



**Independent
Skill Development
Mission**



ISDM (INDEPENDENT SKILL DEVELOPMENT MISSION)

EMBEDDING IMAGES USING AND HANDLING RESPONSIVE IMAGES

CHAPTER 1: INTRODUCTION TO EMBEDDING IMAGES IN HTML

The Importance of Images in Web Design

Images are an essential part of modern web design, making websites visually appealing and engaging. They enhance user experience, help convey information quickly, and improve content readability. In HTML, images are embedded using the `` tag, which allows developers to insert pictures from different sources.

The `` tag is self-closing, meaning it does not require a closing tag. It contains key attributes such as:

- **src (source):** Specifies the image file location.
- **alt (alternative text):** Provides a textual description of the image for accessibility and SEO.
- **width and height:** Define the image dimensions in pixels or percentages.

A basic example of embedding an image in HTML:

```

```

In this example:

✓ The `src` attribute points to "image.jpg".

- ✓ The alt attribute describes the image, improving accessibility.
- ✓ The width and height attributes control the display size.

Without images, websites can appear plain and less engaging. However, properly optimizing and handling images ensures better user experience and performance.

CHAPTER 2: BEST PRACTICES FOR EMBEDDING IMAGES

Using Descriptive alt Text

The alt attribute is crucial for accessibility and SEO. It provides alternative text if the image fails to load and helps screen readers interpret content for visually impaired users.

Example:

```

```

If the image does not load, the browser will display the text “Company Logo” instead.

Benefits of alt text:

- ✓ Improves accessibility for visually impaired users.
- ✓ Enhances SEO by helping search engines understand image content.
- ✓ Provides context when images fail to load.

Choosing the Right Image Format

Different image formats are suited for various use cases:

- **JPEG (JPG):** Best for photographs and images with gradients.
- **PNG:** Supports transparency and is ideal for logos and icons.
- **GIF:** Suitable for animations but not ideal for complex images.
- **SVG:** Best for scalable vector graphics, such as icons.

- **WebP:** A modern format that provides high-quality images with reduced file size.

Example of using an SVG image for a scalable logo:

```

```

Using the right format ensures faster loading times and better quality.

CHAPTER 3: HANDLING RESPONSIVE IMAGES

Why Responsive Images Matter

Responsive images adjust to different screen sizes, ensuring that users on mobile, tablet, and desktop devices see properly sized images. Without responsive design, large images may slow down page speed on mobile devices, leading to poor user experience.

Using the srcset Attribute

The srcset attribute allows browsers to choose the most appropriate image based on screen resolution.

Example of srcset:

```

```

How it works:

- ✓ The browser selects "small.jpg" by default.
- ✓ If the screen width is **768px or larger**, "medium.jpg" is displayed.
- ✓ If the screen width is **1200px or larger**, "large.jpg" is used.

This technique ensures optimal image quality without unnecessary loading times.

Using CSS for Responsive Images

Another way to make images responsive is through CSS. The max-width: 100% property ensures that images do not exceed their container size.

Example:

```
<style>
```

```
img {  
    max-width: 100%;  
    height: auto;  
}
```

```
</style>
```

```

```

Benefits of using CSS for responsive images:

- ✓ Ensures images scale proportionally.
- ✓ Prevents images from overflowing the container.
- ✓ Enhances mobile-friendliness without extra code.

CHAPTER 4: CASE STUDY – OPTIMIZING IMAGES FOR AN E-COMMERCE WEBSITE

Scenario

XYZ Clothing, an online fashion retailer, faced issues with slow website loading times and poor mobile experience. Customers reported that large product images took too long to load, especially on mobile devices.

Challenges Identified

1. The website used **high-resolution images** without optimization, slowing down performance.
2. Images were **not responsive**, causing display issues on smaller screens.
3. The lack of **alt text** affected accessibility and SEO rankings.

Solution Implemented

To improve image loading speed and responsiveness, the development team:

- ✓ **Converted JPEG images to WebP** for better compression and quality.
- ✓ **Implemented srcset** to serve different image sizes based on screen resolution.
- ✓ **Added alt text** to improve accessibility and SEO.

Final Optimized Code

```

```

Results

- 📈 **40% faster page load times** on mobile devices.
- 📈 **Increased sales conversions** due to a better user experience.
- 📈 **Improved SEO ranking**, leading to higher traffic.

This case study highlights how responsive image handling can improve website performance and customer satisfaction.

CHAPTER 5: EXERCISE

Questions

1. What is the purpose of the tag in HTML?
2. How does the alt attribute improve accessibility and SEO?
3. Explain the difference between srcset and max-width: 100% for responsive images.
4. Why should WebP format be used instead of JPEG or PNG?
5. In the case study, how did XYZ Clothing benefit from responsive image optimization?

PRACTICAL TASK

- Create a **product showcase page** with the following:
 - An **optimized image** using srcset.
 - A **responsive layout** using CSS.
 - Proper **alt text** for accessibility.

USING AUDIO & VIDEO ELEMENTS (`<AUDIO>`, `<VIDEO>`) WITH DIFFERENT FORMATS

CHAPTER 1: INTRODUCTION TO AUDIO AND VIDEO ELEMENTS IN HTML

Multimedia elements play a crucial role in modern web design, enhancing user experience by incorporating **audio** and **video** content directly into webpages. The `<audio>` and `<video>` elements in HTML5 provide a standardized way to embed media without relying on external plugins like Flash. These elements enable **seamless streaming, playback controls, and cross-browser compatibility**, making them essential for websites that offer educational content, entertainment, podcasts, and interactive learning materials.

One of the significant advantages of using the `<audio>` and `<video>` elements is their support for multiple file formats, ensuring smooth playback across different devices and browsers. Common audio formats include **MP3, WAV, and Ogg**, while widely used video formats include **MP4, WebM, and Ogg Theora**. These elements also support additional attributes like `controls`, `autoplay`, `loop`, and `muted`, allowing developers to customize user interactions.

