HTML 5

1) HTML Audio Tag

-> The HTML <audio> elemnts is used to play an audio file on aweb page.

-> There are three supported file format for HTML 5 audio tag

A) mp3

B) wav

C) ogg

-> syntax :

```
<audio controls>
    <source src="audio.mp3" type="audio"> audio is here
</audio>
```

-> <audio>:

The main tag for embedding audio.

controls:

Adds playback controls like play, pause, and volume.

<source>:

Speciflies the audio file and its type.

-> ATtributes

A) controls :

Display the built audio controls.

B) autoplay :

automaticaly starts playing the audio when the page loads.

C) loop :

plays the audio continuously in a loop.

D) muted :

starts the audio in a muted state

E) preloadd :

specific if and how the audio should be preload

2) HTML Video Tag

-> The HTML <video> element is used to embed videos on a web page.

-> There are three supported file format for HTML 5 video tag

A) mp4

B) wab5

C) ogg

-> syntax:

```
<video controls>
  <source src="video.mp4" type="video/mp4">
  Your browser does not support the video element.
</video>
```

-> <video>:

The main tag for embedding video.

controls:

Adds playback controls like play, pause, and volume.

<source>:

Specifies the video file and its type.

-> Attributes:

A) controls:

Displays built-in video controls.

B) autoplay:

Automatically starts playing the video when the page loads.

C) loop:

Plays the video continuously in a loop.

D) muted:

Starts the video in a muted state.

E) preload:

Specifies if and how the video should be preloaded.

D) src :

It specifies the source URL of the video file.

F) width :

It is used to set the width of the video player

E) poster :

It specifies the image which is displayed on the screen when the video is not played.

3) HTML SVG

-> The HTML SVG stands for Scalable Vector Graphics.

-> HTML SVG is a modularized language which is used to describe graphics in XML.

-> SVG is mostly used for vector type diagrams like pie charts, 2-Dimensional graphs

in an X,Y coordinate system etc.

-> SVG Shapes:

A) circle :

- Syntax

```
<svg width="200" height="200">
    <circle cx="100" cy="100" r="50" fill="blue" />
</svg>
```

- cx:

  X-coordinate of the circle's center.

- cy:

  Y-coordinate of the circle's center.

- r:

  Radius of the circle.

B) Rectangle :

- Syntax

```
<svg width="200" height="200">
  <rect x="50" y="50" width="100" height="100" fill="green" />
</svg>
```

- x, y:
Top-left corner of the rectangle.
- width, height:
Dimensions of the rectangle.

C) Line:

- Syntax

```
<svg width="200" height="200">
  <line x1="0" y1="0" x2="200" y2="200" stroke="red" stroke-width="2" />
</svg>
```

- x1, y1:
Starting point.
- x2, y2:
Ending point.
- stroke:
Line color.

D) Polygon:

- Syntax

```
<svg width="200" height="200">
  <polygon points="50,150 150,150 100,50" fill="orange" />
</svg>
```

- points:
List of X,Y coordinates for the polygon's vertices.

E) Path:

- Syntax

```
<svg width="200" height="200">
  <path d="M10 80 C 40 10, 65 10, 95 80 S 150 150, 180 80" stroke="purple" fill="transparent" />
</svg>
```

- d:
Path commands for complex shapes (ex, M for move, L for line, C for curve).

-> Advantages of SVG:

Scalability:
SVG images do not lose quality when resized.
Styleable:
Can be styled with CSS (e.g., changing colors, adding effects).
Interactive:
Supports interactivity and animations using JavaScript or CSS.
Lightweight:
Smaller file size compared to bitmap images for simple graphics.

4) SVG Gradient

-> A gradient is a smooth transition from one color to another.

-> In addition, several color transitions can be applied to the same element.

-> There are two main types of gradients in SVG:

A) Linear Gradient:

- Creates a gradient that transitions along a straight line.

-EX

```
<svg width="200" height="200">
  <defs>
    <linearGradient id="linearGradient1" x1="0%" y1="0%" x2="100%" y2="0%">
      <stop offset="0%" style="stop-color:blue;stop-opacity:1" />
      <stop offset="100%" style="stop-color:red;stop-opacity:1" />
    </linearGradient>
  </defs>
  <rect x="10" y="10" width="180" height="180" fill="url(#linearGradient1)" />
</svg>
```

B) Radial Gradient:

- Creates a gradient that radiates outward in a circular pattern.

- EX

```
<svg width="200" height="200">
  <defs>
    <radialGradient id="radialGradient1" cx="50%" cy="50%" r="50%">
      <stop offset="0%" style="stop-color:yellow;stop-opacity:1" />
      <stop offset="100%" style="stop-color:green;stop-opacity:1" />
    </radialGradient>
  </defs>
  <circle cx="100" cy="100" r="80" fill="url(#radialGradient1)" />
</svg>
```

5) HTML Canvas

-> The <canvas> element in HTML is used to draw graphics, animations, and other visual content using JavaScript. It's a versatile tool for creating dynamic, programmatically controlled visual elements on a web page.

