

Virtual Piano - Complete Code Explanation

FILE STRUCTURE OVERVIEW

Your project has 3 files:

1. **index.html** - The structure (HTML)
2. **style.css** - The styling (CSS)
3. **script.js** - The logic and functionality (JavaScript)

FILE 1: INDEX.HTML - THE STRUCTURE

Head Section

```
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Virtual Piano</title>
  <link rel="stylesheet" href="style.css">
</head>
```

..

- Tells browser this is HTML5 (latest standard)

```
<html lang="en">
```

- Opens HTML document, language is English

```
<meta charset="UTF-8">
```

- Character encoding: supports all languages, symbols, emojis

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

- CRITICAL for mobile!
- Prevents page from being zoomed out on mobile devices
- Ensures text is readable on phones

```
<link rel="stylesheet" href="style.css">
```

- Imports styling from style.css file

Body Section - Header

```
<body>
  <div>
    <div>
      <h1> Virtual Piano</h1>
      <p>Click keys or use keyboard to play (A-J, W, E, T, Y, U)</p>
    </div>
```

```
</div>
```

- Wrapper that centers and contains all content

```
<div>
```

- Top section with title and instructions

```
<h1> and <p>
```

- h1 = large heading
- p = paragraph text

Body Section - Piano Container

```
</p><div>
  <div></div>
```

```
</div></div>
```

- IMPORTANT: This is where all piano keys go!
- id="keyboard" means JavaScript finds this element
- Currently empty, JavaScript will add keys here

Body Section - Controls

```
<div>
  <div>
    <label for="volume">Volume:</label>
    <input type="range" id="volume" min="0" max="1" step="0.01" value=
    <span>50%</span>
  </div>
```

```
<input type="range">
```

- Creates a slider control

Attributes:

- min="0" - lowest volume
- max="1" - highest volume (100%)

- `step="0.01"` - each movement = 1% increment
- `value="0.5"` - starts at 50%

`50%`

- Shows current volume percentage
- JavaScript updates this when slider moves

Body Section - Waveform Control

```
<div>
  <label for="waveform">Waveform:</label>
  <select id="waveform">
    <option value="sine">Sine</option>
    <option value="triangle">Triangle</option>
    <option value="square">Square</option>
    <option value="sawtooth" selected>Sawtooth</option>
  </select>
</div>
```

Waveform types:

- **Sine** = smooth, pure tone (like a whistle)
- **Triangle** = warmer (between sine and square)
- **Square** = harsh, buzzy (like old video games)
- **Sawtooth** = bright, buzzy (like synthesizer) - default

Body Section - Octave Control

```
<div>
  <label for="octave">Octave:</label>
  <input type="range" id="octave" min="0" max="1" step="1" value="0" />
  <span>Octave 1</span>
</div>
```

`step="1"`

- Only 2 positions: 0 (Octave 1) or 1 (Octave 2)
- Jumps between values, doesn't slide smoothly

Body Section - Shortcuts Display

```
<div>
  <div>🖱️ Keyboard Shortcuts</div>
  <div></div>
</div>
</div>
</div>
```

```
<script src="script.js"></script>
</body>
</html>
```

id="shortcutsGrid"

- JavaScript fills this with keyboard shortcut items

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

- MUST be at end of HTML so elements load first
- Imports JavaScript file

FILE 2: STYLE.CSS - THE STYLING

Universal Reset

```
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}
```

Purpose:

- `margin: 0;` - removes space around all elements
- `padding: 0;` - removes space inside all elements
- `box-sizing: border-box;` - borders included in element width

Body Styling

```
body {
  font-family: -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto, sans-serif;
  background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
  min-height: 100vh;
  display: flex;
  justify-content: center;
  align-items: center;
  padding: 20px;
  user-select: none;
}
```

`font-family` - tries fonts in order:

- Mac system font → Chrome Mac → Windows → Android → fallback

`background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);`

- 135deg = diagonal angle

- Purple-blue → darker purple

```
display: flex; justify-content: center; align-items: center;
```

