Lecture 14
Not Your Father's TV: Video ...
With Special Guest Star "Canvas" [2]

Samkeun Kim <skim@hknu.ac.kr>

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

국립한경대학교 HANKYONG NATIONAL UNIV.

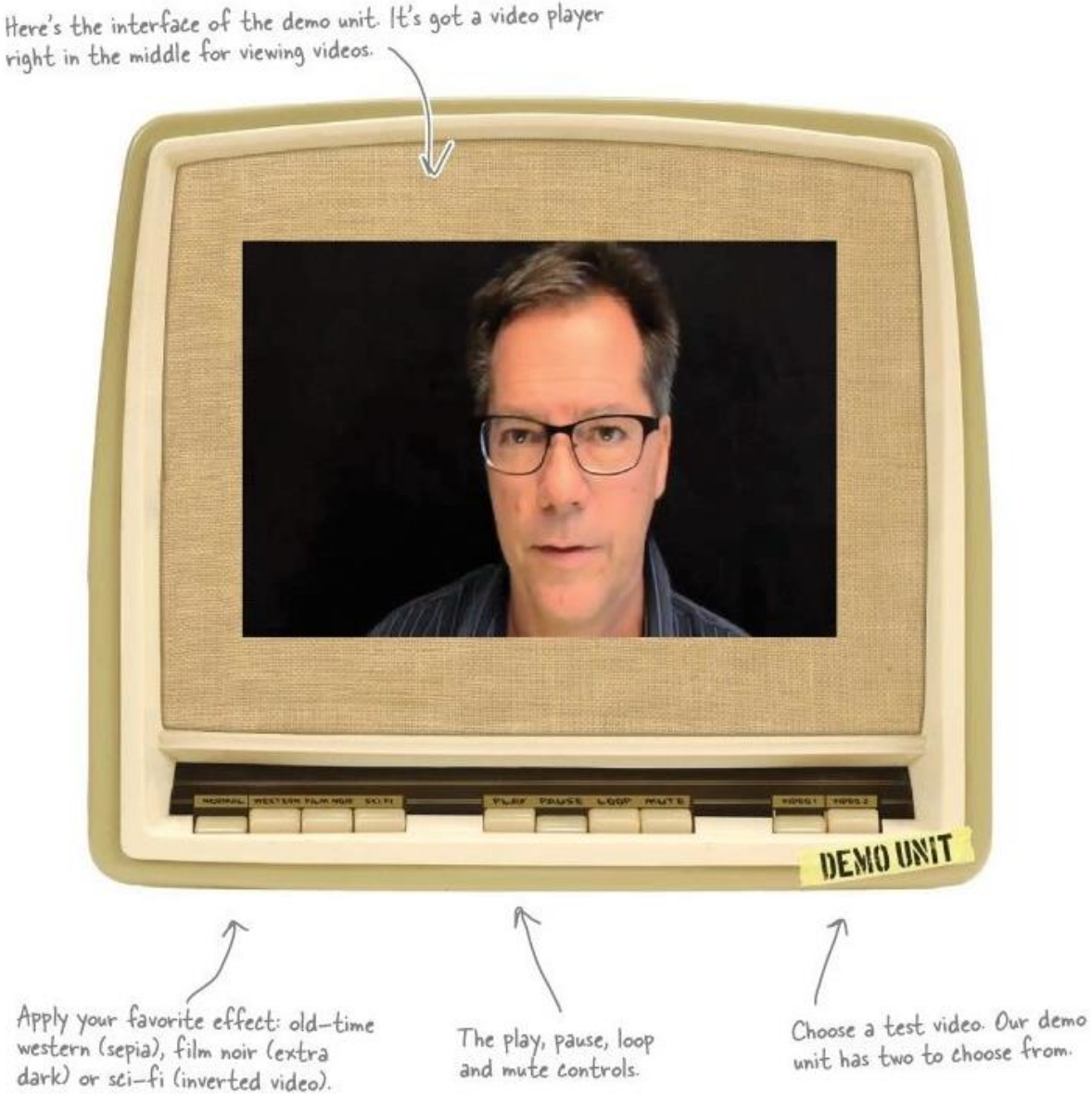
*At some point we have to treat you like a **real** developer.*

We need your help!

This just in... we just got the contract to build the **Starring You Video** software for their new video booth. What on earth is that? Oh, just the latest HTML5-enabled video messaging booth—a customer enters an enclosed video booth and shoots their own video message.

We're going to get a partly functional **demo unit** and a few test video files, and we'll write all our code using those. Then when we're done, the Starring You folks can just point the code to the just-captured real video. And of course, remember that all this has to be done using HTML5.

Step inside the booth, let's take a look...



Demo Unit

```
<!doctype html>
<html lang="en">
<head>
    <title>Starring YOU Video Booth</title>
    <meta charset="utf-8">
    <link rel="stylesheet" href="videobooth.css">
    <script src="videobooth.js"></script>
</head>
<body>
<div id="booth">
    <div id="console">
        <div id="videoDiv">
            <video id="video" width="720" height="480"></video>
        </div>
        <div id="dashboard">
            <div id="effects">
                <a class="effect" id="normal"></a>
                <a class="effect" id="western"></a>
                <a class="effect" id="noir"></a>
                <a class="effect" id="scifi"></a>
            </div>
            <div id="controls">
                <a class="control" id="play"></a>
                <a class="control" id="pause"></a>
                <a class="control" id="loop"></a>
                <a class="control" id="mute"></a>
            </div>
            <div id="videoSelection">
                <a class="videoSelection" id="video1"></a>
                <a class="videoSelection" id="video2"></a>
            </div>
    </div>
```

HTML5, of course.

And all the styling is done for us! Here's the CSS file.

And here's the JavaScript file, we're going to need to write most of this. We'll take an in-depth look, but it looks they've just written the code to control the buttons on the interface so far...

Here's the main interface, we've got the console itself, which looks like it is divided into the video display and a dashboard, with three sets of buttons grouped into "effects", "controls" and "videoSelection".

They've already got a video player installed...good, we're going to need that.

Here are all the effects.

These are all just HTML anchors. We'll see how we tie into these in a sec...

And the player controls.