Integrating multimedia elements into a webpage enhances accessibility and engagement. Websites such as **YouTube, Spotify, and online learning platforms** extensively utilize audio and video features to deliver content. Understanding how to implement and optimize these elements is essential for web developers aiming to create immersive and interactive user experiences.

CHAPTER 2: THE `<AUDIO>` ELEMENT FOR EMBEDDING SOUND

Using the <audio> Tag for Audio Playback

The <audio> tag allows web developers to embed sound clips, background music, or voice recordings into a webpage. It supports multiple audio formats to ensure compatibility across various browsers.

Example of a Basic Audio Player

<audio controls>

```
<source src="audio-file.mp3" type="audio/mp3">
```

```
<source src="audio-file.ogg" type="audio/ogg">
```

```
<source src="audio-file.wav" type="audio/wav">
```

Your browser does not support the audio element.

</audio>

This example provides multiple formats to ensure that the audio plays on different browsers. If a browser does not support a particular format, it tries the next available format.

Supported Audio Formats and Their Use Cases

Different browsers support different audio formats, making it essential to include multiple sources:

Format	File Extension	Description	Supported Browsers
MP3	.mp3	Most commonly used, compressed, and widely supported	All modern browsers
WAV	.wav	High-quality, uncompressed format	Chrome, Edge, Safari, Firefox

Ogg	.ogg	Open-source format, good for web usage	Firefox, Chrome, Edge
-----	------	--	-----------------------

Using the <audio> element with different formats ensures that the content reaches the widest possible audience.

CHAPTER 3: THE <VIDEO> ELEMENT FOR EMBEDDING VIDEOS

Using the <video> Tag for Playing Video Files

The <video> tag allows embedding video content directly on a webpage. It supports multiple video formats, providing seamless playback across various browsers.

Example of a Basic Video Player

```
<video controls width="600">  
  <source src="video-file.mp4" type="video/mp4">  
  <source src="video-file.webm" type="video/webm">  
  <source src="video-file.ogv" type="video/ogg">  
  Your browser does not support the video element.  
</video>
```

This example ensures compatibility by providing multiple formats. The controls attribute adds play, pause, volume, and fullscreen options.

Supported Video Formats and Their Use Cases

Like audio, different browsers support different video formats:

Format	File Extension	Description	Supported Browsers
--------	----------------	-------------	--------------------

MP4	.mp4	Most widely used format, supports high compression	All modern browsers
WebM	.webm	Open-source format, optimized for web streaming	Chrome, Firefox, Edge
Ogg Theora	.ogg	Open-source format, less commonly used	Firefox, Chrome

By including multiple formats in a <video> element, developers ensure smooth playback for users regardless of their browser or device.

CHAPTER 4: CUSTOMIZING AUDIO AND VIDEO PLAYBACK

Adding Custom Controls and Autoplay Features

The <audio> and <video> elements support various attributes that allow customization of playback behavior.

Common Attributes for <audio> and <video>

Attribute	Description
controls	Adds play, pause, volume, and fullscreen controls
autoplay	Automatically starts playback when the page loads
loop	Repeats the media file indefinitely
muted	Starts playback without sound
poster (for <video>)	Displays an image before the video starts

Example with Custom Attributes

```
<video controls autoplay loop muted width="600"
poster="thumbnail.jpg">

  <source src="promo-video.mp4" type="video/mp4">

  <source src="promo-video.webm" type="video/webm">

</video>
```

This video starts automatically, loops continuously, is muted by default, and displays a **poster image** before playback.

CHAPTER 5: CASE STUDY – IMPLEMENTING A VIDEO STREAMING FEATURE

Problem Statement

An **educational platform** wanted to improve user engagement by embedding video lectures directly on its website. Initially, they relied on **YouTube embeds**, but this limited control over content delivery. The platform decided to use **HTML5 video elements** to host and stream videos seamlessly.

Implementation

1. **Uploaded lecture videos in MP4 and WebM formats** to ensure browser compatibility.
2. **Used the <video> tag with controls and poster attributes** to provide user-friendly playback.
3. **Implemented autoplay and loop features** for introductory videos on course pages.
4. **Used responsive design techniques** to optimize video playback across different screen sizes.

Final Code Implementation

```
<video controls poster="lecture-thumbnail.jpg" width="800">
```

```
<source src="lecture-video.mp4" type="video/mp4">
```

```
<source src="lecture-video.webm" type="video/webm">
```

Your browser does not support this video format.

```
</video>
```

Results

- **Increased engagement by 40%** as students found in-site videos more accessible.
- **Reduced dependency on third-party platforms**, ensuring better control over content.
- **Enhanced learning experience** through responsive and high-quality video integration.

CHAPTER 6: EXERCISE – IMPLEMENTING AUDIO AND VIDEO ELEMENTS

1. **Create an HTML page with an embedded podcast using the <audio> element.** Ensure it includes multiple formats for compatibility.
2. **Design a simple video player using the <video> element.** Add autoplay, controls, and a poster image.
3. **Implement a background music player for a portfolio website.** Ensure it plays audio continuously across different pages.
4. **Develop a streaming page for an online course platform.** Include multiple video formats for better accessibility.

SVG (SCALABLE VECTOR GRAPHICS) VS CANVAS FOR WEB GRAPHICS

CHAPTER 1: INTRODUCTION TO WEB GRAPHICS

The Importance of Web Graphics

Web graphics play a crucial role in modern web design, making websites visually appealing and interactive. Graphics are used for icons, charts, animations, data visualization, and interactive elements. Two primary technologies used for rendering graphics on the web are **SVG (Scalable Vector Graphics)** and **Canvas (HTML5 Canvas API)**. Both technologies allow developers to create rich visuals, but they function differently and serve distinct purposes.

SVG is an **XML-based** vector graphics format that maintains quality regardless of screen size or resolution. It is widely used for **icons, logos, illustrations, and complex graphical elements** that need to be scalable without pixelation.

On the other hand, Canvas provides a **bitmap-based** drawing API that allows developers to create **dynamic, pixel-based graphics** on a web page. It is commonly used for **game development, real-time visualizations, and animations** that require fast rendering.

