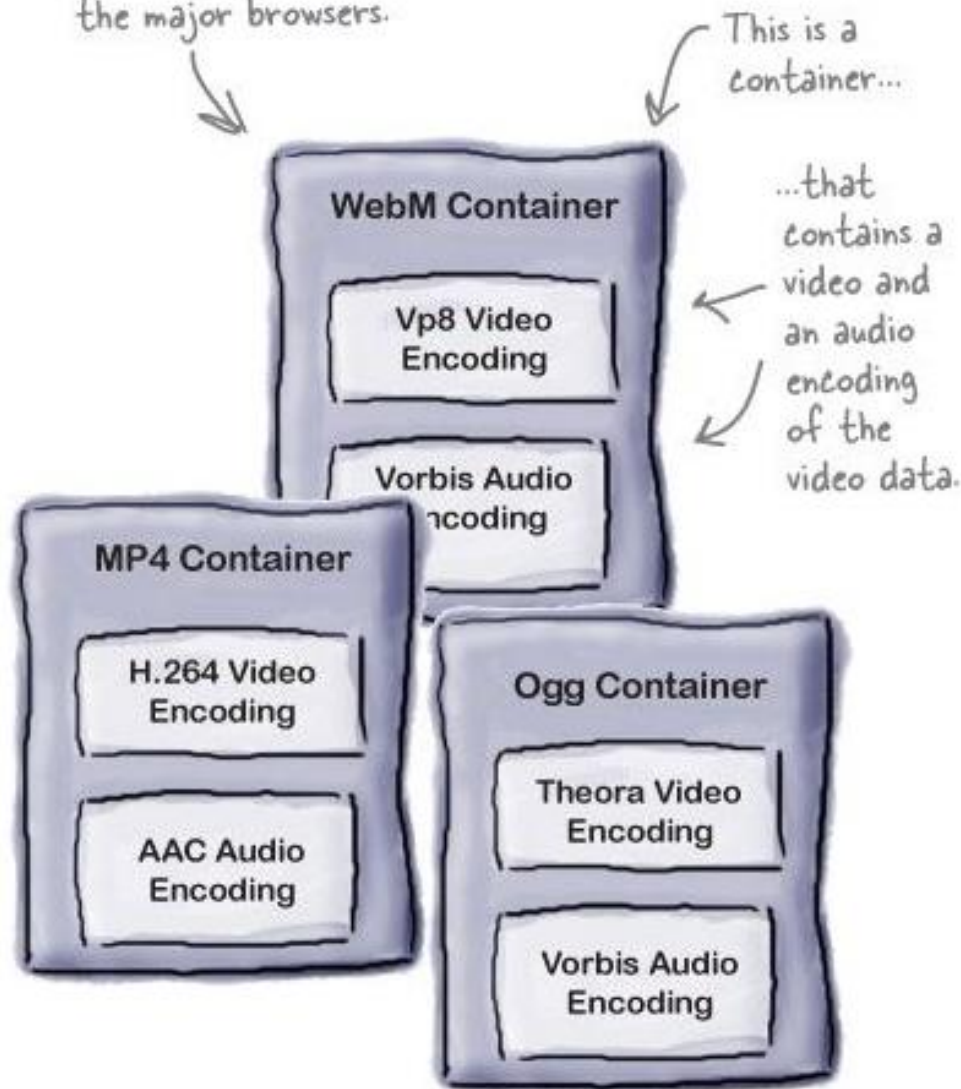
모든 것이 video 엘리먼트와 그 속성들만큼 깔끔하게 정돈되어 있기를 원하지만 사실은 웹 상에서 비디오 포맷은 약간 복잡하게 얽혀있다.

What's a video format?

비디오 파일은 두 파트로 나뉜다: video 파트와 audio 파트

- 각각은 특정 **인코딩 타입**에 의해 인코딩된다.
(사이즈를 줄이고 더 효율적으로 플레이되도록 하기 위해)
- 바로 이 인코딩 부분에서 **의견 일치**를 볼 수 없었다.
(어떤 브라우저 메이커는 H.264를 선호하고, 어떤 경우는 VP8을 선호하고, 또 다른 경우는 오픈 소스인 Theora를 선호한다.)
- **Container**로서 알려진 video/audio 인코딩을 저장하고 있는 파일이 **자신만의 포맷**을 가진다.

