Module No. 1 – Part 1

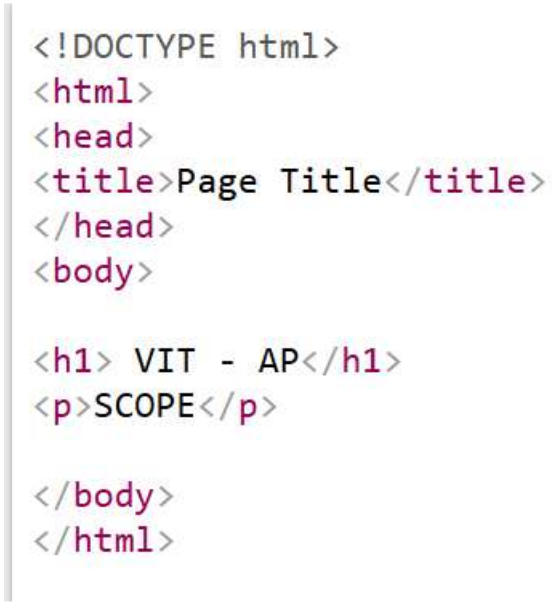
Introduction to Web Design

Introduction to HTML 5, syntax, attributes, events, SVG, Web storage, Introduction to Canvas, Audio & Video, Geolocations, Drag & Drop, Web workers, working with Fonts, working with other graphics.

Introduction to HTML 5

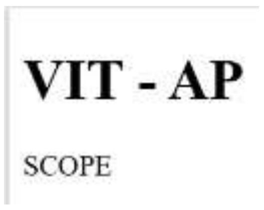
- **HTML5**
- HTML5 provides details of all 40+ HTML tags including audio, video, header, footer, data, data list, article etc.
- HTML5 is a next version of HTML.
- Here, you will get some brand-new features which will make HTML much easier. These new introducing features make your website layout clearer to both website designers and users.
- **What is HTML?**
- HTML stands for Hyper Text Markup Language.
- HTML is the standard markup language for creating Web pages.
- HTML describes the structure of a Web page.
- HTML elements tell the browser how to display the content.
- HTML consists of a series of elements.

HTML Basic Example



```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>

<h1> VIT - AP</h1>
<p>SCOPE</p>

</body>
</html>
```

Output:



```
VIT - AP

SCOPE
```

- **<u>From the above Example:</u>**

- The <!DOCTYPE html> declaration defines that this document is an HTML5 document.
- The <html> element is the root element of an HTML page.
- The <head> element contains meta information about the HTML page.
- The <title> element specifies a title for the HTML page (which is shown in the browser's title bar or in the page's tab).
- The <body> element defines the document's body, and is a container for all the visible contents, such as headings, paragraphs, images, hyperlinks, tables, lists, etc.
- The <h1> element defines a large heading.
- The <p> element defines a paragraph.

Syntax

- **<u>What is an HTML Element?</u>**

- **An HTML element is defined by a start tag, some content, and an end tag:**
  <tagname> Content goes here... </tagname>
- **The HTML element is everything from the start tag to the end tag:**
  <h1>My First Heading</h1>
  <p>My first paragraph. </p>

Start tag	Element content	End tag
<h1>	My First Heading	</h1>
<p>	My first paragraph.	</p>
<br>	*none*	*none*

HTML Basic Tags

- HTML Basic Tags are the fundamental elements of HTML used for defining the structure of the document. These are letters or words enclosed by angle brackets (< and >). Usually, most of the HTML tags contains an opening and a closing tag. Each tag has a different meaning and the browser reads the tags and displays the contents enclosed by it accordingly.

Tags	Description
<!--...-->	Specifies a comment. Comments are annotations within the code that are note displayed in the browser window.
<!DOCTYPE>	Specifies the document type. This is the initial declaration required for the specification of document type and html version.
<html>	The html tags are the container that contains all the other HTML tags. It consist of opening and closing angle brackets surrounding a tag name.
<head>	Specifies information about the document. Html <head> is a tag specifies as a metadata container.
<title>	Specifies the document title. <title> is placed within the <head> specifies the concise and descriptive label that appears in the browser's title bar or tab when the webpage is opened.
<body>	Specifies the body element. The HTML <body> tag serves as the container for all visible content within a webpage.
<h1> to <h6>	Specifies header 1 to header 6. It represents the main title or headline of a section or the entire page.
<p>	Specifies a paragraph. This tag serves as a container for blocks of text and content specifying the web page.
<br>	Inserts a single line break. It is a self-closing element used to create a forced line break within text content.

Attributes

- **HTML attributes are special words placed inside the opening tag of an HTML element to define its characteristics. Each attribute has two parts:**
    - Attribute name
    - Attribute value (separated by an equal sign = and enclosed in double quotes " ").
- **All HTML elements can have attributes:**
    - The href attribute of <a> specifies the URL of the page the link goes to.
    - The src attribute of <img> specifies the path to the image to be displayed.
    - The width and height attributes of <img> provide size information for images.
    - The alt attribute of <img> provides an alternate text for an image.
    - The style attribute is used to add styles to an element, such as color, font, size, and more.
    - The lang attribute of the <html> tag declares the language of the Web page.
    - The title attribute defines some extra information about an element.

HTML attributes are special words used within an element's opening tag to provide **additional information** or configure its behavior and display. They consist of a name and a value pair, typically written as `name="value"`.

Key Characteristics

- **Syntax:** Attributes are always included within the start tag of an HTML element (e.g., `<a href="...">`).
- **Name/Value Pairs:** Most attributes use a name, an equal sign, and a value enclosed in quotes (double quotes are recommended for safety and consistency, but single quotes are also valid).
- **Boolean Attributes:** Some attributes, like `disabled` or `required`, are "boolean" and only require the attribute's name to be present to indicate a true value. They do not need a value, or can use the name as their value (e.g., `<input required>`).

- **Purpose:** Attributes modify an element's default functionality, provide links to resources, define styles, or enable accessibility features.

Common HTML Attributes

Attributes can be categorized as global (usable on most elements) or specific to certain elements.

Global Attributes

These attributes can be used on nearly all HTML elements:

- **`class`** : Specifies one or more class names for an element, primarily used for styling with CSS or selecting with JavaScript.
- **`id`** : Specifies a unique identifier for an element, allowing specific targeting with CSS or JavaScript.
- **`style`** : Allows for inline CSS styling declarations to be applied directly to an element.
- **`title`** : Provides extra information about an element, often displayed as a tooltip when the user hovers over it.
- **`data-*`** : Used to store custom data private to the page or application.
- **`lang`** : Declares the language of the element's content, which helps with accessibility and search engines.
- **`hidden`** : A boolean attribute that prevents an element from being displayed.

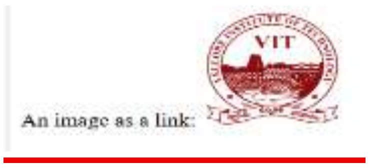
Element-Specific Attributes

These attributes are essential for the specific functionality of certain tags:

- **`href` (for `<a>` and `<link>`):** Specifies the destination URL of a hyperlink or linked resource.
- **`src` (for `<img>` , `<audio>` , `<video>` , `<script>`):** Specifies the source file or URL for media or script files.
- **`alt` (for `<img>`):** Provides alternative text for an image, essential for screen readers and when the image cannot be displayed.