And the two demo videos to test with.

```
45 a.effect, a.control, a.videoSelection {
46     display: block;
47     width: 64px;
48     height: 56px;
49 }
50
51 div#effects {
52     position: absolute;
53     background: url(images/effectbuttons.png) no-repeat top left;
54     top: 0px;
55     left: 0px;
56     width: 259px;
57     height: 56px;
58 }
59 a#normal {
60     position: relative;
61     width: 68px;
62     top: 0px;
63 }
64 a#western {
65     position: absolute;
66     width: 62px;
67     top: 0px;
68     left: 68px;
69 }
70 a#noir {
71     position: absolute;
72     top: 0px;
73     left: 130px;
74 }
75 a#scifi {
76     position: absolute;
77     top: 0px;
78     left: 194px;
79 }
80
```

```
81 div#controls {
82     position: absolute;
83     background: url(images/playpausemute.png) no-repeat top left;
84     top: 0px;
85     left: 350px;
86     width: 259px;
87     height: 56px;
88 }
89 a#play {
90     position: relative;
91     top: 0px;
92 }
93 a#pause {
94     position: absolute;
95     top: 0px;
96     left: 64px;
97 }
98 a#loop {
99     position: absolute;
100    top: 0px;
101    left: 128px;
102 }
103 a#mute {
104    position: absolute;
105    top: 0px;
106    left: 192px;
107 }
```

```
109 div#videoSelection {
110     position: absolute;
111     background: url(images/video1video2buttons.pn
112     top: 0px;
113     left: 700px;
114     width: 137px;
115     height: 56px;
116 }
117 a#video1 {
118     position: relative;
119     width: 68px;
120     top: 0px;
121 }
122 a#video2 {
123     position: absolute;
124     top: 0px;
125     width: 70px;
126     left: 68px;
127 }
128
129 a:hover {
130     cursor: pointer;
131 }
```

. => class
# => id
(예: a.control => a 태그의 control 클래스를 의미)

http://ksamkeun.dothome.co.kr/wp/hfhtml5/ch8/videobooth.css

Inspecting the rest of the factory code

And now the JavaScript...

```
window.onload = function() {
    var controlLinks = document.querySelectorAll("a.control");
    for (var i = 0; i < controlLinks.length; i++) {
        controlLinks[i].onclick = handleControl;
    }

    var effectLinks = document.querySelectorAll("a.effect");
    for (var i = 0; i < effectLinks.length; i++) {
        effectLinks[i].onclick = setEffect;
    }

    var videoLinks = document.querySelectorAll("a.videoSelection");
    for (var i = 0; i < videoLinks.length; i++) {
        videoLinks[i].onclick = setVideo;
    }

    pushUnpushButtons("video1", []);
    pushUnpushButtons("normal", []);
}
```

Here's the function that is invoked when the page is fully loaded.

Each for statement loops over the elements of one group of buttons.

The onclick handler for each button in the player controls is set to the handleControl handler.

And the handler for effects is set to setEffect.

And finally the handler for video selection is set to setVideo.

Once we've done all the ground work we use a helper function to visually depress the "video1" button, and the "normal" (no filter) button in the interface.

videobooth.js

querySelectorAll

document.querySelectorAll:

CSS selector와 일치하는 엘리먼트를 선택한다는 점을 제외하고는 document.**getElementsByTagName**과 비슷하다.

이 메소드는 CSS selector 인자와 일치하는 엘리먼트 객체의 노드 목록을 반환한다.

```
var elementarray = document.querySelectorAll("selector");
```

Looking at the button handlers

```
<div id="controls">
   <a class="control" id="play"></a>
   <a class="control" id="pause"></a>
   <a class="control" id="loop"></a>
   <a class="control" id="mute"></a>
</div>
```

The **target** is the element you clicked on that generated the event

우선, 핸들러를 호출한 엘리먼트의 id를 검색하여 알아낸다!

```
function handleControl(e) {
    var id = e.target.getAttribute("id");

    if (id == "play") {
        pushUnpushButtons("play", ["pause"]);

    } else if (id == "pause") {
        pushUnpushButtons("pause", ["play"]);

    } else if (id == "loop") {
        if (isButtonPushed("loop")) {
            pushUnpushButtons("", ["loop"]);
        } else {
            pushUnpushButtons("loop", []);
        }
    } else if (id == "mute") {
        if (isButtonPushed("mute")) {
            pushUnpushButtons("", ["mute"]);
        } else {
            pushUnpushButtons("mute", []);
        }
    }
}
```

일단, id를 알아 내었다면 그 엘리먼트가 play, pause, loop, mute인지를 안다!

어떤 버튼이 push되었느냐에 따라 push된 버튼을 반영하기 위해 인터페이스를 변경한다.
예를 들어, pause가 push되었다면 play는 push되지 않아야 한다!

Play, pause: radio buttons

Loop, mute: toggle buttons

setEffect/ setVideo handlers

```
function setEffect(e) {
    var id = e.target.getAttribute("id");

    if (id == "normal") {
        pushUnpushButtons("normal", ["western", "noir", "scifi"]);

    } else if (id == "western") {
        pushUnpushButtons("western", ["normal", "noir", "scifi"]);

    } else if (id == "noir") {
        pushUnpushButtons("noir", ["normal", "western", "scifi"]);

    } else if (id == "scifi") {
        pushUnpushButtons("scifi", ["normal", "western", "noir"]);

    }
}
```

We'll be adding code to each case to handle implementing the appropriate special effect filter.

```
function setVideo(e) {
    var id = e.target.getAttribute("id");
    if (id == "video1") {
        pushUnpushButtons("video1", ["video2"]);
    } else if (id == "video2") {
        pushUnpushButtons("video2", ["video1"]);
    }

}
```

Helper Functions

```
function pushUnpushButtons(idToPush, idArrayToUnpush) {
    if (idToPush != "") {
        var anchor = document.getElementById(idToPush);
        var theClass = anchor.getAttribute("class");
        if (!theClass.indexOf("selected") >= 0) {
            theClass = theClass + " selected";
            anchor.setAttribute("class", theClass);
            var newImage = "url(images/" + idToPush + "pressed.png)";
            anchor.style.backgroundImage = newImage;
        }
    }

    for (var i = 0; i < idArrayToUnpush.length; i++) {
        anchor = document.getElementById(idArrayToUnpush[i]);
        theClass = anchor.getAttribute("class");
        if (theClass.indexOf("selected") >= 0) {
            theClass = theClass.replace("selected", "");
            anchor.setAttribute("class", theClass);
            anchor.style.backgroundImage = "";
        }
    }
}
```

isButtonPushed simply checks to see if a button is pushed. It takes the id of an anchor...

```
function isButtonPushed(id) {
    var anchor = document.getElementById(id);
    var theClass = anchor.getAttribute("class");
    return (theClass.indexOf("selected") >= 0);
}
```

