# HTML5 Complete Guide

## Table of Contents



---

# HTML5 Overview

## What is HTML5?

HTML5 stands for **HyperText Markup Language version 5**. It represents the latest and most advanced version of HTML, the standard markup language used to create and structure content on the web.

## Key Features of HTML5

HTML5 introduced revolutionary changes to web development, including:

1. **New Semantic Elements**: Tags that provide meaning to the structure of web content
2. **Enhanced Form Controls**: New input types and attributes for better user experience
3. **Native Media Support**: Built-in audio and video playback without plugins
4. **Graphics Capabilities**: Canvas and SVG for creating dynamic graphics
5. **Powerful JavaScript APIs**: Local storage, geolocation, drag-and-drop, and more

## Why HTML5 Matters

**Before HTML5**, developers had to rely on third-party plugins like Flash for multimedia content. HTML5 eliminated this dependency by providing native support for audio, video, and interactive graphics. This made the web more accessible, faster, and more standardized across different browsers and devices.

**HTML5 Benefits:**

- **Cross-platform compatibility**: Works seamlessly across desktop, mobile, and tablet devices
- **Better performance**: Native implementations are faster than plugin-based solutions
- **Improved accessibility**: Semantic tags help screen readers and assistive technologies
- **SEO advantages**: Search engines better understand semantic markup
- **Offline capabilities**: Applications can work without internet connection using local storage
- **Reduced dependency**: No need for third-party plugins like Flash or Silverlight

---

# HTML Tag Types

## Opening and Closing Tags

Most HTML elements use a pair of tags: an opening tag and a closing tag that wraps content.

**Syntax:**

```
<tagname>Content goes here</tagname>
```

**Examples:**

```
<p>This is a paragraph</p>
<h1>This is a heading</h1>
<div>This is a division</div>
<span>This is an inline element</span>
```

**Key Characteristics:**

- The opening tag marks where the element begins
- The closing tag (with forward slash `/`) marks where it ends
- Content between tags is affected by the element's styling and behavior
- Tags can be nested inside other tags to create document structure

## Self-Closing Tags (Void Elements)

Some HTML elements don't contain any content and therefore don't need a closing tag. These are called **void elements** or **self-closing tags**.

**Syntax:**

```
<tagname />
```

**Common Self-Closing Tags:**

```
<img src="image.jpg" alt="Description" />
<input type="text" name="username" />
<br />
<hr />
<meta charset="UTF-8" />
<link rel="stylesheet" href="styles.css" />
```

**Important Notes:**

- In HTML5, the trailing slash `/` is optional for void elements
- Both `<input>` and `<input />` are valid
- These elements cannot have child elements
- All configuration is done through attributes

---

# Semantic Tags

## What Are Semantic Tags?

Semantic tags are HTML elements that carry **meaning** about the content they contain. Unlike generic containers like `<div>` or `<span>`, semantic tags describe the purpose and role of content sections in a web page.

## Complete List of HTML5 Semantic Tags

**Structural Semantic Tags:**

- `<header>`: Introductory content or navigational aids for a section or page
- `<nav>`: Navigation links section
- `<main>`: The dominant content of the document body
- `<section>`: A thematic grouping of content, typically with a heading
- `<article>`: Self-contained composition intended to be independently distributable
- `<aside>`: Content tangentially related to the main content (sidebars)
- `<footer>`: Footer information for a section or page

**Content Semantic Tags:**

- `<figure>`: Self-contained content like images, diagrams, code listings
- `<figcaption>`: Caption or legend for a figure element
- `<details>`: Disclosure widget that user can open/close
- `<summary>`: Summary, caption, or legend for a details element
- `<mark>`: Highlighted or marked text
- `<time>`: Specific time or datetime
- `<datalist>`: Predefined options for input controls

## Why Semantic Tags Matter

### 1. Accessibility (Screen Readers)

**Problem with Non-Semantic HTML:**

```
<div id="navigation">
  <div class="link">Home</div>
  <div class="link">About</div>
</div>
```

Screen readers for visually impaired users cannot understand the purpose of these divs. They're just generic containers.

**Solution with Semantic HTML:**

```
<nav>
  <a href="/">Home</a>
  <a href="/about">About</a>
</nav>
```

Screen readers can now announce "Navigation landmark" and help users quickly jump to navigation sections. This dramatically improves the browsing experience for people using assistive technologies.

**Accessibility Benefits:**

- Screen readers can identify page regions (navigation, main content, complementary content)
- Users can skip to specific sections using keyboard shortcuts
- Proper heading hierarchy helps users understand content structure
- ARIA (Accessible Rich Internet Applications) roles are automatically implied

### 2. Search Engine Optimization (SEO)

Search engines use semantic tags to better understand your content's structure and importance.

**How It Helps SEO:**

- Search engines prioritize content in `<main>` and `<article>` tags
- `<header>` and `<h1>-<h6>` tags signal content hierarchy
- `<nav>` helps search engines understand site structure
- Proper semantic structure can improve search rankings
- Rich snippets and featured results are more likely with semantic markup

**Example:**

```
<article>
  <header>
    <h1>Understanding Machine Learning</h1>
    <time datetime="2024-01-15">January 15, 2024</time>
  </header>
  <p>Machine learning is a subset of artificial intelligence...</p>
</article>
```

Google understands this is an article with a title, publication date, and content—potentially displaying it as a rich result.