-> EX

<canvas id="myCanvas" width="400" height="300" style="border:1px solid #000;"></canvas>

-> Common Canvas API Methods:

A) **Drawing Rectangles:**

- fillRect(x, y, width, height): Fills a rectangle.
- strokeRect(x, y, width, height): Outlines a rectangle.
- clearRect(x, y, width, height): Clears part of the canvas.
- EX

```
<script>
ctx.fillStyle = 'green';
ctx.fillRect(20, 20, 150, 100);
ctx.strokeStyle = 'red';
ctx.strokeRect(20, 20, 150, 100);
ctx.clearRect(30, 30, 50, 50);
</script>
```

B) Drawing Lines:

- beginPath(): Starts a new path.
- moveTo(x, y): Moves the starting point
- .lineTo(x, y): Draws a line to a specified point
- .stroke(): Renders the outline of the path.
- EX

```
<script>
    ctx.beginPath();
    ctx.moveTo(50, 50);
    ctx.lineTo(200, 50);
    ctx.lineTo(200, 200);
    ctx.closePath(); // Connects the last point to the first
    ctx.stroke(); // Draws the outline
</script>
```

C) Drawing Circles and Arcs:
-> arc(x, y, radius, startAngle, endAngle, counterclockwise): Draws an arc or circle.
-> EX

```
     <script>
    ctx.beginPath();
    ctx.arc(150, 150, 50, 0, Math.PI * 2); // Full circle
    ctx.fillStyle = 'yellow';
    ctx.fill();
    ctx.stroke();
</script>
```

D) Drawing Text:
-> fillText(text, x, y): Fills text.s
-> trokeText(text, x, y): Outlines text.
-> EX

```
<script>
    ctx.font = '30px Arial';
    ctx.fillStyle = 'black';
    ctx.fillText('Hello, Canvas!', 50, 50);
    ctx.strokeStyle = 'blue';
    ctx.strokeText('Hello, Canvas!', 50, 100);
</script>
```

E) Inserting Images:
-> drawImage(image, x, y, width, height): Draws an image onto the canvas.
-> EX

```
 <img id="myImage" src="image.jpg" style="display:none;">
<script>
    const img = document.getElementById('myImage');
    img.onload = function () {
        ctx.drawImage(img, 10, 10, 200, 150);
    };
</script>
```

Advantages of Canvas:
Flexible Graphics: Supports freehand drawing, animations, and complex visualizations.
High Performance: Suitable for gaming and interactive visuals.
Customizable: Can be styled and manipulated entirely with JavaScript.

6) URL Encode

-> URL stands for Uniform Resource Locator. It is actually a web address.

-> Syntax

scheme://prefix.domain:port/path/filename

-> A) scheme :

is used to define the type of Internet service (most common is http or https)

B) prefix :

is used to define a domain prefix (default for http is www).

C) domain :

is used to define the Internet domain name

D) port :

is used to define the port number at the host (default for http is 80).

E) path :

is used to define a path at the server (If omitted: the root directory of the site).

F) filename ;

is used to define the name of a document or resource

URL ENCODE :

-> URL encoding is used to convert non-ASCII characters into a format that can be used over the Internet because a URL is sent over the Internet by using the ASCII character-set only. If a URL contains characters outside the ASCII set, the URL has to be converted.

-> In URL encoding, the non-ASCII characters are replaced with a "%" followed by hexadecimal digits.

-> URLs cannot contain spaces. URL encoding normally replaces a space with a plus (+) sign, or %20.

-> Ex. © will be replaced with %C2%A9

7) Difference between HTML and XHTML

1) HTML

-> Standard markup language for web pages.

-> Less strict, forgiving of errors.

-> Tags and attributes are not case-sensitive

-> Some tags (e.g., <img>,
) can omit /.

-> Attributes may or may not use quotes.

-> No strict rules for attribute names.

-> Can use <!DOCTYPE HTML> or omit it.

-> Browsers are forgiving; minor errors are ignored.

-> More lenient with whitespace.

-> Optional (e.g., <img>).

-> EX

```
<!DOCTYPE html>
<html>
<head>
    <title>HTML Example</title>
</head>
<body>
        <img src="image.jpg">
        <p>This is an HTML example.</p>
</body>
</html>
```

-> **Advantages of HTML:**

Flexibility: Easier for beginners, as browsers are forgiving of errors.

Widespread Support: Compatible with all browsers and devices.

Less Overhead: Less strict rules mean faster development.

2) XHTML

-> A stricter, XML-based version of HTML.

-> Strict and follows XML rules.

-> Tags and attributes must be in lowercase.

-> All tags must be properly closed (e.g., <img />).

-> All attribute values must be quoted.

-> Attribute names must follow XML naming rules.

-> Must have a <!DOCTYPE> declaration.

-> Browsers require well-formed syntax.

-> Treats whitespace more strictly as per XML.

-> Mandatory (e.g., <img />).