- **width and height (for <img>, <canvas>, <video>, etc.):** Specifies the dimensions of the element.
- **value (for <input>, <button>, <option>):** Defines the initial or submitted value of a form control or list item.
- **disabled (for form elements):** A boolean attribute that makes the form element unusable and un-clickable.
- **Note : Example Programs Required for above all.**

```
<!DOCTYPE html>
<html>
<body>
<p>
An image as a link: <a href="https://www.vitap.ac.in/">
<img border="0" alt="VIT-AP" src=" " width="100" height="100">
</a>
</p>
</body>
</html>
```



HTML forms

- HTML forms, defined using the <form> tag, are essential for collecting user input on web pages. They include interactive controls like text fields, emails, passwords, checkboxes, radios, and buttons.
- **Example :**

```
<html>
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>HTML Form</title>
  <style>
Page Centering -- Centers the form vertically and horizontally.
    body {
      display: flex;
      justify-content: center;
      align-items: center;
      height: 100vh;
      margin: 0;
      background-color: #f0f0f0;
    }
Form Styling -- Adds a white card-style form, Soft shadow and Rounded corners.
    form {
      width: 400px;
      background-color: #fff;
      padding: 20px;
      border-radius: 8px;
      box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    }
```

<fieldset> groups related form elements.

<legend> names the group.

```
fieldset {
   border: 1px solid black;
   padding: 10px;
   margin: 0;
}
legend {
   font-weight: bold;
   margin-bottom: 10px;
}
label {
   display: block;
   margin-bottom: 5px;
}
```

Input Boxes -- Makes all inputs the same width & adds padding and space between inputs.

```
input[type="text"],
input[type="email"],
input[type="password"],
textarea,
input[type="date"] {
   width: calc(100% - 20px);
   padding: 8px;
   margin-bottom: 10px;
   box-sizing: border-box;
   border: 1px solid #ccc;
   border-radius: 4px;
}
```

Radio Button Alignment -- Keeps the labels (Male/Female/Others) inline with the radio buttons.

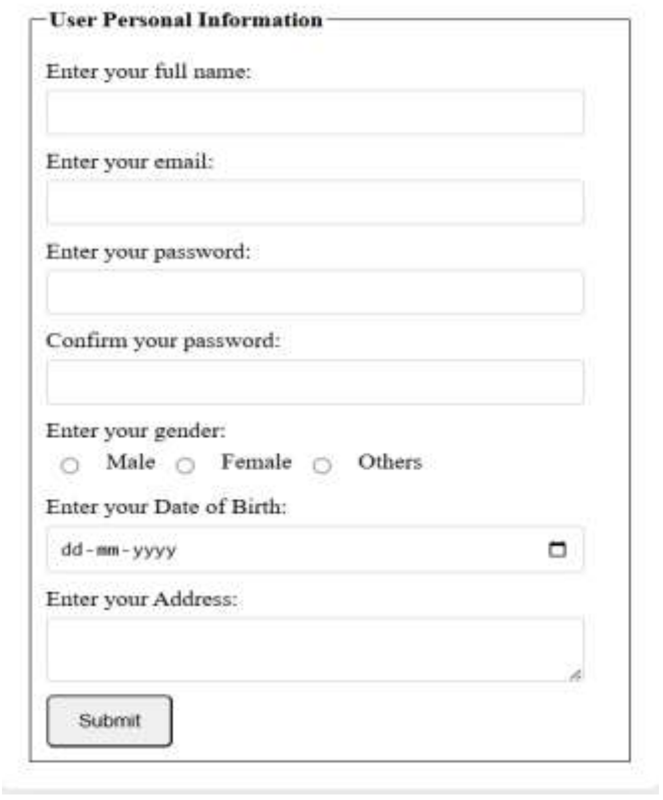
```
.gender-group {
   margin-bottom: 10px;
```

```
        }
        .gender-group label {
           display: inline-block;
           margin-left: 10px;
        }
        input[type="radio"] {
           margin-left: 10px;
           vertical-align: middle;
        }
```

Submit Button -- Makes the button clickable and visually attractive.

```
        input[type="submit"] {
           padding: 10px 20px;
           border-radius: 5px;
           cursor: pointer;
        }
    </style>
</head>
<body>
    <form>
        <fieldset>
           <legend>User Personal Information</legend>
           <label for="name">Enter your full name:</label>
           <input type="text" id="name" name="name" required />
           <label for="email">Enter your email:</label>
           <input type="email" id="email" name="email" required />
           <label for="password">Enter your password:</label>
           <input type="password" id="password" name="pass" required />
           <label for="confirmPassword">Confirm your password:</label>
           <input type="password" id="confirmPassword" name="confirmPass" required
/>
           <label>Enter your gender:</label>
           <div class="gender-group">
```

```
            <input type="radio" name="gender" value="male" id="male" required />
            <label for="male">Male</label>
            <input type="radio" name="gender" value="female" id="female" />
            <label for="female">Female</label>
            <input type="radio" name="gender" value="others" id="others" />
            <label for="others">Others</label>
          </div>
          <label for="dob">Enter your Date of Birth:</label>
          <input type="date" id="dob" name="dob" required />
          <label for="address">Enter your Address:</label>
          <textarea id="address" name="address" required></textarea>
          <input type="submit" value="Submit" />
        </fieldset>
      </form>
    </body>
    </html>
```



User Personal Information

Enter your full name:

Enter your email:

Enter your password:

Confirm your password:

Enter your gender:
○ Male ○ Female ○ Others

Enter your Date of Birth:
dd-mm-yyyy

Enter your Address:

Submit

HTML form control fields

- HTML form control fields are interactive elements used within an

  HTML <form> element to collect user input. These elements enable users to provide information that can then be sent to a server for processing.
- **Key HTML elements that serve as form control fields include:**
- <input>:

  This is a versatile element used for various types of input, determined by

  its type attribute. Examples include:
    - type="text": For single-line text input.
    - type="password": For password input, masking the characters.
    - type="checkbox": For selecting multiple options from a list.
    - type="radio": For selecting a single option from a group.
    - type="submit": For a button that submits the form data.
    - type="file": For uploading files.
    - type="email", type="number", type="date", type="color", etc., for specific data types with built-in validation and specialized user interfaces.
- <textarea>:

  Used for multi-line text input, such as comments or descriptions.
- <select>:

  Creates a dropdown list from which the user can choose one or more options.
    - <option>: Represents an individual item within a <select> list.
    - <optgroup>: Used to group related options within a <select> list.
- <button>:

  Used to create clickable buttons for various actions, including submitting the form, resetting fields, or triggering client-side scripts.
- <label>:

  Provides a descriptive label for a form control, improving accessibility and user experience by associating text with an input field.
- <fieldset> and <legend>:

Used to group related form controls together visually and semantically, with <legend> providing a caption for the group.

- **There are different types of form controls that we can use to collect data using HTML form:**
- Text Input Controls
- Checkboxes Control
- Radio Buttons Control
- Select Box Control
- File Select Box
- Button Control
- Hidden Form Control
- Datetime Controls
- Date Control
- Month Control
- Week Control
- Time Control
- Number Control
- Range Control
- Email Control
- URL Control

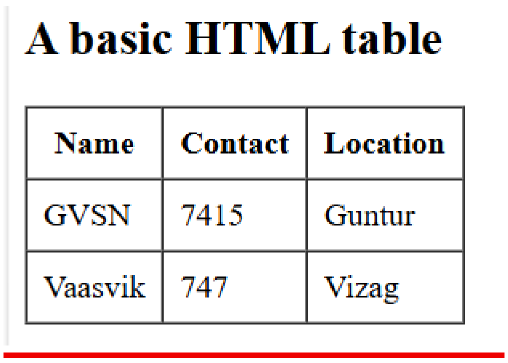
HTML Tables

- HTML tables help organize data into rows and columns, making information easy to read and compare. They are useful for displaying schedules, price lists, product details, and more.
    - Can include text, images, links, and other elements.
    - Built using tags like <table>, <tr>, <th>, and <td>.
    - Allow clear presentation of data for comparison.
    - Can be styled with CSS for better design and readability.
- **HTML Tables Tags:**
- Here, is the list of all the tags that we used in table formation in html.