## 3. Code Readability and Maintainability

**Non-Semantic Approach:**

```
<div class="top-section">
  <div class="menu-items">...</div>
</div>
<div class="main-content-area">
  <div class="content-block">...</div>
</div>
<div class="bottom-section">...</div>
```

**Semantic Approach:**

```
<header>
  <nav>...</nav>
</header>
<main>
  <article>...</article>
```

```
</main>
<footer>...</footer>
```

The semantic version is immediately understandable without reading class names or comments. Any developer can quickly grasp the page structure.

## Practical Example: Complete Page Structure

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>News Website</title>
</head>
<body>
  <header>
    <h1>Daily News</h1>
    <nav>
      <a href="/">Home</a>
      <a href="/world">World</a>
      <a href="/tech">Technology</a>
    </nav>
  </header>

  <main>
    <article>
      <header>
        <h2>Breaking: New Discovery in Science</h2>
        <time datetime="2024-01-15">January 15, 2024</time>
      </header>
      <p>Scientists have discovered...</p>
      <footer>
        <p>Written by Jane Doe</p>
      </footer>
    </article>

    <aside>
      <h3>Related Articles</h3>
      <ul>
        <li><a href="#">Previous Discovery</a></li>
      </ul>
    </aside>
  </main>

  <footer>
```

```
    <p>&copy; 2024 Daily News. All rights reserved.</p>
  </footer>
</body>
</html>
```

---

# HTML5 Input Types

HTML5 dramatically expanded the types of input fields available in forms, making them more powerful, user-friendly, and reducing the need for JavaScript validation.

## Traditional Input Types (Pre-HTML5)

Before HTML5, we had limited input types:

- `text` : Plain text input
- `password` : Masked text input
- `submit` : Submit button
- `reset` : Reset button
- `radio` : Radio button
- `checkbox` : Checkbox
- `color` : Color

## New HTML5 Input Types

### 1. Number Input

**Purpose**: Accept numeric values with built-in increment/decrement controls.

**Syntax:**

```
<input type="number" min="10" max="100" step="10">
```

**Attributes:**

- `min` : Minimum acceptable value
- `max` : Maximum acceptable value
- `step` : Legal number intervals (e.g., step="10" allows 10, 20, 30...)
- `value` : Default value

**JavaScript Methods:**

```
var numInput = document.getElementById("numInp");

// Increase value by step amount
numInput.stepUp();      // Increase by 1 (default step)
numInput.stepUp(10);    // Increase by 10

// Decrease value by step amount
numInput.stepDown();    // Decrease by 1
numInput.stepDown(5);   // Decrease by 5

// Get numeric value (not string)
console.log(numInput.valueAsNumber);  // Returns actual number
console.log(typeof numInput.valueAsNumber);  // "number"

// vs regular value property
console.log(numInput.value);  // Returns string "50"
console.log(typeof numInput.value);  // "string"
```

**Browser Behavior:**

- Desktop browsers show up/down spinner arrows
- Mobile browsers display numeric keyboard
- Invalid entries are automatically prevented or highlighted

## 2. Range Input (Slider)

**Purpose**: Select a numeric value from a range using a slider control.

**Syntax:**

```
<input type="range" min="10" max="100" step="10" value="50">
<p id="rangeValue"></p>
```

**JavaScript Integration:**

```
var rangeInput = document.getElementById("rangeInp");
var rangeValue = document.getElementById("rangeVal");

// Update display on input (fires continuously as user drags)
rangeInput.addEventListener("input", function() {
    rangeValue.innerText = rangeInput.value;
});

// vs change event (fires only when user releases)
```

```
rangeInput.addEventListener("change", function() {
    rangeValue.innerText = rangeInput.value;
});
```

**Difference Between `input` and `change` Events:**

- `input`: Fires continuously as the slider moves (real-time updates)
- `change`: Fires only when the user finishes adjusting (releases mouse/touch)

**Use Cases:**

- Volume controls
- Brightness adjustments
- Price range filters
- Zoom controls

## 3. Email Input

**Purpose**: Validate email addresses with built-in pattern matching.

**Syntax:**

```
<input type="email" name="userEmail" required>
```

**Features:**

- Automatic validation for email format
- Mobile devices show email-optimized keyboard (with @ symbol)
- Can use `pattern` attribute for custom validation
- `multiple` attribute allows comma-separated emails

**Custom Pattern Example:**

```
<input type="email"
       pattern="[A-Za-z0-9]+@[A-Za-z]+\.[a-z]{2,3}">
```

**Pattern Explanation:**

- `[A-Za-z0-9]+`: One or more letters or numbers before @
- `@`: Required @ symbol
- `[A-Za-z]+`: One or more letters for domain name
- `\.`: Required dot (escaped with backslash)

- `[a-z]{2,3}` : 2 or 3 lowercase letters for extension (.com, .org, .uk)

## 4. URL Input

**Purpose**: Validate URLs and provide appropriate keyboard on mobile devices.

**Syntax:**