Three different video formats in use across the major browsers.



HTML5 스펙: 모든 비디오 포맷 허용

실제로 지원되는 포맷들을 결정하는 것은 **브라우저 구현체**에 달려있다.

각 포맷은 컨테이너 타입(WebM, MP4, Ogg)과 비디오/오디오 인코딩(VP8, Vorbis)으로 구성된다.

The contenders

Today we have **three** main contenders:

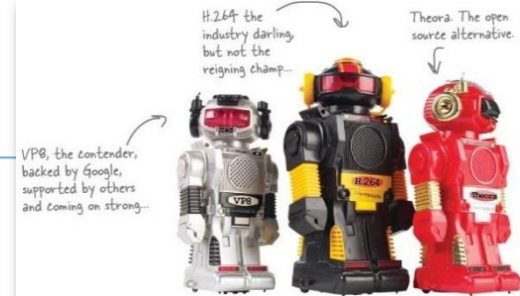
NOTE

MP4 container with H.264 Video and AAC Audio

H.264 is licensed by the MPEG-LA group.

There is more than one kind of H.264; each is known as a “profile.”

MP4/H.264 is supported by Safari and IE9+. You may find support in some versions of Chrome.



NOTE

WebM container with VP8 Video and Vorbis Audio

WebM was designed by Google to work with VP8 encoded videos.

WebM/VP8 is supported by Firefox, Chrome and Opera.

You'll find WebM formatted videos with the .webm extension.

NOTE

Ogg container with Theora Video and Vorbis Audio

Theora is an open source codec.

Video encoded with Theora is usually contained in an Ogg file, with the .ogv file extension.

Ogg/Theora is supported by Firefox, Chrome and Opera.

How to juggle all those formats...

<video> 엘리먼트 안에서 <source> 엘리먼트 사용:

Notice we're removing the src attribute from the <video> tag...



```
<video src="video/preroll.mp4" id="video"
      poster="video/prerollposter.jpg" controls
      width="480" height="360">
```

```
<source src="video/preroll.mp4">
<source src="video/preroll.webm">
<source src="video/preroll.ogv">
```



```
<p>Sorry, your browser doesn't support the video element</p>
```

```
</video>
```

This is what the browser shows if it doesn't support video.

... and adding three source tags each with their own src attribute, each with a version of the video in a different format.



The browser starts at the top and work its way down until it finds a format it can play.



For each source the browser loads the metadata of the video file to see if it can play it (which can be a lengthy process, although we can make it easier on the browser... see the next page).