Tag	Description
<table>	Defines the structure for organizing data in rows and columns within a web page.
<tr>	Represents a row within an HTML table containing individual cells.
<th>	Shows a table header cell that typically holds titles or headings.
<td>	Represents a standard data cell, holding content or data.
<caption>	Provides a title or description for the entire table.
<thead>	Defines the header section of a table, often containing column labels.
<tbody>	Represents the main content area of a table, separating it from the header or footer.
<tfoot>	Specifies the footer section of a table, typically holding summaries or totals.
<col>	Defines attributes for table columns that can be applied to multiple columns simultaneously.
<colgroup>	Groups together a set of columns in a table to which you can apply formatting or properties collectively.

- **Example Program required for above all .**

```
<!DOCTYPE html>
<html>
<head>
  <title>A basic HTML table</title>
</head>
<body>

<h2>A basic HTML table</h2>

<table border="1" cellspacing="0" cellpadding="8">
  <tr>
    <th>Name</th>
    <th>Contact</th>
    <th>Location</th>
  </tr>
  <tr>
    <td>GVSN</td>
    <td>7415</td>
    <td>Guntur</td>
  </tr>
  <tr>
    <td>Vaasvik</td>
    <td>747</td>
    <td>Vizag</td>
  </tr>
</table>

</body>
</html>
```



A basic HTML table

Name	Contact	Location
GVSN	7415	Guntur
Vaasvik	747	Vizag

Events

HTML events are "things" that happen to HTML elements, **allowing for dynamic and interactive web pages.** JavaScript is commonly used to "react" to these events by executing specific code when an event occurs.

- Events in HTML5 are **actions or occurrences** that happen in the browser. They are usually triggered by:
    - User actions (clicking, typing, scrolling)
    - Browser actions (page load, errors)
    - System actions (network changes, media playback)
- HTML events allow developers to make websites **interactive** using JavaScript.

Common HTML Events and their uses:

- **Click Events:**
    - **onclick:** Occurs when an element is clicked.
    - **ondblclick:** Occurs when an element is double-clicked.
    - **Example:** Triggering a function when a button is clicked to display a message.

- **Mouse Events:**
    - **onmouseover:** Occurs when the mouse pointer moves over an element.
    - **onmouseout:** Occurs when the mouse pointer moves out of an element.
    - **onmousedown:** Occurs when a mouse button is pressed down over an element.
    - **onmouseup:** Occurs when a mouse button is released over an element.
    - **Example:** Changing the appearance of an image when the mouse hovers over it.

- **Keyboard Events:**
    - **onkeydown:** Occurs when a key is pressed down.
    - **onkeyup:** Occurs when a key is released.
    - **onkeypress:** Occurs when a key is pressed and held down.
    - **Example:** Validating input as a user types in a text field.

- **Form Events:**
    - **onchange:** Occurs when the value of an input element changes (e.g., a dropdown selection or text input).
    - **onsubmit:** Occurs when a form is submitted.
    - **onreset:** Occurs when a form is reset.
    - **Example:** Performing client-side validation before submitting a form.

- **Load Events:**
    - **onload:** Occurs when an object (like a page or an image) has finished loading.
    - **onunload:** Occurs when a page is unloaded (e.g., when navigating away).
    - **Example:** Executing a script after the entire page content is loaded.

- **Note : Example Programs Required for above all.**

```
<!DOCTYPE html>
<html>
<body>

<button onclick="showMessage()">Click Me</button>

<script>
  function showMessage() {
    alert("Button was clicked!");
  }
</script>

</body>
</html>
```

SVG

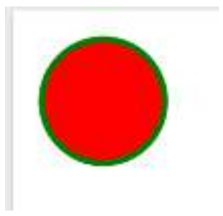
- SVG stands for Scalable Vector Graphics.
- SVG is used to define vector-based graphics for the Web.
- SVG defines graphics in XML format.
- Each element and attribute in SVG files can be animated.
- SVG integrates with other standards, such as CSS, DOM, XSL and JavaScript.

- **The <svg> Element:**
  - The HTML <svg> element is a container for SVG graphics.
  - SVG has several methods for drawing paths, rectangles, circles, polygons, text, and much more.
- **Drawing basic shapes: SVG provides elements for drawing common shapes like:**
  - <circle>: For circles.
  - <rect>: For rectangles.
  - <line>: For straight lines.
  - <ellipse>: For ellipses.
  - <polygon>: For shapes with multiple straight sides.
  - <polyline>: For connected straight line segments.

- **SVG Circle - <circle>**
  - The `<circle>` element is used to create a circle.
  - The `<circle>` element has three basic attributes to position and set the size of the circle:

Attribute	Description
r	Required. The radius of the circle
cx	The x-axis center of the circle. Default is 0
cy	The y-axis center of the circle. Default is 0

```
<!DOCTYPE html>
<html>
<body>

<svg width="200" height="200">
  <circle cx="100" cy="100" r="40"
    Stroke = “green” stroke-width=”4” fill="red" />
</svg>

</body>
</html>
```



- Start with the <svg> root element.
- The width and height of the SVG image is defined with the width and height attributes.
- Since SVG is an XML dialect, always bind the namespace correctly with the xmlns attribute.
- The namespace "http://www.w3.org/2000/svg" identifies SVG elements inside an HTML document.
- The <circle> element is used to draw a circle.
- The cx and cy attributes define the x and y coordinates of the center of the circle. If omitted, the circle's center is set to (0, 0).
- The r attribute defines the radius of the circle.
- The stroke and stroke-width attributes control how the outline of a shape appears. We set the outline of the circle to a 4px green "border".
- The fill attribute refers to the color inside the circle. We set the fill color to yellow.
- The closing </svg> tag closes the SVG image.

- **SVG Rectangle - <rect>**
    - The <rect> element is used to create a rectangle and variations of a rectangle shape.
    - The <rect> element has six basic attributes to position and shape the rectangle:

Attribute	**Description**
width	Required. The width of the rectangle
height	Required. The height of the rectangle
x	The x-position for the top-left corner of the rectangle
y	The y-position for the top-left corner of the rectangle
rx	The x radius (used to round the corners). Default is 0
ry	The y radius (used to round the corners). Default is 0

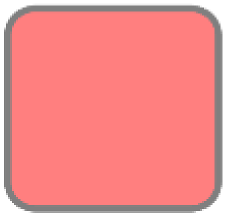
```
<!DOCTYPE html>
<html>
<body>

<svg width="200" height="120">
  <rect x="20" y="20" width="160" height="80" fill="blue" />
</svg>

</body>
</html>
```

Code explanation:

- The width and height attributes of the <rect> element define the height and the width of the rectangle
- The x and y attributes define the x- and y-position of the top-left corner of the rectangle (x="10" places the rectangle 10px from the left margin and y="10" places the rectangle 10px from the top margin) in the SVG canvas
- The rx and ry attributes defines the radius of the corners of the rectangle
- The fill attribute defines the fill color of the rectangle

- **A Rectangle with Rounded Corners:**
    - Last example, create a rectangle with rounded corners:

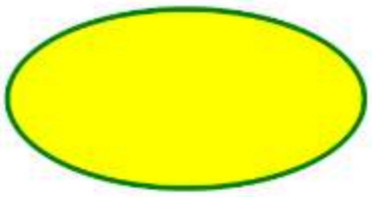
```
<!DOCTYPE html>
<html>
<body>
<h2> A rectangle with rounded corners</h2>
<svg width="200" height="120">
  <rect x="20" y="20" width="160" height="80" rx="15" ry="15"
     Style=”fill:red;stroke:black;stroke-width:5” />
</svg>

</body>
</html>
```



- **SVG Ellipse - <ellipse>**
    - The <ellipse> element is used to create an ellipse.
    - An ellipse is closely related to a circle. The difference is that an ellipse has an x and a y radius that differs from each other, while a circle has equal x and y radius.
    - The <ellipse> element has four basic attributes to position and set the size of the ellipse:

Attribute	Description
rx	Required. The x radius of the ellipse
ry	Required. The y radius of the ellipse
cx	The x-axis center of the ellipse. Default is 0
cy	The y-axis center of the ellipse. Default is 0

```
<!DOCTYPE html>
<html>
<body>

<svg width="200" height="150">
  <ellipse cx="100" cy="75" rx="60" ry="40"
     Style=”fill:yellow;stroke:green;stroke-width:4” />
</svg>

</body>
</html>
```



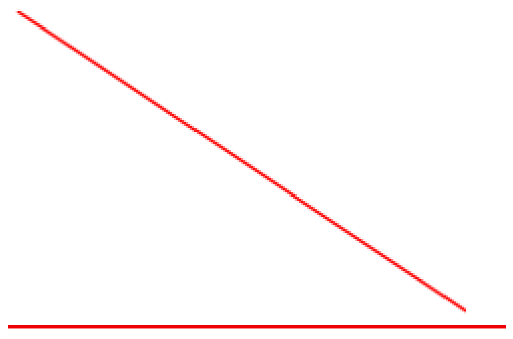
Code explanation:

- The rx attribute defines the x (horizontal) radius.
- The ry attribute defines the y (vertical) radius.
- The cx attribute defines the x-axis center of the ellipse.
- The cy attribute defines the y-axis center of the ellipse.

- **SVG Line - <line>**
    - The <line> element is used to create a line.
    - The <line> element creates a line between the start position (x1,y1) and the end position (x2,y2).
    - The <line> element has four basic attributes to position and set the length of the line:

Attribute	Description
x1	The start of the line on the x-axis
y1	The start of the line on the y-axis
x2	The end of the line on the x-axis
y2	The end of the line on the y-axis

```
<!DOCTYPE html>
<html>
<body>

<svg width="200" height="100">
  <line x1="20" y1="50" x2="180" y2="50"
        stroke="red" stroke-width="3" />
</svg>

</body>
</html>
```



Code explanation:

- The <x1> and <y1> attributes define the start of the line (0,0).
- The <x2> and <y2> attributes define the end of the line (300,200).

- **SVG Polygon - <polygon>**
    - The <polygon> element is used to create a graphic that contains at least three sides.
    - Polygons are made of straight lines, and the shape is "closed" (it automatically connects the last point with the first).
    - The <polygon> element has one basic attribute that defines the points of the polygon:

Attribute	Description
points	Required. The list of points of the polygon. Each point must contain an x coordinate and a y coordinate

Web storage

- With web storage, applications can store data locally within the user's browser.
- Before HTML5, application data had to be stored in cookies, included in every server request. Web storage is more secure, and large amounts of data can be stored locally, without affecting website performance.
- Unlike cookies, the storage limit is far larger (at least 5MB) and information is never transferred to the server.
- Web storage is per origin (per domain and protocol). All pages, from one origin, can store and access the same data.
- Web storage provides two objects for storing data in the browser:

1. Local Storage:
    - ➢ Stores data permanently (until manually cleared).
    - ➢ Data remains even after closing the browser.
    - ➢ SAVING data to localStorage using:
        - o localStorage.setItem("key", "value");
    - ➢ READING data from localStorage using:
        - o var name = localStorage.getItem("key");
    - ➢ REMOVING data from localStorage using:
        - o localStorage.removeItem("key");
2. Session Storage:
    - ➢ Stores data temporarily.
    - ➢ Data is deleted when the browser tab is closed.
    - ➢ window.sessionStorage → returns Storage object

Example 1:

Test Web Storage API Support: This program checks browser support for Web Storage API and shows the result on the webpage:

```
<!DOCTYPE html>
<html>
<head>
   <title>Web Storage API Test</title>
</head>
<body>

<h1>Browser test for Web Storage API</h1>

<div id="result"></div>

<script>
   const x = document.getElementById("result");
   if (typeof(Storage) !== "undefined") {
      x.innerHTML = "Your browser supports Web storage!";
   } else {
      x.innerHTML = "Sorry, no Web storage support!";
   }
</script>

</body>
</html>
```

Code Explenation:

- `<div id="result">`: Creates an empty container where the message will be displayed. **id="result"** is used to access this element using JavaScript.
- `const x = document.getElementById("result")`: Finds the **<div>** with id **result** and stores it in a variable named **x**.
- `if (typeof(Storage) !== "undefined")`: Checks whether the browser supports Web Storage. **Storage** is a built-in object for **localStorage** and **sessionStorage**.

Example 2: Checks Web Storage support, Saves data using localStorage, Retrieves and displays the stored data even after page refresh.

```
<!DOCTYPE html>
<html>
<head>
  <title>Web Storage Example</title>
</head>
<body>

<div id="result"></div>

<script>
  const x = document.getElementById("result");

  // Check browser support
  if (typeof(Storage) !== "undefined") {

    // Store data in localStorage
    localStorage.setItem("lastname", "sivan");
    localStorage.setItem("bgcolor", "yellow");

    // Retrieve data from localStorage
    x.innerHTML = localStorage.getItem("lastname");
    x.style.backgroundColor = localStorage.getItem("bgcolor");

  } else {
    x.innerHTML = "Sorry, no Web storage support!";
  }
</script>

</body>
</html>
```

Example explained:

- localStorage.setItem("lastname", "GVSN"): Stores the value "**GVSN**" with the key **"lastname"** in localStorage.

- localStorage.setItem("bgcolor", "yellow"): Stores the value **"yellow"** with the key **"bgcolor"** in localStorage.
- x.innerHTML = localStorage.getItem("lastname"): Retrieves the value of **"lastname"** from **localStorage** and displays it inside the **div**.
- x.style.backgroundColor = localStorage.getItem("bgcolor"): Gets the stored background color and applies it to the **div**.

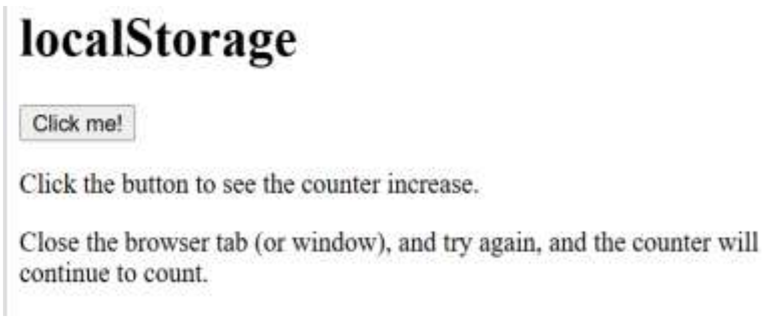
Example 3: Counting Clicks with localStorage :

- The following example counts the number of times a user has clicked a button. In this code the value string is converted to a number to be able to increase the counter:

```
<!DOCTYPE html>
<html>
<head>
  <title>localStorage Click Counter</title>

  <script>
    function clickCounter() {
      const x = document.getElementById("result");

      // Check browser support
      if (typeof(Storage) !== "undefined") {

        // Check if clickcount already exists
        if (localStorage.clickcount) {
          localStorage.clickcount = Number(localStorage.clickcount) + 1;
        } else {
          localStorage.clickcount = 1;
        }

        // Displays the total number of clicks on the webpage
        x.innerHTML = "You have clicked the button " + localStorage.clickcount + " time(s)!";
      } else {
        x.innerHTML = "Sorry, no Web storage support!";
      }
    }
  </script>
</head>

<body>

  <h1>localStorage</h1>

  <button onclick="clickCounter()" type="button">
    Click me!
  </button>

  <p id="result"></p>

  <p>Click the button to see the counter increase.</p>
  <p>
    Close the browser tab (or window), and try again,
    and the counter will continue to count.
  </p>

</body>
</html>
```

Example explained:

- Every button click is stored in localStorage. The count remains even after refreshing or reopening the browser.
- function clickCounter(): Defines a function named **clickCounter** that runs when the button is clicked.
- const x = document.getElementById("result"): Gets the paragraph with id result and stores it in variable **x**.
- if (localStorage.clickcount): Checks whether **clickcount** already exists in **localStorage**.
- localStorage.clickcount = Number(localStorage.clickcount) + 1: Converts stored value to a number and Increases the count by **1**.
- else { localStorage.clickcount = 1; }: If no value exists, it starts the counter at **1**.