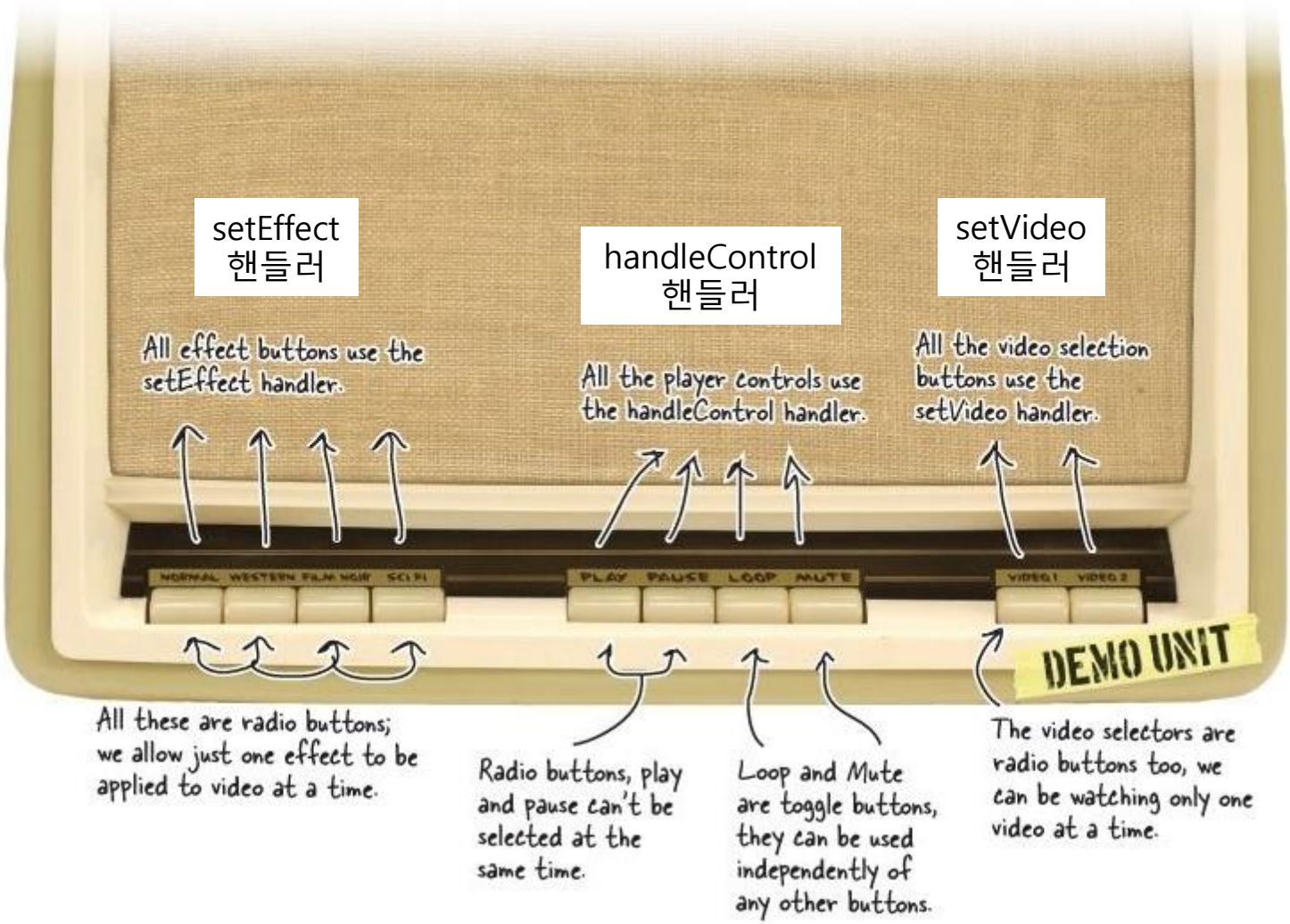
... grabs the anchor...

... gets the class of that anchor...

... and returns true if the anchor has the "selected" class.

Sharpen your pencil

아래 버튼들이 Toggle 버튼인지 아니면 Radio 버튼인지 구분해보라.



anchor 태그를 가진 들
로부터 버튼들을 어떻게
가져왔는지 . . . ?

That would be the power of CSS.

We set the background image of the console to the booth console (no buttons).

The <video> element is in a which is positioned relative to the console. Then, the <video> element is absolute positioned so it sits in the middle of the console.

We position the dashboard relative to the console and then position the s for each group of buttons relative to the dashboard.

Each button group gets a background image for all the unpushed buttons.

Each "button" anchor is positioned within the for the group, and given a width and height to match the button it corresponds to. When you click on a "button", we give that anchor a background image of a pushed in button to cover up the unpushed button.

Getting our demo videos ready...

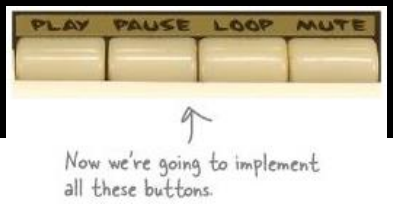
```
var videos = {video1: "video/demovideo1", video2: "video/demovideo2"};

window.onload = function() {

    var video = document.getElementById("video");
    video.src = videos.video1 + getFormatExtension();
    video.load();

    var controlLinks = document.querySelectorAll("a.control");
    for (var i = 0; i < controlLinks.length; i++) {
        controlLinks[i].onclick = handleControl;
    }
    var effectLinks = document.querySelectorAll("a.effect");
    for (var i = 0; i < effectLinks.length; i++) {
        effectLinks[i].onclick = setEffect;
    }
    var videoLinks = document.querySelectorAll("a.videoSelection");
    for (var i = 0; i < videoLinks.length; i++) {
        videoLinks[i].onclick = setVideo;
    }
    pushUnpushButtons("video1", []);
    pushUnpushButtons("normal", []);
}
```

Here, we're getting the video element, and setting its source to the first video in the array with a playable extension.

Then we go ahead and load the video so if the user clicks play, it's ready to go.

2개의 데모 비디오로 테스트해 보자!

Implementing the video controls



Buttons handler

```
function handleControl(e) {
    var id = e.target.getAttribute("id");
    var video = document.getElementById("video");

    if (id == "play") {
        pushUnpushButtons("play", ["pause"]);
        if (video.ended) {
            video.load();
        }
        video.play();
    } else if (id == "pause") {
        pushUnpushButtons("pause", ["play"]);

    } else if (id == "loop") {
        if (isButtonPushed("loop")) {
            pushUnpushButtons("", ["loop"]);
        } else {
            pushUnpushButtons("loop", []);
        }
    } else if (id == "mute") {
        if (isButtonPushed("mute")) {
            pushUnpushButtons("", ["mute"]);
        } else {
            pushUnpushButtons("mute", []);
        }
    }
}
```

We need a reference to the video object.

This should be pretty simple. If the user has pressed play, then call the play method on the video object.

But we'll warn you, there's one edge case here about to bite us, so let's go ahead and take care of it: If we've played a video, and let that video play through to the end, then to start it playing again, we have to load it again first. We check to make sure the video ran through to the end (and wasn't just paused), because we only want to load again in that case. If it's paused, we can just play without loading.

Implementing the rest of the video controls

```
function handleControl(e) {
    var id = e.target.getAttribute("id");
    var video = document.getElementById("video");

    if (id == "play") {
        pushUnpushButtons("play", ["pause"]);
        video.load();
        video.play();
    } else if (id == "pause") {
        pushUnpushButtons("pause", ["play"]);
        video.pause();

    } else if (id == "loop") {
        if (isButtonPushed("loop")) {
            pushUnpushButtons("", ["loop"]);
        } else {
            pushUnpushButtons("loop", []);
        }
        video.loop = !video.loop;

    } else if (id == "mute") {
        if (isButtonPushed("mute")) {
            pushUnpushButtons("", ["mute"]);
        } else {
            pushUnpushButtons("mute", []);
        }
        video.muted = !video.muted;

    }
}
```

If the user pauses the video, then use the video object's pause method.

For looping we've got a boolean property named loop in the video object. We just set it appropriately...

...and to do that we'll keep you on your toes by using the boolean "!" (not) operator, which just flips the boolean value for us.

And mute works the same way: we just flip the current value of the mute property when the button is pressed.

실습과제 14-1 Another test drive!

Make sure you've got all the code changes typed in. Load **videobooth.html** into your browser and give your control buttons a test. You should see video start playing, be able to pause it, mute it, or even put it in a loop. Of course, you can't select the other demo video yet or add an effect, but we're getting there!

http://ksamkeun.dothome.co.kr/wp/hfhtml5/ch8/videobooth.html

Taking care of a loose end...