```
<input type="url" name="website" placeholder="https://example.com">
```

**Features:**

- Validates URL format automatically
- Mobile keyboards include forward slash and .com shortcuts
- Accepts various protocols (http, https, ftp, etc.)

## 5. Search Input

**Purpose**: Optimized for search queries with special styling and behavior.

**Syntax:**

```
<input type="search" name="query" placeholder="Search...">
```

**Special Behaviors:**

- Browsers may add a clear button (X) to quickly erase input
- Rounded corners styling in some browsers
- May store recent searches (browser-dependent)
- Better semantic meaning for screen readers

## 6. Tel (Telephone) Input

**Purpose**: Optimized for entering phone numbers.

**Syntax:**

```
<input type="tel" name="phone" pattern="[0-9]{3}-[0-9]{3}-[0-9]{4}">
```

**Features:**

- Mobile devices show numeric telephone keyboard
- Pattern attribute for format validation

- No automatic validation (phone formats vary by country)

## 7. Color Input

**Purpose**: Provide a color picker interface.

**Syntax:**

```
<input type="color" name="themeColor" value="#ff0000">
```

**Features:**

- Opens native color picker dialog
- Returns hexadecimal color value (#RRGGBB)
- Default value must be in hex format
- Great for theme customization, drawing apps, etc.

**JavaScript Usage:**

```
var colorInput = document.getElementById("colorPicker");
colorInput.addEventListener("change", function() {
    document.body.style.backgroundColor = colorInput.value;
});
```

## 8. Date and Time Inputs

HTML5 provides multiple date/time input types for different use cases.

**Date Input:**

```
<input type="date" name="birthday">
```

- Shows calendar picker
- Format: YYYY-MM-DD
- Validates date values

**Time Input:**

```
<input type="time" name="appointment">
```

- Shows time picker (12 or 24-hour based on locale)
- Format: HH:MM or HH:MM:SS

**Datetime-local Input:**

```
<input type="datetime-local" name="eventDateTime">
```

- Combined date and time picker
- No timezone (local time only)
- Format: YYYY-MM-DDTHH:MM

**Month Input:**

```
<input type="month" name="creditCardExpiry">
```

- Pick month and year only
- Format: YYYY-MM
- Useful for credit card expiration, subscription billing

**Week Input:**

```
<input type="week" name="weekNumber">
```

- Pick week of the year
- Format: YYYY-W##
- Useful for scheduling, reports by week

**Note:** The `datetime` input type (with timezone) was removed from HTML5 specification. Use `datetime-local` instead.

## Common Input Attributes (HTML5)

These attributes work across multiple input types:

**required**: Makes field mandatory

```
<input type="text" required>
```

**placeholder**: Shows hint text when field is empty

```
<input type="text" placeholder="Enter username">
```

**autocomplete**: Controls browser's autocomplete behavior

```
<input type="text" autocomplete="off">
```

- `on` : Enable autocomplete (default)
- `off` : Disable autocomplete
- Specific values: `name` , `email` , `tel` , `street-address` , etc.

**autofocus**: Automatically focus this field when page loads

```
<input type="text" autofocus>
```

- Only one element per page should have autofocus
- Improves user experience for forms

**pattern**: Regular expression for validation

```
<input type="text" pattern="[A-Za-z]{3,}">
```

**min, max, step**: For numeric and date inputs

```
<input type="number" min="1" max="100" step="5">
```

---

# Media Elements - Audio

HTML5 introduced native audio support, eliminating the need for Flash or other plugins.

## Basic Audio Element

**Syntax:**

```
<audio controls>
  <source src="audio.mp3" type="audio/mpeg">
  <source src="audio.ogg" type="audio/ogg">
  Your browser does not support the audio element.
</audio>
```

## Audio Attributes

**controls**: Display browser's default playback controls

```
<audio controls src="song.mp3"></audio>
```

**autoplay**: Start playing automatically when page loads

```
<audio autoplay src="song.mp3"></audio>
```

- **Note**: Modern browsers restrict autoplay for better user experience
- Usually requires `muted` attribute to work

**loop**: Repeat audio indefinitely

```
<audio loop src="background.mp3"></audio>
```

**muted**: Start with audio muted

```
<audio muted autoplay src="song.mp3"></audio>
```

**preload**: How the audio should be loaded

```
<audio preload="auto" src="song.mp3"></audio>
```

- `none` : Don't preload
- `metadata` : Load only metadata (duration, etc.)
- `auto` : Load entire file (default)

## Multiple Source Formats

Different browsers support different audio formats. Providing multiple sources ensures compatibility.

