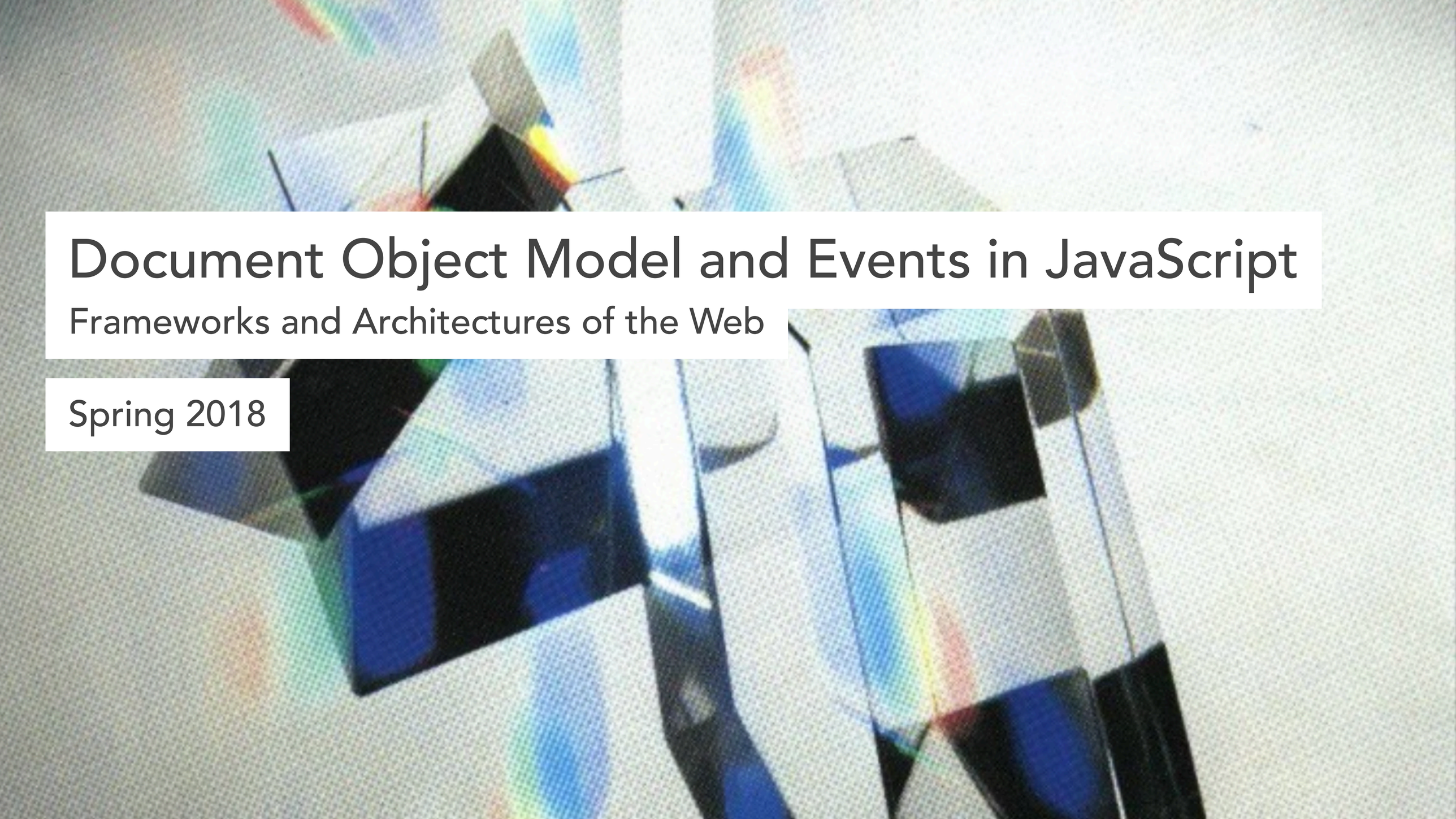


Document Object Model and Events in JavaScript

Frameworks and Architectures of the Web

Spring 2018

Today's Program

The Document Object Model

Events in JavaScript

Exercises

Opening
Phillip Glass
Glassworks

Course Outline

Foundation

Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11	Week 12
Introduction, HTML Syntax, Structure and Semantics	Interface Principles, Design Patterns and Aesthetics	HTML and CSS Preliminaries	CSS Layout & Positioning	Mobile and Responsive Design, Forms and Data Validation	Introduction to JavaScript	Document Object Model and Events in JavaScript	CSS3, Graphics and Media and Advanced JavaScript

