HTML5 Audio & Video

<audio>

HTML5 introduced the **<audio>** element to include audio files in your pages.
The **<audio>** element has a number of attributes which allow you to control audio playback:

src
This attribute specifies the path to the audio file.

controls
This attribute indicates whether the player should display controls. If you do not use this attribute, no controls will be shown by default. You can also specify your own controls using JavaScript.

autoplay
The presence of this attribute indicates that the audio should start playing automatically. (It is considered better practice to let visitors choose to play audio.)

preload
This attribute indicates what the browser should do if the player is not set to **autoplay**.

loop
This attribute specifies that the audio track should play again once it has finished.

<video>

The **<video>** element is used to add video to webpage. It has a number of attributes which allow you to control video playback:

src
This attribute specifies the path to the video.

poster
This attribute allows you to specify an image to show while the video is downloading or until the user tells the video to play.

width, height
These attributes specify the size of the player in pixels.

controls
When used, this attribute indicates that the browser should supply its own controls for playback.

autoplay
When used, this attribute specifies that the file should play automatically.

loop
When used, this attribute indicates that the video should start playing again once it has ended.

preload
This attribute indicates what the browser should do if the player is not set to **autoplay**.

In HTML5 you do not need to supply values for all attributes, such as the controls, autoplay, preload and loop. These attributes are like on / off switches. If the attribute is present, it turns that option on. If the attribute is omitted, the option is turned off.
If the browser does not support the **<video>** element or the format of video used, it will display whatever is between the opening **<video>** and closing **</video>** tags.

Refer audio.html & video.html

Canvas

HTML5 element <canvas> gives you an easy and powerful way to draw graphics using JavaScript. It can be used to draw graphs, make photo compositions or do simple (and not so simple) animations. The <canvas> element is only a container for graphics. You must use JavaScript to actually draw the graphics.

A canvas is a rectangular area on an HTML page. By default, a canvas has no border and no content. Here is a simple <canvas> element which has only two specific attributes width and height plus all the core HTML5 attributes like id, name and class, etc.

```
<canvas id = "mycanvas" width = "100" height = "100"></canvas>
```

The <canvas> is initially blank, and to display something, a script first needs to access the rendering context and draw on it. The canvas element has a DOM method called getContext, used to obtain the rendering context and its drawing functions. This function takes one parameter '2d'.

Canvas has several methods for drawing paths, boxes, circles, text, and adding images.

canvas.html

```
<!DOCTYPE HTML>
<html>
  <head>
    <script type = "text/javascript">
      function drawShape()
            {
        // Get the canvas element using the DOM
        var canvas = document.getElementById('mycanvas');

        // Make sure we don't execute when canvas isn't supported
        if (canvas.getContext)
              {
          // use getContext to use the canvas for drawing
          var ctx = canvas.getContext('2d');

          // Draw shapes
          ctx.fillRect(0,0,100,100);
          ctx.clearRect(25,25,50,50);
          ctx.strokeRect(100,100,50,50);

          ctx.moveTo(170, 10);
          ctx.lineTo(170, 100);
          ctx.stroke();

          ctx.beginPath();
          ctx.arc(300, 100, 50, 0, 2 * Math.PI);
          ctx.stroke();
        }
        else
          alert('Your does not support canvas element');
      }
    </script>
  </head>

  <body  onload = "drawShape();">
    <canvas id = "mycanvas" width="400" height="200" style="border:1px solid
    black">Your browser does not support the HTML5 canvas tag.</canvas>
  </body>

</html>
```

HTML5 Geolocation API

HTML5 Geolocation API lets you share your location with your favorite web sites. A JavaScript program can then capture your latitude and longitude and can be sent to backend web server and do fancy location-aware things like finding local businesses or showing your location on a map. Today most of the browsers and mobile devices support Geolocation API.

geolocation.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Geolocation</title>
  </head>

  <body>
        <button onclick="getLocation()">Get Location</button>
        <p id="msg"></p>
        <script>
                var x = document.getElementById("msg");

       function getLocation()
       {
                if (navigator.geolocation)
                        navigator.geolocation.watchPosition(showPosition , errorHandler);
                else
                        x.innerHTML = "Geolocation is not supported by this browser.";
                }

                function showPosition(position)
                {
                        x.innerHTML = "Latitude: " + position.coords.latitude +
                                "<br>Longitude: " + position.coords.longitude;
                }