-> EX

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
 <head>
     <title>XHTML Example</title>
  </head>
   <body>
       <img src="image.jpg" alt="Example image" />
         <p>This is an XHTML example.</p>
    </body>
    </html>
```

-> **Advantages of XHTML:**

Strict Syntax: Enforces cleaner, more consistent code.
XML Compatibility: Easier integration with XML tools and technologies.
Future-proof: Encourages better coding practices.

8) API's in HTML

-> API stands for Application Programming Interface.

-> A Web API is an application programming interface for the Web

-> A Browser API can extend the functionality of a web browser.

-> A Server API can extend the functionality of a web server.

-> API's help to access data

-> **Common HTML APIs**

A) Geolocation API

- Allows web applications to access the user's geographic location (with user permission).
- **Use Cases**:

Maps, location-based services.

- <button onclick="getLocation()">Get Location</button>

```
<p id="output"></p>
<script>
    function getLocation() {
        if (navigator.geolocation) {
            navigator.geolocation.getCurrentPosition((position) => {
                document.getElementById("output").innerText =
                    `Latitude: ${position.coords.latitude}, Longitude: ${position.coords.longitude}`;
            });
        } else {
            alert("Geolocation is not supported by your browser.");
        }
    }
</script>
```

B) Web Storage API

- With web storage, web applications can store data locally within the user's browser.
- Web storage is more secure, and large amounts of data can be stored locally, without affecting website performance
- **Types**:

localStorage:

Data persists even after the browser is closed.

sessionStorage:

Data persists only until the browser tab is closed.

-> <button onclick="storeData()">Store Data</button>
<button onclick="retrieveData()">Retrieve Data</button>

<script>
  function storeData() {
    localStorage.setItem("name", "John Doe");
    alert("Data Stored!");
  }
  function retrieveData() {
    const name = localStorage.getItem("name");
    document.getElementById("storageOutput").innerText = `Stored Name: ${name}`;
  }
</script>

C) Canvas API
- Used for rendering 2D graphics and animations.
- **Use Cases**:
  Games, charts, custom graphics.
- **Example**:
  <canvas id="myCanvas" width="200" height="100" style="border:1px solid;"></canvas>
<script>
  const canvas = document.getElementById("myCanvas");
  const ctx = canvas.getContext("2d");
  ctx.fillStyle = "blue";
  ctx.fillRect(20, 20, 100, 50);
</script>

D) **4. Drag and Drop API**
- Enables drag-and-drop functionality for HTML elements.
- **Use Cases**: File uploads, interactive UIs.
- **Example**:
<div id="drag" draggable="true" style="width:100px; height:100px; background:blue;"
  ondragstart="drag(event)">
<div id="drop" style="width:200px; height:200px; border:1px solid black;"
  ondrop="drop(event)" ondragover="allowDrop(event)">
<script>
  function allowDrop(event) {
    event.preventDefault();
  }

  function drag(event) {
    event.dataTransfer.setData("text", event.target.id);
  }

  function drop(event) {
    event.preventDefault();
    const data = event.dataTransfer.getData("text");
    event.target.appendChild(document.getElementById(data));
  }
</script>

E) Media API

Allows control over audio and video playback.

Use Cases: Custom media players.

Example:

```
<video id="myVideo" width="320" height="240" controls>
    <source src="video.mp4" type="video/mp4">
</video>
<button onclick="playVideo()">Play</button>
<button onclick="pauseVideo()">Pause</button>
<script>
    const video = document.getElementById("myVideo");
    function playVideo() {
        video.play();
    }
    function pauseVideo() {
        video.pause();
    }
</script>
```

6. File API

Provides file-related operations, such as reading file content.

Use Cases: File uploads, file previews.

Example:

```
<input type="file" id="fileInput" />
<pre id="fileOutput"></pre>

<script>
    document.getElementById("fileInput").addEventListener("change", function(event) {
        const file = event.target.files[0];
        const reader = new FileReader();
        reader.onload = function(e) {
            document.getElementById("fileOutput").textContent = e.target.result;
        };
        reader.readAsText(file);
    });
</script>
```

7. Notification API

Allows web apps to send notifications to the user.

Use Cases: Alerts, reminders.

Example:

```
<button onclick="showNotification()">Show Notification</button>

<script>
    function showNotification() {
        if (Notification.permission === "granted") {
            new Notification("Hello! This is a notification.");
        } else if (Notification.permission !== "denied") {
```

```
 Notification.requestPermission().then((permission) => {
            if (permission === "granted") {
                new Notification("Hello! This is a notification.");
            }
        });
    }
  }
</script>
```

8. Fetch API

Allows fetching resources (e.g., data from a server).

Use Cases: Making HTTP requests.

Example

```
<button onclick="fetchData()">Fetch Data</button>
<pre id="fetchOutput"></pre>

<script>
   function fetchData() {
      fetch("https://jsonplaceholder.typicode.com/posts/1")
         .then((response) => response.json())
         .then((data) => {
            document.getElementById("fetchOutput").textContent = JSON.stringify(data, null, 2);
         });
   }
</script>
```