```
<audio controls>
  <source src="audio.mp3" type="audio/mpeg">   <!-- Chrome, Safari, Edge -->
  <source src="audio.ogg" type="audio/ogg">    <!-- Firefox, Opera -->
  <source src="audio.wav" type="audio/wav">    <!-- Fallback -->
  Your browser does not support the audio element.
</audio>
```

**Common Audio Formats:**

- **MP3** (audio/mpeg): Most widely supported, good compression

- **OGG** (audio/ogg): Open format, good quality
- **WAV** (audio/wav): Uncompressed, large files, best quality
- **AAC** (audio/aac): Advanced Audio Coding, better than MP3
- **WebM** (audio/webm): Modern format, good compression

## JavaScript Audio Control

The audio element provides extensive JavaScript API for custom controls.

### Loading and Playing Audio

```
var audio = document.querySelector("audio");

// Set audio source
audio.src = "song.mp3";

// Load the audio file
audio.load();

// Play audio
audio.play();

// Pause audio
audio.pause();
```

### Play/Pause Toggle

```
function playAudio() {
    if (audio.paused) {
        audio.play();
    }
}

function pauseAudio() {
    if (!audio.paused) {
        audio.pause();
    }
}
```

**Key Property:**

- `audio.paused` : Returns `true` if audio is paused, `false` if playing

### Stop Functionality

HTML5 audio doesn't have a native stop method. To stop:

```
function stopAudio() {
    audio.pause();         // Pause playback
    audio.currentTime = 0; // Reset to beginning
}

// Alternative using load()
function stopAudio() {
    audio.load();  // Reloads the audio, resetting it
    audio.pause(); // Ensure it's paused
}
```

## Volume Control

Volume is controlled with a number between 0.0 (silent) and 1.0 (maximum).

```
var volumeRange = document.getElementById("volumeRange");

volumeRange.addEventListener("input", function() {
    audio.volume = volumeRange.value;
});
```

### HTML for Volume Slider:

```
<input type="range" id="volumeRange" min="0" max="1" step="0.01" value="1">
```

### Volume Properties:

- `audio.volume` : Get/set volume (0.0 to 1.0)
- `audio.muted` : Get/set mute status (true/false)

## Mute Toggle

```
function muteAudio() {
    audio.muted = !audio.muted;  // Toggle mute status
}
```

## Time Control (Seeking)

Control playback position using `currentTime` (in seconds).

```
var timeRange = document.getElementById("timeRange");

// Seek to specific time when user drags slider
timeRange.addEventListener("input", function() {
    audio.currentTime = timeRange.value;
});

// Update slider as audio plays
audio.addEventListener("timeupdate", function() {
    timeRange.value = audio.currentTime;
});
```

**Important Event - loadedmetadata:**

```
audio.addEventListener("loadedmetadata", function() {
    console.log(audio.duration);  // Total duration in seconds
    timeRange.max = audio.duration;  // Set slider maximum
});
```

This event fires when audio metadata (duration, dimensions for video) is loaded.

**Time Properties:**

- `audio.currentTime` : Current playback position (seconds)
- `audio.duration` : Total duration (seconds, available after metadata loads)

## Playback Speed Control

Change how fast audio plays (0.25x to 4x typically).

```
var speedSelect = document.getElementById("speed");

speedSelect.addEventListener("change", function() {
    audio.playbackRate = speedSelect.value;
});
```

**HTML:**

```
<select id="speed">
  <option value="0.25">0.25x</option>
  <option value="0.5">0.5x</option>
  <option value="1" selected>Normal</option>
  <option value="1.25">1.25x</option>
```

```
  <option value="1.5">1.5x</option>
  <option value="2">2.0x</option>
</select>
```

**Property:**

- `audio.playbackRate` : Playback speed (1.0 = normal speed)

## Playlist Implementation

Creating next/previous functionality with multiple audio files.

```
var sounds = [
    'media/song1.mp3',
    'media/song2.mp3',
    'media/song3.mp3',
    'media/song4.mp3'
];

var audio = document.querySelector("audio");
var index = 0;
var size = sounds.length;

// Initialize with first song
audio.src = sounds[0];
audio.load();

function nextAudio() {
    if (index < size - 1) {
        index++;
    } else {
        index = 0;  // Loop back to first song
    }

    audio.src = sounds[index];
    audio.load();
    audio.play();
}

function prevAudio() {
    if (index > 0) {
        index--;
    } else {
        index = size - 1;  // Loop to last song
    }
```

```
    audio.src = sounds[index];
    audio.load();
    audio.play();
}
```

**Key Steps:**

1. Change `audio.src` to new file
2. Call `audio.load()` to load new file
3. Call `audio.play()` to start playback

## Essential Audio Properties

**Read-Only Properties:**

- `audio.duration` : Total length (seconds)
- `audio.paused` : Is audio paused?
- `audio.ended` : Has audio finished playing?
- `audio.currentSrc` : Currently loaded source URL

**Read-Write Properties:**

- `audio.currentTime` : Current position (seconds)
- `audio.volume` : Volume level (0.0 to 1.0)
- `audio.playbackRate` : Playback speed
- `audio.muted` : Mute status (boolean)
- `audio.src` : Audio source URL

## Important Audio Events

```
audio.addEventListener("loadedmetadata", function() {
    // Metadata loaded (duration available)
});

audio.addEventListener("timeupdate", function() {
    // Fires periodically as audio plays
    // Use for progress bars
});

audio.addEventListener("ended", function() {
    // Audio finished playing
    // Good for auto-playing next track
});
```