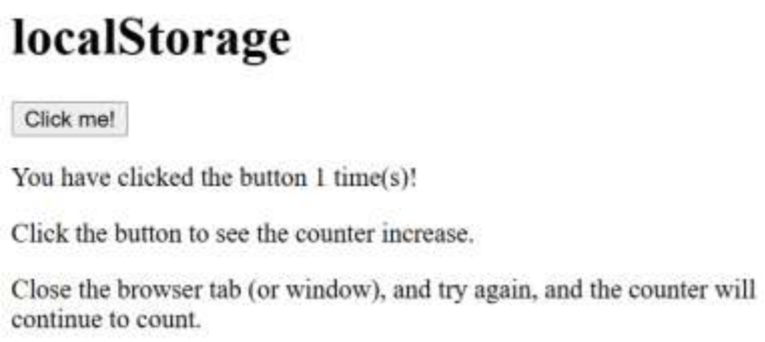


localStorage

Click me!

Click the button to see the counter increase.

Close the browser tab (or window), and try again, and the counter will continue to count.



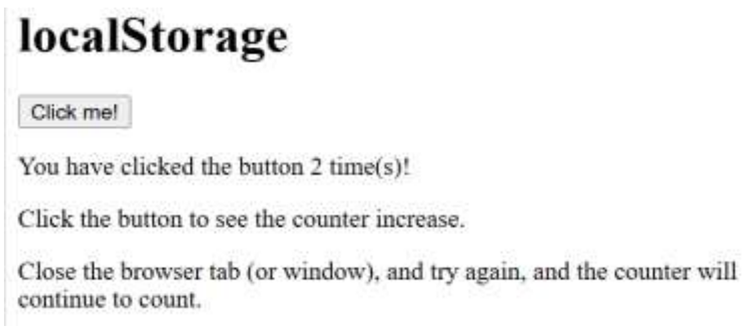
localStorage

Click me!

You have clicked the button 1 time(s)!

Click the button to see the counter increase.

Close the browser tab (or window), and try again, and the counter will continue to count.



localStorage

Click me!

You have clicked the button 2 time(s)!

Click the button to see the counter increase.

Close the browser tab (or window), and try again, and the counter will continue to count.

The sessionStorage Object :

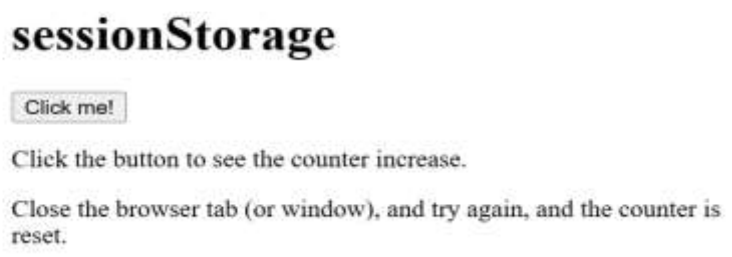
- The sessionStorage object is equal to the localStorage object, except that it stores the data for only one session! The data is deleted when the user closes the specific browser tab.

Example: Counting Clicks with sessionStorage:

The following example counts the number of times a user has clicked a button, in the current session:

```
<!DOCTYPE html>
<html>
<head>
  <title>sessionStorage Click Counter</title>

  <script>
    function clickCounter() {
      const x = document.getElementById("result");

      // Check browser support
      if (typeof(Storage) !== "undefined") {

        // Check if clickcount exists in sessionStorage
        if (sessionStorage.clickcount) {
          sessionStorage.clickcount =
            Number(sessionStorage.clickcount) + 1;
        } else {
          sessionStorage.clickcount = 1;
        }

        // Display result
        x.innerHTML = "You have clicked the button " +  sessionStorage.clickcount +
                " time(s) in this session!";
      } else {
        x.innerHTML = "Sorry, no Web storage support!";
      }
    }
  </script>
</head>

<body>

  <h1>sessionStorage</h1>

  <button onclick="clickCounter()" type="button">
    Click me!
  </button>
```

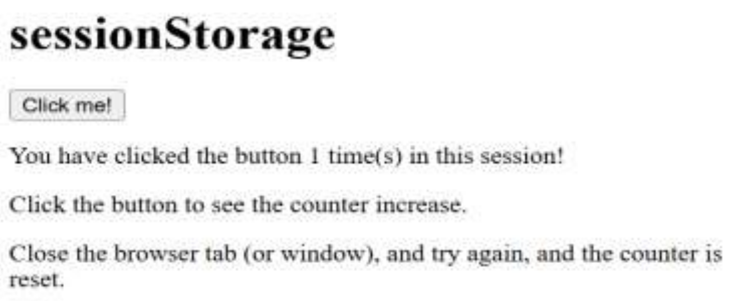
```
<p id="result"></p>

   <p>Click the button to see the counter increase.</p>
   <p>
      Close the browser tab (or window), and open it again.
      The counter will reset because sessionStorage is
cleared.
   </p>

</body>
</html>
```



sessionStorage

Click the button to see the counter increase.

Close the browser tab (or window), and try again, and the counter is reset.

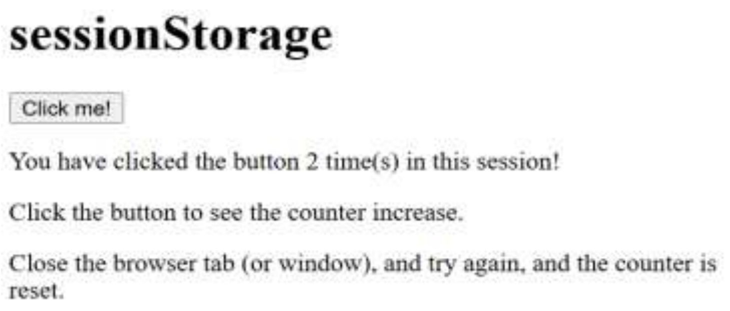


sessionStorage

You have clicked the button 1 time(s) in this session!

Click the button to see the counter increase.

Close the browser tab (or window), and try again, and the counter is reset.



sessionStorage

You have clicked the button 2 time(s) in this session!

Click the button to see the counter increase.

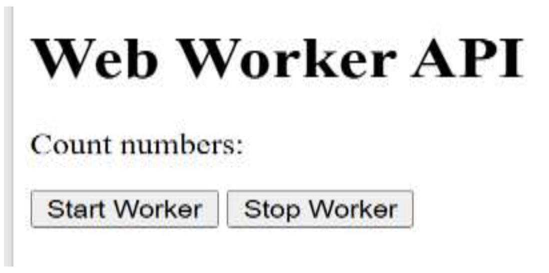
Close the browser tab (or window), and try again, and the counter is reset.

Web Workers

- Web Workers allow JavaScript to run in the background without blocking the main webpage.
- Normally, JavaScript runs on a single thread. If a heavy task runs (large calculation, loop), the page may freeze. Web Workers solve this by running tasks separately.
- Runs in background thread, Does not block the UI, Used for heavy computations.
- A Web Worker is a JavaScript script that runs in a separate background thread, independently of the main browser process and the user interface. This allows web applications to perform computationally expensive tasks or process large amounts of data without blocking the main thread, thus keeping the web page responsive and improving performance.