비디오가 play되고 있고 loop가 선택되지 않은 상태에서 비디오 play가 종료되고 나면 Play 버튼은 눌러진 상태로 남아있게 된다.

Play 버튼이 누르기 전 상태로 되돌아 와 있는게 좋지 않을까?

Ended 이벤트에 리스너를 추가함으로써 간단하게 해결할 수 있다.
window.onload 핸들러 bottom 부분에 아래 코드를 추가하자:

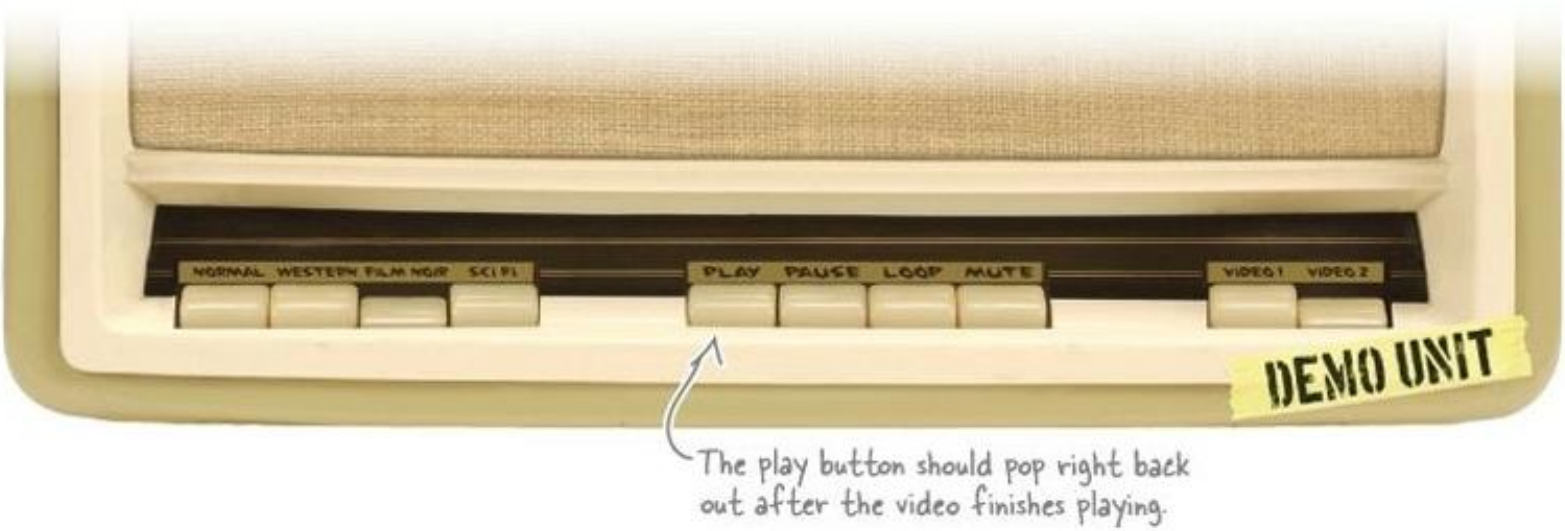
```
video.addEventListener("ended", endedHandler, false);
```

이제 비디오가 종료될 때마다 호출되는 핸들러를 작성해 보자:

```
function endedHandler() {
    pushUnpushButtons("", ["play"]);
}
```

And another...

Okay, **make the changes**, save the code and reload.
Now start a video and let it play to its conclusion without the loop button pressed, and at the end you should see the play button pop back out on its own.



Switching test videos


```
var videos = {video1: "video/demovideo1", video2: "video/demovideo2"};

function setVideo(e) {
    var id = e.target.getAttribute("id");
    var video = document.getElementById("video");

    if (id == "video1") {
        pushUnpushButtons("video1", ["video2"]);
    } else if (id == "video2") {
        pushUnpushButtons("video2", ["video1"]);
    }
    video.src = videos[id] + getFormatExtension();
    video.load();
    video.play();

    pushUnpushButtons("play", ["pause"]);
}
```

And here's the handler again.

Again, we need a reference to the video object.

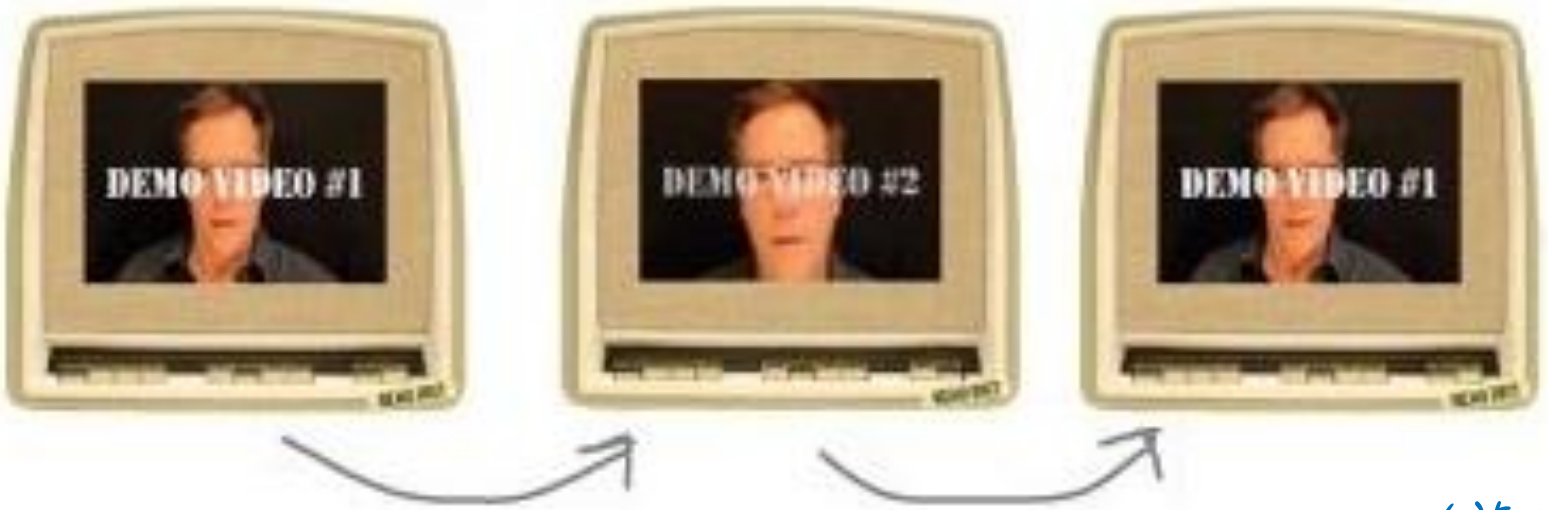
Then we still update the buttons in the same way we were, no changes there.

Then we use the source id of the button (the id attribute of the anchor) to grab the correct video filename, and add on our browser-aware extension. Notice we're using the [ ] notation with our videos object, using the id string as the property name.

Once we have the correct video path and filename, we load and play the video.