To choose the best option for a project, developers must understand the differences in how **SVG and Canvas handle graphics, their performance, flexibility, and compatibility**.

CHAPTER 2: UNDERSTANDING SVG (SCALABLE VECTOR GRAPHICS)

What is SVG?

SVG (Scalable Vector Graphics) is an XML-based format for defining two-dimensional graphics that can be scaled without losing quality. Unlike

raster graphics (JPEG, PNG), which consist of pixels, **SVG uses mathematical equations** to define shapes, lines, and colors.

SVG images are **resolution-independent**, meaning they remain sharp on all devices, including high-resolution Retina displays. This makes them an excellent choice for icons, logos, charts, and scalable UI elements.

Features of SVG

- ✓ **Scalable without quality loss** – Ideal for high-resolution displays.
- ✓ **Editable with CSS and JavaScript** – Can be manipulated like HTML elements.
- ✓ **Small file sizes for simple graphics** – Reduces bandwidth usage.
- ✓ **Better accessibility** – Text inside SVGs is selectable and searchable.
- ✓ **Works well for static graphics** – Best for logos, charts, and vector illustrations.

Example of an SVG Image

```
<svg width="200" height="200" viewBox="0 0 100 100"
xmlns="http://www.w3.org/2000/svg">

  <circle cx="50" cy="50" r="40" stroke="black" stroke-width="2"
fill="blue"/>

</svg>
```

Explanation

- This code **creates a blue circle** with a black border.
- **cx and cy** define the circle's center.
- **r** defines the radius.
- **stroke and fill** set the border and fill colors.

SVGs are often used for **vector-based graphics**, such as UI elements, logos, and diagrams.

CHAPTER 3: UNDERSTANDING CANVAS

What is Canvas?

The HTML5 <canvas> element provides a **bitmap-based** drawing API, allowing developers to **draw and manipulate graphics dynamically** using JavaScript. Unlike SVG, which is XML-based and scales indefinitely, Canvas **renders graphics pixel by pixel** on a fixed-sized area.

Canvas is widely used in **game development, real-time animations, and data visualization** where performance and fast rendering are critical.

Features of Canvas

- ✓ **Supports real-time rendering** – Ideal for animations and interactive graphics.
- ✓ **Handles thousands of objects efficiently** – Better for complex, dynamic graphics.
- ✓ **Allows low-level drawing** – Gives developers complete control over pixels.
- ✓ **Uses JavaScript for rendering** – Requires scripting for interaction.

Example of a Canvas Drawing

```
<canvas id="myCanvas" width="200" height="200"></canvas>
```

```
<script>
```

```
    var canvas = document.getElementById("myCanvas");
```

```
    var ctx = canvas.getContext("2d");
```

```
    ctx.beginPath();
```

```
ctx.arc(100, 100, 40, 0, 2 * Math.PI);  
  
ctx.fillStyle = "blue";  
  
ctx.fill();  
  
ctx.stroke();  
  
</script>
```

Explanation

- The <canvas> element creates a **drawing space** of 200x200 pixels.
- The getContext("2d") method initializes a **2D drawing surface**.
- The arc() function **draws a circle** filled with blue color.
- The stroke() function **outlines the shape**.

Canvas is ideal for **dynamic graphics, animations, and real-time visualizations** where frame-by-frame updates are needed.

CHAPTER 4: SVG VS. CANVAS – KEY DIFFERENCES

Feature	SVG	Canvas
Rendering Type	Vector-based	Pixel-based
Performance	Best for simple, static graphics	Best for dynamic, complex graphics
Scalability	Scales without quality loss	Fixed resolution, may pixelate
Manipulation	Can be styled with CSS & JavaScript	Requires JavaScript for rendering

Interactivity	Built-in event handling	No built-in event handling
Use Cases	Logos, icons, charts, UI elements	Games, animations, real-time visualizations

✓ **Use SVG for:** Logos, UI elements, charts, scalable illustrations.

✓ **Use Canvas for:** Animations, games, data visualizations, real-time graphics.

CHAPTER 5: CASE STUDY – CHOOSING SVG VS. CANVAS FOR AN INTERACTIVE WEB DASHBOARD

Scenario

TechCorp, a data analytics company, was developing an **interactive dashboard** to visualize large datasets. The dashboard required:




1. **Static visual elements (logos, icons, navigation UI).**
2. **Real-time graphs and charts that update dynamically.**
3. **Interactive maps with zooming and panning.**

Solution

✓ **SVG was used** for the static elements (company logo, icons, UI elements) since they needed to be sharp and scalable.

✓ **Canvas was used** for the real-time charts and interactive maps since it handled large datasets efficiently.

Results

-  **40% faster rendering** of real-time charts.
-  **Improved user experience** with smooth zooming and panning.
-  **Optimized performance** by combining both technologies effectively.

This case study demonstrates how SVG and Canvas can be used **together** for an optimized web experience.

CHAPTER 6: EXERCISE

Questions

1. What is the main difference between SVG and Canvas?
2. Why is SVG preferred for logos and UI elements?
3. How does Canvas handle dynamic animations better than SVG?
4. What are some use cases where SVG and Canvas can be combined?
5. In the case study, why did TechCorp use both SVG and Canvas?

PRACTICAL TASK

- **Create an SVG-based logo** with a company name and a circular icon.
- **Use Canvas** to draw a moving animation (e.g., a bouncing ball).
- **Compare the two approaches** and identify which works better for different use cases.

IMPLEMENTING IMAGE MAPS AND FIGURE ELEMENTS IN HTML

CHAPTER 1: INTRODUCTION TO IMAGE MAPS AND FIGURE ELEMENTS

Images play a crucial role in web design, enhancing user engagement, visual appeal, and interactivity. However, in many cases, static images alone are not sufficient, and users need interactive elements for better navigation and user experience. This is where **image maps** and **figure elements** come into play.

An **image map** is a feature that allows different sections of an image to be clickable, each leading to different links or actions. This is particularly useful for interactive site maps, infographics, and advertisements. By using the `<map>` and `<area>` elements, developers can define multiple clickable regions within a single image, improving usability and navigation efficiency.

