Web Designing Assignment Term-1

Module (HTML5)-3

1. **What are the new tags added in HTML5?**

Ans. HTML5 introduced several new tags to improve the semantic structure of web pages and support new functionalities. Here's a list of some of the key new tags added in HTML5:

1. Structural Tags:
    - **<header>:** Represents introductory content typically at the beginning of a section or a page.
    - **<nav>:** Defines a section of navigation links.
    - **<section>:** Represents a generic section of a document or application.
    - **<article>:** Represents an independent piece of content, such as a blog post or a news article.
    - **<aside>:** Represents content that is tangentially related to the content around it, such as sidebars.
    - **<footer>:** Represents a footer for its nearest sectioning content or root element.
    - **<main>:** Represents the main content of the <body> of a document or application.
    - <**figure>:** Represents self-contained content, such as images, videos, or diagrams.
    - **<figcaption>:** Represents a caption or legend for the content of a <figure>.
2. Media Tags:
    - **<audio>:** Embeds sound content into documents.
    - **<video>:** Embeds video content into documents.
3. Interactive Tags:
    - **<details>:** Represents a disclosure widget from which the user can obtain additional information or controls.

    - **<summary>:** Represents a summary or a legend for the content of a <details> element.
    - **<dialog>:** Represents a dialog box or other interactive component.
4. Form Tags:
    - **<datalist>:** Specifies a list of pre-defined options for input controls.
    - **<output>:** Represents the result of a calculation or user action.
5. Semantic Tags:
    - **<time>:** Represents a specific period in time or a duration.
    - **<mark>:** Represents text highlighted for reference purposes.
    - **<progress>:** Represents the progress of a task.
    - **<meter>:** Represents a scalar measurement within a known range.
6. Other Tags:
    - **<canvas>:** Used to draw graphics, animations, or other visual images on the fly.
    - **<ruby>:** Represents a ruby annotation, used for showing pronunciation or providing additional annotations to East Asian text.

These tags were introduced to provide more semantic meaning to the structure of web documents, improve accessibility, and facilitate the development of web applications with richer multimedia and interactive capabilities.

2. How to embed audio and video in a webpage?

Ans. Embedding Audio

To embed audio in a webpage, you use the <audio> element. Here's a basic example:

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

- <audio>: This element is used to embed audio content into the document.
- controls: This attribute displays audio controls such as play, pause, and volume.
- <source>: This is a child element of <audio> that specifies the audio file to be played and its type.
    - src: Specifies the URL of the audio file.

    - type: Specifies the MIME type of the audio file.

In the example above:

- Replace "audio-file.mp3" with the path to your audio file.
- You can include multiple <source> elements to provide different formats (e.g., MP3, OGG) for broader browser compatibility.

Embedding Video

To embed video in a webpage, you use the <video> element. Here's a basic example:

```
<video controls width="640" height="360">
<source src="video-file.mp4" type="video/mp4">
Your browser does not support the video tag.
</video>
```

- <video>: This element is used to embed video content into the document.
- controls: This attribute displays video controls such as play, pause, and volume.
- width and height: These attributes specify the dimensions of the video player.
- <source>: This is a child element of <video> that specifies the video file to be played and its type.
    - src: Specifies the URL of the video file.
    - type: Specifies the MIME type of the video file.

In the example above:

- Replace "video-file.mp4" with the path to your video file.
- Similar to <audio>, you can include multiple <source> elements to provide different video formats (e.g., MP4, WebM) for broader browser compatibility.

Additional Attributes

Both <audio> and <video> elements support additional attributes and events for customization and interaction, such as:

- autoplay: Automatically starts playback of the media when the page loads.
- loop: Causes the media to replay from the beginning when finished.
- preload: Specifies how much of the media data to preload (none, metadata, auto).

3. Semantic elements in HTML5?

Ans. Semantic elements in HTML5 are tags that provide meaning to the content they contain, making it easier for both browsers and developers to understand the structure and purpose of the web page. These elements are designed to replace generic tags that were traditionally used for layout purposes without conveying any specific meaning. Here are some of the key semantic elements introduced in HTML5:

1. <header>:
    - Represents introductory content or a group of introductory content.
    - Typically contains navigation links, headings, logos, etc.
2. <nav>:
    - Defines a section of navigation links.
    - Intended to contain major navigation blocks.
3. <main>:
    - Represents the main content of the <body> of a document or application.
    - Should not include content that is repeated across multiple pages, such as navigation links.
4. <section>:
    - Represents a thematic grouping of content, typically with a heading.
    - Used to divide a document into sections that can be styled or navigated separately.
5. <article>:
    - Represents independent, self-contained content, such as a blog post or a news article.
    - Should be able to be distributed independently from the rest of the content.
6. <aside>:
    - Represents content that is tangentially related to the content around it, such as sidebars or pull quotes.
    - Typically used for content that is not essential to the main content but enhances it.
7. <footer>:
    - Represents a footer for its nearest sectioning content or root element.