                function errorHandler(error)
                {
                        if(error.code==error.PERMISSION_DENIED)
                                x.innerHTML = "Location request denied";
                        else if(error.code==error.POSITION_UNAVAILABLE)
                                x.innerHTML = "Location information not available";
                        else if(error.code==error.TIMEOUT)
                                x.innerHTML = "Location request timed out";
                        else if(error.code==error.UNKNOWN_ERROR)
                                x.innerHTML = "Unknown error occured";
                }
        </script>
  </body>
</html>
<!-- Difference between getCurrentPosition() & watchPosition() -->
```

CSS3 and Responsive Web Design

Responsive web design (RWD) is an approach to web design that makes web pages look good on a variety of devices (desktops, tablets and phones) and screen sizes. In Responsive Web Design we use HTML and CSS to automatically resize, hide, shrink, or enlarge a website to make it look good on all devices (desktops, tablets and phones). The goal of responsive design is to build web pages that detect the visitor's screen size and orientation and change the layout accordingly. RWD is implemented through:

- Fluid grids, where elements are resized through the use of relative units, like percentages.
- Flexible images, which are also sized in relative units
- Media queries, which allow a website to determine device type, screen size and browser capabilities, enabling the delivery of different style rules based on these characteristics

Viewport

Viewport relates to the content area within the browser window, excluding the toolbars, tabs, and so on. It's important to understand that viewport and screen size are not the same thing. More succinctly, it relates to the area where a website actually displays. Screen size refers to the physical display area of a device.

Media Queries: Supporting Differing Viewports

CSS3 Media queries can be used to check many things, such as:

- width and height of the viewport
- width and height of the device
- orientation (is the tablet/phone in landscape or portrait mode?)
- resolution

Using Media Query module, with just a few lines of CSS we can change the way content displays based upon things such as viewport width, screen aspect ratio, orientation (landscape or portrait), and so on.

Fluid Layout / Liquid Layout

A fluid layout is a type of webpage design in which layout of the page resizes as the window size is changed. This is accomplished by defining areas of the page using percentages instead of fixed pixel widths.

Most webpage layouts include one, two, or three columns. In the early days of web design, when most users had similar screen sizes, web developers would assign the columns fixed widths. For example, a fixed layout may include three columns that have widths of 200px, 600px, and 200px. While this layout might look great on a 1024x768 screen, it might look small on a 1920x1080 screen and would not fit on a 800x600 screen.

Fluid layouts solve this problem by using percentages to define each area of the layout. Consider the following example:

Fixed Layout	Fluid Layout
.left, .right { width: 200px; } .middle { width: 600px; }	.left, .right { width: 20%; } .middle { width: 60%; }

Refer mq_bgcolor.html, mq_minmax.html

mq_navbar.html

```
<!DOCTYPE html>
<head>
	<title>Media Queries - Navigation Bar</title>
	<style>
	* {
	font-family: verdana;
	font-size: 20px;
	color: white;
	}

	ul {
	list-style-type: none;
	margin: 0;
	padding: 0;
	}

	li {
	float: left;
	display: block;
	padding: 8px;
	background-color: green;
	text-align: center;
	}
	a {
	text-decoration: none;
	}
	@media screen and (max-width: 600px) {
	li {
		float: none;
		border: solid 1px white;
		width: 100%;
	}
	}
</style>
</head>
<body>
<ul>
  <li><a href="#">Home</a></li>
  <li><a href="#">SPA</a></li>
  <li><a href="#">OOPM</a></li>
  <li><a href="#">DS</a></li>
  <li><a href="#">OCJP</a></li>
</ul>
</body>
</html>
```

mq_columns.html

```
<!DOCTYPE html>
<head>
	<title>Media Queries - Columns</title>
	<style>
	* {
	box-sizing: border-box;
	 }

	/* Create four equal columns that floats next to each other */
	.column {
	float: left;
	width: 20%;
	padding: 20px;
	}

	@media screen and (max-width: 800px) {
	.column {
	width: 50%;
	}
	#ip {
	display:none;
	}
	}

	@media screen and (max-width: 600px) {
	.column {
	width: 100%;
	}
	}
 </style>
</head>
<body>
  <h2>Important Programming Subjects</h2>

  <section class="column" style="background-color:#aaa;">
  <p>SPA is a subject of sem 2 and contains the C programming language.
  It is the foundation of programming.</p>
  </section>

  <section class="column" style="background-color:#bbb;">
  <p>DS & AOA are subjects of sem 3 and  4 respectively and
  very important subjects for placement in dream companies.</p>
  </section>

  <section class="column" style="background-color:#ccc;">
  <p>OOPM is a subject of sem 3 and contains Java Programming.
  It is the first subject where you will learn java programming. </p>
  </section>

  <section class="column" style="background-color:#ddd;">
  <p>OCJP stands for Oracle Certified Java Programmer. It is not a
  subject of MU. It is an international certification exam on Java</p>
  </section>
```

```
   <section class="column" id="ip" style="background-color:#eee;">
   <p>This is a subject of sem 5. It is one of the best subjects of MU.
   In this subject you will learn HTML, CSS, Javascript, JQuery, Php. </p>
   </section>

 </body>
</html>
```

mq_image.html

```
<!DOCTYPE html>
<head>
        <title>Media Queries - Fluid Images</title>
        <style>
         img {
        float: left;
        width: 50%;
        }
        @media screen and (max-width: 750px) {
        img {
        width: 100%;
        }
        }
  </style>
 </head>
 <body>
        <img src="images/ds.jpg" title="DS Batch Mulund" />
        <img src="images/oopm.jpg" title="OOPM Batch Mulund" />
   <p>DS & OOPM Batches in Study Circle Mulund.</p>
 </body>
</html>
```

mq_orientation.html

```
<!DOCTYPE html>
<html>
<head>
        <title>Media Queries - Orientation</title>
        <style>
                body {
                background-color: lightgreen;
                }
                @media only screen and (orientation: portrait) {
                body {
                background-color: lightblue;
                }
                }
        </style>
</head>
<body>
        <h2>Internet Programming</h2>
        <p>Orientation is portrait when height is more than width, else orientation is
        landscape.</p>
</body></html>
```