On the other hand, the `<figure>` element provides a **semantic way** to group images with captions and descriptions, improving accessibility and readability. It allows images, illustrations, and even code snippets to be grouped with `<figcaption>`, making it easier for users to understand the context of the visual representation.

These elements improve user experience and functionality, making web content more interactive and informative. Understanding how to use them effectively is essential for any web developer looking to enhance website design and usability.

CHAPTER 2: UNDERSTANDING AND IMPLEMENTING IMAGE MAPS

What is an Image Map?

An image map is an image that contains multiple clickable areas, each leading to different destinations. This is accomplished using the `<map>`

element, which defines the clickable regions, and the <area> element, which specifies the coordinates and actions of each clickable section.

How Image Maps Work in HTML

To create an image map:

1. Define the element and associate it with a <map> element using the usemap attribute.
2. Inside the <map> element, use <area> elements to define clickable regions.
3. Each <area> must have a shape attribute (rectangle, circle, or polygon) and coords (coordinates specifying the clickable region).

Example of an Image Map

```

```

```
<map name="worldmap">
```

```
  <area shape="rect" coords="50,50,200,200"  
  href="https://www.usa.gov" alt="USA">
```

```
  <area shape="circle" coords="300,150,50" href="https://www.uk.gov"  
  alt="UK">
```

```
  <area shape="poly" coords="400,100,450,150,420,200"  
  href="https://www.india.gov" alt="India">
```

```
</map>
```

Types of Shapes in Image Maps

Shape	Attribute Value	Description

Rectangle	rect	Defines a rectangular clickable area using x1, y1, x2, y2 coordinates.
Circle	circle	Defines a circular area using x, y, radius.
Polygon	poly	Defines a complex shape using multiple coordinate points.

Benefits of Image Maps

- **Enhances Navigation:** Users can click on different parts of an image to navigate different pages.
- **Improves Interactivity:** Ideal for infographics and educational diagrams.
- **Saves Space:** Multiple links can be embedded in a single image instead of using multiple separate images.

CHAPTER 3: THE <FIGURE> AND <FIGCAPTION> ELEMENTS IN HTML

What is the <figure> Element?

The <figure> element is used to group media content like images, videos, illustrations, or even code snippets along with their descriptions. It improves web accessibility and helps search engines understand the relationship between images and text.

The <figcaption> element, used inside <figure>, provides a caption or description for the image, making it more informative.

How to Use the <figure> and <figcaption> Elements

Example of a Figure with a Caption

```
<figure>
```

```
  
```

```
<figcaption>A breathtaking view of the sunset over the  
mountains.</figcaption>
```

```
</figure>
```

This example ensures that the image and its description remain together, improving readability and accessibility.

Benefits of Using `<figure>` and `<figcaption>`

- **Improves SEO:** Search engines can better understand images with proper captions.
- **Enhances Readability:** Keeps captions visually close to images.
- **Accessibility:** Helps visually impaired users using screen readers to understand image content.
- **Better Structure:** Clearly groups images with related descriptions, making the webpage more organized.

CHAPTER 4: COMBINING IMAGE MAPS WITH FIGURE ELEMENTS

Using `<figure>` with `<figcaption>` can enhance image maps by adding context to the image.

Example of an Image Map with a Figure Element

```
<figure>
```

```

```

```
<figcaption>Clickable interactive map of downtown city  
locations.</figcaption>
```

```
</figure>
```

```
<map name="citymap">

    <area shape="rect" coords="30,40,100,100" href="library.html"
alt="Library">

    <area shape="circle" coords="200,150,40" href="museum.html"
alt="Museum">

</map>
```

This combination provides a **structured and accessible** way to present an image with clickable areas and a descriptive caption.

CHAPTER 5: CASE STUDY – IMPLEMENTING AN INTERACTIVE CAMPUS MAP

Problem Statement

A **university website** needed an interactive campus map to help students locate buildings and departments. Instead of listing addresses as text, they wanted an image map that students could click on to get information about different locations.

Implementation Steps

1. **Designed a high-resolution campus map** with labeled sections.
2. **Used the <map> and <area> elements** to define clickable locations.
3. **Added <figure> and <figcaption>** for better readability and accessibility.
4. **Linked each section to relevant webpages**, such as department pages, canteen menus, and parking information.

Final Code Implementation

```
<figure>
```

```

```

```
<figcaption>Click on different buildings to get more
details.</figcaption>
```

```
</figure>
```

```
<map name="campus">
```

```
<area shape="rect" coords="50,50,150,150" href="library.html"
alt="Library">
```

```
<area shape="circle" coords="250,100,50" href="cafeteria.html"
alt="Cafeteria">
```

```
<area shape="poly" coords="400,200,450,250,500,220" href="science-
dept.html" alt="Science Department">
```

```
</map>
```

Results

- **Student navigation improved by 60%** as they could visually locate buildings.
- **Website engagement increased by 40%**, reducing the need for printed maps.
- **Enhanced accessibility**, making the map usable for visually impaired students with screen readers.

CHAPTER 6: EXERCISE – IMPLEMENTING IMAGE MAPS AND FIGURE ELEMENTS

1. **Create an interactive travel map** where users can click on different countries to learn more about them.

2. **Design an e-commerce product image map** with different clickable sections for product details, such as size, color, and availability.
3. **Implement a <figure> element** that includes an image of a historical monument with a caption describing its history.
4. **Develop a user guide image map** where users can click on different sections of a machine diagram to get more information about each part.

ADDING CAPTIONS, SUBTITLES, AND ACCESSIBILITY FEATURES

CHAPTER 1: INTRODUCTION TO CAPTIONS, SUBTITLES, AND ACCESSIBILITY

Why Are Captions and Subtitles Important?

Captions and subtitles play a crucial role in improving web accessibility by making multimedia content available to a wider audience, including individuals with hearing impairments and those who speak different languages. They also help users understand content in noisy or quiet environments where audio may not be available.