Web Project - Design, Wireframes and Interactive Prototype

Web Project - Impleme

Source: https://dev.to/reverentgeek/do-you-hate-javascript

Understanding the DOM

Understanding ancestors, parents and children.

CodePen: https://codepen.io/timwray87/pen/eMpQVm

```
<html lang="en">
<head>
   <title>A Simple HTML Document</title>
   …
</head>
<body>
   <div id="main">
      <p>This is my HTML document.</p>
      <p>There are many others like this.</p>
      <p>But this one is mine.</p>
      <ul>
         <li class="anecdote">There are no right or wrong answers.</li>
         <li class="anecdote">Just true and false.</li>
      </ul>
   </div>
</body>
</html>
```

Selecting an Element by ID

```
var el = document.getElementById('main'); // returns DOMElement
```

```
<html lang="en">
<head>
   <title>A Simple HTML Document</title>
   …
</head>
<body>
   <div id="main">
      <p>This is my HTML document.</p>
      <p>There are many others like this.</p>
      <p>But this one is mine.</p>
      <ul>
         <li class="anecdote">There are no right or wrong answers.</li>
         <li class="anecdote">Just true and false.</li>
      </ul>
   </div>
</body>
</html>
```

Selecting an Element by Tag Name

```
var el = document.getElementsByTagName('p'); // returns array
```

```
<html lang="en">
<head>
   <title>A Simple HTML Document</title>
   …
</head>
<body>
   <div id="main">
      <p>This is my HTML document.</p>
      <p>There are many others like this.</p>
      <p>But this one is mine.</p>
      <ul>
         <li class="anecdote">There are no right or wrong answers.</li>
         <li class="anecdote">Just true and false.</li>
      </ul>
   </div>
</body>
</html>
```

Selecting an Element by Class

```
var el = document.getElementsByClassName('anecdote'); // returns array
```

```
<html lang="en">
<head>
   <title>A Simple HTML Document</title>
   …
</head>
<body>
   <div id="main">
      <p>This is my HTML document.</p>
      <p>There are many others like this.</p>
      <p>But this one is mine.</p>
      <ul>
         <li class="anecdote">There are no right or wrong answers.</li>
         <li class="anecdote">Just true and false.</li>
      </ul>
   </div>
</body>
</html>
```

document.querySelector()

Selects a method using a CSS query, returns a DOMElement

document.querySelectorAll()

Selects a method using a CSS query, returns an array

Selecting an Element using document.querySelector

```
var el = document.querySelector('p'); // returns DOMElement
```

```
<html lang="en">
<head>
   <title>A Simple HTML Document</title>
   …
</head>
<body>
   <div id="main">
      <p>This is my HTML document.</p>
      <p>There are many others like this.</p>
      <p>But this one is mine.</p>
      <ul>
         <li class="anecdote">There are no right or wrong answers.</li>
         <li class="anecdote">Just true and false.</li>
      </ul>
   </div>
</body>
</html>
```

Selecting Elements using document.querySelectorAll

```
var el = document.querySelectorAll('p'); // returns array
```

```
<html lang="en">
<head>
   <title>A Simple HTML Document</title>
   …
</head>
<body>
   <div id="main">
      <p>This is my HTML document.</p>
      <p>There are many others like this.</p>
      <p>But this one is mine.</p>
      <ul>
         <li class="anecdote">There are no right or wrong answers.</li>
         <li class="anecdote">Just true and false.</li>
      </ul>
   </div>
</body>
</html>
```

