Objectives:

- Working with Multimedia
- Forms and Input Elements
- Semantic HTML
- Svg and Maps

Multimedia:

Introduction:

In HTML, we can do much more than just creating paragraphs, lists, and tables. We can also incorporate multimedia elements into our webpages to make them more engaging and interactive. For instance, we can display images, play audio files, and embed videos.

Images:

We can add images to our web page by using the `<img>` tag. This tag accepts three attributes:

- **src**: Represents the source and location of the image. It can be a URL or a relative path to the image.
- **title**: Represents the title that will be displayed when we hover over the image.
- **alt**: Provides alternative text that is displayed if the image fails to load.

Example :

```
<img src="./cat.png" title="cat image" alt="cat picture">
```

Working with Paths:

As mentioned earlier, for the `src` attribute, we can use either a URL or a file path to locate the image. In HTML, paths are relative to the current file's location. To navigate **up** to a parent directory, we use `..`. To navigate **deeper** into a subdirectory, we simply use `/`.

Example:

Let’s suppose we have the following folder structure:

```
project/
├── images/
│   └── cat.jpg
└── pages/
    └── index.html
```

If we want to select the image `cat.jpg` from the `images` folder (which is in the parent directory) while working in the `pages` folder, we would use the following path in the `src` attribute:

```
<img src="../images/example.jpg" alt="Example Image">
```

figure and figcaption:

HTML5 provides us with two additional tags for working with images and other media:
the `<figure>` tag and the `<figcaption>` tag.
The `<figure>` tag is used to encapsulate images, diagrams, illustrations, or other media content.
The `<figcaption>` tag is used to add a small descriptive text or caption to the content inside the `<figure>` tag.

```
<figure>
    <img src="cat.jpg" alt="cat image">
    <figcaption>This is cute cat image.</figcaption>
</figure>
```

Audio :

We use the `<audio>` tag to insert and play audio files on our web pages. This tag comes with several attributes to control the audio playback and behavior.

- **src:** specifies the path (URL or file path) to the audio file.
- **autoplay:** determines whether the audio will start playing automatically as soon as it is ready.
- **loop:** specifies whether the audio will repeat continuously after it ends
- **controls:** displays a small controller (play, pause, volume, etc.) for the audio file.
- **preload:** specifies how the audio file should be loaded when the page loads. It can take one of the following values:
    - `auto` : The browser should load the entire audio file.
    - `metadata` : The browser should only load metadata (e.g., duration).

    - `none` : The browser should not load the audio file until the user clicks play.

Example:

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

`<source>` tag:

When displaying audio files on a webpage, we may encounter issues where the web browser does not support the audio format. To address this problem, we can use the `<source>` tag to provide alternative audio formats. This ensures that the audio file can be played across all browsers. The `<source>` tag accepts two key attributes:

- **src:** specifies the path (URL or file path) to the audio file.
- **type:** specifies the MIME type of the audio file (e.g., `audio/mpeg` for MP3, `audio/ogg` for OGG).

```
<audio controls>
    <source src="audio.mp3" type="audio/mpeg">
    <source src="audio.ogg" type="audio/ogg">
</audio>
```

Videos:

To include videos in our webpage, HTML provides us with the `<video>` tag. This tag is similar to the `<audio>` tag and shares many of the same attributes, making it easy to embed and control video content.

- **src:** specifies the path (URL or file path) to the video file.
- **autoplay:** determines whether the video will start playing automatically as soon as it is ready.
- **loop:** specifies whether the video will repeat continuously after it ends
- **controls:** displays a small controller (play, pause, volume, etc.) for the video file.
- **preload:** specifies how the video file should be loaded when the page loads. It can take one of the following values:
    - `auto` : The browser should load the entire video file.
    - `metadata` : The browser should only load metadata (e.g., duration).
    - `none` : The browser should not load the video file until the user clicks play.
- **poster**: specifies an image to display as a placeholder before the video is played.

Example :

```
<video src="video.mp4" controls autoplay>
</video>
```

We can use the `<source>` tag for videos as well, to provide alternative source files. This ensures that if the browser does not support the current video format, it can fall back to another supported format.

