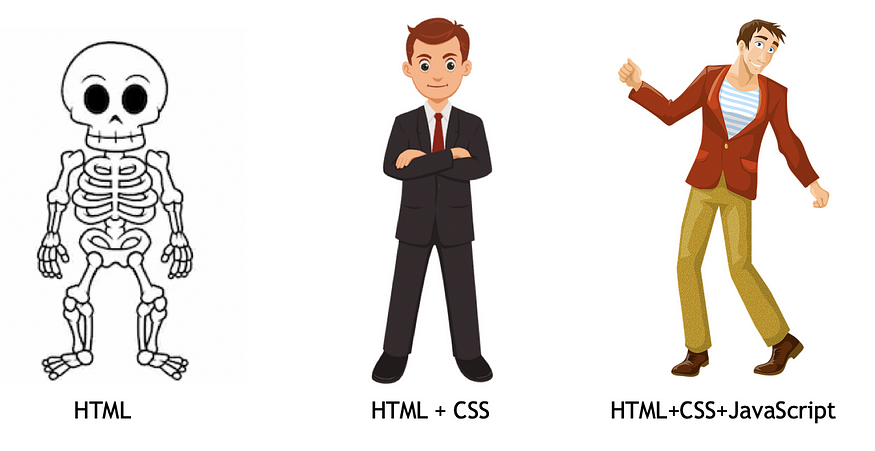
***HTML***

While HTML is a skeleton structure of a document, CSS beautifies it and Javascript makes it dynamic



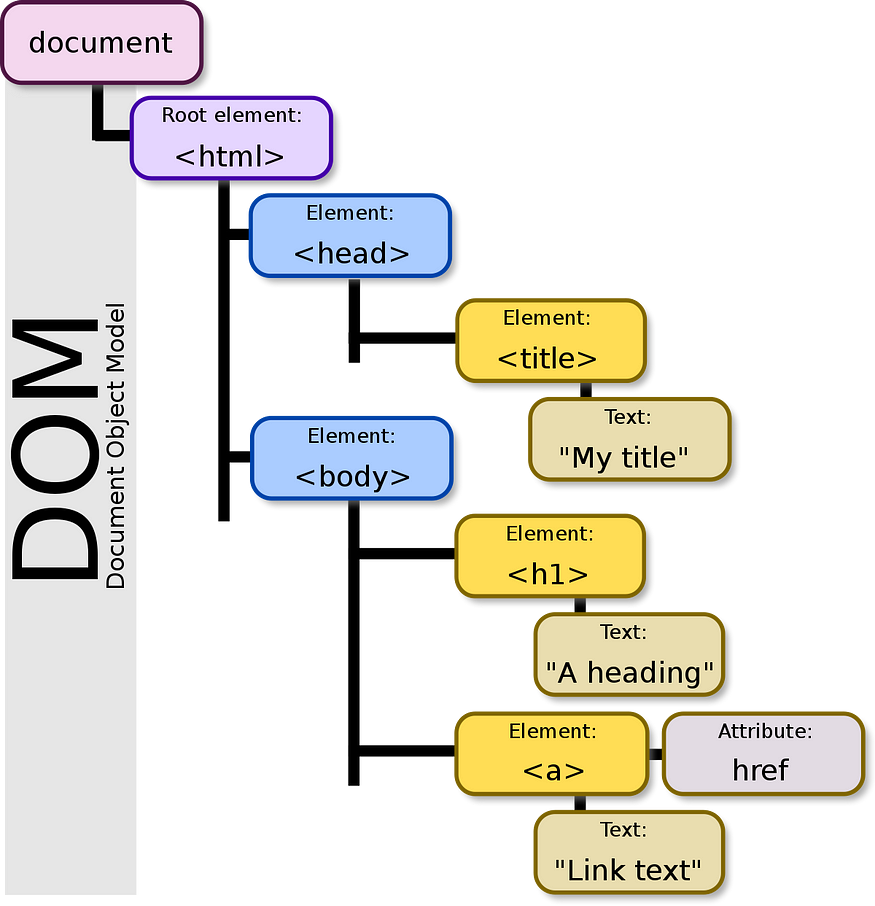
Advantages and Disadvantages of HTML:

|  |  |
| --- | --- |
| ***Advantages*** | ***Disadvantages*** |
| HTML is a light web programming language | It is difficult to read as it contains a lot of codewhich creates complexity |
| It can create static and plain web pages | Back-end languages such as PHP and JAVA are necessary to build dynamic websites and applications. |
| HTML is easy to use and analyze | We have to write a lot of code to create a webpage |
| It provides the structure of the webpage and CSS provides styling to the webpage. | HTML alone is insufficient for creating visually appealing websites. |
| All browsers support HTML. | It Takes time to load |

1. **HTML Document, DOM**

When an html document is loaded into a web browser, it becomes a document object.

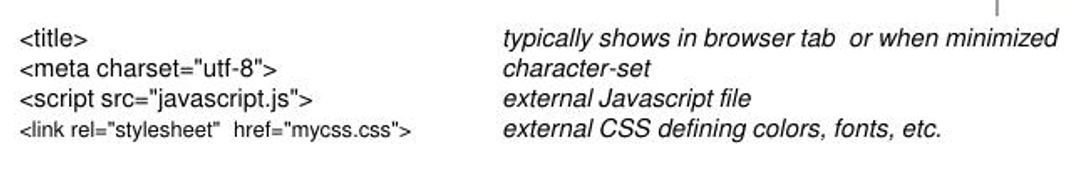
<!DOCTYPE html>  
<html>  
<head>  
<meta charset="UTF-8">  
<title>*Title of the document*</title>  
</head>  
  
<body>  
*Content of the document......*  
</body>  
  
</html>



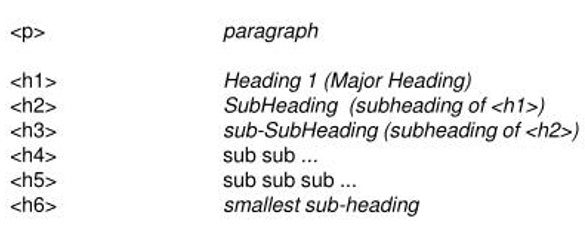
## Understanding HTML Tags:

HTML tags help web browsers convert HTML documents into web pages.

Common HTML tags in <head>:



Common HTML tags in <body>:



## Understanding HTML attributes:

* HTML attributes are special words which provide additional information about the elements.
* Each element or tag can have attributes, which defines the behaviour of that element.
* Attributes should always be applied with start tag.
* The Attribute should always be applied with its name and value pair.
* The Attributes name and values are case sensitive, and it is recommended to be written in Lowercase only.
* You can add multiple attributes in one HTML element, but need to give space between two attributes.

## Syntax:

## <element attribute\_name="value">content</element>

## <a> tag:

The <a> tag in HTML is used to create hyperlinks, which are clickable elements that navigate to another webpage, file, or location within the same webpage.

Here's how the <a> tag is structured:

<a href="URL">Link Text</a>

* href: The href attribute specifies the destination of the link. It can be a URL (web address), a file path, or an anchor within the same page.
* Link Text: The text or content that is displayed as the clickable link.

**Additional Attributes:**

The <a> tag can also have additional attributes:

target: Specifies where to open the linked document (\_self, \_blank, \_parent, \_top, or a custom frame name).

**\_self (default):** Opens the link in the same tab or window where the link was clicked.

**\_blank:** Opens the link in a new tab or window. This is commonly used when linking to external websites or documents.

**\_parent:** Opens the link in the parent frame, which is useful if you are working with nested frames.

**\_top:** Opens the link in the full body of the window, effectively breaking out of any frames.

title: Provides additional information about the link (often displayed as a tooltip).