In web development, HTML5 provides built-in support for captions and subtitles using the `<track>` element within the `<video>` tag. This enables developers to add **closed captions (CC)** and **subtitles** to videos, making content more inclusive.

Beyond captions and subtitles, other accessibility features like **ARIA (Accessible Rich Internet Applications) attributes**, keyboard navigation, and screen reader compatibility ensure that websites are usable for people with disabilities. Proper implementation of these features not only enhances usability but also complies with accessibility standards such as the **Web Content Accessibility Guidelines (WCAG)**.

For example, a basic video with subtitles can be implemented as follows:

```
<video controls>
```

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

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

```
</video>
```

This example demonstrates how subtitles are added to a video, allowing users to toggle captions on and off as needed.

CHAPTER 2: UNDERSTANDING CAPTIONS AND SUBTITLES IN HTML5

Difference Between Captions and Subtitles

Although often used interchangeably, **captions** and **subtitles** serve different purposes:

✓ **Captions:** Provide a text representation of all audio elements, including dialogue, background sounds, and speaker identification. These are primarily used by individuals with hearing impairments.

✓ **Subtitles:** Display dialogue in text format but do not include background sounds or speaker identification. They are used for translation purposes or when users do not understand the spoken language.

Using the <track> Element for Captions and Subtitles

The <track> element allows developers to add captions or subtitles to videos using a **WebVTT (.vtt) file**.

Example of Adding Subtitles

```
<video controls>

  <source src="example.mp4" type="video/mp4">

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

  <track src="subtitles_es.vtt" kind="subtitles" srclang="es"
  label="Spanish">

</video>
```

Example .vtt File for English Subtitles

WEBVTT

00:00:01.000 --> 00:00:05.000

Welcome to our web development tutorial.

00:00:06.000 --> 00:00:10.000

In this lesson, we will learn about HTML5 captions and subtitles.

- ✓ **srclang="en"** specifies the language of the subtitles.
- ✓ Users can select the subtitle language using the video player controls.

This method allows video content to be **accessible and user-friendly** across different languages.

CHAPTER 3: ENHANCING ACCESSIBILITY WITH ARIA AND KEYBOARD NAVIGATION

ARIA (Accessible Rich Internet Applications)

ARIA attributes improve web accessibility by enhancing dynamic content interaction for screen readers and assistive technologies.

- ✓ **aria-label** – Provides a descriptive label for elements.
- ✓ **aria-hidden="true"** – Hides unnecessary elements from screen readers.
- ✓ **role="button"** – Helps screen readers recognize interactive elements.

Example of ARIA for Accessible Buttons

```
<button aria-label="Play Video">Play</button>
```

This ensures that assistive technologies can describe the button's function to users.

Keyboard Navigation and Focus Control

Not all users navigate websites with a mouse; some rely on keyboards. Implementing **focus management** ensures smooth navigation using the Tab key.

✓ **tabindex="0"** – Makes an element focusable with the keyboard.

✓ **tabindex="-1"** – Removes an element from the focus order.

Example of Focusable Elements

```
<a href="about.html" tabindex="0">About Us</a>
```

```
<button tabindex="0">Click Me</button>
```

Proper keyboard navigation enhances accessibility, ensuring a seamless experience for all users.

CHAPTER 4: CASE STUDY – IMPROVING VIDEO ACCESSIBILITY IN AN EDUCATIONAL PLATFORM

Scenario

EduTech, an online learning platform, faced accessibility challenges with its video lectures. Users with hearing impairments struggled to understand the lessons, and international students requested subtitles in multiple languages.

Challenges Identified

1. **No captions** – Users with hearing disabilities couldn't follow the content.
2. **Lack of multilingual subtitles** – Non-English speakers had difficulty understanding lectures.
3. **No keyboard support** – Users relying on keyboards found it hard to navigate videos.

Solution Implemented

- ✓ **Added closed captions using the <track> element** for English and Spanish subtitles.
- ✓ **Implemented ARIA attributes** to improve screen reader compatibility.
- ✓ **Enabled keyboard shortcuts** for video playback controls.

Final Optimized Code

```
<video controls>
```

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

```
  <track src="captions_en.vtt" kind="captions" srclang="en"
label="English">
```

```
  <track src="captions_es.vtt" kind="captions" srclang="es"
label="Spanish">
```

```
</video>
```

```
<button aria-label="Pause Video"
onclick="document.querySelector('video').pause()">Pause</button>
```

```
<button aria-label="Play Video"
onclick="document.querySelector('video').play()">Play</button>
```

RESULTS

- 📈 **30% increase in engagement** among hearing-impaired users.
- 📈 **Improved user satisfaction** with multilingual subtitles.
- 📈 **Enhanced accessibility compliance** with WCAG guidelines.

This case study highlights how **captions, subtitles, and ARIA attributes** significantly improve digital accessibility.

CHAPTER 5: EXERCISE

Questions

1. What is the difference between captions and subtitles?
2. How does the <track> element improve video accessibility?
3. What are ARIA attributes, and how do they help screen readers?
4. How does keyboard navigation contribute to web accessibility?
5. In the case study, how did EduTech improve accessibility for its users?

PRACTICAL TASK

- **Create a video player** with:
 - Captions and subtitles in two languages.
 - ARIA attributes for accessible controls.
 - Keyboard navigation support.

ADDING CAPTIONS, SUBTITLES, AND ACCESSIBILITY FEATURES

CHAPTER 1: INTRODUCTION TO CAPTIONS, SUBTITLES, AND ACCESSIBILITY

Why Are Captions and Subtitles Important?

Captions and subtitles play a crucial role in improving web accessibility by making multimedia content available to a wider audience, including individuals with hearing impairments and those who speak different languages. They also help users understand content in noisy or quiet environments where audio may not be available.

In web development, HTML5 provides built-in support for captions and subtitles using the `<track>` element within the `<video>` tag. This enables developers to add **closed captions (CC)** and **subtitles** to videos, making content more inclusive.

Beyond captions and subtitles, other accessibility features like **ARIA (Accessible Rich Internet Applications) attributes**, keyboard navigation, and screen reader compatibility ensure that websites are usable for people with disabilities. Proper implementation of these features not only enhances usability but also complies with accessibility standards such as the **Web Content Accessibility Guidelines (WCAG)**.