```
audio.addEventListener("play", function() {
    // Playback started
});

audio.addEventListener("pause", function() {
    // Playback paused
});

audio.addEventListener("volumechange", function() {
    // Volume or mute status changed
});

audio.addEventListener("error", function() {
    // Error loading or playing audio
    console.error("Audio error:", audio.error);
});
```

## Building Custom Audio Player

Complete example combining all concepts:

```
var audio = document.querySelector("audio");
var playBtn = document.getElementById("play");
var pauseBtn = document.getElementById("pause");
var stopBtn = document.getElementById("stop");
var muteBtn = document.getElementById("mute");
var volumeSlider = document.getElementById("volume");
var timeSlider = document.getElementById("time");
var currentTimeDisplay = document.getElementById("currentTime");
var durationDisplay = document.getElementById("duration");

// Initialize when metadata loads
audio.addEventListener("loadedmetadata", function() {
    timeSlider.max = audio.duration;
    durationDisplay.textContent = formatTime(audio.duration);
});

// Update UI as audio plays
audio.addEventListener("timeupdate", function() {
    timeSlider.value = audio.currentTime;
    currentTimeDisplay.textContent = formatTime(audio.currentTime);
});

// Control functions
playBtn.addEventListener("click", function() {
```

```
    audio.play();
});

pauseBtn.addEventListener("click", function() {
    audio.pause();
});

stopBtn.addEventListener("click", function() {
    audio.pause();
    audio.currentTime = 0;
});

muteBtn.addEventListener("click", function() {
    audio.muted = !audio.muted;
    muteBtn.textContent = audio.muted ? "Unmute" : "Mute";
});

volumeSlider.addEventListener("input", function() {
    audio.volume = volumeSlider.value;
});

timeSlider.addEventListener("input", function() {
    audio.currentTime = timeSlider.value;
});

// Utility function to format seconds as MM:SS
function formatTime(seconds) {
    var min = Math.floor(seconds / 60);
    var sec = Math.floor(seconds % 60);
    return min + ":" + (sec < 10 ? "0" : "") + sec;
}
```

---

## Media Elements - Video

HTML5 video works similarly to audio but with additional visual features.

### Basic Video Element

```
<video controls width="640" height="360">
  <source src="video.mp4" type="video/mp4">
  <source src="video.webm" type="video/webm">
  <source src="video.ogg" type="video/ogg">
```

```
  Your browser does not support the video element.
</video>
```

## Video Attributes

All audio attributes apply to video, plus:

**width/height**: Set video dimensions

```
<video controls width="800" height="450" src="movie.mp4"></video>
```

**poster**: Thumbnail image shown before playback

```
<video controls poster="thumbnail.jpg" src="movie.mp4"></video>
```

## Video Formats and Codecs

### Common Video Formats:

- **MP4** (video/mp4): H.264 codec, most compatible
- **WebM** (video/webm): VP8/VP9 codec, open format, good compression
- **OGG** (video/ogg): Theora codec, open format

### Browser Support Strategy:

```
<video controls>
  <source src="video.mp4" type="video/mp4">   <!-- Safari, Chrome, Edge -->
  <source src="video.webm" type="video/webm"> <!-- Chrome, Firefox, Opera -->
  <source src="video.ogg" type="video/ogg">   <!-- Firefox, Opera -->
</video>
```

## Subtitle/Caption Tracks

The `<track>` element provides subtitles, captions, and other timed text.

```
<video controls>
  <source src="movie.mp4" type="video/mp4">

  <track kind="subtitles"
         src="subtitles_en.vtt"
         srclang="en"
         label="English"
         default>
```

```
  <track kind="subtitles"
         src="subtitles_ar.vtt"
         srclang="ar"
         label="Arabic">

  <track kind="subtitles"
         src="subtitles_fr.vtt"
         srclang="fr"
         label="French">
</video>
```

**Track Attributes:**

- `kind` : Type of track
  - `subtitles` : Translation of dialogue
  - `captions` : Transcription including sound effects
  - `descriptions` : Text descriptions for visually impaired
  - `chapters` : Chapter titles for navigation
  - `metadata` : Track metadata (not visible to user)
- `src` : Path to WebVTT file
- `srclang` : Language code (en, ar, fr, es, etc.)
- `label` : User-visible label in track menu
- `default` : Make this track active by default

## WebVTT Format

WebVTT (Web Video Text Tracks) is the format for subtitle files.

**Example subtitles_en.vtt:**

```
WEBVTT

00:00:00.000 --> 00:00:03.000
Welcome to our video tutorial.

00:00:03.500 --> 00:00:07.000
Today we'll learn about HTML5 video.