<a href="https://www.example.com" target="\_blank" title="Visit Example Website">Example Website</a>

## Images and attributes in HTML:

* src: Specifies the URL of the image.
* alt: Specifies alternative text for the image, used by screen readers and displayed if the image fails to load.
* width: Specifies the width of the image in pixels.
* height: Specifies the height of the image in pixels.
* title: Specifies a title for the image, typically displayed as a tooltip when the user hovers over the image.

## Understanding HTML elements:

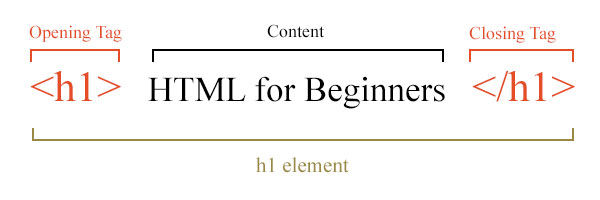
## An element is a collection of start tag, attributes, end tag, content between them.

## The diagram below displays an HTML header element. As we can see, the header element is made up of:

## An opening tag (<h1>)

## The content (“HTML for Beginners” text)

## A closing tag (</h1>)



**5.a. Nested Elements**: HTML allows to keep one element inside another element.

**5.b. Empty Elements**: Contains only opening tag. Do not require content and closing tag.

* <br>: Represents a line break.
* <hr>: Represents a thematic break or horizontal rule.
* <img>: Embeds an image into the document.
* <meta>: Provides metadata about the HTML document.

**5.c. Block level Elements**:

In HTML, block-level elements are elements that typically start on a new line and stretch out to the left and right as far as they can. They form the primary building blocks of the document's layout, and they typically contain inline elements and other block-level elements. Block-level elements are often used for major structural elements of the page, such as headings, paragraphs, lists, navigation menus, and divisions.

Here are some common block-level elements in HTML:

* <div>: The generic block-level container used for grouping elements and applying styles.
* <p>: Represents a paragraph of text.
* <hr>: Horizontal line
* <h1>, <h2>, <h3>, <h4>, <h5>, <h6>: Heading elements, with <h1> being the highest level and <h6> being the lowest.
* <ul>: Defines an unordered list.
* <ol>: Defines an ordered list.
* <li>: Represents a list item within <ul> or <ol> elements.
* <nav>: Defines a section of navigation links.
* <header>: Represents the header of a document or a section.
* <footer>: Represents the footer of a document or a section.
* <section>: Represents a thematic grouping of content.
* <article>: Represents an independent piece of content that can stand alone.
* <main>: Represents the main content of the document.

Block-level elements typically accept width, height, margin, and padding properties, and they naturally create a visual separation between content sections. By default, block-level elements have a line break before and after them in the document flow.

In contrast, inline elements do not start on a new line and only take up as much width as necessary. They are often used within block-level elements to style text or create small content elements within larger blocks.

**Note:**

**<div class="container">**: A <div> is used as a wrapper to control the overall layout, providing a container that can be centered or have its width managed via CSS.

**5.d. Inline Elements:**

Inline elements in HTML are elements that do not start on a new line and only take up as much width as necessary. Unlike block-level elements, inline elements can appear within a block-level element or other inline elements, and they typically do not create a line break before or after themselves. Inline elements are often used to style and format smaller pieces of content within larger blocks of text.

Here are some common inline elements in HTML along with their purposes:

* <span>: The generic inline container used to apply styles or manipulate text within a larger block of content.
* <a>: Defines a hyperlink, allowing users to navigate to another web page or resource.
* <strong>: Indicates strongly emphasized text, typically rendered as bold by default.
* <em>: Indicates text that is emphasized, typically rendered as italic by default.
* <img>: Embeds an image within the text flow.
* <br>: Inserts a line break within the text flow.
* <input>: Defines an input control within a form, such as text input, checkboxes, or radio buttons.
* <label>: Associates a label with a form control, providing a description or instruction for the control.
* <span>: Used for grouping inline elements or applying styles to specific parts of text.
* <code>: Renders text as computer code, typically in a monospaced font.