For example, a basic video with subtitles can be implemented as follows:

```
<video controls>
```

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

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

```
</video>
```


This example demonstrates how subtitles are added to a video, allowing users to toggle captions on and off as needed.

CHAPTER 2: UNDERSTANDING CAPTIONS AND SUBTITLES IN HTML5

Difference Between Captions and Subtitles

Although often used interchangeably, **captions** and **subtitles** serve different purposes:

✓ **Captions:** Provide a text representation of all audio elements, including dialogue, background sounds, and speaker identification. These are primarily used by individuals with hearing impairments.

✓ **Subtitles:** Display dialogue in text format but do not include background sounds or speaker identification. They are used for translation purposes or when users do not understand the spoken language.

Using the <track> Element for Captions and Subtitles

The <track> element allows developers to add captions or subtitles to videos using a **WebVTT (.vtt) file**.

Example of Adding Subtitles

```
<video controls>
  <source src="example.mp4" type="video/mp4">
  <track src="subtitles_en.vtt" kind="subtitles" srclang="en"
label="English">
  <track src="subtitles_es.vtt" kind="subtitles" srclang="es"
label="Spanish">
</video>
```

Example .vtt File for English Subtitles

WEBVTT

00:00:01.000 --> 00:00:05.000

Welcome to our web development tutorial.

00:00:06.000 --> 00:00:10.000

In this lesson, we will learn about HTML5 captions and subtitles.

- ✓ **srclang="en"** specifies the language of the subtitles.
- ✓ Users can select the subtitle language using the video player controls.

This method allows video content to be **accessible and user-friendly** across different languages.

CHAPTER 3: ENHANCING ACCESSIBILITY WITH ARIA AND KEYBOARD NAVIGATION

ARIA (Accessible Rich Internet Applications)

ARIA attributes improve web accessibility by enhancing dynamic content interaction for screen readers and assistive technologies.

- ✓ **aria-label** – Provides a descriptive label for elements.
- ✓ **aria-hidden="true"** – Hides unnecessary elements from screen readers.
- ✓ **role="button"** – Helps screen readers recognize interactive elements.

Example of ARIA for Accessible Buttons

```
<button aria-label="Play Video">Play</button>
```

This ensures that assistive technologies can describe the button's function to users.

Keyboard Navigation and Focus Control

Not all users navigate websites with a mouse; some rely on keyboards. Implementing **focus management** ensures smooth navigation using the Tab key.

✓ **tabindex="0"** – Makes an element focusable with the keyboard.

✓ **tabindex="-1"** – Removes an element from the focus order.

Example of Focusable Elements

```
<a href="about.html" tabindex="0">About Us</a>
```

```
<button tabindex="0">Click Me</button>
```

Proper keyboard navigation enhances accessibility, ensuring a seamless experience for all users.

CHAPTER 4: CASE STUDY – IMPROVING VIDEO ACCESSIBILITY IN AN EDUCATIONAL PLATFORM

Scenario

EduTech, an online learning platform, faced accessibility challenges with its video lectures. Users with hearing impairments struggled to understand the lessons, and international students requested subtitles in multiple languages.

Challenges Identified

1. **No captions** – Users with hearing disabilities couldn't follow the content.
2. **Lack of multilingual subtitles** – Non-English speakers had difficulty understanding lectures.

3. **No keyboard support** – Users relying on keyboards found it hard to navigate videos.

Solution Implemented

- ✓ **Added closed captions using the <track> element** for English and Spanish subtitles.
- ✓ **Implemented ARIA attributes** to improve screen reader compatibility.
- ✓ **Enabled keyboard shortcuts** for video playback controls.

Final Optimized Code

```
<video controls>
```

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

```
  <track src="captions_en.vtt" kind="captions" srclang="en"
  label="English">
```




```
  <track src="captions_es.vtt" kind="captions" srclang="es"
  label="Spanish">
```

```
</video>
```

```
<button aria-label="Pause Video"
onclick="document.querySelector('video').pause()">Pause</button>
```

```
<button aria-label="Play Video"
onclick="document.querySelector('video').play()">Play</button>
```

Results

-  **30% increase in engagement** among hearing-impaired users.
-  **Improved user satisfaction** with multilingual subtitles.
-  **Enhanced accessibility compliance** with WCAG guidelines.

This case study highlights how **captions, subtitles, and ARIA attributes** significantly improve digital accessibility.

CHAPTER 5: EXERCISE

Questions

1. What is the difference between captions and subtitles?
2. How does the <track> element improve video accessibility?
3. What are ARIA attributes, and how do they help screen readers?
4. How does keyboard navigation contribute to web accessibility?
5. In the case study, how did EduTech improve accessibility for its users?

PRACTICAL TASK

- **Create a video player** with:
 - Captions and subtitles in two languages.
 - ARIA attributes for accessible controls.
 - Keyboard navigation support.

LAZY LOADING TECHNIQUES FOR FASTER PAGE LOAD SPEED

CHAPTER 1: INTRODUCTION TO LAZY LOADING

In modern web development, optimizing page load speed is essential for improving **user experience, search engine rankings, and overall website performance**. One of the most effective techniques to achieve this is **lazy loading**, which helps defer the loading of non-critical resources like images, videos, and scripts until they are needed.

Lazy loading works by **loading only the content that is visible on the user's screen** while delaying the loading of elements that are offscreen until they come into view. This technique significantly reduces the initial page load time, saves bandwidth, and **improves website performance**, especially on **mobile devices and low-speed networks**.

Without lazy loading, a webpage loads **all elements at once**, even those that might not be immediately visible. This can slow down the website and increase the time a user has to wait before interacting with the content. By implementing lazy loading, websites can ensure that only essential content is prioritized, leading to **faster load times, improved user engagement, and reduced bounce rates**.

Web developers can implement lazy loading using various techniques, including **native lazy loading attributes in HTML, JavaScript event listeners, and third-party libraries**. Understanding these methods allows developers to build **efficient, high-performance websites** that provide seamless user experiences across different devices and network conditions.