A List of Plays

CodePen: https://codepen.io/timwray87/pen/ZxbmZQ

```
<h2>Selected Shakespeare Plays</h2>
<ul id="selected-plays">
  <li>Comedies
    <ul>
      <li><a href="/asyoulikeit/">As You Like It</a></li>
      <li>All's Well That Ends Well</li>
      <li>A Midsummer Night's Dream</li>
      <li>Twelfth Night</li>
    </ul>
  </li>
  <li>Tragedies
    <ul>
      <li><a href="hamlet.pdf">Hamlet</a></li>
      <li>Macbeth</li>
      <li>Romeo and Juliet</li>
    </ul>
  </li>
  <li>Histories
    <ul>
      <li>Henry IV (<a href="mailto:henryiv@king.co.uk">email</a>)
        <ul>
            <li>Part I</li>
            <li>Part II</li>
        </ul>
      </li>
      <li>
        <a href="http://www.shakespeare.co.uk/henryv.htm">Henry V</a>
      </li>
      <li>Richard II</li>
    </ul>
  </li>
</ul>
```

Selected Shakespeare Plays

- Comedies
  - As You Like It
  - All's Well That Ends Well
  - A Midsummer Night's Dream
  - Twelfth Night
- Tragedies
  - Hamlet
  - Macbeth
  - Romeo and Juliet
- Histories
  - Henry IV (email)
    - Part I
    - Part II
  - Henry V
  - Richard II

```css
.horizontal {
  float: left;
  list-style: none;
  margin: 10px;
}
```

```js
document.querySelectorAll('#selected-plays > li').forEach(function(el) {
  el.classList.add('horizontal');
});
```

Let’s add this CSS Class

But we only want to do this for top level list items.

“within the element whose ID is `#selected-plays`, I’m going select all list items li who are direct child descendants (>) of `#selected-plays`“

“I’m then going to add the `.horizontal` style class (`classList.add()`) to these elements“

Selected Shakespeare Plays

Comedies

- As You Like It
- All's Well That Ends Well
- A Midsummer Night's Dream
- Twelfth Night

Tragedies

- Hamlet
- Macbeth
- Romeo and Juliet

Histories

- Henry IV (email)
  - Part I
  - Part II
- Henry V
- Richard II

```
.sub-level {
  background: #ccc;
}
```

Now we want to apply this style (grey shading) to all remaining list items

```
document.querySelectorAll('#selected-plays > li').forEach(function(el) {
  el.classList.add('horizontal');
});

document.querySelectorAll('#selected-plays > li > ul > li').forEach(function(el) {
  el.classList.add('sub-level');
});
```

Selected Shakespeare Plays

Comedies

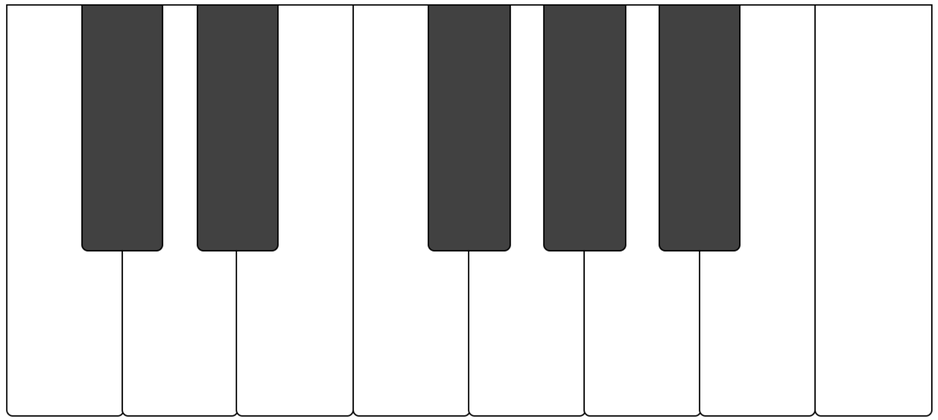
- As You Like It
- All's Well That Ends Well
- A Midsummer Night's Dream
- Twelfth Night

Tragedies

- Hamlet
- Macbeth
- Romeo and Juliet

Histories

- Henry IV (email)
  - Part I
  - Part II
- Henry V
- Richard II



The Piano

CodePen: https://codepen.io/timwray87/pen/dmYwjw

Events in JavaScript

In Web Applications, we add interactivity through the use of events.