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

Forms:

Introduction:

We use forms when we want to collect information from users. Examples of forms include login pages, registration pages, contact forms, and surveys. In HTML, forms are created using the `<form>` tag. Inside this tag, we can add various form elements such as input fields, buttons, and dropdowns. The `<form>` tag has two primary attributes:

- **action**: specifies the URL or path where the form data will be sent for processing.`
- **method**: specifies the HTTP method used to send the form data. The two most common methods are:
    - **`GET`** : Appends the form data to the URL (visible in the address bar). it used for non-sensitive data.
    - **`POST`** : Sends the form data in the request body (not visible in the URL). it used for sensitive or large amounts of data.

```
<form action="/login" method="post">


</form>
```

Forms Elements:

Text input:

The most common and widely used form element is the **text input**. It is a box where users can enter text. To create a text input, we use the `<input>` tag and set the `type` attribute to `"text"`.

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

Password input:

Another important form element is the **password input**. It is used to securely collect sensitive information, such as passwords, from users. To create a password input, we use the `<input>` tag and set the `type` attribute to `"password"`.

```
<input type="password">
```

Number input

HTML also allows us to create **number input fields**. These fields are used to collect numeric values from users, such as age, quantity, or ratings. To create a number input, we use the `<input>` tag and set the `type` attribute to `"number"`. This input type comes with special additional attributes, such as `min`, `max`, and `step`, which allow us to define the minimum and maximum values, as well as the increment or decrement step for the number.

```
<input type="number" min="1" max="10" step="1">
```

Email input:

We can use a **text input** to allow users to enter their email address, but HTML provides a specialized input type for this purpose: the **email input**.To create an email input, we use the `<input>` tag and set the `type` attribute to `"email"`.

```
<input type="email">
```

Search input:

The **search input** is used when we want to allow users to search for specific data. This input field often includes a list of previously searched items. To create a search input, we use the `<input>` tag and set the `type` attribute to `"search"`.

```
<input type="search" >
```

Url input:

We use the **URL input** to allow users to enter a web address (URL). It functions similarly to a text input but includes additional validation to ensure the input is a valid URL format. To create a URL input, we use the `<input>` tag and set the `type` attribute to `"url"`.

```
<input type="url">
```

Date input:

If we want to allow users to input a date as part of their information, we can use the **date input**. This input type enables users to select a date from a calendar picker, making it easy and intuitive to enter dates. To create a date input, we use the `<input>` tag and set the `type` attribute to `"date"`.

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

Color input:

The **color input** is a special input type that allows users to select a color from a color picker or palette. To create a color input, we use the `<input>` tag and set the `type` attribute to `"color"`.

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

Range input:

We use the **range input** when we want to include a slider or range selector in our webpage. This input type allows users to select a value within a specified range by dragging a slider. It is commonly used for controlling settings like volume levels in audio or video players or brightness adjustments. To create a range input, we use the `<input>` tag and set the `type` attribute to `"range"`, we also add some attributes to control the behavior and appearance of the range input.

- **min**: specifies the minimum value of the range (default is 0).
- **max**: specifies the maximum value of the range (default is `100`).
- **step**: specifies the increment or decrement step for the range. For example, if `step="10"`, the slider will move in increments of 10.

```
<input type="range" min="0" max="100" step="2">
```

Checkbox input:

Checkboxes are used when we want to allow users to select one or more options from a set of related choices. A common example is an online food ordering form, where users can use checkboxes to select extra ingredients they want to add to their order. To create a checkbox, we use the `<input>` tag and set the `type` attribute to `"checkbox"`.

```
<input type="checkbox">
```

Radio input:

Unlike checkboxes, **radio buttons** are used when we want the user to select **only one option** from a set of choices. A common example is a gender selection field in a form, where the user can choose either "Male" or "Female" but cannot select both. To create a radio button, we use the `<input>` tag and set the `type` attribute to `"radio"`.

```
<input type="radio">
```

File input:

The **file input** is used when we want to allow users to upload files to our form. This input type provides a way for users to select and upload files from their device, such as images, documents, or videos. To create a file input, we use the `<input>` tag and set the `type` attribute to `"file"`.

```
<input type="file">
```

Submit input:

The final input type is the **submit input**. This represents a button that is typically added at the end of a form. When the user clicks this button, the form data is submitted to the server for processing. To create a submit button, we use the `<input>` tag and set the `type` attribute to `"submit"`.

```
<input type="submit">
```

Input attributes:

name

The `name` **attribute** is one of the most important attributes for form elements. It is used to identify the data when the form is submitted. The value of this attribute is sent to the server, allowing the server to access and process the form data.

- For **checkboxes**, if you want multiple checkboxes to be grouped together (e.g., selecting multiple options), you should give them the **same** `name` **attribute**. When the form is submitted, the values of all selected checkboxes will be sent under the same name.
- For **radio buttons**, giving them the **same** `name` **attribute** ensures that only one option can be selected at a time. This is because radio buttons are designed to allow users to choose **either one option or another** within the same group.
  Example: checkbox

```
cheese: <input type="checkbox" name="toppings" >
pepperoni: <input type="checkbox" name="toppings">
```

Example: radio

```
male: <input type="radio" name="gender">


female: <input type="radio" name="gender">
```

value:

We use the `value` **attribute** to set a default value for form elements. This attribute is also used for **checkboxes** and **radio buttons** to specify the value that will be sent to the server when the user selects them.
Example: text input

```
<input type="text" name="username" value="JohnDoe">
```

Example: checkbox

```
<input type="checkbox" name="toppings" value="cheese">
<input type="checkbox" name="toppings" value="pepperoni">
```

Example: radio

```
<input type="radio" name="gender" value="male">
```

```
<input type="radio" name="gender" value="female">
```

required:

We use the **`required` attribute** on input fields to make them mandatory. This ensures that the user cannot submit the form without filling out the required fields.

placeholder:

We use the **`placeholder` attribute** to add placeholder text to an input field. This text provides a hint or example to the user about what data to input. The placeholder text is displayed inside the input field and disappears when the user starts typing.

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

disabled:

We use the **`disabled` attribute** to make an input field inactive. When an input field is disabled, the user cannot interact with it—they cannot select it, type into it, or modify its value.

```
<input type="text" name="username" value="Ali" disabled>
```

checked:

The final attribute we will discuss is the **`checked` attribute**. This attribute is used to pre-select a **radio button** or **checkbox** when the form loads.

```
<input type="radio" name="gender" value="male" checked>

<input type="radio" name="gender" value="female">
```

Labels:

The `<label>` tag can help us when working with forms. We can put text inside it as a label for the form element. We can even do more than that and make the browser autofocus or select the input element when we click on that label. To create a label for an element, we first create a `<label>` tag with the descriptive text inside it. Then, we set the `for` attribute of the `<label>` tag to the same value as the `id` attribute of the input element.

```
 <label for="username">Username:</label>
<input type="text" id="username" name="username" required>
```

We can omit using the `for` and `id` attributes by placing both the text and the input element inside the `<label>` tag.

```
<label>
        <input type="checkbox" name="subscribe"> Subscribe to newsletter
</label>
```

Text areas:

A **textarea** is a large text input field that allows users to enter multi-line text, such as comments or descriptions. It is created using the `<textarea>` tag. This tag supports additional attributes like `rows` and `cols`, which control the size of the textarea:

- The `rows` attribute specifies the number of visible text lines (height).
- The `cols` attribute specifies the number of visible characters per line (width).

```
<label for="comment">Comment:</label>
<textarea id="comment" name="comment" rows="5" cols="40"></textarea>
```

Select:

An additional element for our forms is the **`<select>` tag**. This element is used to create a dropdown list, allowing users to select one or more options from a list. Inside the `<select>` tag, we add `<option>` tags to define the available choices, each with its own value.

```
<label for="country">Country:</label>
<select id="country" name="country">
    <option value="usa">United States</option>
    <option value="ca">Canada</option>
    <option value="uk">United Kingdom</option>
</select>
```

datalist:

The **`<datalist>` tag** in HTML is used to provide a list of predefined options for an input field. It works with the `<input>` tag to create a dropdown list of suggestions that users can choose from, while still allowing them to enter custom text if needed. To create a `<datalist>`, we first create an input field and give it an `id`. Then, below it, we create a `<datalist>` tag and set its `id` attribute to match the `id` of the input field. Finally, inside the `<datalist>` tag, we add our options.

```
    <label for="fruit">Choose a fruit:</label>
    <input type="text" id="fruit" name="fruit" list="fruits">
    <datalist id="fruits">
        <option value="Apple">
        <option value="Banana">
        <option value="Orange">
        <option value="Mango">
        <option value="Strawberry">
    </datalist>
```

buttons:

We can create more buttons beyond just the submit button. These buttons can be created using the `<button>` tag. The `<button>` tag is versatile and allows us to create buttons with different types and functionalities, such as submit, reset, or generic buttons.

```
<button type="submit">Submit</button>
<button type="reset">Reset</button>
<button type="button">Click Me</button>
```

Semantic HTML:

Introduction

Semantic HTML is a new feature added in HTML5. It introduced special tags that are representative of their content, allowing developers to structure web pages more meaningfully. Instead of only using `<div>` tags to group and structure our webpage, it provides us with the following tags: `<header>`, `<nav>`, `<main>`, `<article>`, `<section>`, `<aside>`, and `<footer>`.

`<nav>`

We use the **`<nav>` tag** to create a navigation bar that allows users to navigate through the pages of a website. Inside the `<nav>` tag, we typically include a list of links, often structured using an unordered list (`<ul>`) and list items (`<li>`).

```
<nav>
    <ul>
        <li><a href="/">Home</a></li>
        <li><a href="/about">About</a></li>
        <li><a href="/services">Services</a></li>
        <li><a href="/contact">Contact</a></li>
    </ul>
</nav>
```

`<header>`

The `<header>` **tag** is used to create the header section of a webpage. This section typically contains introductory content, such as a logo, a brief description of the website, or an image slider that displays general information about the page. The header is the first thing users see, so it should make strong first impression.

```
<header>
    <img src="logo.png" alt="Website Logo">
    <h1>Welcome to My Website</h1>
    <p>Discover amazing content and services tailored just for you.</p>
</header>
```

`<main>`

The `<main>` **tag** comes after the `<header>` and is used to contain the **main content** of the webpage. This is where the primary information, such as articles, blog posts, or key features, is placed.

```
<main>
    <h1>About Us</h1>
    <p>We are a company dedicated to providing high-quality services.</p>

    <section>
        <h2>Our Services</h2>
        <p>Here are the services we offer...</p>
    </section>

    <article>
        <h2>Customer Testimonials</h2>
        <p>Read what our customers have to say about us.</p>
    </article>
</main>
```

`<article>`

With the `<article>` **tag**, we can add blog posts, news articles, forum posts, or product descriptions. It helps structure the content semantically, making it easier for search engines and screen readers to understand.

```
<article>
    <h2>How to Use Semantic HTML</h2>
    <p>Semantic HTML is essential for creating accessible and well-structured web
pages...</p>
</article>
```

`<section>`

We use the `<section>` **tag** to group content on our webpages and divide it into sections of related objects. Sections are generally placed inside the `<main>` tag.

```
<form>
    <section>
        <h3>Personal Information</h3>
        <label for="name">Name:</label>
        <input type="text" id="name" name="name">
    </section>
    <section>
        <h3>Contact Information</h3>
        <label for="email">Email:</label>
        <input type="email" id="email" name="email">
    </section>
</form>
```

`<aside>`

We use the `<aside>` **tag** to create content that is related to the main content but not part of it. For example, this can include sidebars, pull quotes, advertisements, or additional information that complements the primary content but is not essential to it.