- Centers content both horizontally and vertically

```
min-height: 100vh;
```

- Full screen height (vh = viewport height)

```
user-select: none;
```

- Prevents text selection (important for piano keys)

Header Styling

```
.header h1 {
  font-size: clamp(28px, 8vw, 48px);
  font-weight: 700;
  text-shadow: 0 2px 10px rgba(0, 0, 0, 0.2);
}
```

```
clamp(28px, 8vw, 48px)
```

- RESPONSIVE sizing!
- Minimum: 28px
- Preferred: 8% of screen width
- Maximum: 48px
- Automatically scales between min and max

```
font-weight: 700;
```

- Bold text

```
text-shadow
```

- Creates shadow effect for depth

Piano Container

```
.piano {
  background: linear-gradient(to bottom, #1a1a2e, #16213e);
  border-radius: 20px;
  padding: clamp(15px, 3vw, 30px);
  box-shadow: 0 20px 60px rgba(0, 0, 0, 0.4);
  border: 1px solid rgba(255, 255, 255, 0.1);
  margin: 0 auto;
  max-width: 1400px;
}
```

```
border-radius: 20px;
```

- Round corners by 20 pixels

box-shadow

- Drop shadow makes it look floating

margin: 0 auto;

- Centers horizontally

max-width: 1400px;

- Never wider than 1400px

Keyboard Container

```
.keyboard {  
  position: relative;  
  display: flex;  
  flex-direction: column;  
  padding: clamp(10px, 2vw, 20px);  
  background: rgba(0, 0, 0, 0.3);  
  border-radius: 12px;  
  align-items: center;  
  min-height: 300px;  
}
```

position: relative;

- Parent for absolutely positioned black keys

flex-direction: column;

- Stack items vertically

align-items: center;

- Center items horizontally

Keys Wrapper

```
.keyboard-keys-wrapper {  
  position: relative;  
  display: flex;  
  gap: 0;  
  justify-content: center;  
  align-items: flex-start;  
  width: 100%;  
  flex-wrap: wrap;  
}
```

gap: 0;

- No space between keys (they touch)

align-items: flex-start;

- Align to top (black keys positioned on top)

flex-wrap: wrap;

- Keys wrap to next line on mobile

White Keys

```
.key-white {  
  width: clamp(35px, 6vw, 70px);  
  aspect-ratio: 1 / 4.67;  
  min-height: 150px;  
  background: linear-gradient(to bottom, #ffffff 0%, #f5f5f5 95%, #e0e0e0 100%);  
  border: 1px solid #999;  
  border-radius: 0 0 8px 8px;  
  cursor: pointer;  
  transition: all 0.05s ease;  
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);  
  user-select: none;  
  z-index: 1;  
}
```

aspect-ratio: 1 / 4.67;

- Height = width × 4.67 (keeps keys proportional)

border-radius: 0 0 8px 8px;

- Round bottom corners only (not top)

transition: all 0.05s ease;

- Smooth animation when state changes

z-index: 1;

- White keys behind black keys

White Keys - Hover and Active States

```
.key-white:hover:not(.active) {  
  background: lighter;  
  box-shadow: bigger shadow;  
}  
  
.key-white.active {  
  background: darker;  
  transform: translateY(4px);  
}
```

:hover - when mouse over

:not(.active) - but not if currently pressed

.active - JavaScript adds this class when pressing

transform: translateY(4px);

- Move down 4px (simulates key press)

Black Keys

```
.key-black {  
  width: clamp(20px, 3.5vw, 40px);  
  aspect-ratio: 1 / 4.5;  
  background: linear-gradient(to bottom, #222 0%, #000 100%);  
  position: absolute;  
  top: 0;  
  z-index: 10;  
  left: [JavaScript sets this];  
}
```

position: absolute;

- Positioned over white keys

z-index: 10;

- On top of white keys

left: [JavaScript calculated]