Events are “things that happen”, such as mouse clicks, pressing a key on a keyboard, or downloading data via a network request.

We add interactivity by “listening” to events on DOM elements. For example, a button would ‘listen’ to a click, and then your program could do something in response to that click, such as displaying a message or altering the state of your application.

```
<span id="note-indicator"></span>
<div class="keyboard">
  <button id="C4" class="key white" onclick="playNote('C4')"></button>
  <button id="Db4" class="key black" onclick="playNote('Db4')"></button>
  …
</div>


function playNote(note) {
  console.log('Playing note: ' + note);
}
```

Click Events

One of the most common events in Web Applications are mouse click, this is handled via the `click` / `onclick` event.

In our piano example, we are going to write a function called `playNote()` that ‘plays’ a specified note, depending on the key that was clicked on.

We’ll add an `onclick` attribute of each key calls a function within our JavaScript file, passing in the parameter indicating the note to be played.

The example only shows the first two notes, write the `onclick` attribute for all keys.

```
<span id="note-indicator"></span>
<div class="keyboard">
  <button id="C4" class="key white" onclick="playNote('C4')"></button>
  <button id="Db4" class="key black" onclick="playNote('Db4')"></button>
  …
</div>


function playNote(note) {
  console.log('Playing note: ' + note);
  document.getElementById('note-indicator').innerHTML = note;
}
```

Altering the DOM

Naturally we might want to let the user know that something happened when we clicked on each note.

When a user clicks on each note, we'll display it to the user via altering the HTML content of the note-indicator element.

```
<span id="note-indicator"></span>
<div class="keyboard">
  <button id="C4" class="key white" onclick="playNote('C4')"></button>
  <button id="Db4" class="key black" onclick="playNote('Db4')"></button>
  …
</div>


// Event registration via getElementsByClassName

var whiteKeys = document.getElementsByClassName('white');
for (i = 0; i < whiteKeys.length; i++) {
  var key = whiteKeys[i];
  key.addEventListener('click', function() {
    console.log('You are playing a white key');
  });
}

var blackKeys = document.getElementsByClassName('black');
for (i = 0; i < blackKeys.length; i++) {
  var key = blackKeys[i];
  key.addEventListener('click', function() {
    console.log('You are playing a black key');
  });
}

function playNote(note) {
  console.log('Playing note: ' + note);
  document.getElementById('note-indicator').innerHTML = note;
}
```

Using addEventListener

So far, we've set up event listeners by writing attributes inline on the elements themselves.

Another way of setting up events is via the use of `addEventListener` within your JavaScript code. First you select the elements, then you use `addEventListener` to tell the element to listen to one or more events.

In this case, we'll use `getElementsByClassName` to select the white keys and the black keys, and we'll add event listeners to set up and respond to events.

```
<span id="note-indicator"></span>
<div class="keyboard">
  <button id="C4" class="key white"></button>
  <button id="Db4" class="key black"></button>
  …
</div>


// Event registration via getElementsByClassName

var whiteKeys = document.getElementsByClassName('white');
for (i = 0; i < whiteKeys.length; i++) {
  var key = whiteKeys[i];
  key.addEventListener('click', function() {
    console.log('You are playing a white key');
  });
}

var blackKeys = document.getElementsByClassName('black');
for (i = 0; i < blackKeys.length; i++) {
  var key = blackKeys[i];
  key.addEventListener('click', function() {
    console.log('You are playing a black key');
  });
}

// Event registration via querySelectorAll

document.querySelectorAll('.key').forEach(function(key) {
  key.addEventListener('click', function() {
    playNote(key.id);
  })
});

function playNote(note) {
  console.log('Playing note: ' + note);
  document.getElementById('note-indicator').innerHTML = note;
}
```

Using addEventListener

We'll also refactor our code, so that we no longer us onclick attributes for each key, and instead bind the click events programmatically via addEventListener.

There are advantages and disadvantages to using attribute bindings vs. addEventListener. Some prefer addEventListener, whereas frameworks (such as React) use the former attribute binding style.