These elements can be combined and nested within each other to create rich and expressive content within HTML documents. Inline elements are commonly used for styling individual words or phrases, adding links and images to text, creating form elements, and highlighting specific portions of content.

Understanding the distinction between block-level and inline elements is crucial for structuring HTML documents effectively and applying appropriate styles and formatting to different parts of the content.

**5.d. Semantic Elements: (Def: relating** **to meaning)**

Semantic elements in HTML are elements that carry meaning. They are used to define the structure of a web page and convey the purpose of the content to both the browser and the developer. Semantic elements provide context and clarity to the content, making it more understandable for search engines, assistive technologies, and other developers who may work on the project.

Some commonly used semantic elements in HTML5 include:

<header>: Defines a header for a document or section. Typically contains introductory content, such as headings, logos, and navigation menus.

<nav>: Represents a section of navigation links. Used for navigation menus, table of contents, or other sets of links to navigate the document or related documents.

<main>: Specifies the main content of the document. It should not contain content that is repeated across multiple pages, such as headers, footers, or navigation menus.

<section>: Defines a section of related content. It groups together thematic content, such as chapters, headers, or footers.

<article>: Represents a self-contained piece of content that can be distributed or reused independently, such as blog posts, news articles, or comments.

<aside>: Defines content aside from the main content, such as sidebars, pull quotes, or tangentially related information.

<footer>: Defines the footer of a document or section. Typically contains metadata, copyright information, contact details, or links to related content.

<figure> and <figcaption>: Used together to represent self-contained content, such as images, diagrams, illustrations, or code snippets, along with captions.

<details> and <summary>: Defines additional details or disclosures that the user can view or hide. The summary represents a visible heading for the details.

Using semantic elements not only helps to structure the document logically but also improves accessibility, search engine optimization (SEO), and code maintainability. Semantic HTML communicates the meaning and purpose of content more effectively, leading to a better user **experience and making the document easier to understand and maintain.**

**Difference between Semantic elements and <div> tags:**

Using <div> elements versus semantic elements in HTML affects both the structure and accessibility of your web pages. Let's break down the differences using the example layouts provided:

### 1. **Meaning and Clarity**

* **Semantic Elements (<header>, <nav>, <section>, etc.):**
  + **Purpose**: These elements convey meaning about the content they contain. For instance:
    - <header> indicates the top section of a page or a section, usually containing a logo, navigation links, or a title.
    - <nav> specifies a block of navigation links.
    - <section> represents a thematic grouping of content.
    - <article> is intended for self-contained content that could be distributed or reused independently.
    - <aside> typically contains content related to the main content, like sidebars.
    - <footer> represents the bottom section of a page, usually containing contact information or copyright notices.
  + **Clarity**: When other developers or tools (like search engines or screen readers) look at your HTML, they can understand the purpose of each part of your page more easily because the tags themselves carry meaning.
  + **Example**: Using <header>, a browser or assistive technology knows that this section contains introductory content or navigational aids for the page.
* **Non-Semantic Elements (<div>):**
  + **Purpose**: <div> is a generic container element that doesn't convey any specific meaning about the content inside it.
  + **Clarity**: You need to rely on classes, IDs, or comments to explain the purpose of each <div>. This can make the HTML less self-explanatory.
  + **Example**: A <div class="header"> does not inherently tell a browser or assistive technology that it is a header; it's just a block-level element.

### 2. **SEO (Search Engine Optimization)**

* **Semantic Elements**: Search engines use semantic elements to better understand the structure and content of your page. For example, a properly used <nav> tells search engines that the links inside it are important for navigating the site, which can improve how your site is indexed and ranked.
* **Non-Semantic Elements**: Using <div> alone doesn't provide the same level of information to search engines, potentially leading to less efficient indexing.