`<footer>`

We use the `<footer>` **tag** to add information such as copyright notices, contact details, or links to related pages. We always place it at the bottom of the webpage.

```
<footer>
    <p>&copy; 2023 My Website. All rights reserved.</p>
    <p>Contact: <a href="mailto:info@example.com">info@example.com</a></p>
    <nav>
        <ul>
            <li><a href="/privacy">Privacy Policy</a></li>
            <li><a href="/terms">Terms of Service</a></li>
        </ul>
    </nav>
```

Comment

Comments are used to add hints and notes for ourselves or other developers to read and remember what we were doing. They can also be used to create a "to-do" block, reminding us or other developers of tasks that need to be completed or features that need to be added. Comments will not be displayed to users who visit our page. We create comments using `<!-- -->`. Anything placed between these symbols will be treated as a comment.

```
<!--
comment here
-->
```

`<details>` and `<summary>`

We use the `<details>` and `<summary>` tags when we want to create an interactive element that displays a description and reveals more details when the user clicks on it. To create this, we first add the `<details>` tag. Inside it, we place the `<summary>` tag, which contains the text that will be displayed to the user. After the `<summary>` tag, we add the content that we want to reveal when the user clicks on the summary.

```
<details>
    <summary>Click to expand</summary>
    <p>This is the hidden content that will be revealed when the summary is
```

```
clicked.</p>
</details>
```

Svg and Maps:

Svg:

SVG (Scalable Vector Graphics) is a special HTML5 feature that allows us to draw vector images directly within a webpage. These images are unique because they are **resolution-independent**, meaning they can be resized without losing quality. Unlike raster images (e.g., JPEG, PNG), SVG images do not become pixelated when zoomed in, making them ideal for logos, icons, and other graphics that need to look sharp on all devices and screen sizes.

Creating Svg:

We create SVG images by using the **`<svg>` tag** with `height` and `width` attributes to define the size of the image. Inside the `<svg>` tag, we can create and draw various elements such as circles, rectangles, lines, and paths to build the desired graphic.

Drawing lines

We draw lines using the **`<line>` tag** in SVG. To draw a line, we need to specify the following basic attributes:

- **x1**: Represents the starting x-coordinate of the line.
- **x2**: Represents the ending x-coordinate of the line.
- **y1**: Represents the starting y-coordinate of the line.
- **y2**: Represents the ending y-coordinate of the line.
- **stroke**: Represents the color of the line.
- **stroke-width**: Represents the width of the line.
  Example:

```
<svg width="200" height="200">
    <line x1="10" y1="10" x2="190" y2="190" stroke="black" stroke-width="2" />
</svg>
```

The origin `(0, 0)` is at the **top-left corner** of the SVG canvas. Increasing the **x-coordinate** moves to the **right**, and increasing the **y-coordinate** moves **downward**.

Drawing rectangles

We draw rectangles using the `<rect>` **tag** in SVG. This tag accepts the following attributes:

- **x**: The starting x-coordinate of the rectangle.
- **y**: The starting y-coordinate of the rectangle.
- **width**: Defines the width of the rectangle.
- **height**: Defines the height of the rectangle.
- **fill**: The fill color of the rectangle (how the rectangle will be filled).
- **stroke**: The color of the rectangle's border.
- **stroke-width**: The width of the rectangle's border.
  Example:

```
<svg width="200" height="200">
    <rect x="10" y="10" width="180" height="180" fill="lightblue" stroke="black"
stroke-width="2" />
</svg>
```

Drawing circle

If we want to draw circle we use the `<circle>` **tag** in SVG. This tag accepts the following attributes:

- `cx` : The x-coordinate of the center of the circle.
- `cy` : The y-coordinate of the center of the circle.
- `r` : The radius of the circle.
- `fill` : The fill color of the circle (how the circle will be filled).
- `stroke` : The color of the circle's border.
- `stroke-width` : The width of the circle's border.
  Example:

```
<svg width="200" height="200">
    <circle cx="100" cy="100" r="50" fill="lightblue" stroke="black" stroke-
width="2" />
</svg>
```

Writing text:

We add text to SVG graphics using the `<text>` **tag**. This tag allows us to display text within an SVG image and accepts the following attributes:

- `x` : The x-coordinate of the starting point of the text.