실습과제 14-2

Make these changes to your setVideo function and then load your page again. You should now be able to easily switch between video sources.



자신의 dothome URL을 설명문서에 기술할 것!!

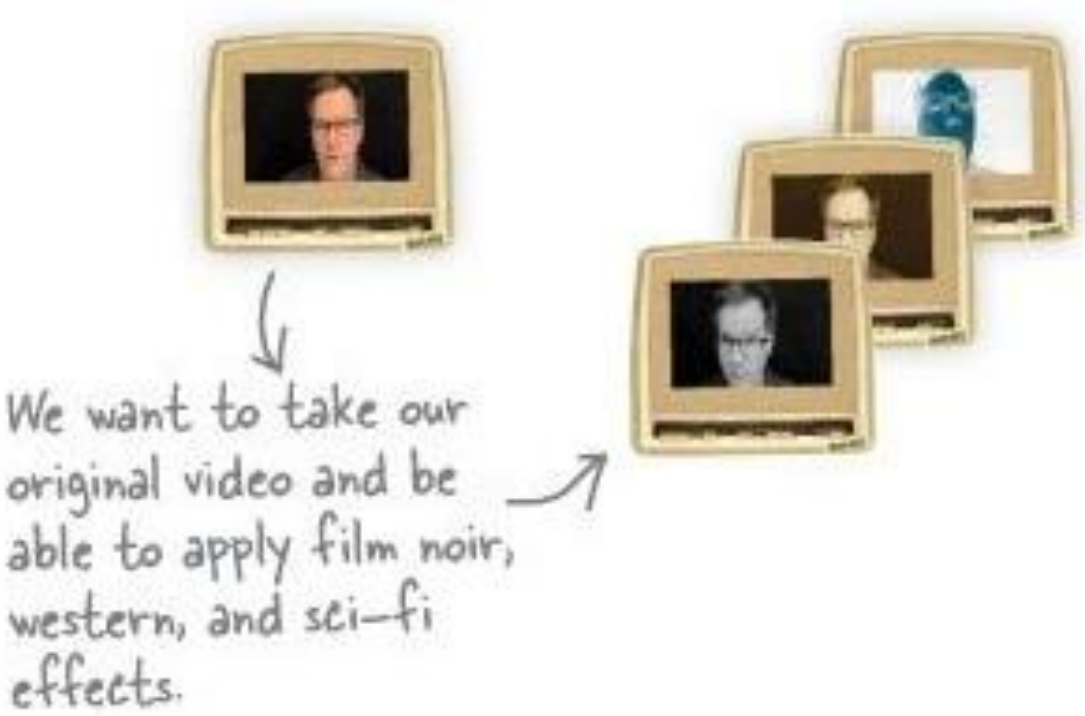
http://ksamkeun.dothome.co.kr/wp/hfhtml5/ch8/videobooth2.html

It’s time for special effects

무비 이펙트 추가:

이제 원본 비디오에 다양한 효과를 적용시켜 보자: **film noir**, **western**, **sci-fi** effect.
그러나 **Video API**를 아무리 살펴봐도 어떠한 이펙트 관련 메소드를 찾아 볼 수 없다.

So how are we going to add those effects?

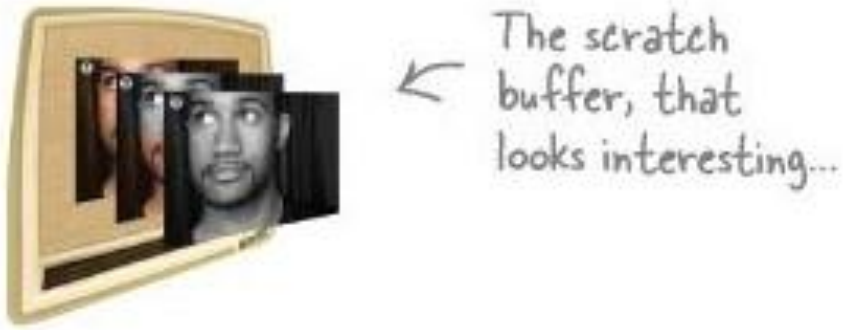


FX plan

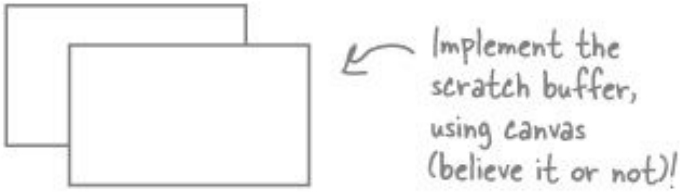
High level plan of attack:

1. 먼저 이펙트를 제어하는 버튼들을 연결한다.

2. 비디오 프로세싱에 관한 약간의 지식을 습득하고 이펙트를 추가하기 위한 "**scratch buffer**" 기법을 살펴보자.



3. 다음은 **scratch buffer**를 구현한다 => **video**와 **canvas**를 함께 구동되도록 해준다.



4. 각 이펙트에 대한 함수 구현: **western**, **film noir**, **sci-fi**

```
function noir(pos, r, g, b, data) {
  ...
}
```

5. 모두 함께 묶어 테스트한다.

Time to get those effects buttons working

Holds a function that knows how to process the video data

effectFunction: global variable

```
var effectFunction = null;

function setEffect(e) {
    var id = e.target.getAttribute("id");

    if (id == "normal") {
        pushUnpushButtons("normal", ["western", "noir", "scifi"]);
        effectFunction = null;
    } else if (id == "western") {
        pushUnpushButtons("western", ["normal", "noir", "scifi"]);
        effectFunction = western;
    } else if (id == "noir") {
        pushUnpushButtons("noir", ["normal", "western", "scifi"]);
        effectFunction = noir;
    } else if (id == "scifi") {
        pushUnpushButtons("scifi", ["normal", "western", "noir"]);
        effectFunction = scifi;
    }
}
```

Here's our setEffect handler again. Remember this is called whenever the user clicks on a effect button.

For each button press we set the effectFunction variable to the appropriate function (all of which we still need to write).

If the effect is no effect, or normal, we just use null as the value.