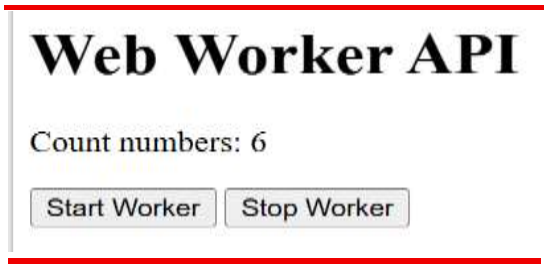
```
<!DOCTYPE html>
<html>
<body>

<p>Count numbers: <output id="result"></output></p>
//<output> shows the result sent by the Web Worker
<button onclick="startWorker()">Start Worker</button>
<button onclick="stopWorker()">Stop Worker</button>

<script>
let w;      //Declares a variable w to store the Web Worker object.

function startWorker() {
  const x = document.getElementById("result");
  if (typeof(Worker) !== "undefined") {
    if (typeof(w) == "undefined") {
      w = new Worker("demo_workers.js");
    }
    w.onmessage = function(event) {
      x.innerHTML = event.data;
    };
  } else {
    x.innerHTML = "Sorry! No Web Worker support.";
  }
}
```

```
function stopWorker() {
  w.terminate();
  w = undefined;
}
</script>

</body>
</html>
```



Web Worker API

Count numbers:



Web Worker API

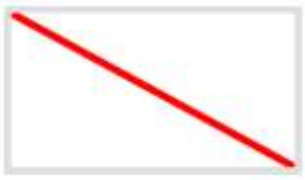
Count numbers: 6

Introduction to Canvas

- The HTML <canvas> element is used to draw graphics, on the fly, via JavaScript.
- The <canvas> element is only a container for graphics. You must use JavaScript to actually draw the graphics.
- Canvas has several methods for drawing paths, boxes, circles, text, and adding images.
- Canvas is supported by all major browsers.

Example 1 : Draw a Line

```
<!DOCTYPE html>
<html>
<body>
<canvas id="myCanvas" width="200" height="100" style="border:5px solid #d3d3d3;">
</canvas>

<script>
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
ctx.moveTo(0, 0);
ctx.lineTo(200, 100);
ctx.strokeStyle = "red";
ctx.lineWidth = 5;
ctx.stroke();
</script>
```



Example explained:

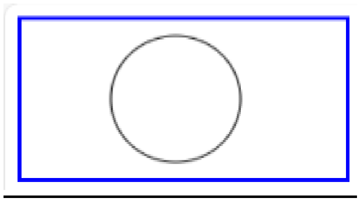
- <canvas id="myCanvas" width="200" height="100" style="border:5px solid #d3d3d3;">
    - Creates a canvas area for drawing
    - id="myCanvas" → used to access the canvas in JavaScript
    - width="200" → canvas width is 200 pixels
    - height="100" → canvas height is 100 pixels

  - style="border:5px solid #d3d3d3;" → adds a light gray border around the canvas

- var c = document.getElementById("myCanvas"): Finds the **<canvas>** element with id **myCanvas** and stores it in variable **c**.
- var ctx = c.getContext("2d"): Gets the **2D drawing** context of the canvas, **ctx** is used to draw shapes, lines, text, etc.
- ctx.moveTo(0, 0): Moves the drawing cursor to position (0, 0). This is the starting point of the line (top-left corner).
- ctx.lineTo(200, 100): Draws a line from the current position to point (200, 100).
- ctx.stroke(): Makes the drawn line visible on the canvas.

Example 2 : Draw a Circle

```
<!DOCTYPE html>
<html>
<body>
<canvas id="myCanvas" width="200" height="100" style="border:1px solid #0000FF;">
</canvas>

<script>
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
ctx.beginPath();
ctx.arc(95, 50, 40, 0, 2 * Math.PI);
ctx.stroke();
</script>

</body>
</html>
```



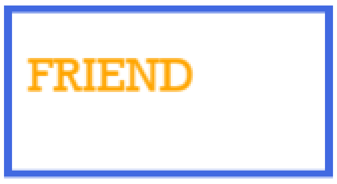
Example explained:

- ctx.beginPath(): Starts a new drawing path.
- ctx.arc(95, 50, 40, 0, 2 * Math.PI): Draws a circle:
  - 95, 50 → center of the circle (x, y)
  - 40 → radius of the circle

- 0 → starting angle
- 2 * Math.PI → ending angle (full circle)

Example 3 : Draw a Text

```
<!DOCTYPE html>
<html>
<body>
<canvas id="myCanvas" width="200" height="100" style="border:5px solid #4169E1;">
</canvas>

<script>
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
ctx.fillStyle = "orange";
ctx.font = "30px Rockwell";
ctx.fillText("FRIEND", 10, 50);
</script>

</body>
</html>
```

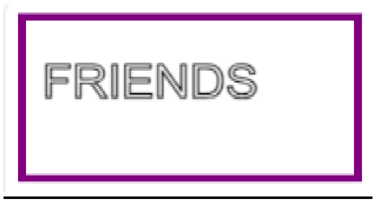


Example explained:

- ctx.font = "30px Rockwell ": Sets the text size to 30 pixels and sets the font style to Rockwell.
- ctx.fillText("FRIEND", 10, 50): Draws the text **"FRIEND"** on the canvas
    - 10 → x-position (distance from left)
    - 50 → y-position (distance from top)
    - Text is **filled** with the current color (default is black)

Example 4 : Stroke Text

```
<!DOCTYPE html>
<html>
<body>
<canvas id="myCanvas" width="200" height="100" style="border: 5px solid #800080;">
</canvas>

<script>
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
ctx.font = "30px Arial";
ctx.strokeText("FRIENDS", 10, 50);
</script>

</body>
</html>
```



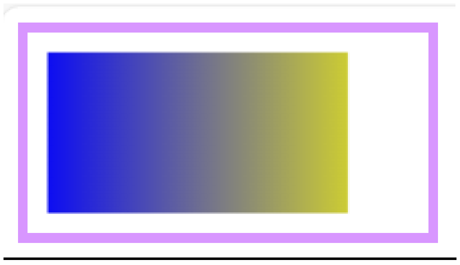
Example explained:

- **ctx.strokeText("FRIENDS", 10, 50):** Draws the **outline (border)** of the text **"FRIENDS"** on the canvas.
  - 10 → x-position (from the left)
  - 50 → y-position (from the top)
  - Text is drawn as **outline only**, not filled
- fillText() → fills the text
- strokeText() → draws only the text border

Example 5: Draw Linear Gradient

```
<!DOCTYPE html>
<html>
<body>
<canvas id="myCanvas" width="200" height="100" style="border: 5px solid #D896FF;">
</canvas>

<script>
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");

// Create gradient
var grd = ctx.createLinearGradient(0, 0, 200, 0);
grd.addColorStop(0, "blue");
grd.addColorStop(1, "yellow");

// Fill with gradient
ctx.fillStyle = grd;
ctx.fillRect(10, 10, 150, 80);
</script>

</body>
</html>
```

```
ctx.fillStyle = grd;
// Draw circle
ctx.beginPath();
ctx.arc(100, 50, 40, 0, 2 * Math.PI);
ctx.fill();
```

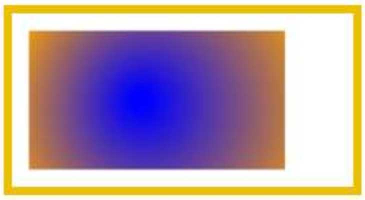


Example explained:

- var grd = ctx.createLinearGradient(0, 0, 200, 0): Creates a linear color gradient
  - Starts at point (0, 0)
  - Ends at point (200, 0) (horizontal gradient)
- grd.addColorStop(0, "blue"): Sets the **starting color** of the gradient to red.
- grd.addColorStop(1, "yellow"): Sets the ending color of the gradient to white.
- ctx.fillStyle = grd: Applies the gradient as the fill color.
- ctx.fillRect(10, 10, 150, 80): Draws a filled rectangle:
  - 10, 10 → starting position (x, y)
  - 150 → width; 80 → height
  - Rectangle is filled with the gradient