CHAPTER 2: HOW LAZY LOADING WORKS

The Concept Behind Lazy Loading

Lazy loading delays the loading of resources **until the user needs them**, rather than loading everything when the page initially loads. This is particularly useful for media-heavy websites that include numerous images, videos, and third-party scripts.

Comparison: Traditional Loading vs. Lazy Loading

Feature	Traditional Loading	Lazy Loading
Page Speed	Slower due to loading all resources upfront	Faster since only necessary elements load initially
Bandwidth Usage	High because all images, videos, and scripts are downloaded	Reduced because only needed content is loaded
User Experience	Users may wait longer for the entire page to load	Improved because essential content loads first
SEO Benefits	May impact search engine rankings due to slow speed	Improved rankings as Google prefers fast-loading pages

How Browsers Handle Lazy Loading

Modern browsers now support **native lazy loading** for images and iframes using the `loading="lazy"` attribute. This allows web developers to enable lazy loading without relying on JavaScript or third-party libraries.

Example of Native Lazy Loading in HTML

```

```

With this implementation, the browser loads the image **only when the user scrolls close to it**, reducing the initial page load time.

CHAPTER 3: IMPLEMENTING LAZY LOADING TECHNIQUES

1. Native Lazy Loading in HTML

Modern browsers support **native lazy loading** through the `loading="lazy"` attribute, which can be applied to `` and `<iframe>` elements.

Example of Lazy Loading for Images

```

```

```

```

Example of Lazy Loading for Iframes

```
<iframe src="https://www.example.com" loading="lazy"></iframe>
```

This method is **simple, efficient, and requires no JavaScript**, but it depends on browser support.

2. Lazy Loading with JavaScript (Intersection Observer API)

For more control, developers use the **Intersection Observer API**, which detects when an element enters the viewport and then loads it.

Example of JavaScript-Based Lazy Loading

```

```

```
<script>
```

```
document.addEventListener("DOMContentLoaded", function () {
```

```
    let lazyImages = document.querySelectorAll(".lazy");
```

```
    let observer = new IntersectionObserver((entries, observer) => {
```

```
        entries.forEach(entry => {
```

```
            if (entry.isIntersecting) {
```

```
                let img = entry.target;
```



```
        img.src = img.dataset.src;

        img.classList.remove("lazy");

        observer.unobserve(img);
    }

    });

});

lazyImages.forEach(image => {

    observer.observe(image);

});

});

</script>
```

Here, images are initially set with a data-src attribute instead of src. When they come into view, JavaScript replaces data-src with src, making them load dynamically.

3. Using Third-Party Libraries for Lazy Loading

For developers looking for ready-made solutions, third-party libraries like **Lazysizes.js** provide robust lazy loading implementations.

Example Using Lazysizes.js

```
<script
src="https://cdnjs.cloudflare.com/ajax/libs/lazysizes/5.2.2/lazysizes.min.js
"></script>


```

Lazysizes automatically detects and loads images only when they are about to appear in the viewport.

CHAPTER 4: CASE STUDY – OPTIMIZING AN E-COMMERCE WEBSITE WITH LAZY LOADING

Problem Statement

An e-commerce company noticed that its website was **loading too slowly**, especially on mobile devices. The product pages contained **dozens of high-resolution images**, and loading them all at once was **causing delays and increasing bounce rates**.

Challenges Faced

1. **Slow initial page load time**, leading to a poor user experience.
2. **Increased bandwidth usage**, especially for mobile users.
3. **Lower conversion rates** due to impatient users leaving before the page fully loaded.

Solution: Implementing Lazy Loading

The development team implemented **lazy loading for product images** using the `loading="lazy"` attribute for modern browsers and a **JavaScript-based fallback for older browsers**.

Final Implementation

```

```

```
<script>
```

```
document.addEventListener("DOMContentLoaded", function () {
```

```
    let lazyImages = document.querySelectorAll(".lazyload");
```

```
let observer = new IntersectionObserver((entries, observer) => {  
  entries.forEach(entry => {  
    if (entry.isIntersecting) {  
      let img = entry.target;  
      img.src = img.dataset.src;  
      observer.unobserve(img);  
    }  
  });  
});  
  
lazyImages.forEach(image => {  
  observer.observe(image);  
});  
});  
</script>
```

Results Achieved

- **Page load time improved by 50%**, leading to **faster navigation**.
- **Bandwidth usage decreased**, improving performance for mobile users.
- **Conversion rates increased by 30%**, as users were more likely to complete purchases.

CHAPTER 5: EXERCISE – IMPLEMENTING LAZY LOADING IN WEB PROJECTS

1. **Add lazy loading to an image-heavy blog post** using the `loading="lazy"` attribute.
2. **Implement JavaScript-based lazy loading** for a gallery page using the Intersection Observer API.
3. **Optimize an iframe-heavy website** by using lazy loading for embedded videos.
4. **Compare traditional loading vs. lazy loading** by measuring page speed before and after implementation.

ASSIGNMENT SOLUTION: CREATING AN INTERACTIVE IMAGE GALLERY WITH LAZY LOADING AND RESPONSIVE MEDIA ELEMENTS

Step-by-Step Guide

Creating an interactive image gallery with **lazy loading** and **responsive media elements** ensures optimal performance, faster loading times, and a seamless user experience across different devices. This guide will walk you through **structuring, styling, and enhancing** the gallery using HTML, CSS, and JavaScript.

STEP 1: SETTING UP THE BASIC HTML STRUCTURE

Begin by creating an HTML document and setting up the gallery structure.

Code:

```
<!DOCTYPE html>

<html lang="en">
<head>
  <meta charset="UTF-8">

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

  <title>Interactive Image Gallery</title>

  <link rel="stylesheet" href="styles.css">

</head>
```

```
<body>

<h1>Interactive Image Gallery</h1>

<div class="gallery">

</div>

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

</body>

</html>
```

Explanation:

- ✓ The <h1> tag adds a heading to the gallery.
- ✓ The .gallery <div> contains multiple elements for the images.