See: https://stackoverflow.com/questions/6348494/addeventlistener-vs-onclick/35093997#35093997

```
// Event registration via getElementsByClassName

var whiteKeys = document.getElementsByClassName('white');
for (i = 0; i < whiteKeys.length; i++) {
  var key = whiteKeys[i];
  key.addEventListener('click', function() {
    console.log('You are playing a white key');
  });
}

var blackKeys = document.getElementsByClassName('black');
for (i = 0; i < blackKeys.length; i++) {
  var key = blackKeys[i];
  key.addEventListener('click', function() {
    console.log('You are playing a black key');
  });
}

// Event registration via querySelectorAll

document.querySelectorAll('.key').forEach(function(key) {
  key.addEventListener('click', function() {
    playNote(key.id);
  })
});

function playNote(note) {
  console.log('Playing note: ' + note);
  document.getElementById('note-indicator').innerHTML = note;
  // Play the note
  var noteURL = 'https://itu.dk/people/tiwr/piano/' + note + '.mp3';
  var noteSound = new Audio(noteURL);
  noteSound.play();
}
```

Add Audio Playback

Let's alter our `playNote()` function so that it actually plays audio. We'll use the Web Audio API to load a sound file from a URL, then play that sound file.

```
// Event registration via getElementsByClassName

var whiteKeys = document.getElementsByClassName('white');
for (i = 0; i < whiteKeys.length; i++) {
  var key = whiteKeys[i];
  key.addEventListener('click', function() {
    console.log('You are playing a white key');
  });
}

var blackKeys = document.getElementsByClassName('black');
for (i = 0; i < blackKeys.length; i++) {
  var key = blackKeys[i];
  key.addEventListener('click', function() {
    console.log('You are playing a black key');
  });
}

// Event registration via querySelectorAll

document.querySelectorAll('.key').forEach(function(key) {
  key.addEventListener('mouseover', function() {
    playNote(key.id);
  })
});

function playNote(note) {
  console.log('Playing note: ' + note);
  document.getElementById('note-indicator').innerHTML = note;
  // Play the note
  var noteURL = 'https://itu.dk/people/tiwr/piano/' + note + '.mp3';
  var noteSound = new Audio(noteURL);
  noteSound.play();
}
```

Different Events

You can play around with different kinds of events. For example, in the part of the code where we register the event for each key, change the `click` event to `mouseover`. Run your code and ‘play’ your piano. What happens?