Example 6: Draw Circular Gradient

```
<!DOCTYPE html>
<html>
<body>
<canvas id="myCanvas" width="200" height="100" style="border: 5px solid #EFBF04;">
</canvas>

<script>
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");

// Create gradient
var grd = ctx.createRadialGradient(75, 50, 5, 90, 60, 100);
grd.addColorStop(0, "blue");
grd.addColorStop(1, "orange");

// Fill with gradient
ctx.fillStyle = grd;
ctx.fillRect(10, 10, 150, 80);
</script>

</body>
</html>
```

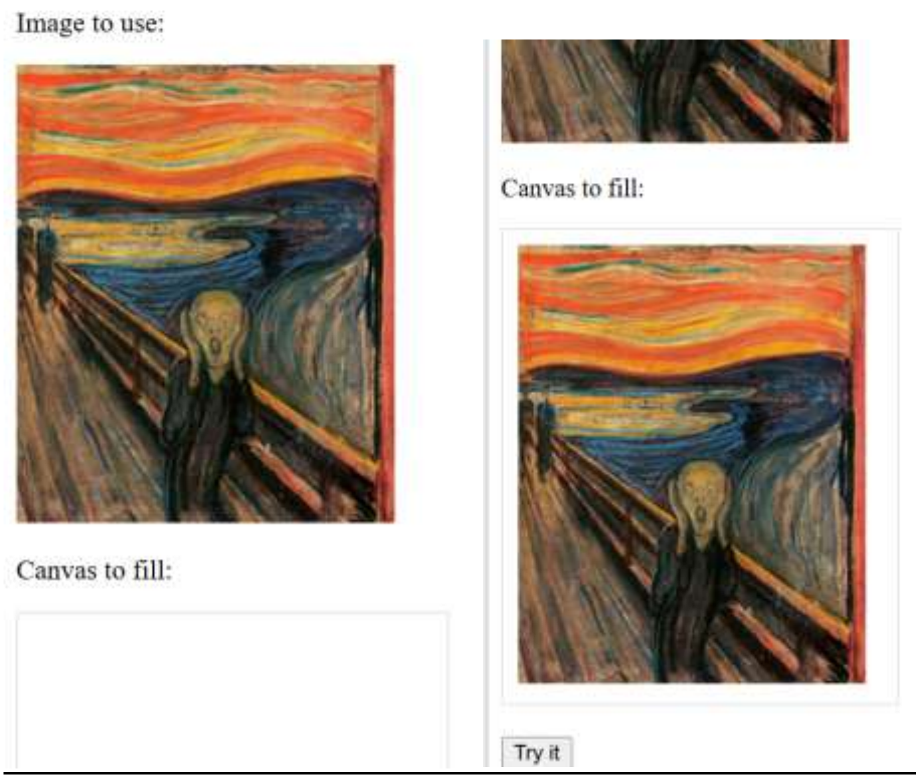


Example explained:

- var grd = ctx.createRadialGradient(75, 50, 5, 90, 60, 100): Creates a radial (circular) gradient:
    - 75, 50 → center of the inner circle
    - 5 → radius of the inner circle
    - 90, 60 → center of the outer circle
    - 100 → radius of the outer circle

Example 6: Draw Image

```
<!DOCTYPE html>
<html>
<body>
<canvas id="myCanvas" width="200" height="100" style="border:   5px solid #D3AF37;">
</canvas>

<script>
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
var img = document.getElementById("scream");
ctx.drawImage(img, 10, 10);
</script>

</body>
</html>
```



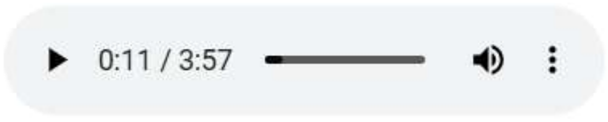
Example explained:

- ctx.drawImage(img, 10, 10): Draws the image onto the canvas
  - img → image to be drawn
  - 10, 10 → position on the canvas (x, y from top-left)

Audio

- The HTML <audio> element is used to play an audio file on a web page.
- The controls attribute adds audio controls, like play, pause, and volume.
- The <source> element allows you to specify alternative audio files which the browser may choose from. The browser will use the first recognized format.
- The text between the <audio> and </audio> tags will only be displayed in browsers that do not support the <audio> element.

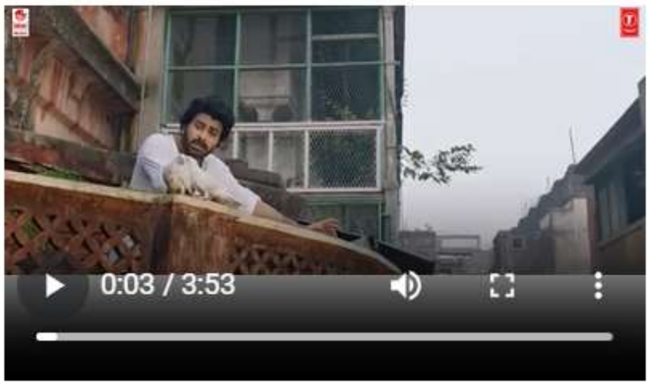
```
<!DOCTYPE html>
<html>
<body>

<audio controls>
  <source src="D:\VSN\Audio Songs\Annamayya\adivo alladigo.mp3" type="audio/ogg">
//Used if above OGG is not supported
  <source src="horse.mp3" type="audio/mpeg">
Your browser does not support the audio element.
</audio>

</body>
</html>
```



Video

- The HTML <video> element is used to show a video on a web page.
- The controls attribute adds video controls, like play, pause, and volume.
- It is a good idea to always include width and height attributes. If height and width are not set, the page might flicker while the video loads.
- The <source> element allows you to specify alternative video files which the browser may choose from. The browser will use the first recognized format.
- The text between the <video> and </video> tags will only be displayed in browsers that do not support the <video> element.

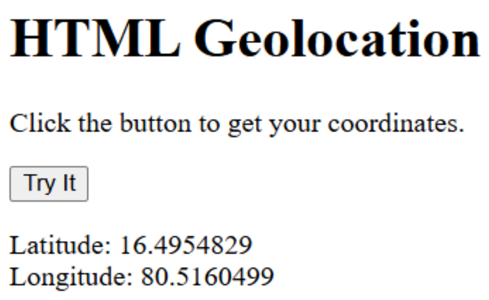
```
<!DOCTYPE html>
<html>
<body>

<video width="320" height="240" controls>
  <source src="D:\VSN\vaasvik sngs\padi padi leche manasu_Kallolam.mp4" type="video/mp4">
//Specifies an OGG video file, Used if MP4 is not supported
  <source src="movie.ogg" type="video/ogg">
Your browser does not support the video tag.
</video>

</body>
</html>
```



Geolocations

- The Geolocation API is used to access the user's current location.
- Since this can compromise privacy, the location is not available unless the user approves it.
- The Geolocation API is accessed via a call to navigator.geolocation. This will cause the browser to ask the user for permission to access their location data. If the user accept, the browser will search for the best available functionality on the device to access this information (for example GPS).
- The getCurrentPosition() method is used to return the user's current location.
- The example below returns the latitude and longitude of the user's current location:

```
<!DOCTYPE html>
<html>
<body>

<h1>HTML Geolocation</h1>
<p>Click the button to get your coordinates.</p>

<button onclick="getLocation()">Try It</button>

<p id="demo"></p>

<script>
const x = document.getElementById("demo");

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

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

function Error() {
alert(“sorry, no position is available”);
</script>
</body>

</html>
```



HTML Geolocation

Click the button to get your coordinates.