- ✓ Each `` initially loads a **low-resolution placeholder**, and the `data-src` attribute stores the **actual image** for lazy loading.
 - ✓ The **class "lazy"** is used to target images for lazy loading.
 - ✓ The **JavaScript file (script.js)** is linked at the end to implement lazy loading functionality.
-

STEP 2: STYLING THE IMAGE GALLERY WITH CSS

Now, style the gallery for a **responsive and interactive layout**.

Code:

```
body {  
    font-family: Arial, sans-serif;  
    text-align: center;  
    background-color: #f5f5f5;  
    margin: 0;  
    padding: 0;  
}  
  
h1 {  
    margin-top: 20px;  
    font-size: 24px;  
    color: #333;  
}
```

```
.gallery {  
  display: grid;  
  grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));  
  gap: 15px;  
  padding: 20px;  
  max-width: 1000px;  
  margin: auto;  
}
```

```
.gallery img {  
  width: 100%;  
  height: auto;  
  display: block;  
  border-radius: 8px;  
  transition: transform 0.3s ease-in-out;  
}
```

```
.gallery img:hover {  
  transform: scale(1.05);  
}
```

Explanation:

- ✓ The body background is set to **light gray** for a modern look.
 - ✓ The **grid layout** (grid-template-columns) ensures **responsive arrangement** of images.
 - ✓ The auto-fit and minmax(200px, 1fr) ensure images adjust based on screen width.
 - ✓ The images **scale up slightly on hover** (transform: scale(1.05)) for an **interactive effect**.
 - ✓ The border-radius: 8px; rounds the edges for a **modern aesthetic**.
-

STEP 3: IMPLEMENTING LAZY LOADING WITH JAVASCRIPT

Lazy loading ensures images are loaded **only when they come into view**, improving performance and reducing unnecessary bandwidth usage.

Code:

```
document.addEventListener("DOMContentLoaded", function () {  
    const lazyImages = document.querySelectorAll("img.lazy");  
  
    const imageObserver = new IntersectionObserver((entries, observer) => {  
        entries.forEach(entry => {  
            if (entry.isIntersecting) {  
                let img = entry.target;  
  
                img.src = img.dataset.src; // Replace placeholder with actual  
                image  
  
                img.classList.remove("lazy");  
  
                observer.unobserve(img);  
            }  
        });  
    });
```

```
    }  
  });  
});  
  
lazyImages.forEach(img => {  
  imageObserver.observe(img);  
});  
});
```

Explanation:

- ✓ Uses IntersectionObserver to detect when an image enters the **viewport**.
- ✓ Replaces the placeholder.jpg with the actual image (data-src).
- ✓ Stops observing the image after loading to optimize performance.
- ✓ Reduces **initial page load time**, improving **user experience**.

STEP 4: ENSURING MOBILE RESPONSIVENESS

For a fully responsive gallery, add **CSS media queries**.

Code:

```
@media (max-width: 768px) {  
  .gallery {  
    grid-template-columns: repeat(auto-fit, minmax(150px, 1fr));  
  }  
}
```

Explanation:

- ✓ On **small screens**, the gallery adjusts with **smaller image sizes** for better viewing.
 - ✓ Ensures a **mobile-friendly layout**.
-

STEP 5: ADDING LIGHTBOX EFFECT FOR IMAGE ZOOM

To enhance interactivity, implement a **lightbox effect** so users can **click an image** to **view it larger**.

Updated HTML:

```
<div id="lightbox" class="hidden">  
  <span id="close">&times;</span>  
  <img id="lightbox-img" src="" alt="Expanded View">  
</div>
```

Updated CSS:

```
#lightbox {  
  position: fixed;  
  top: 0;  
  left: 0;  
  width: 100%;  
  height: 100%;  
  background: rgba(0, 0, 0, 0.8);  
  display: flex;  
  justify-content: center;
```

```
align-items: center;

visibility: hidden;

opacity: 0;

transition: opacity 0.3s ease-in-out;

}
```

```
#lightbox img {

    max-width: 80%;

    max-height: 80%;

    border-radius: 8px;

}
```

```
#lightbox.visible {

    visibility: visible;

    opacity: 1;

}
```

```
#close {

    position: absolute;

    top: 20px;

    right: 30px;

    font-size: 30px;
```

```
color: white;

cursor: pointer;

}
```

Updated JavaScript:

```
const lightbox = document.getElementById("lightbox");
const lightboxImg = document.getElementById("lightbox-img");
const closeBtn = document.getElementById("close");

document.querySelectorAll(".gallery img").forEach(img => {
  img.addEventListener("click", function () {
    lightbox.classList.add("visible");
    lightboxImg.src = this.src;
  });
});

closeBtn.addEventListener("click", function () {
  lightbox.classList.remove("visible");
});
```

Explanation:

- ✓ Clicking an image **opens it in a lightbox** for a **larger view**.
- ✓ The lightbox **fades in smoothly** with opacity transition.
- ✓ Clicking the **close button (x)** hides the lightbox.

Final Features Implemented

- ✓ **Lazy Loading** – Images load only when they appear in view.
- ✓ **Responsive Design** – Grid layout adjusts to different screen sizes.
- ✓ **Hover Effects** – Slight image zoom on hover.
- ✓ **Lightbox Effect** – Users can view images in a larger format.

CONCLUSION

By following these steps, we have successfully created a **fully interactive image gallery** that:

- 🎯 **Loads faster** using lazy loading.
- 🎯 **Adjusts to all screen sizes** (responsive design).
- 🎯 **Provides an enhanced user experience** with hover effects and a lightbox feature.

This optimized gallery is **ideal for portfolios, product showcases, and photography websites.** 🚀

Assignment Submission Checklist

- ✓ Basic HTML structure created.
- ✓ CSS for styling and responsiveness applied.
- ✓ JavaScript for lazy loading implemented.
- ✓ Lightbox effect added for interactive viewing.
- ✓ Proper testing done across different screen sizes.

ISDM-NxT