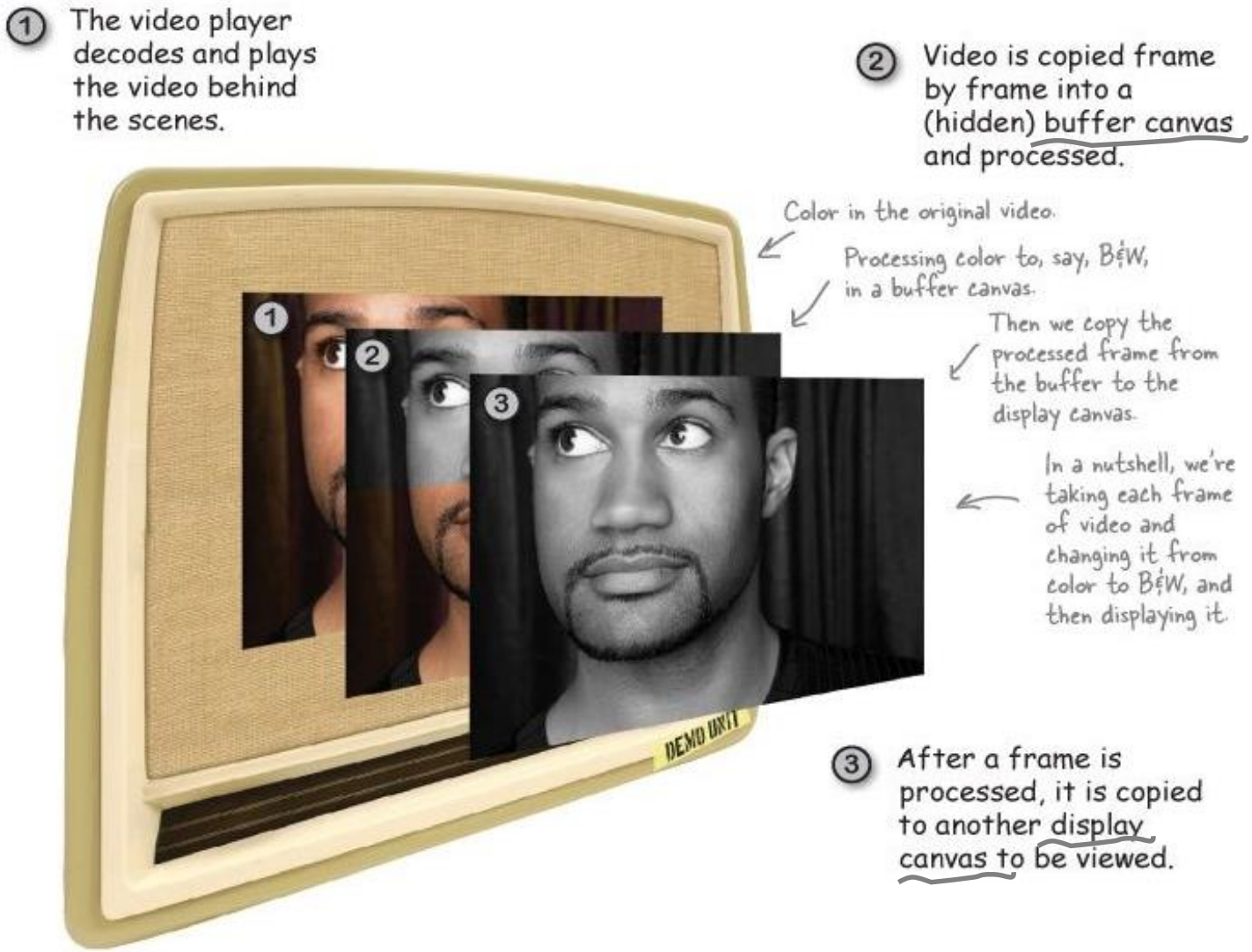
Otherwise we set effectFunction to an appropriately named function that will do the work of applying the effect.

We still need to write these effects functions. So, let's see how we process video so we can apply effects to it!

videobooth.js

How video processing works

video와 **canvas**를 함께 동작시키는 방법:



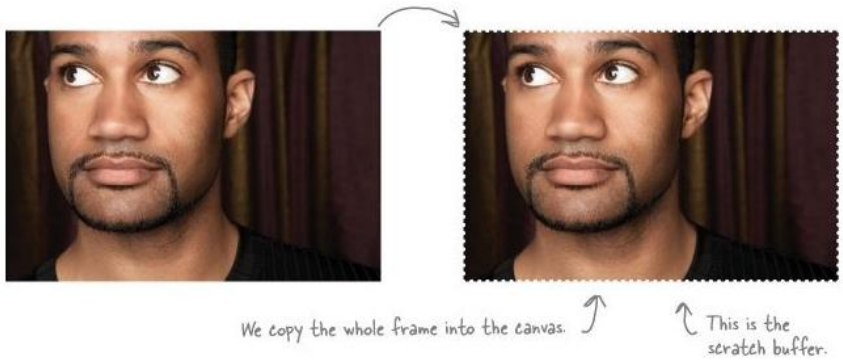
How to process video using a scratch buffer

비디오를 처리하고 보여주기 위해 왜 2개의 캔바스를 사용할까?

"**scratch buffer**" 기법 사용:
버퍼에 있는 비디오의 한 프레임을 처리한 후에 디스플레이 캔버스에 카피!!

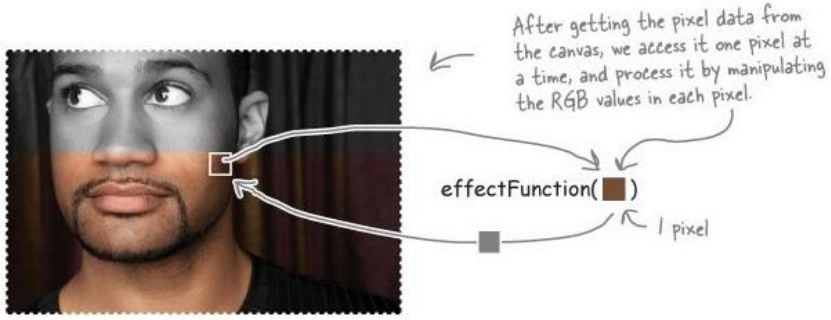
1. 브라우저가 비디오를 일련의 프레임으로 분해한다. 각 프레임은 특정 시간에서의 비디오의 스냅샷이다.

2. 각 프레임이 분해되면서 scratch buffer로서 동작하는 캔버스에 카피된다.

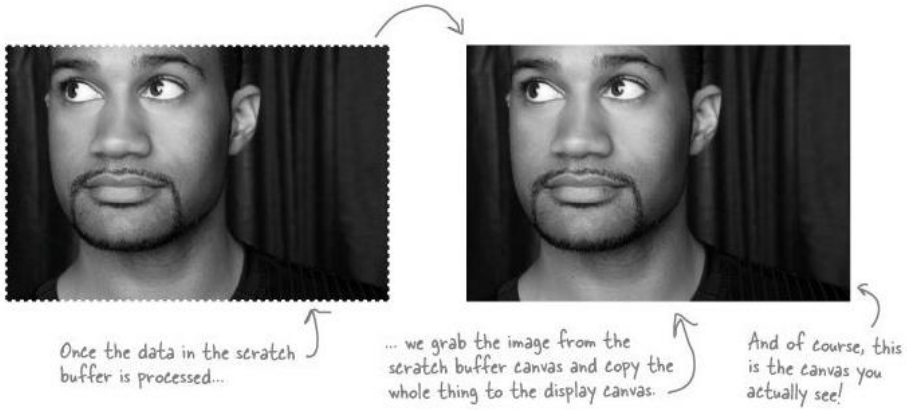


How to process video using a scratch buffer

3. Scratch buffer에서 비디오 처리를 위해 각 픽셀을 앞의 이펙트 함수 (effectFunction)에 픽셀 단위로 통과시킨다.



4. Scratch buffer의 모든 픽셀이 처리된 후에 그들을 scratch buffer 캔버스에서 display 캔버스로 카피한다.



5. 모든 프레임에 대해 반복한다.