Bullet Points

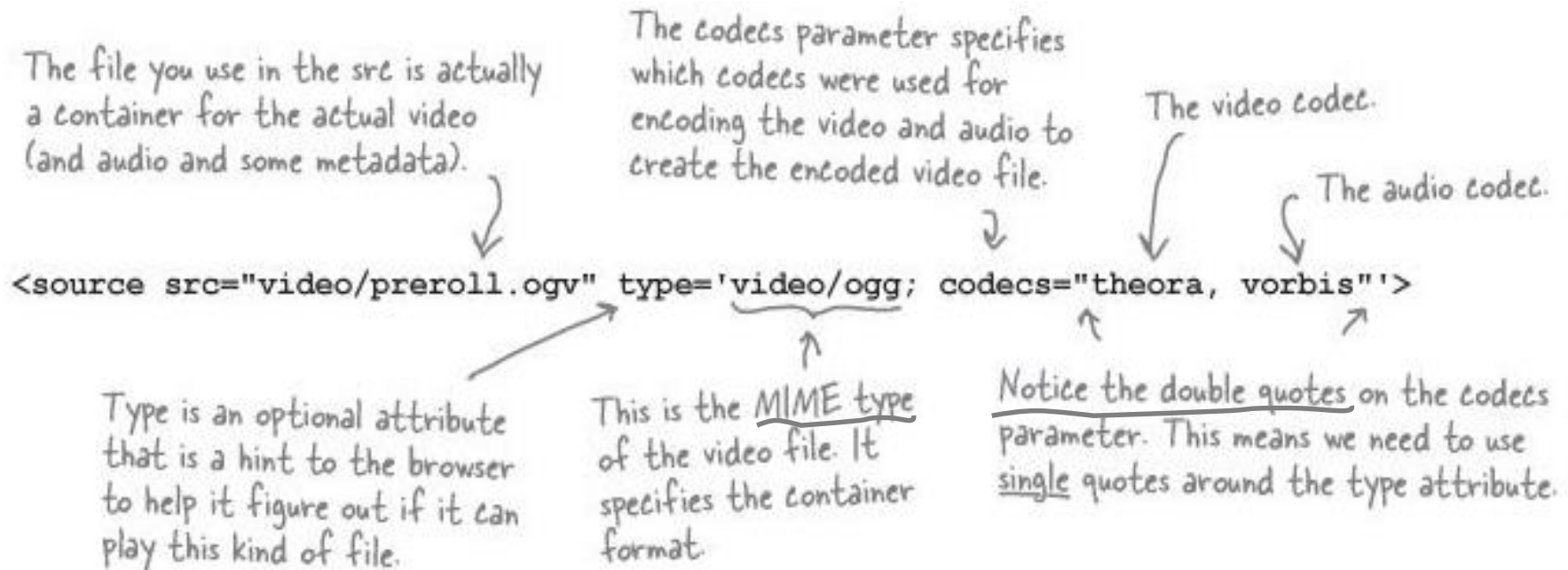
The **container** is the file format that's used to package up the video, audio and metadata information. Common container formats include: MP4, WebM, Ogg and Flash Video.

The **codec** is the software used to encode and decode a specific encoding of video or audio. Popular web codecs include: H.264, VP8, Theora, AAC, and Vorbis.

The browser decides what video it can decode and not all browser makers agree, so if you want to support everyone, you need **multiple encodings**.

How to be even **more specific** with your video formats

비디오 파일의 **타입**과 **코덱**에 관한 정보를 브라우저에게 제공할 수 있다:



<source> 엘리먼트를 이용하여 세 가지 타입 모두 포함시킬 수 있다:

```
<video id="video" poster="video/prerollposter.jpg" controls width="480" height="360">
  <source src="video/preroll.mp4" type='video/mp4; codecs="avc1.42E01E, mp4a.40.2"'>
  <source src="video/preroll.webm" type='video/webm; codecs="vp8, vorbis"'>
  <source src="video/preroll.ogv" type='video/ogg; codecs="theora, vorbis"'>
  <p>Sorry, your browser doesn't support the video element</p>
</video>
```

↗
If you don't know the codecs parameters, then you can leave them off and just use the MIME type. It will be a little less efficient, but most of the time, that's okay.

↖
The codecs for mp4 are more complicated than the other two because h.264 supports various "profiles," different encodings for different uses (like high bandwidth vs. low bandwidth). So, to get those right, you'll need to know more details about how your video was encoded.

I think Flash video is still important and I want to make sure I have a fallback if my users' browsers don't support HTML5 video.



HTML5 video를 지원하지 않는 브라우저를
위해 Flash video도 계속 사용하고 싶은데?

No problem.

선호하는 비디오 플레이어 (HTML5 or Flash or another)가 지원되지 않을 경우 또다른 비디오 플레이어를 이용하도록 해주는 **대비책**이 있다.

```
<video poster="video.jpg" controls>  
  <source src="video.mp4">  
  <source src="video.webm">  
  <source src="video.ogv">  
  <object>...</object>  
</video>
```

↑ Insert your <object> element inside the <video> element. If the browser doesn't know about the <video> element, the <object> will be used.

I was told there would be APIs?

<video> 엘리먼트

모든 종류의 비디오 동작을 구현하는데 사용될 수 있는 풍부한 API 제공
methods, properties and events

Use these Properties



videoWidth	loop
videoHeight	muted
currentTime	paused
duration	readyState
ended	seeking
error	volume

These are **all properties** of the <video> element object.