00:00:07.500 --> 00:00:11.000
Let's get started!
```

**Format Explanation:**

- First line must be `WEBVTT`
- Each subtitle has:
  - Start time --> End time (HH:MM:SS.MMM format)
  - Text content (can be multiple lines)
  - Blank line separator

## Changing Subtitles with JavaScript

```
function changeSubtitle(src, lang) {
    const track = document.getElementById("subtitleTrack");

    // Update track source and language
    track.src = src;
    track.srclang = lang;

    // Reload track by toggling mode
    track.track.mode = "disabled";
    track.track.mode = "showing";
}

// Usage
changeSubtitle('/media/subtitles_ar.vtt', 'ar');  // Switch to Arabic
changeSubtitle('/media/subtitles_en.vtt', 'en');  // Switch to English
changeSubtitle('/media/subtitles_fr.vtt', 'fr');  // Switch to French
```

**Track Mode Property:**

- `"disabled"` : Track is not active
- `"hidden"` : Track is active but not displayed
- `"showing"` : Track is active and displayed

**Why toggle mode?**
When changing track source, toggling the mode forces the browser to reload and display the new subtitle file.

## Fullscreen API

Allow video to take over the entire screen.

```
var video = document.querySelector("video");

function toggleFullscreen() {
    if (!document.fullscreenElement) {
```

```
        // Enter fullscreen
        video.requestFullscreen();
    } else {
        // Exit fullscreen
        document.exitFullscreen();
    }
}
```

**Key Points:**

- `document.fullscreenElement` : Returns element currently in fullscreen (null if none)
- `element.requestFullscreen()` : Make element fullscreen (called on video element)
- `document.exitFullscreen()` : Exit fullscreen (called on document)

**Cross-browser Compatibility:**

```
function enterFullscreen(element) {
    if (element.requestFullscreen) {
        element.requestFullscreen();
    } else if (element.webkitRequestFullscreen) { // Safari
        element.webkitRequestFullscreen();
    } else if (element.mozRequestFullScreen) { // Firefox
        element.mozRequestFullScreen();
    } else if (element.msRequestFullscreen) { // IE/Edge
        element.msRequestFullscreen();
    }
}
```

## Picture-in-Picture API

Allow video to play in a floating window while user browses other content.

```
var video = document.querySelector("video");
var pipButton = document.getElementById("picInPic");

function togglePictureInPicture() {
    if (!document.pictureInPictureElement) {
        // Enter Picture-in-Picture
        video.requestPictureInPicture();
        pipButton.innerHTML = "Exit Picture in Picture";
    } else {
        // Exit Picture-in-Picture
        document.exitPictureInPicture();
        pipButton.innerHTML = "Picture in Picture";
```

```
    }
}
```

**Key Points:**

- `document.pictureInPictureElement` : Returns element currently in PiP (null if none)
- `video.requestPictureInPicture()` : Enter PiP mode
- `document.exitPictureInPicture()` : Exit PiP mode
- User can resize and move the floating window
- Video controls remain functional in PiP mode

**Detecting PiP Support:**

```
if ('pictureInPictureEnabled' in document) {
    // PiP is supported
    pipButton.style.display = 'block';
} else {
    // PiP not supported
    pipButton.style.display = 'none';
}
```

**PiP Events:**

```
video.addEventListener('enterpictureinpicture', function() {
    console.log('Video entered PiP mode');
});

video.addEventListener('leavepictureinpicture', function() {
    console.log('Video left PiP mode');
});
```

## All Video JavaScript Properties

**Inherited from Audio:**
All audio properties and methods work with video:

- `play()` , `pause()` , `load()`
- `currentTime` , `duration` , `volume`
- `playbackRate` , `muted` , `paused`
- All audio events

**Video-Specific Properties:**

- `video.videoWidth` : Intrinsic width of video
- `video.videoHeight` : Intrinsic height of video
- `video.poster` : Get/set poster image URL

## Complete Video Player Example

```
var video = document.querySelector("video");

// Playback controls
var playBtn = document.getElementById("play");
var pauseBtn = document.getElementById("pause");
var stopBtn = document.getElementById("stop");
var fullscreenBtn = document.getElementById("fullscreen");
var pipBtn = document.getElementById("pip");

// Progress tracking
var progressBar = document.getElementById("progress");
var currentTimeSpan = document.getElementById("current");
var durationSpan = document.getElementById("duration");

// Initialize
video.addEventListener("loadedmetadata", function() {
    progressBar.max = video.duration;
    durationSpan.textContent = formatTime(video.duration);
});

// Update progress
video.addEventListener("timeupdate", function() {
    progressBar.value = video.currentTime;
    currentTimeSpan.textContent = formatTime(video.currentTime);
});

// Playback controls
playBtn.addEventListener("click", () => video.play());
pauseBtn.addEventListener("click", () => video.pause());
stopBtn.addEventListener("click", function() {
    video.pause();
    video.currentTime = 0;
});

// Seeking
progressBar.addEventListener("input", function() {
    video.currentTime = progressBar.value;
});
```

```
// Fullscreen
fullscreenBtn.addEventListener("click", function() {
    if (!document.fullscreenElement) {
        video.requestFullscreen();
    } else {
        document.exitFullscreen();
    }
});

// Picture-in-Picture
pipBtn.addEventListener("click", function() {
    if (!document.pictureInPictureElement) {
        video.requestPictureInPicture();
    } else {
        document.exitPictureInPicture();
    }
});