Implementing a scratch buffer with Canvas

Scratch buffer 기법을 구현하기 위해서는 2개의 캔버스가 필요하다:

계산용과 **디스플레이용**

HTML 파일 **videobooth.html**: 이 파일을 열고 "videoDiv" ID로 를 찾아서 <video> 아래에 2개의 캔버스 엘리먼트를 추가한다.

```
<div id="videoDiv">
    <video id="video" width="720" height="480"></video>
    <canvas id="buffer" width="720" height="480"></canvas>
    <canvas id="display" width="720" height="480"></canvas>
</div>
```

We're adding two canvas elements, one for the buffer and one to display.

Notice they're exactly the same size as the video element.

How to position the video and canvases

```
div#videoDiv {
    position: relative;
    width: 720px;
    height: 480px;
    top: 180px;
    left: 190px;
}
video {
    background-color: black;
}
div#videoDiv canvas {
    position: absolute;
    top: 0px;
    left: 0px;
}
```

The videoDiv is positioned relative to the element it's in (the console), at 180px from the top and 190px from the left, which places it in the center of the console.

We set the width and height equal to the width and height of the <video> and the two <canvas> elements.

The <video> is the first element in the videoDiv so it's automatically positioned at the top left of the . We set the background to black so that if we have letter-boxing or pillar-boxing, the space is black.

The two <canvas> elements in the videoDiv are positioned absolutely with respect to the videoDiv (their parent), so by placing the <canvas> elements at 0px from the top, and 0px from the left, they are in exactly the same position as the <video> and the videoDiv.

videobooth.css

Writing the code to process the video

window.onload 핸들러 끝에 아래 코드 부분을 추가한다(**videobooth2.js**):

```
video.addEventListener("play", processFrame, false);
```

When the video begins playing it will call the function processFrame.

processFrame 함수 => 비디오 픽셀을 처리하여 디스플레이용 캔버스에 둔다:

```
function processFrame() {
    var video = document.getElementById("video");
    if (video.paused || video.ended) {
        return;
    }
    var bufferCanvas = document.getElementById("buffer");
    var displayCanvas = document.getElementById("display");
    var buffer = bufferCanvas.getContext("2d");
    var display = displayCanvas.getContext("2d");
}
```

First grab the video object...

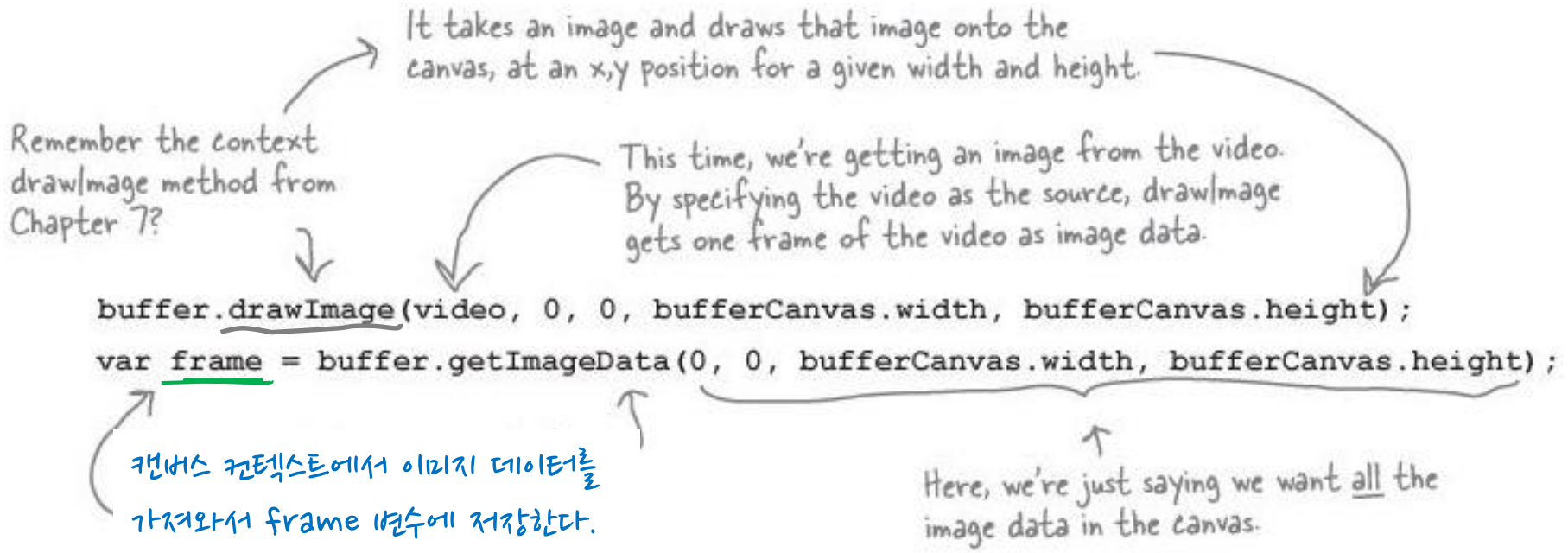
... and check to see if the video is still playing. If it isn't then we've got no work to do, just return.

Then grab a reference to both canvas elements and also to their contexts, we're going to need those.

How to create the buffer

버퍼를 생성하기 위해 현재의 비디오 프레임을 취하여 buffer 캔버스에 복사할 필요가 있다.
일단 캔버스에 프레임을 복사했다면 프레임 안에서 데이터를 처리할 수 있다.

⇒ **processFrame** 함수 아래 부분에 아래 코드 부분을 추가하자:



사실, 각 픽셀은 4개의 값을 갖는다: R, G, B, Alpha(투명도, 여기서는 사용하지 않음)

```
buffer.drawImage(video, 0, 0, bufferCanvas.width, displayCanvas.height);
var frame = buffer.getImageData(0, 0, bufferCanvas.width, displayCanvas.height);

var length = frame.data.length / 4;

for (var i = 0; i < length; i++) {
    var r = frame.data[i * 4 + 0];
    var g = frame.data[i * 4 + 1];
    var b = frame.data[i * 4 + 2];
    if (effectFunction) {
       effectFunction(i, r, g, b, frame.data);
    }
}
display.putImageData(frame, 0, 0);
```

First, we find out the length of the frame data. Notice that the data is in a property of frame, frame.data, and length is a property of frame.data. The length is actually four times longer than the size of the canvas because each pixel has four values: RGBA.

Now we loop over the data and get the RGB values for each pixel. Each pixel takes up four spaces in the array, so we grab r from the first position, g from the second, and b from the third.

Then, we call the effectFunction (if it's not null, which it will be if the "Normal" button is pressed), passing in the position of the pixel, the RGB values, and the frame.data array. The effect function will update the frame.data array with new pixel values, processed according to the filter function assigned to effectFunction.

At this point the frame data has been processed, so we use the context putImageData method to put the data into the display canvas. This method takes the data in frame and writes it into the canvas at the specified x, y position.

We've processed one frame, what next?

방금 처리한 것은 단 하나의 프레임일 뿐이다.

모든 프레임을 처리할 필요가 있다.