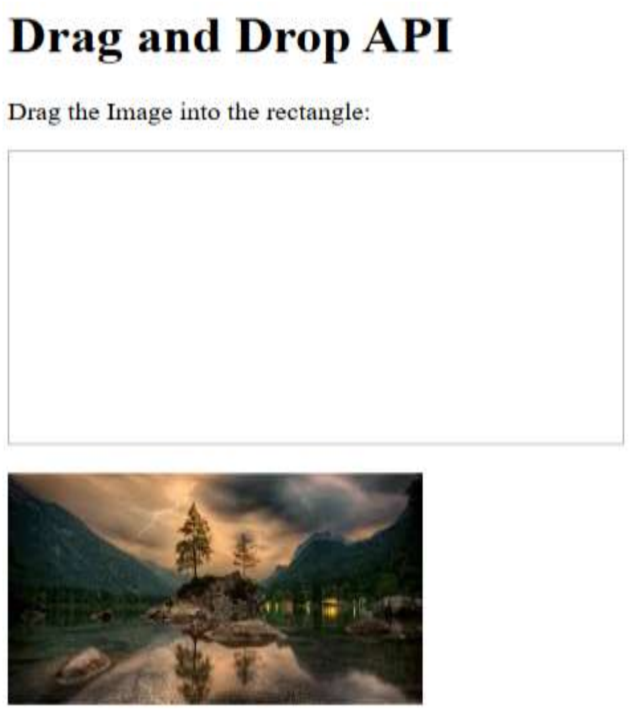
```
Latitude: 16.4954829
Longitude: 80.5160499
```

Drag & Drop

- Drag and drop is a very common feature. It is when you "grab" an object and drag it to a different location.

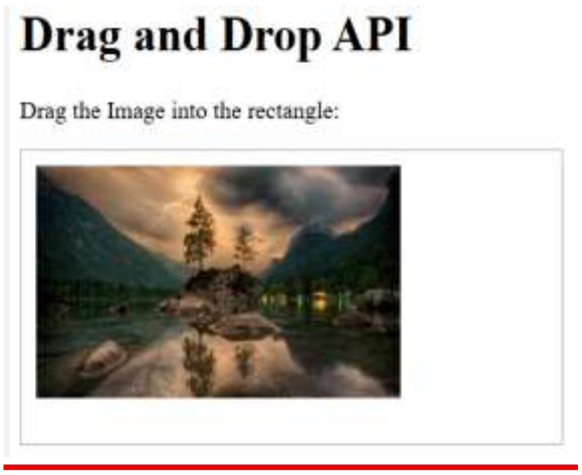
```
<!DOCTYPE html>
<html>
<head>
  <style>
    #div1 {
      width: 350px;
      height: 170px;
      padding: 10px;
      border: 1px solid #aaaaaa;
    }
  </style>

  <script>
    function dragstartHandler(ev) {
      ev.dataTransfer.setData("text", ev.target.id);
    }

    function dragoverHandler(ev) {
      ev.preventDefault();
    }

    function dropHandler(ev) {
      ev.preventDefault();
      const data = ev.dataTransfer.getData("text");
      ev.target.appendChild(document.getElementById(data));
    }
  </script>
</head>
```

```
<body>

   <h1>Drag and Drop API</h1>

   <p>Drag the Image into the rectangle:</p>

   <div id="div1"
       ondrop="dropHandler(event)"
       ondragover="dragoverHandler(event)">
   </div>

   <br>

   <img id="img1"
       src="D:\$iv@\VIT-AP\2025-26 Winter
Sem\WT\Examples\nature.jpg"
       draggable="true"
       ondragstart="dragstartHandler(event)"
       width="250"
       height="150">

</body>
</html>
```



Drag and Drop API

Drag the Image into the rectangle:



Drag and Drop API

Drag the Image into the rectangle:

Example explained:

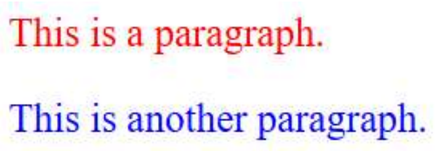
- #div1 : Selects the element with id **div1**.
- width: 350px: Sets the width of the drop box to 350 pixels.
- height: 70px: Sets the height of the drop box to 70 pixels
- padding: 10px: Adds space inside the box.
- border: 1px solid #aaaaaa: Adds a light gray border around the box
- function dragstartHandler(ev): Function runs when dragging starts.
- ev.dataTransfer.setData("text", ev.target.id): Stores the ID of the dragged element.
- function dragoverHandler(ev): Function runs when a draggable item is over the drop area.
- ev.preventDefault(): Allows dropping by preventing default behavior.
- function dropHandler(ev): Function runs when the element is dropped.
- const data = ev.dataTransfer.getData("text"): Retrieves the stored element ID.
- ev.target.appendChild(document.getElementById(data)): Moves the dragged element into the drop box.
- ev is an event object that the browser automatically sends to the function when an event happens (like drag, drop, click).

Working with Fonts

- Not Supported in HTML5.
- The <font> tag was used in HTML 4 to specify the font face, font size, and color of text.

Example 1:

```
<!DOCTYPE html>
<html>
<body>

<p style="color:red">This is a paragraph.</p>
<p style="color:blue">This is another paragraph.</p>

</body>
</html>
```

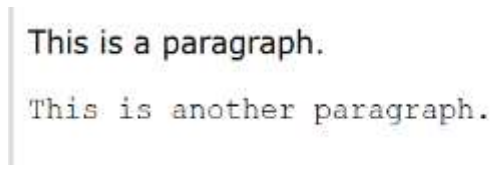


This is a paragraph.

This is another paragraph.

Example 2:

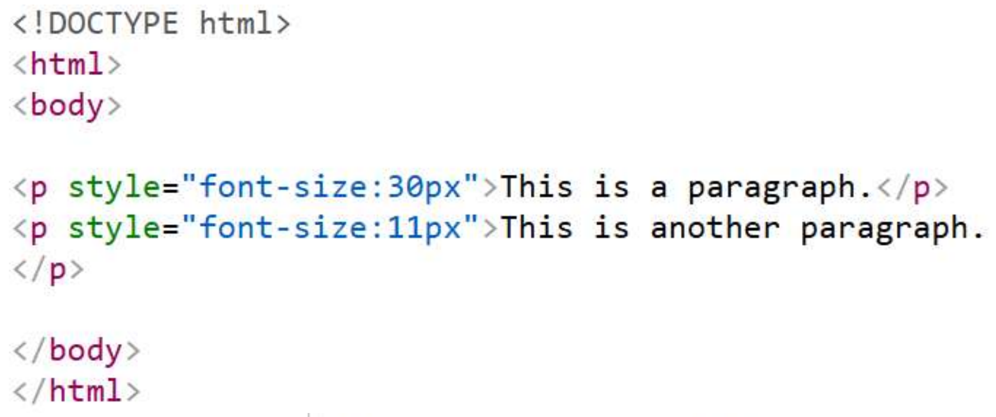
```
<!DOCTYPE html>
<html>
<body>

<p style="font-family:verdana">This is a paragraph.
</p>
<p style="font-family:'Courier New'">This is another
paragraph.</p>

</body>
</html>
```



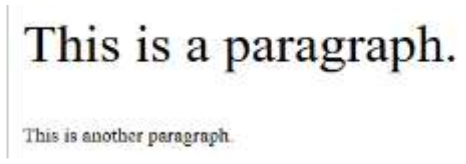
This is a paragraph.

This is another paragraph.

Example 3:



```
<!DOCTYPE html>
<html>
<body>

<p style="font-size:30px">This is a paragraph.</p>
<p style="font-size:11px">This is another paragraph.
</p>

</body>
</html>
```



This is a paragraph.

This is another paragraph.

Working with other graphics

Few Examples Required