- `y` : The y-coordinate of the baseline of the text.
- `font-family` : The font family of the text (e.g., Arial, Times New Roman).
- `font-size` : The size of the text.
- `fill` : The fill color of the text.
- `stroke` : The color of the text's outline.
- `stroke-width` : The width of the text's outline.
  Example:

```
<svg width="200" height="100">
    <text x="10" y="50" font-family="Arial" font-size="20" fill="black">Hello!
</text>
</svg>
```

Working with path

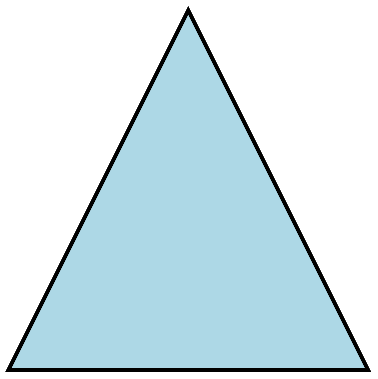
Sometimes, we want to create more **complex shapes** and objects beyond simple circles or rectangles. For this, we can use the **`<path>` tag** in SVG. The `<path>` tag allows us to define intricate shapes using a series of commands and coordinates. These commands describe how to move, draw lines, and create curves to form the desired shape.
The **`<path>` tag** in SVG uses several attributes to define and style complex shapes:

- **fill**: specifies the color used to fill the shape.
- **stroke**: specifies the color of the shape's outline.
- **stroke-width**: specifies the thickness of the shape's outline.

4. **d**: this special attribute contains a series of commands and coordinates that define how the shape is drawn. The commands include:
   - `M` **(Move To)**: Moves the pen to a specific point without drawing.
   - `L` **(Line To)**: Draws a straight line to a specific point.
   - `C` **(Curve To)**: Draws a cubic Bézier curve.
   - `S` **(Smooth Curve To)**: Draws a smooth cubic Bézier curve.
   - `Z` **(Close Path)**: Closes the path by drawing a line back to the starting point.
   Example:
   creating triangle

```
<svg width="200" height="200">
    <path d="M100 10 L190 190 L10 190 Z" fill="lightblue" stroke="black" stroke-
width="2" />
</svg>
```

- `M100 10` : Move the pen to the starting position `(100, 10)`.
- `L190 190` : Draw a line from the current position to `(190, 190)`.
- `L10 190` : Draw a line from the current position to `(10, 190)`.
- `Z` : Close the path by drawing a line back to the starting point.
  output:



Maps:

The `<map>` feature is a special tool that can be added to images to make them more interactive and dynamic. It allows you to define specific areas within an image, and when users click on those areas, they can be directed to specific links or perform certain actions. This feature is particularly useful for creating image-based navigation, diagrams, or interactive infographics.

Creating map:

To create a map, first, add an image to your webpage and give it a special attribute called `usemap`. The value of `usemap` should be the name of the map, starting with a hashtag (e.g., `#mapname`). After this, create the `<map>` element and give it a `name` attribute that matches the value you used in `usemap`. Inside the `<map>` element, add `<area>` tags. These tags are used to define the clickable areas within the image that users can interact with.

```
<img src="world-map.jpg" alt="World Map" usemap="#worldmap">
<map name="worldmap">
 <!--
 here we declare the areas
 -->
</map>
```

Creating area:

The `<area>` tag accepts four key attributes:

1. `href` : Specifies the link that the user will navigate to when clicking on the area.
2. `alt` : Provides alternative text that describes the area. This text is displayed if the image cannot be loaded and is also used by screen readers for accessibility.
3. `shape` : Defines the shape of the clickable area. It can be one of the following:
    - `rect` (rectangle)
    - `circle` (circle)
    - `poly` (polygon)
4. `coords` : Specifies the coordinates of the area. The format of the coordinates depends on the chosen shape:
    - For `rect` : `x1,y1,x2,y2` (top-left and bottom-right corners).
    - For `circle` : `x,y,radius` (center coordinates and radius).
    - For `poly` : `x1,y1,x2,y2,x3,y3,...` (vertices of the polygon).
    Among the previous shape Polygons (`poly`) are the most powerful because they can be customized to create complex and irregular shapes. However, creating polygons can be difficult and challenging, especially when determining the precise coordinates for each vertex. To simplify this process, we can use online tools like Image-Map.net, which allow you to visually draw and generate the coordinates for your polygons.
    Example :

```
  <img src="world-map.jpg" alt="World Map" usemap="#worldmap">
  <map name="worldmap">
    <area shape="rect" coords="50,50,200,200" href="north-america.html" alt="North
America">

    <area shape="circle" coords="300,150,50" href="europe.html" alt="Europe">
    <area shape="poly" coords="400,50,450,100,400,150,350,100" href="asia.html"
alt="Asia">
  </map>
```

Task

Objective:

Create a basic travel website with multiple pages, incorporating multimedia, forms, semantic HTML, and SVG.

Pages:

1. **Home Page (index.html):**
    - Show a header with a navigation bar.

- Include a large image or a video showcasing a travel destination.
- Add a brief introduction to the website using semantic HTML (`<main>` , `<article>` , `<section>`).
- Include a form to subscribe to a newsletter (email input, submit button).
- Add a footer with copyright information and links to the "About Us" and "Contact Us" pages.

2. **Destinations Page (destinations.html):**
   - Show a header with a navigation bar.
   - Display a list of travel destinations. Each destination should be presented as a `<figure>` with an `<img>` and a `<figcaption>` .
   - Use an image map on one of the destination images to link to more details about specific areas within that destination.
   - Include a simple audio player with a relaxing travel-themed song.
3. **Booking Page (booking.html):**
   - Show a header with a navigation bar.
   - Create a booking form with the following fields:
     - Name (text input, required, with placeholder)
     - Email (email input, required)
     - Destination (select dropdown with a few destination options)
     - Travel Date (date input)
     - Number of Travelers (number input, with min and max)
     - Comments (textarea)
     - Preferences (checkboxes for options like "Window Seat," "Vegetarian Meal")
     - Travel Type (radio buttons for "Business," "Leisure")
     - Submit button
   - Use `<label>` tags to clearly associate labels with form fields.
4. **About Us Page (about.html):**
   - Show a header with a navigation bar.
   - Provide information about the travel agency using semantic HTML (`<main>` , `<article>` , `<section>`).
   - Include an SVG graphic (e.g., a simple airplane or globe) created using `<svg>` , `<path>` , or other SVG elements.
   - Use `<details>` and `<summary>` to add expandable sections for FAQs.
5. **Contact Us Page (contact.html):**
   - Show a header with a navigation bar.
   - Include contact information (address, phone, email).
   - Add a simple contact form (name, email, message textarea, submit button).

Requirements:

- **Multimedia:**
  - Use `<img>`, `<audio>`, and `<video>` tags.
  - Demonstrate correct use of `src`, `alt`, `title`, `controls`, `autoplay`, `loop`, `poster`, and `<source>` tags.
  - Use `<figure>` and `<figcaption>` where appropriate.
  - Practice using relative file paths for images and other media.
- **Forms:**
  - Implement all the form elements mentioned in the lecture (text, password, number, email, etc.).
  - Use appropriate attributes (`name`, `value`, `required`, `placeholder`, `disabled`, `checked`).
  - Use `<label>`, `<textarea>`, `<select>`, `<option>`, `<datalist>`, and `<button>` tags.
  - Organize the forms logically.
- **Semantic HTML:**
  - Structure the pages using `<header>`, `<nav>`, `<main>`, `<article>`, `<section>`, `<aside>` (if applicable), and `<footer>`.
  - Use comments (``) to annotate your code.
  - Employ `<details>` and `<summary>` where suitable.
- **SVG and Maps:**
  - Create a simple SVG graphic.
  - Implement an image map on one of the images.
  - Use the `<map>`, `<area>`, `usemap`, `shape`, and `coords` attributes.

Solution:

You can find our solution here:
https://alitigui.github.io/Front_end_solutions/Lecture3/solution/index.html