Call these **Methods**

play ← plays your video
pause ← pauses your video
load ← loads your video
canPlayType ← helps you determine which video types you can play, programmatically



Catch these **Events**

play	abort
pause	waiting
progress	loadeddata
error	loadedmetadata
timeupdate	volumechange
ended	

A little content “programming” on Webville TV

비디오 playlist를 제공하는 스케줄을 프로그래밍할 수 있다.

Webville TV playlist schedule:

1. Show a little preshow to the audience, you know, **the Coke and popcorn ads**, the audience etiquette, and so on...
2. Show our first feature, titled **Are you Popular?** Trust us, you'll enjoy it.
3. And then show our featured presentation, **Destination Earth**, presented in full technicolor. Created by the American Petroleum Institute, what on earth could be their message? Watch and find out.

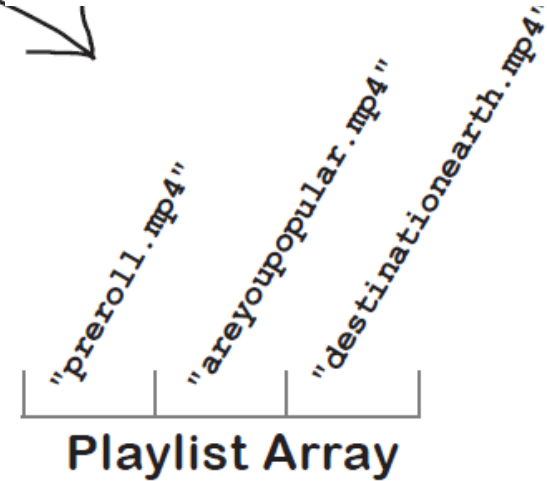


Playlist Pseudo-code

Create array of playlist videos
Get video from DOM
Set event handler on video for "ended" event

Create variable position = 0
Set video source to playlist position 0
Play the video

<video> 엘리먼트에 오로지 한 개의 비디오만을 지정할 수 있기 때문에 약간의 코딩이 필요하다.



Here's our handler to deal with video ending.

Ended Event

Every time a video finishes playing the ended event occurs...

... which calls the ended event handler.

Ended Event Handler Pseudo-code

Increment position by one
Set video to next playlist position
Play the next video

Implementing Webville TV's **playlist**

Webville TV playlist를 구현하기 위해 JavaScript와 video API를 이용한다.

JavaScript 파일 링크 추가 (in webvilletv.html) :

```
<script src="webvilletv.js"></script>
```

기존 **<video>** 엘리먼트로부터 아래 내용을 삭제:

```
<video controls autoplay src="video/preroll.mp4" width="480" height="360"  
  poster="images/prerollposter.jpg" id="video">  
</video>
```

Also remove any `<source>` elements you might have been experimenting with.

새로운 파일 **webvilletv.js**를 생성한다.

```
var position = 0;
var playlist;
var video;

window.onload = function() {
    playlist = ["video/preroll.mp4",
               "video/areyoupopular.mp4",
               "video/destinationearth.mp4"];
    video = document.getElementById("video");
    video.addEventListener("ended", nextVideo, false);

    video.src = playlist[position];
    video.load();
    video.play();
}
```

← First let's define a variable to keep track of which video we're playing; we'll name this position.

← And we need a variable to hold the video playlist array.

← And also a variable to hold a reference to the video element.

← We'll set up our playlist with three videos.

← Grab the video element.

← And add a handler for the video ended event. Yes, this looks different than what we're used to—hold on one sec, we'll talk about this on the next page.

← Now let's set the src for the first video.

← And load the video and play it!

So what's up with that event handler code?

지금까지는 이벤트가 발생할 때 호출되는 핸들러 함수를 항상 다음처럼 할당하였다:

```
video.onended = nextVideo;
```

그러나 이번에는 약간 다르게 수행할 것이다.

You can use the `addEventListener` method to add an event handler.

This is the function we're going to call when the event happens.

```
video.addEventListener("ended", nextVideo, false);
```

This is the object on which we're listening for the event.

This is the event we're listening for. Notice we don't put an "on" before the event name like we do with handlers that we set with properties (like `onload`).

The third parameter controls some advanced methods of getting events if it is set to true. Unless you're writing advanced code you'll always set this to false.