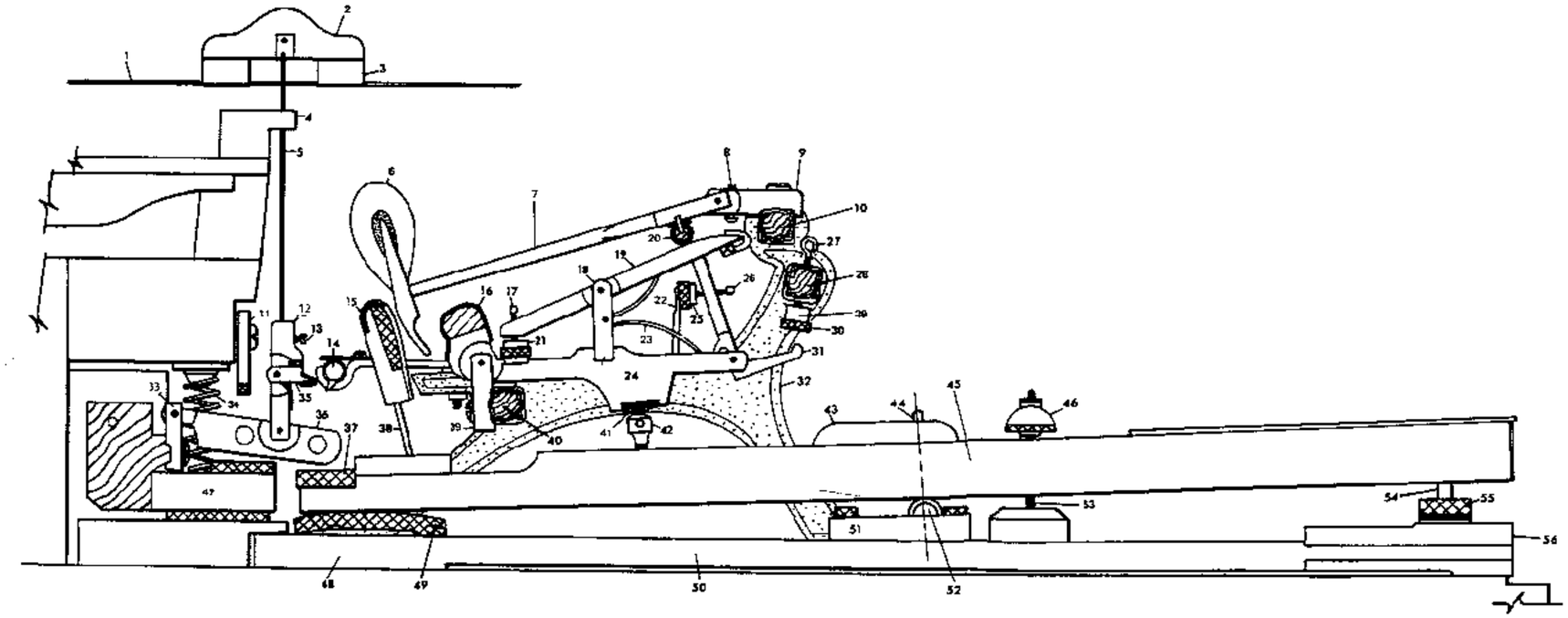
Keyboard Events

Event Name	Fired When
`keydown`	ANY key is pressed
`keypress`	ANY key except Shift, Fn, CapsLock is in pressed position. (Fired continously.)
`keyup`	ANY key is released

Mouse Events

Event Name	Fired When
`mouseenter`	A pointing device is moved onto the element that has the listener attached.
`mouseover`	A pointing device is moved onto the element that has the listener attached or onto one of its children.
`mousemove`	A pointing device is moved over an element. (Fired continously as the mouse moves.)
`mousedown`	A pointing device button is pressed on an element.
`mouseup`	A pointing device button is released over an element.
`auxclick`	A pointing device button (ANY non-primary button) has been pressed and released on an element.
`click`	A pointing device button (ANY button; soon to be primary button only) has been pressed and released on an element.
`dblclick`	A pointing device button is clicked twice on an element.
`contextmenu`	The right button of the mouse is clicked (before the context menu is displayed).
`wheel`	A wheel button of a pointing device is rotated in any direction.
`mouseleave`	A pointing device is moved off the element that has the listener attached.
`mouseout`	A pointing device is moved off the element that has the listener attached or off one of its children.
`select`	Some text is being selected.

https://developer.mozilla.org/en-US/docs/Web/Events



Damping the Notes

Image source: http://www.concertpitchpiano.com/GrandActionModel.html

The Mechanics of the Piano

When you play the piano, the note plays when you hold down the key, and the note stops when you release the key.

We're going to replicate this functionality within our project, although it would somewhat increase its complexity.

There are two things we need to do first:
1) Create a `stopNote()` method.
2) Store the state of each note (whether it is playing or not).

For the latter, we'll use a model, see:
https://codepen.io/timwray87/pen/qoOLMM

Using a Model

https://codepen.io/timwray87/pen/qoOLMM

First, we'll add the model to our application. Copy and paste the JavaScript code from the CodePen to the beginning of your JavaScript section.

The mode is a list of objects that stores data about each note:
- As its **key**, the name of the note.
- As its **value**, its keyboard mapping, and it's state (whether it's playing or not).

```
function playNote(note) {
  console.log('Playing note: ' + note);
  document.getElementById(note).classList.add('pressed');
  document.getElementById('note-indicator').innerHTML = note;
  // Play the note
  var noteURL = 'https://itu.dk/people/tiwr/piano/' + note + '.mp3';
  var noteSound = new Audio(noteURL);
  keyMap[note].audio = noteSound;
  noteSound.play();
}

function stopNote(note) {
  console.log('Stopping note: ' + note);
  document.getElementById(note).classList.remove('pressed');
  var notePlaying = keyMap[note].audio;
  if (notePlaying) {
    // Stop the note
    notePlaying.pause();
    notePlaying.currentTime = 0;
    keyMap[note].audio = undefined;
  }
}
```

The stopNote() function

The playNote() function gets called when the key is pressed down

When the playNote() function runs, we'll need to store a reference to the audio being played back within our keyMap, so we can stop it at a later point.

