Module (HTML5)-3

1. What are the new tags added in HTML5?

- HTML5 introduced several new tags to enhance the functionality and semantic structure of web pages. Here is a list of the key new tags added in HTML5:
- **Structural and Semantic Tags**
- **<article>**: Represents independent content, such as a blog post or news article.
- **<section>**: Defines a section of a document, typically with a heading.
- **<nav>**: Represents a section of navigation links.
- **<header>**: Specifies the header of a document or section.
- **<footer>**: Defines the footer of a document or section.
- **<aside>**: Represents content indirectly related to the main content, such as sidebars.
- **<main>**: Represents the main content of the document, excluding headers, footers, and sidebars.
- **<figure>**: Encapsulates media content, like images or diagrams, along with captions.
- **<fig caption>**: Provides a caption for the content inside <figure>.
- **Multimedia Tags**
- **<audio>**: Embeds audio content.
- **<video>**: Embeds video content.
- **<source>**: Specifies multiple media resources for <audio> and <video>.
- **<track>**: Provides text tracks for <video> and <audio> elements, like subtitles.
- **Interactive and Application Tags**
- **<canvas>**: Used for drawing graphics and animations via JavaScript.
- **<svg>**: Embeds scalable vector graphics.
- **<mark>**: Highlights text for reference or importance.
- **<progress>**: Represents the completion progress of a task.
- **<meter>**: Displays a scalar measurement, such as a temperature or disk usage.
- **<time>**: Represents a specific point in time or duration.
- **<output>**: Displays the result of a calculation or user action.
- **Form-Related Tags**
- **<data-list>**: Provides an autocomplete feature for input elements.

- **<keygen>** *(Deprecated)*: Used for generating key pairs in forms.
- **<output>**: Displays calculation results from scripts.
- **<details>**: Used to create a disclosure widget for additional details.
- **<summary>**: Specifies a summary or legend for the <details> element.
- These tags make HTML5 more semantic, interactive, and capable of handling multimedia and modern web functionalities.

2. How to embed audio and video in a webpage?
- The <video> tag is used to embed video content. Like <audio>, it supports multiple source formats for compatibility.
- **Example:**

- html
- <video controls width="600" height="400">
- <source src="video-file.mp4" type="video/mp4">
- <source src="video-file.webm" type="video/webm">
- Your browser does not support the video element.
- </video>

- **Explanation:**
- **controls**: Adds playback controls like play, pause, and volume.
- **width and height**: Set the dimensions of the video.
- **<source>**: Specifies the video file and its format.
- Fallback text is displayed if the browser doesn't support the <video> element.
- ---
- **Optional Attributes**
- For both <audio> and <video>, you can use the following attributes:
- **autoplay**: Starts playback automatically when the page loads.
- **loop**: Repeats the media continuously.
- **muted**: Starts playback with the audio muted.
- **preload**: Suggests how the browser should load the media:
- auto (default): Loads the media when the page loads.
- metadata: Loads only metadata like duration.
- none: Doesn't load until the user initiates playback.
- **Example with Autoplay and Loop:**

- Html
- <video autoplay loop muted>
- <source src="background-video.mp4" type="video/mp4">
- Your browser does not support the video element.
- </video>

- ---

- By using these tags, you can effectively add multimedia content to your webpage, enhancing user engagement.

3. Semantic element in HTML5?

- **Semantic elements** in HTML5 are elements that clearly define their meaning and role in the context of a webpage. They improve the readability of the HTML code for developers and enhance accessibility and SEO by providing meaningful structure to content.
  List of Semantic Elements in HTML5
  Structural Elements

1. **<header>**: Represents the header of a document or section, often containing introductory content or navigation links.
2. **<nav>**: Defines a block of navigation links.
3. **<section>**: Represents a thematic grouping of content, typically with a heading.
4. **<article>**: Represents independent content, such as a blog post or news article.
5. **<aside>**: Represents content indirectly related to the main content, like sidebars or advertisements.
6. **<footer>**: Represents the footer of a document or section, typically containing metadata, navigation links, or copyright information.
7. **<main>**: Represents the main content of the document, excluding headers, footers, and sidebars.