How to write the “end of video” handler

이제 video의 **ended** 이벤트에 대한 핸들러를 작성해 보자:

핸들러는 비디오 플레이어가 **현재 비디오 파일의 끝에 도달할 때마다** 호출될 것이다.

```
function nextVideo() {  
    position++;  
  
    if (position >= playlist.length) {  
        position = 0;  
    }  
  
    video.src = playlist[position];  
    video.load();  
    video.play();  
}
```

First, increment the position in the playlist array.

And if we hit the end of the list, we'll just loop back around by setting the position to zero again.

Now let's set the source of the player to the next video.

And finally, let's load and start the new video playing.

실습과제 13-2: Another test drive...

Can you believe we're ready for a test drive? **All we did was use the API to set up a video to play**, then we made sure we had an event listener ready to handle the situation when the video ends, which it does by starting the next video in the playlist. Make sure you've got the changes made to your HTML file, type in your new JavaScript code and give it a test drive.

Here's what we see, feel free to scrub ahead in the video to see the video change from one to another without watching the whole show.



=> 과제 제출 시 비디오 파일 제외하고 제출!!

file:///E:/practice/wp/hfhtml5/ch8/webvillev_playlist.html

How the **canPlayType** method works

canPlayType 메소드 => 브라우저가 비디오 포맷을 플레이할 수 있는 지를 결정

<source> 태그에서 사용했던 것과 같은 포맷 형태를 취한다:

- ✓ 포맷의 MIME타입만 제공하면 => Empty string, "maybe"를 반환
- ✓ 특정 타입의 코덱도 함께 제공하면 => Empty string, "maybe" or "probably"를 반환

If we pass just the short form of a format then we can only get "" or "maybe" as a result.

`video.canPlayType("video/ogg")`

`video.canPlayType('video/ogg; codecs="theora, vorbis"')`

But if we pass the specific type with a codec, we might then get "", "maybe" or "probably" as an answer.

Empty string if the browser knows it can't play the video.

The string "maybe" if the browser thinks it can possibly play the video.

The string "probably" if the browser is confident it can play the video.

canPlayType 사용

- ⇒ MIME 타입("video/mp4", "video/webm", "video/ogg")만을 사용하여 해당 브라우저에 적절한 **파일 확장자를 반환**해 주는 새로운 함수를 생성한다.
- ⇒ 코덱을 지정하지 않기 때문에 반환되는 값은 empty string 아니면 "maybe" 둘 뿐이다:

```
function getFormatExtension() {  
    if (video.canPlayType("video/mp4") != "") {  
        return ".mp4";  
    } else if (video.canPlayType("video/webm") != "") {  
        return ".webm";  
    } else if (video.canPlayType("video/ogg") != "") {  
        return ".ogv";  
    }  
}
```

↑ For most use cases, if you don't know the codecs, it's good enough to be "maybe" confident.

getFormatExtension 함수 통합

```
window.onload = function() {  
    playlist = ["video/preroll",  
                "video/areyoupopular",  
                "video/destinationearth"];  
    video = document.getElementById("video");  
    video.addEventListener("ended", nextVideo, false);  
    video.src = playlist[position] + getFormatExtension();  
    video.load();  
    video.play();  
}
```

Remove the file extensions.
We're figuring these out
programmatically now.

```
function nextVideo() {  
    position++;  
    if (position >= playlist.length) {  
        position = 0;  
    }  
    video.src = playlist[position] + getFormatExtension();  
    video.load();  
    video.play();  
}
```


실습과제 13-3

canPlayType 함수를 추가하고 앞의 내용을 변경한 다음 webvilletv.html 파일을 다시 로드하시오. 동작하는가? 이제 코드는 최상의 포맷을 알아낼 것이다. 브라우저가 선택한 비디오를 알고 싶다면 **window.onload** 및 **nextVideo** 함수에 **Alert**를 추가하면 된다; video.play() 다음에, 그리고 각 함수의 맨 아래에 추가하시오.

```
alert("Playing " + video.currentSrc);
```

=> 과제 제출 시 비디오 파일 제외하고 제출!!

Q & A