### 3. **Accessibility**

* **Semantic Elements**: Screen readers and other assistive technologies rely on semantic elements to navigate a web page. For example, users can jump directly to the <nav> to start navigating the site, or skip to the <main> content. This greatly enhances the experience for users with disabilities.
* **Non-Semantic Elements**: With just <div>, assistive technologies don't have the same clues about the page structure. Users might have a harder time navigating the page, leading to a poorer experience.

### 4. **Code Maintainability**

* **Semantic Elements**: Code with semantic elements is easier to maintain because the structure is more understandable at a glance. Developers can quickly identify which part of the content they are dealing with.
* **Non-Semantic Elements**: While <div> is versatile, overuse can lead to "div soup," where the purpose of each <div> is unclear, making the code harder to read, maintain, and debug.

### Example Recap:

In your example:

* Using semantic elements like <header>, <nav>, <section>, <aside>, <article>, and <footer> clearly defines the structure of the page, helping both developers and machines understand the content better.
* Using <div> elements with classes (e.g., <div class="header">, <div class="nav">) achieves the same visual layout but lacks the inherent meaning, making the code less intuitive for both developers and assistive technologies.

### Conclusion:

Using semantic elements is generally preferred for better readability, maintainability, SEO, and accessibility. However, <div> elements are still useful for creating generic containers or when no ––appropriate semantic tag exists.

1. **HTML Lists:**

HTML Lists are used to specify lists of information. All lists may contain one or more list elements. There are three different types of HTML lists:

1. Ordered List or Numbered List (ol) - all the list items are marked with numbers by default.
2. Unordered List or Bulleted List (ul) - all the list items are marked with bullets.
3. Description List or Definition List (dl)

**Definition List:**

This type of list is typically used when you have a term and its corresponding definition .

The HTML definition list contains following three tags:

1. <dl> tag defines the start of the list.
2. <dt> tag defines a term.
3. <dd> tag defines the term definition (description).

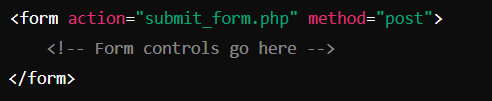
**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*<form>\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

### HTML Forms and Form Controls

**HTML Forms** are a fundamental part of web development, used to collect user input. A form can contain various types of form controls like text fields, radio buttons, checkboxes, and submit buttons. The data entered in these controls can be sent to a server for processing.

#### Basic Structure of an HTML Form

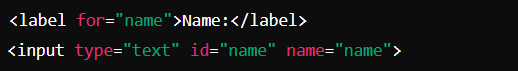
A form is created using the <form> tag, which acts as a container for input elements. The form's attributes define how the data is sent.

****

* **action**: Specifies the URL where the form data will be sent.
* **method**: Defines how the data will be sent (e.g., GET, POST).

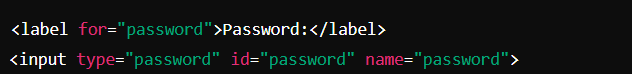
### Common Form Controls

1. **Text Input (<input type="text">)**
   * Allows the user to enter a single line of text.
   * Example:

****

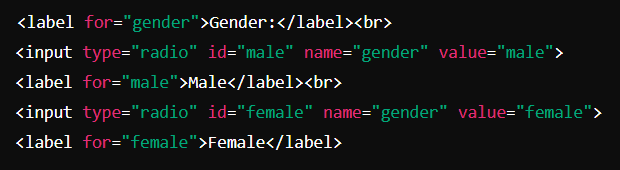
1. **Password Input (<input type="password">)**

* Similar to text input but hides the characters.
* Example:



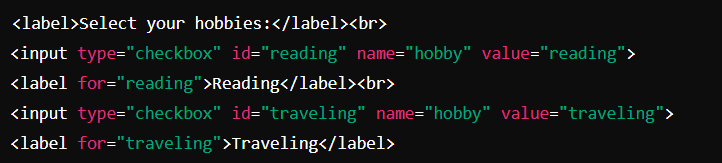
1. **Radio Buttons (<input type="radio">)**

* Allows the user to select one option from a group.
* Example:



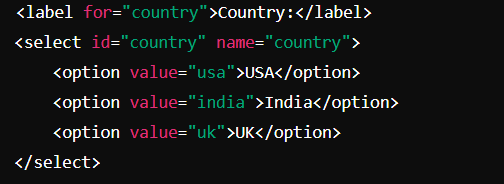
1. **Checkboxes (<input type="checkbox">)**

* Lets the user select multiple options from a list.
* Example:



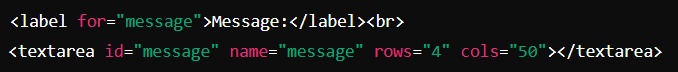
1. **Dropdown List (<select>)**

* Provides a dropdown menu of options.
* Example



1. **Textarea (<textarea>)**

* Allows the user to enter multiple lines of text.
* Example



1. **Submit Button (<input type="submit">)**

* Submits the form data to the server.
* Example:



1. **Reset Button (<input type="reset">)**

* Resets the form fields to their default values.
* Example:



### Understanding the Form Data Flow

When a user submits the form:

1. The browser collects the data entered into the form controls.
2. The data is encoded as a key-value pair (name and value).
3. It is sent to the server using the specified method (GET or POST).
4. The server processes the data, typically storing it in a database or using it to perform some action.

### Additional Form Attributes and Controls

* **<input type="email">**: Validates that the entered text is a properly formatted email address.
* **<input type="url">**: Ensures the input is a valid URL.
* **<input type="date">**: Displays a date picker.
* **<fieldset> and <legend>**: Group related form controls together.

**Form Validation**: You can also add attributes like required, minlength, maxlength, and pattern to form controls for basic client-side validation.

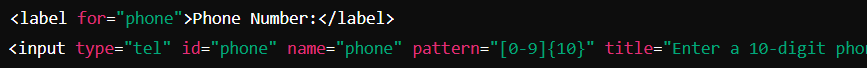
The pattern attribute in HTML is used to specify a regular expression that the input element's value must match for the form to be submitted. It’s primarily used for client-side validation of the data entered by the user, ensuring that it adheres to a specific format.



### How it Works

* **Regular Expression**: The value of the pattern attribute is a regular expression, a sequence of characters that defines a search pattern. The input will be validated against this pattern.
* **Input Types**: The pattern attribute can be used with input types like text, search, tel, url, email, and password.
* **Validation**: If the input doesn’t match the pattern, the form will not submit, and a message will be displayed to the user. The title attribute can be used to provide a custom error message.

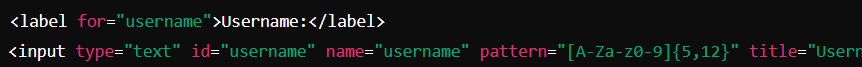
Example 1: Validating a Phone Number



### How it Works

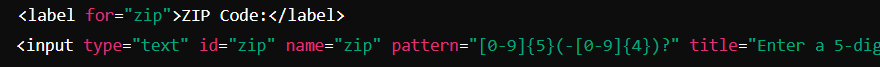
* **Pattern**: [0-9]{10} ensures that the user enters exactly 10 digits.
* **Error Message**: If the user enters something other than a 10-digit number, the form will display the message provided in the title attribute.

### Example 2: Validating an Alphanumeric Username



* **Pattern**: [A-Za-z0-9]{5,12} ensures that the username is between 5 and 12 characters long and consists only of letters (both uppercase and lowercase) and numbers.
* **Error Message**: The title attribute explains the requirement to the user.