여기서 다시 **setTimeout**을 사용한다.

processFrame 함수 끝부분에 다음을 추가하자:

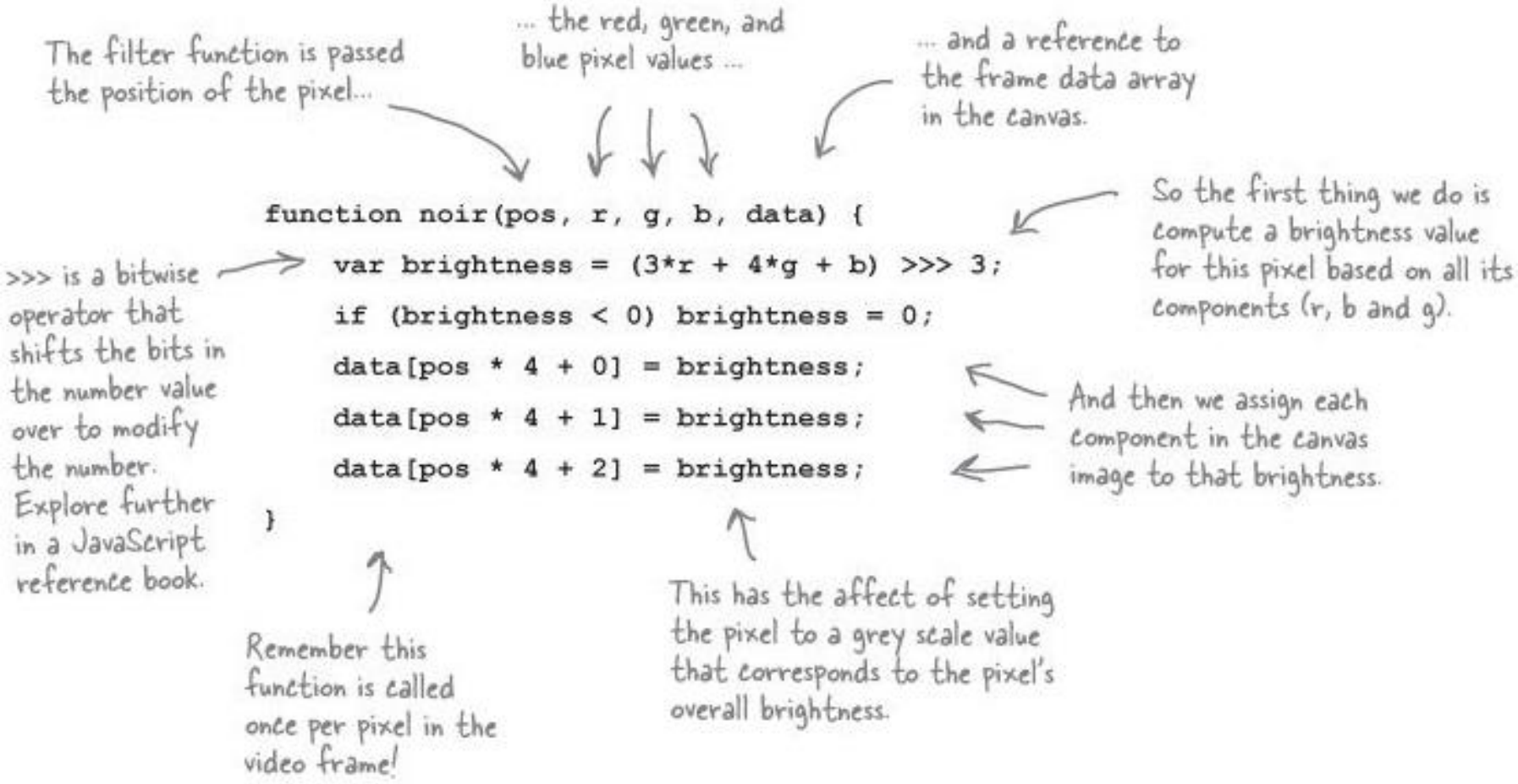
```
setTimeout(processFrame, 0);
```

Tells JavaScript to run processFrame again as soon as possible!

Now we need to write some effects

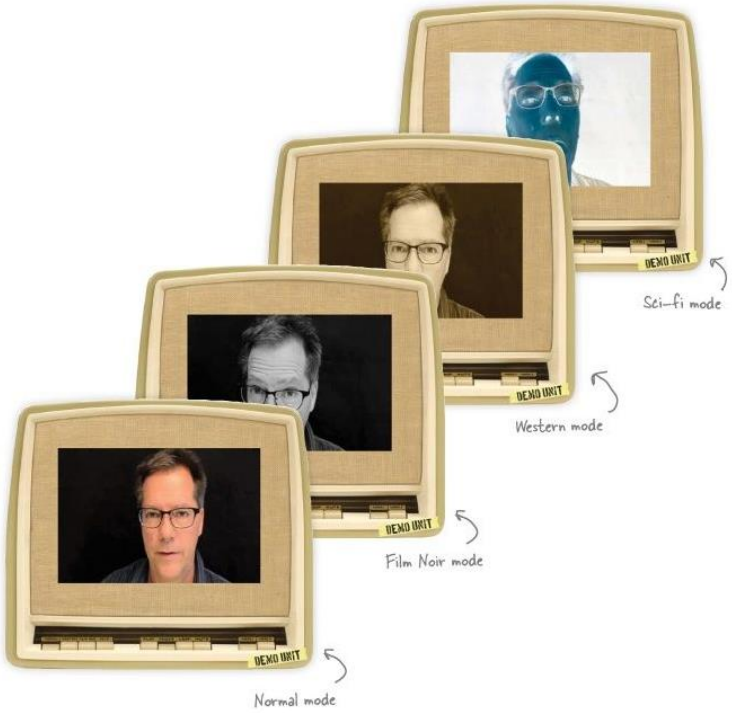
마지막으로 비디오 이펙트를 작성한다:

Film Noir 필터를 살펴보자(a fancy name for black and white):



실습과제 14-3 The Big Test Drive

This is it! We have this code wrapped up and ready to ship off to **Starring You Video**. Go ahead and double check that you've got all the code typed in, save, and load **videobooth3.html**. Then have fun playing around with your new app!




```
function western(pos, r, g, b, data) {
    var brightness = (3*r + 4*g + b) >>> 3;
    data[pos * 4 + 0] = brightness+40;
    data[pos * 4 + 1] = brightness+20;
    data[pos * 4 + 2] = brightness-20;
}

function scifi(pos, r, g, b, data) {
    var offset =  pos * 4;
    data[offset] = Math.round(255 - r) ;
    data[offset+1] = Math.round(255 - g) ;
    data[offset+2] = Math.round(255 - b) ;
}
```

자신의 dothome URL을 설명문서에 기술할 것!!

실습과제 14-4

Starring You Video 에 흑백 카툰 effectFunction을 추가하시오.
해당하는 모든 버튼 및 기능들도 구현하시오.

```
function bwcartoon(pos, r, g, b, outputData) {
    var offset =  pos * 4;
    if( outputData[offset] < 120 ) {
        outputData[offset] = 80;
        outputData[++offset] = 80;
        outputData[++offset] = 80;
    } else {
        outputData[offset] = 255;
        outputData[++offset] = 255;
        outputData[++offset] = 255;
    }
    outputData[++offset] = 255;
    ++offset;
}
```

자신의 dothome URL을 설명문서에 기술할 것!!

Q & A