- JavaScript positions each black key

Controls Section

```
.controls {  
  margin-top: clamp(15px, 3vw, 30px);  
  display: flex;  
  justify-content: center;  
  gap: clamp(10px, 2vw, 20px);  
  flex-wrap: wrap;  
}  
  
.control-group {  
  display: flex;  
  align-items: center;  
  gap: 10px;  
  background: rgba(255, 255, 255, 0.05);  
  padding: 12px 16px;  
  border-radius: 8px;  
  border: 1px solid rgba(255, 255, 255, 0.1);  
}
```

flex-wrap: wrap;

- Controls wrap on mobile

background: rgba(255, 255, 255, 0.05);

- Subtle white tint

Sliders

```
input[type="range"] {  
  width: clamp(80px, 15vw, 140px);  
  background: linear-gradient(to right, #667eea, #764ba2);  
  -webkit-appearance: none;  
  appearance: none;  
}  
  
input[type="range"]::-webkit-slider-thumb {  
  width: 16px;  
  height: 16px;  
  border-radius: 50%;  
  background: white;  
}
```

appearance: none;

- Remove default browser styling

::-webkit-slider-thumb

- Style the draggable circle

FILE 3: SCRIPT.JS - THE LOGIC

Configuration Object

```
const PIANO_CONFIG = {  
  notes: [  
    { note: 'C', key: 'A', frequency: 261.63, isBlack: false },  
    { note: 'C#', key: 'W', frequency: 277.18, isBlack: true },  
    // ... 10 more notes  
  ],  
  notesRow2: [  
    { note: 'C2', key: 'A', frequency: 523.25, isBlack: false },  
    // ... octave 2 (frequencies doubled)  
  ],  
  blackKeyOffsets: {  
    'C#': 41, 'D#': 101, 'F#': 221, 'G#': 281, 'A#': 341  
  },  
  blackKeyOffsetsMobile: { /* smaller offsets */ },  
  blackKeyOffsetsTiny: { /* even smaller */ }  
};
```

Each note has:

- note - musical note name
- key - keyboard key that triggers it
- frequency - Hz value (pitch)

- isBlack - true if black key

Frequency facts:

- C (octave 1) = 261.63 Hz
- C (octave 2) = 523.25 Hz (exactly double)

Class Constructor

```
class VirtualPiano {
  constructor(config) {
    this.config = config;
    this.keyboard = document.getElementById('keyboard');
    this.volumeSlider = document.getElementById('volume');
    this.volumeDisplay = document.getElementById('volumeDisplay');
    this.waveformSelect = document.getElementById('waveform');
    this.octaveSlider = document.getElementById('octave');
    this.octaveLabel = document.getElementById('octaveLabel');

    this.audioContext = null;
    this.mainGainNode = null;
    this.currentOscillators = new Map();
    this.volume = 0.5;
    this.waveform = 'sawtooth';
    this.octave = 0;

    this.init();
  }
}
```

Stores:

- References to HTML elements
- Audio engine components
- Current state (volume, waveform, octave)

Initialization

```
init() {
  this.initializeAudioContext();
  this.createKeys();
  this.createShortcuts();
  this.attachEventListeners();
  this.updateBlackKeyPositions();
}
```

Sequence:

1. Create audio engine
2. Generate piano keys

3. Show keyboard shortcuts
4. Attach event listeners
5. Position black keys

Audio Context Setup

```
initializeAudioContext() {  
  if (!this.audioContext) {  
    const AudioContextClass = window.AudioContext || window.webkitAudioContext;  
    this.audioContext = new AudioContextClass();  
    this.mainGainNode = this.audioContext.createGain();  
    this.mainGainNode.connect(this.audioContext.destination);  
    this.mainGainNode.gain.value = this.volume;  
  }  
}
```

What happens:

1. Check if audio engine already exists
2. Create audio context (sound generator)
3. Create gain node (volume control)
4. Connect to speakers