### Example 3: Validating a ZIP Code



* **Pattern**: [0-9]{5}(-[0-9]{4})? allows for either a 5-digit ZIP code or a 9-digit ZIP+4 code (e.g., 12345 or 12345-6789).
* **Error Message**: The title attribute instructs the user on the correct format.

### Why Use the pattern Attribute?

* **User Experience**: Helps users enter data in the correct format.
* **Client-Side Validation**: Prevents invalid data from being submitted, reducing server load.
* **Custom Validation**: Allows developers to enforce specific input formats based on the needs of the application.

Placeholder attribute:

The placeholder attribute in HTML is used to provide a short, descriptive hint about the expected value of an input field. This hint is displayed inside the input field before the user enters any data. The placeholder text disappears when the user starts typing in the field.

****

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**<label> element:**

In HTML, the <label> element is used to associate a label with a form control, such as an input field, textarea, select element, or button. Labels provide context and improve accessibility by helping users understand the purpose of form controls.

Here are some attributes commonly used with the <label> element:

**for**: The for attribute specifies which form control the label is associated with. It takes the ID of the form control as its value. When a user clicks on the label, the associated form control receives focus or activation. This attribute is essential for accessibility.

**<label for="username">Username:</label>**

**<input type="text" id="username">**

**form:** The form attribute specifies the ID of the form element that the label's associated control belongs to. This attribute is useful when form controls are outside the form element but should still be associated with it.

**html code:**

**<form id="myForm">**

**<label for="email" form="myForm">Email:</label>**

**<input type="email" id="email">**

**</form>**

**title:** The title attribute provides additional information about the label. It usually displays as a tooltip when the user hovers over the label.

**html code:**

**<label for="password" title="Enter your password">Password:</label>**

**<input type="password" id="password">**

**accesskey:** The accesskey attribute allows you to define a keyboard shortcut to focus on the form control associated with the label. It's typically used to enhance accessibility by enabling users to navigate forms using the keyboard.

**Html code:**

**<label for="search" >Search:</label>**

**<input type="text" id="search" accesskey="s">**

These attributes help ensure that form controls are properly associated with their corresponding labels, improving usability and accessibility for users interacting with web forms.

**how accesskey works for <label> in html:**

The accesskey attribute in HTML is used to specify a keyboard shortcut for navigating to an element. When an element has an accesskey attribute, users can activate the shortcut key combination specified in combination with the browser-defined modifier keys (such as Alt on Windows or Control on macOS).

**Here's how accesskey works for the <label> element in HTML:**

**Setting Access Keys:**

You can assign an access key to a <label> element using the accesskey attribute.

The value of the accesskey attribute should be a single character that represents the access key.

For example:

**html code:**

**<label for="search">Search:</label>**

In this example, pressing Alt + S (on Windows) or Control + S (on macOS) would focus on the input field associated with the label.

**Activating the Access Key:**

The exact key combination to activate the access key depends on the browser and operating system.

Typically, users need to press the browser-defined modifier key along with the access key itself.

After pressing the key combination, the browser focuses on the associated form control.

**Accessibility Considerations:**

Access keys can improve accessibility by providing an alternative means of navigation for users who may have difficulty using a mouse.

However, it's important to avoid conflicts with browser or screen reader shortcuts and to ensure that access keys are intuitive and well-documented for users.

**Here's a summary of how accesskey works for <label> elements in HTML:**

It provides a keyboard shortcut to quickly focus on the form control associated with the label.

Users activate the shortcut key combination along with the browser-defined modifier key to access the form control.

Access keys can enhance accessibility by enabling keyboard navigation, but they should be chosen carefully to avoid conflicts and confusion for users.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**<fieldset>:**

In HTML, the <fieldset> element is used to group related form elements together within a web form. It provides a semantic grouping of form controls, making it easier for users to understand the relationships between different parts of the form. Additionally, <fieldset> elements are often used in conjunction with <legend> elements, which provide a title or caption for the group of form controls.