    - Typically contains information about its section such as authorship, copyright information, etc.
8. <figure> and <figcaption>:
    - <figure> represents self-contained content, such as images, videos, diagrams, etc.
    - <figcaption> represents a caption or legend for the content in <figure>.
9. <time>:
    - Represents a specific period in time or a duration either as a date and time or as a machine-readable format.
10. <mark>:

- Represents text highlighted for reference purposes, typically rendered with a yellow background by browsers.

11. <details> and <summary>:

- <details> represents a disclosure widget from which the user can obtain additional information or controls.
- <summary> represents a summary or a legend for the content of a <details> element.

These semantic elements help improve the accessibility, SEO, and maintainability of web pages by providing clear structure and meaning to the content they contain. They also help assistive technologies better understand and navigate the content, improving the overall user experience.

4. Canvas and SVG tags.

Ans. Certainly! The <canvas> and <svg> tags in HTML5 are both used for creating graphics and visual elements on web pages, but they serve different purposes and have distinct methods of implementation.

<canvas>

The <canvas> element provides a drawing surface that allows you to render graphics dynamically using JavaScript. Here are the key characteristics and usage of <canvas>:

- Pixel-Based Drawing: <canvas> provides a bitmap-based rendering context. This means you can draw directly onto the canvas using JavaScript to create shapes, lines, text, images, and animations.

- JavaScript API: It is manipulated through the Canvas API, which provides methods and properties for drawing and manipulating graphics. Common methods include getContext('2d') for 2D graphics and getContext('webgl') for 3D graphics.
- Rendering: All drawing operations (like drawing shapes, paths, text, etc.) are immediate; once drawn, they are fixed onto the canvas.
- Use Cases: <canvas> is ideal for creating complex animations, games, data visualizations, and interactive graphics that require real-time updates and pixel-level control.

Example of <canvas> usage:

```
<canvas id="myCanvas" width="400" height="200"></canvas>
<script>
    var canvas = document.getElementById('myCanvas');
    var ctx = canvas.getContext('2d');

    ctx.fillStyle = 'green';
    ctx.fillRect(10, 10, 100, 100);
    </script>
```

In this example:

- <canvas> element creates a 400px by 200px drawing area.
- JavaScript (getContext('2d')) accesses the 2D drawing context.
- ctx.fillRect() draws a filled rectangle on the canvas.

<svg>

The <svg> element uses XML to define scalable vector graphics. It differs from <canvas> in several ways:

- Vector-Based Graphics: SVG uses XML tags to define shapes, paths, text, and other graphical elements. These elements are scalable without losing quality, making SVG ideal for graphics that need to be resized or printed at different resolutions.
- DOM Integration: SVG elements are part of the document object model (DOM), which means they can be styled with CSS, manipulated with JavaScript, and are accessible to screen readers and search engines.

- Graphics Description: You define shapes and paths using SVG-specific elements like <rect>, <circle>, <path>, <text>, etc., along with attributes like x, y, width, height, fill, stroke, etc.
- Use Cases: SVG is suitable for creating icons, logos, illustrations, charts, maps, and other graphical content that requires scalability and accessibility.

Example of <svg> usage:

```
<svg width="400" height="200">
<rect x="50" y="50" width="100" height="100" fill="blue" />
<circle cx="250" cy="100" r="50" fill="red" />
<text x="150" y="150" fill="black">SVG Example</text>
</svg>
```

In this example:

- <svg> element creates a 400px by 200px SVG drawing area.
- <rect> and <circle> elements define a blue rectangle and a red circle.
- <text> element displays text within the SVG.

Choosing Between <canvas> and <svg>

- Complexity: Use <canvas> for dynamic and complex animations or graphics that require real-time rendering and pixel-level control.
- Scalability: Use <svg> for graphics that need to be scalable and maintain clarity at different sizes and resolutions.
- Interactivity: Both <canvas> and <svg> can be interactive with JavaScript, but <canvas> is typically used for animations and games, while <svg> is more suitable for interactive graphical elements and data visualizations.