Audio chain: Oscillator → Gain → Speakers

Create Piano Keys

```
createKeys() {  
  let wrapper = this.keyboard.querySelector('.keyboard-keys-wrapper');  
  if (!wrapper) {  
    wrapper = document.createElement('div');  
    wrapper.className = 'keyboard-keys-wrapper';  
    this.keyboard.appendChild(wrapper);  
  }  
  
  const mainFragment = document.createDocumentFragment();  
  const currentNotes = this.octave === 0 ? this.config.notes : this.config.notesRow2;  
  
  currentNotes.forEach((note) => {  
    const keyEl = document.createElement('div');  
    const keyClass = note.isBlack ? 'key-black' : 'key-white';  
    keyEl.className = `key ${keyClass}`;  
    keyEl.dataset.note = note.note;  
    keyEl.dataset.frequency = note.frequency;  
    keyEl.innerHTML = `<span>${note.key}</span>`;  
  
    keyEl.addEventListener('mousedown', () => this.playNote(note.frequency, keyEl))  
    keyEl.addEventListener('mouseup', () => this.stopNote(keyEl));  
    keyEl.addEventListener('touchstart', (e) => {  
      e.preventDefault();  
      this.playNote(note.frequency, keyEl);  
    });  
  });  
}
```

```

    });

    mainFragment.appendChild(keyEl);
  });

  wrapper.innerHTML = '';
  wrapper.appendChild(mainFragment);
}

```

Process:

1. Create wrapper div if not exists
2. Create fragment (invisible container)
3. Get current octave notes
4. For each note:
 - Create div element
 - Set class (key-black or key-white)
 - Store data on element
 - Add letter label
 - Add event listeners
 - Add to fragment
5. Clear old keys
6. Add fragment to page (fast batch update)

Why fragment?

- Adding many elements at once is faster than one-by-one
- Prevents multiple page redraws

Event Listeners

```

attachEventListeners() {
  this.volumeSlider.addEventListener('input', (e) => {
    this.volume = parseFloat(e.target.value);
    this.volumeDisplay.textContent = Math.round(this.volume * 100) + '%';
    this.mainGainNode.gain.value = this.volume;
  });

  this.waveformSelect.addEventListener('change', (e) => {
    this.waveform = e.target.value;
  });

  this.octaveSlider.addEventListener('input', (e) => {
    this.octave = parseInt(e.target.value);
    this.octaveLabel.textContent = this.octave === 0 ? 'Octave 1' : 'Octave 2';
    this.createKeys();
    this.createShortcuts();
  });
}

```

```

        this.updateBlackKeyPositions();
    });

    document.addEventListener('keydown', (e) => {
        const currentNotes = this.octave === 0 ? this.config.notes : this.config.notesRow;
        const noteConfig = currentNotes.find(n => n.key.toUpperCase() === e.key.toUpperCase());
        if (noteConfig) {
            const keyEl = document.querySelector(`[data-frequency="${noteConfig.frequency}"]`);
            if (keyEl && !this.currentOscillators.has(keyEl)) {
                this.playNote(noteConfig.frequency, keyEl);
            }
        }
    });
}

```

Volume slider:

- Updates volume value
- Updates display text
- Changes actual audio volume

Waveform selector:

- Changes waveform type
- Takes effect on next key press

Octave slider:

- Changes octave
- Recreates all keys with new frequencies

Keyboard events:

- User presses key → play note
- User releases key → stop note

Play Note

```

playNote(frequency, keyEl) {
    this.initializeAudioContext();
    if (this.currentOscillators.has(keyEl)) return;

    keyEl.classList.add('active');
    const oscillator = this.audioContext.createOscillator();
    oscillator.type = this.waveform;
    oscillator.frequency.value = frequency;
    oscillator.connect(this.mainGainNode);
    this.currentOscillators.set(keyEl, oscillator);
    oscillator.start();
}

```

Process:

1. Make sure audio engine exists
2. Prevent playing same key twice
3. Add 'active' class (CSS highlights key)
4. Create oscillator (sound generator)
5. Set waveform type
6. Set frequency (pitch)
7. Connect to volume control
8. Store oscillator for later stopping
9. Start playing

Stop Note

```
stopNote(keyEl) {  
  if (!this.currentOscillators.has(keyEl)) return;  
  
  const source = this.currentOscillators.get(keyEl);  
  const gainNode = this.audioContext.createGain();  
  gainNode.gain.value = 1;  
  gainNode.gain.exponentialRampToValueAtTime(0.01, this.audioContext.currentTime + 0.1)  
  
  source.disconnect();  
  source.connect(gainNode);  
  gainNode.connect(this.mainGainNode);  
  source.stop(this.audioContext.currentTime + 0.1);  
  
  this.currentOscillators.delete(keyEl);  
  keyEl.classList.remove('active');  
}
```

Process:

1. Get the oscillator for this key
2. Create fade-out gain node
3. Set exponential ramp: 1.0 → 0.01 over 0.1 seconds
4. Disconnect from main gain
5. Connect to fade-out gain
6. Connect fade-out to speakers
7. Stop oscillator after fade
8. Remove from tracking
9. Remove 'active' class

Why exponential ramp?

- Human ears perceive sound logarithmically

- Exponential decrease sounds natural
- Linear decrease sounds glitchy

Initialize Page

```
document.addEventListener('DOMContentLoaded', () => {  
    new VirtualPiano(PIANO_CONFIG);  
});  
  
document.addEventListener('click', () => {  
    const AudioContextClass = window.AudioContext || window.webkitAudioContext;  
    if (AudioContextClass) {  
        const ctx = new AudioContextClass();  
        if (ctx.state === 'suspended') ctx.resume();  
    }  
}, { once: true });
```

DOMContentLoaded:

- Fires when HTML fully loads
- Creates VirtualPiano instance

First click handler:

- Modern browsers require user interaction before audio
- Security feature (prevents ads)
- Resumes audio engine on first click

COMPLETE EXECUTION FLOW

1. Page Loads

- Browser reads HTML
- Loads CSS styles
- Loads JavaScript

2. JavaScript Executes

- DOMContentLoaded event fires
- Creates new VirtualPiano(PIANO_CONFIG)
- Calls init()

3. Initialization Complete

- Audio engine ready
- 12 keys generated and displayed
- Keyboard shortcuts shown
- Event listeners active
- Black keys positioned

4. User Clicks Key

- Mouse click detected
- Event listener fires
- `playNote(frequency, keyEl)` called
- Oscillator created at correct pitch
- Sound plays through speakers
- Key visual feedback added

5. User Releases Key

- Mouse release detected
- Event listener fires
- `stopNote(keyEl)` called
- Fade-out effect starts
- Sound smoothly stops
- Visual feedback removed

6. Ready for Next Note

- Loop back to step 4

KEY CONCEPTS SUMMARY

1. Web Audio API

- Generates sound mathematically
- Not playing audio files
- Creates oscillators at specific frequencies
- Powerful and real-time

2. Responsive Design

- `cLamp()` function scales automatically
- Works on desktop, tablet, mobile
- No media queries needed for simple scaling

3. Event System

- Listens for user actions
- Mouse, touch, keyboard events
- Triggers appropriate functions

4. Maps for Storage

- Tracks active oscillators
- Fast lookups
- Prevents duplicate playing

5. Exponential Ramps

- Natural-sounding fades
- Matches human hearing
- Better than linear transitions

6. Data Attributes

- Store information on HTML elements
- Easy to retrieve
- Keeps HTML and JavaScript connected

7. Document Fragments

- Batch DOM updates
- Faster than individual updates
- Better performance

8. Touch Event Prevention

- `preventDefault()` stops browser defaults
- Allows custom touch behavior
- Prevents text selection and menus

You now understand every line of code in your Virtual Piano!