// Auto-play next video when ended
video.addEventListener("ended", function() {
    // Load and play next video
    playNextVideo();
});

function formatTime(seconds) {
    var h = Math.floor(seconds / 3600);
    var m = Math.floor((seconds % 3600) / 60);
    var s = Math.floor(seconds % 60);

    if (h > 0) {
        return h + ":" + (m < 10 ? "0" : "") + m + ":" + (s < 10 ? "0" : "") +
s;
    }
    return m + ":" + (s < 10 ? "0" : "") + s;
}
```

---

## 3. Geolocation API

**Purpose**: Get user's geographical location (with permission).

**Basic Usage:**

```
if ("geolocation" in navigator) {
    navigator.geolocation.getCurrentPosition(
        // Success callback
        function(position) {
            var latitude = position.coords.latitude;
            var longitude = position.coords.longitude;
            var accuracy = position.coords.accuracy;  // meters

            console.log(`Location: ${latitude}, ${longitude}`);
            console.log(`Accuracy: ${accuracy} meters`);
        },
        // Error callback
        function(error) {
            switch(error.code) {
                case error.PERMISSION_DENIED:
                    console.log("User denied geolocation");
                    break;
                case error.POSITION_UNAVAILABLE:
                    console.log("Location unavailable");
                    break;
                case error.TIMEOUT:
                    console.log("Request timeout");
                    break;
            }
        },
        // Options
        {
            enableHighAccuracy: true,  // Use GPS if available
            timeout: 5000,              // Max wait time (ms)
            maximumAge: 0               // Don't use cached position
        }
    );
} else {
    console.log("Geolocation not supported");
}
```

**Position Object Properties:**

- `coords.latitude` : Latitude in decimal degrees
- `coords.longitude` : Longitude in decimal degrees
- `coords.accuracy` : Accuracy in meters
- `coords.altitude` : Height above sea level (meters, may be null)
- `coords.altitudeAccuracy` : Accuracy of altitude
- `coords.heading` : Direction of travel (degrees, may be null)

- `coords.speed` : Speed in meters/second (may be null)
- `timestamp` : When position was acquired

**Watch Position (Continuous Tracking):**

```
var watchId = navigator.geolocation.watchPosition(
    function(position) {
        updateMapPosition(position.coords.latitude,
position.coords.longitude);
    },
    function(error) {
        console.error(error);
    }
);

// Stop watching
navigator.geolocation.clearWatch(watchId);
```

**Use Cases:**

- Store locator (find nearest location)
- Weather based on location
- Local search results
- Delivery tracking
- Fitness tracking apps

---

# Event Handling

## Event Listeners

Modern way to handle events in JavaScript.

**Syntax:**

```
element.addEventListener(event, callback, options);
```

**Basic Example:**

```
var button = document.getElementById('myButton');

button.addEventListener('click', function() {
```

```
    console.log('Button clicked!');
});
```

**Multiple Listeners:**

```
button.addEventListener('click', handler1);
button.addEventListener('click', handler2);
// Both handlers execute when clicked
```

**Removing Listeners:**

```
function handleClick() {
    console.log('Clicked');
}

button.addEventListener('click', handleClick);
button.removeEventListener('click', handleClick);  // Must use same function
reference
```

## Common Events

**Mouse Events:**

- `click` : Element clicked
- `dblclick` : Double-clicked
- `mousedown` : Mouse button pressed
- `mouseup` : Mouse button released
- `mousemove` : Mouse moved over element
- `mouseenter` : Mouse enters element
- `mouseleave` : Mouse leaves element
- `mouseover` : Mouse over element (bubbles)
- `mouseout` : Mouse out of element (bubbles)

**Keyboard Events:**

- `keydown` : Key pressed
- `keyup` : Key released
- `keypress` : Character typed (deprecated, use keydown)

**Form Events:**

- `submit` : Form submitted

- `input` : Input value changed (real-time)
- `change` : Input value changed (on blur/submit)
- `focus` : Element focused
- `blur` : Element lost focus

**Media Events:**

- `play` : Media started playing
- `pause` : Media paused
- `ended` : Media finished
- `timeupdate` : Playback position changed
- `loadedmetadata` : Metadata loaded
- `volumechange` : Volume changed

## Event Object

Every event handler receives an event object with useful information.