The stopNote() function gets called when the key goes back up again.

The stopNote() function works by looking up the specified key within the keyMap, and if the key is playing, tells the sound associated with the key to stop.

```
// Event registration via querySelectorAll

document.querySelectorAll('.key').forEach(function(key) {
  key.addEventListener('mousedown', function() {
    playNote(key.id);
  });
  key.addEventListener('mouseup', function() {
    stopNote(key.id);
  });
});

function playNote(note) {
  console.log('Playing note: ' + note);
  document.getElementById(note).classList.add('pressed');
  document.getElementById('note-indicator').innerHTML = note;
  // Play the note
  var noteURL = 'https://itu.dk/people/tiwr/piano/' + note + '.mp3';
  var noteSound = new Audio(noteURL);
  keyMap[note].audio = noteSound;
  noteSound.play();
}

function stopNote(note) {
  console.log('Stopping note: ' + note);
  document.getElementById(note).classList.remove('pressed');
  var notePlaying = keyMap[note].audio;
  if (notePlaying) {
    // Stop the note
    notePlaying.pause();
    notePlaying.currentTime = 0;
    keyMap[note].audio = undefined;
  }
}
```

mousedown and mouseup

We'll modify our event listeners so that `playNote()` gets called when the mouse button down, and `stopNote()` gets called when the mouse button goes back up again.

We'll also modify the `playNote()` and `stopNote()` methods so that they add and remove the CSS class pressed to the corresponding note that's being played, so as to give visual feedback to the user.

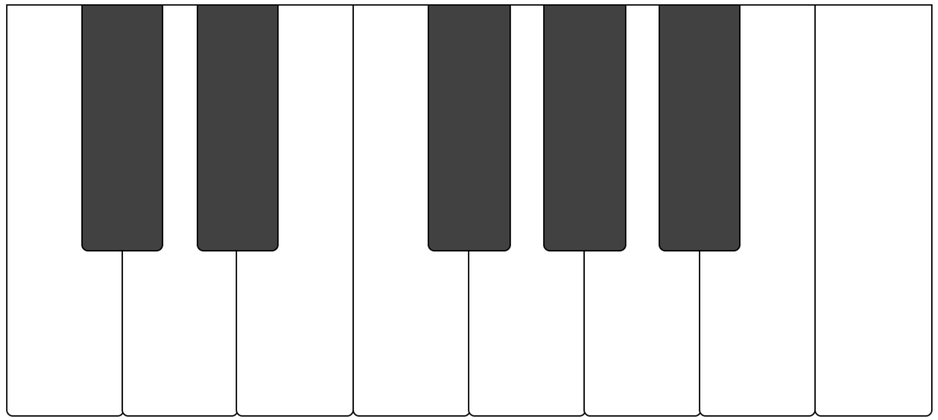
```
// Setup keyboard bindings for keyboard playback

document.addEventListener('keydown', function(e) {
  var keyID = e.key.toUpperCase();
  Object.keys(keyMap).forEach(function(key) {
    // The !keyMap[key].audio condition is required
    // to stop automatic repetition of sounds
    if (keyMap[key].keyboard === keyID && !keyMap[key].audio) {
      playNote(key);
    }
  });
});

document.addEventListener('keyup', function(e) {
  var keyID = e.key.toUpperCase();
  Object.keys(keyMap).forEach(function(key) {
    if (keyMap[key].keyboard === keyID) {
      stopNote(key);
    }
  });
});
```

Keyboard Bindings

Finally, we can set up keyboard event bindings so that we can play the notes with the middle and top row of your keyboard.



The “Complete” Piano Experience

Hold spacebar to press / release the damper pedal

CodePen: https://codepen.io/timwray87/pen/geaLxg