Content-Grouping Elements

8. **<figure>**: Groups media content like images, diagrams, or illustrations with an optional caption.
9. **<fig caption>**: Provides a caption for content inside a <figure>.
10. **<details>**: Used to create a collapsible section of content.
11. **<summary>**: Defines a visible heading for a <details> element.

12. **<dialog>**: Represents a dialog box or interactive component, such as a pop-up or modal.

Text-Level Semantic Elements

13. **<mark>**: Highlights or marks text for reference or emphasis.
14. **<time>**: Represents a specific time or duration.
15. **<abbr>**: Represents an abbreviation or acronym, often with a title attribute to provide the full form.
16. **<address>**: Represents contact information for its nearest ancestor or the entire document.
17. **<cite>**: Represents the title of a work or citation reference.

Benefits of Using Semantic Elements

1. **Improved Accessibility**: Helps screen readers and assistive technologies better understand content.
2. **Enhanced SEO**: Search engines can more effectively parse the content structure.
3. **Code Readability**: Makes the HTML code easier to read and maintain.
4. **Consistent Styling**: Allows easier application of styles by targeting specific semantic elements.

- By incorporating semantic elements, developers create web pages that are both user-friendly and well-structured.

4. Canvas and SVG tags

- Both <canvas> and <svg> are used for creating graphics in HTML5. However, they differ significantly in how they work and their use cases.
-

- **1. <canvas>**
- The <canvas> element is used for **drawing graphics** dynamically via JavaScript. It is a bitmap-based rendering method.
- **Key Features**
- Resolution-dependent (graphics may lose quality when scaled).
- Procedural drawing through JavaScript.
- Ideal for animations, game graphics, or real-time visualizations.
- No built-in support for event handling (requires JavaScript for interactivity).
- **Example**
- html

- <canvas id="myCanvas" width="400" height="200" style="border:1px solid #000;"></canvas>
- <script>
- const canvas = document.getElementById("myCanvas");
- const ctx = canvas.getContext("2d");
-
- // Draw a rectangle
- ctx.fillStyle = "blue";
- ctx.fillRect(50, 50, 150, 100);
-
- // Draw a circle
- ctx.beginPath();
- ctx.arc(200, 100, 40, 0, 2 * Math.PI);
- ctx.fillStyle = "red";
- ctx.fill();
- </script>

- **Pros**
- High performance for rendering complex and dynamic graphics.
- Lightweight and fast for animations and games.
- **Cons**
- Graphics are not scalable (resolution-dependent).
- Requires JavaScript for drawing and interaction.
- ---
- **2. <svg>**
- The <svg> element is used for creating **scalable vector graphics**. It is XML-based and supports declarative graphics creation.
- **Key Features**
- Resolution-independent (scales without losing quality).
- Declarative markup for static images.
- Ideal for logos, icons, and graphs.
- Built-in support for event handling.
- **Example**
- html

- <svg width="400" height="200" style="border:1px solid #000;">
- <!-- Draw a rectangle -->
- <rect x="50" y="50" width="150" height="100" fill="blue" />
-
- <!-- Draw a circle -->
- <circle cx="250" cy="100" r="40" fill="red" />
- </svg>

- **Pros**
- Scalable and resolution-independent graphics.
- Better for static images with interactivity.
- Easier to style with CSS.
- **Cons**
- Performance may degrade for highly dynamic or complex scenes.
- ---

- **Comparison of <canvas> and <svg>**

Feature	<canvas>	<svg>
Type	Bitmap-based graphics	Vector-based graphics
Rendering	Procedural (JavaScript required)	Declarative (XML markup)
Scalability	Resolution-dependent	Resolution-independent
Interactivity	Requires JavaScript	Built-in event handling
Use Case	Animations, games, dynamic graphics	Logos, icons, charts, and graphs
Performance	High for dynamic rendering	Best for static or moderately dynamic

- ---

- **When to Use**
- **Use <canvas>** for games, dynamic visualizations, or high-performance animations.
- **Use <svg>** for scalable, static images like charts, icons, and illustrations.
- Both are powerful tools, and the choice depends on the specific requirements of your project.