```
element.addEventListener('click', function(event) {
    console.log(event.type);         // "click"
    console.log(event.target);       // Element that triggered event
    console.log(event.currentTarget); // Element listener attached to
    console.log(event.clientX);      // Mouse X coordinate
    console.log(event.clientY);      // Mouse Y coordinate

    event.preventDefault();   // Prevent default behavior
    event.stopPropagation();  // Stop event from bubbling
});
```

**Keyboard Event Properties:**

```
element.addEventListener('keydown', function(e) {
    console.log(e.key);       // Key name: "Enter", "a", "Shift"
    console.log(e.code);      // Physical key: "KeyA", "Enter"
    console.log(e.keyCode);   // Numeric code (deprecated)
    console.log(e.ctrlKey);   // Ctrl pressed?
    console.log(e.shiftKey);  // Shift pressed?
    console.log(e.altKey);    // Alt pressed?
});
```

## input vs change Events

**`input` Event:**

- Fires immediately as value changes
- Works for every keystroke, paste, etc.
- Real-time updates

```
input.addEventListener('input', function(e) {
    console.log('Current value:', e.target.value);
    // Updates with every character typed
});
```

**`change` Event:**

- Fires when element loses focus AND value changed
- Or when user commits change (e.g., select option)
- Less frequent updates

```
input.addEventListener('change', function(e) {
    console.log('Final value:', e.target.value);
    // Updates only when user leaves field
});
```

**When to Use:**

- **input**: Live search, character counters, real-time validation
- **change**: Form submissions, selecting options, less intensive operations

## Event Delegation

Handle events on parent element instead of individual children (more efficient).

```
// Instead of this (inefficient for many items):
items.forEach(item => {
    item.addEventListener('click', handleClick);
});

// Use this (single listener on parent):
parent.addEventListener('click', function(e) {
    if (e.target.matches('.item')) {
        // Handle click on item
        console.log('Clicked:', e.target.textContent);
    }
});
```

**Benefits:**

- Better performance with many elements
- Works for dynamically added elements
- Less memory usage

---

## Complete Integration Example

Here's how everything works together in a real application:

```
// ========== Application State ==========
var appState = {
    currentUser: null,
    theme: 'light',
    volume: 1.0,
    playlist: [],
    currentTrack: 0
};

// ========== Initialize from LocalStorage ==========
window.addEventListener('DOMContentLoaded', function() {
    // Load saved preferences
    var savedTheme = localStorage.getItem('theme');
    if (savedTheme) {
        appState.theme = savedTheme;
        applyTheme(savedTheme);
    }

    var savedVolume = localStorage.getItem('volume');
    if (savedVolume) {
        appState.volume = parseFloat(savedVolume);
        audio.volume = appState.volume;
    }

    // Load playlist
    var savedPlaylist = localStorage.getItem('playlist');
    if (savedPlaylist) {
        appState.playlist = JSON.parse(savedPlaylist);
        loadPlaylist();
    }
});

// ========== Save Preferences ==========
```

```
function savePreferences() {
    localStorage.setItem('theme', appState.theme);
    localStorage.setItem('volume', appState.volume);
    localStorage.setItem('playlist', JSON.stringify(appState.playlist));
}

// ========== Theme System ==========
function applyTheme(theme) {
    document.body.className = theme;
    localStorage.setItem('theme', theme);
}

document.getElementById('themeToggle').addEventListener('click', function() {
    appState.theme = appState.theme === 'light' ? 'dark' : 'light';
    applyTheme(appState.theme);
});

// ========== Audio Player with Storage ==========
var audio = document.querySelector('audio');
var volumeSlider = document.getElementById('volume');

volumeSlider.addEventListener('input', function() {
    appState.volume = volumeSlider.value;
    audio.volume = appState.volume;
    localStorage.setItem('volume', appState.volume);
});

// Remember playback position
audio.addEventListener('pause', function() {
    sessionStorage.setItem('playbackPosition', audio.currentTime);
});

// Resume from saved position
audio.addEventListener('loadedmetadata', function() {
    var savedPosition = sessionStorage.getItem('playbackPosition');
    if (savedPosition) {
        audio.currentTime = parseFloat(savedPosition);
    }
});

// ========== Geolocation for Local Content ==========
if ('geolocation' in navigator) {
    navigator.geolocation.getCurrentPosition(function(position) {
        var lat = position.coords.latitude;
        var lng = position.coords.longitude;
```

```
        // Load location-specific content
        loadLocalContent(lat, lng);
    });
}

// ========== Drag and Drop Playlist ==========
var dropZone = document.getElementById('playlist');

dropZone.addEventListener('dragover', function(e) {
    e.preventDefault();
});

dropZone.addEventListener('drop', function(e) {
    e.preventDefault();

    var files = e.dataTransfer.files;

    for (var i = 0; i < files.length; i++) {
        var file = files[i];
        if (file.type.startsWith('audio/')) {
            addToPlaylist(file);
        }
    }

    savePreferences();
});

// ========== Complete Integration ==========
// This example shows how HTML5 features work together:
// - LocalStorage for persistent preferences
// - SessionStorage for temporary state
// - Audio API for playback
// - Drag/Drop for adding content
// - Geolocation for personalization
// - Events for interactivity
```

---

## Summary

HTML5 represents a fundamental shift in web development, providing:

1. **Semantic Markup**: Better structure, accessibility, and SEO
2. **Rich Input Types**: Enhanced forms with built-in validation
3. **Native Media**: Audio and video without plugins

4. **Powerful APIs**: Storage, geolocation, drag-drop, and more
5. **Better User Experience**: Faster, more responsive applications

All these features work together to create modern, powerful web applications that rival native applications in functionality while remaining cross-platform and accessible through a simple web browser.

---


# Abdullah Ali

## Contact : +201012613453