**Here's how the <fieldset> element works:**

**Grouping Form Controls:**

The <fieldset> element allows you to group related form controls, such as radio buttons, checkboxes, text inputs, etc., together within a single container.

This grouping helps organize the form and provides visual cues to users about the relationships between different input fields.

**Adding a Legend:**

The <legend> element is used inside the <fieldset> element to provide a title or caption for the group of form controls.

The <legend> element should be the first child of the <fieldset> element.

It typically appears as a title or label above the group of form controls, providing context for the user.

**Disabling Form Controls:**

You can disable an entire <fieldset> and all its contained form controls by adding the disabled attribute to the <fieldset> element.

This can be useful when you want to prevent users from interacting with a specific section of the form based on certain conditions.

**Here's an example of how <fieldset> and <legend> elements are used:**

**html code:**

**<form>**

**<fieldset>**

**<legend>Personal Information</legend>**

**<label for="name">Name:</label>**

**<input type="text" id="name" name="name"><br>**

**<label for="email">Email:</label>**

**<input type="email" id="email" name="email"><br>**

**</fieldset>**

**<fieldset>**

**<legend>Preferences</legend>**

**<input type="checkbox" id="newsletter" name="newsletter">**

**<label for="newsletter">Subscribe to newsletter</label><br>**

**<input type="checkbox" id="updates" name="updates">**

**<label for="updates">Receive updates</label><br>**

**</fieldset>**

**<input type="submit" value="Submit">**

**</form>**

In this example, we have two <fieldset> elements: one for "Personal Information" and another for "Preferences". Each <fieldset> contains related form controls, and the <legend> element provides a title for each group. This helps improve the organization and usability of the form.

**file form control in html :**

In HTML, the <input> element with the type="file" attribute is used to create a file input control. This control allows users to select one or more files from their local file system to be uploaded to the server when the form is submitted.

Here's how you can use the file input control in HTML:

**html code:**

**<form action="/upload" method="post" enctype="multipart/form-data">**

**<label for="file">Select a file:</label>**

**<br> <input type="file" id="file" name="file"><br>**

**<input type="submit" value="Upload">**

**</form>**

**Explanation of the code:**

<form>: This is the HTML form element. It specifies where the form data will be sent when the form is submitted (action="/upload") and the HTTP method to be used (method="post").

enctype="multipart/form-data": This attribute is required when the form includes file input controls. It specifies how form data should be encoded before sending it to the server.

<label>: This is a label element associated with the file input control. It provides a textual description of the control.

for="file": The for attribute of the label specifies which form control the label is associated with. It matches the id attribute of the file input control.

<input type="file">: This is the file input control. Users can click on it to open a file dialog box where they can select one or more files from their local file system.

id="file": This attribute uniquely identifies the file input control and is used by the associated label element.

name="file": This attribute specifies the name of the file input control. It is used to identify the uploaded file on the server.

<input type="submit">: This is the submit button. When clicked, it submits the form data to the server.

When the form is submitted, the selected file(s) will be uploaded to the server. The server-side code (e.g., PHP, Python, Node.js) can then process the uploaded file(s) according to the application's requirements.

**display: flex;** is a CSS property that is used to define a flex container, which enables a flexible layout mode within its children elements. This property is a part of the CSS Flexible Box Layout Module, commonly known as Flexbox.

When you apply **display: flex;** to an element, it essentially turns that element into a flex container, and its direct children become flex items. Here's a breakdown of how **display: flex;** works:

1. **Flex Container**: The element with **display: flex;** becomes a flex container. This means that it organizes its direct children (flex items) along a flex line.
2. **Flex Items**: The children of the flex container become flex items. These items can be aligned and arranged in various ways within the flex container.

Once a container is set to **display: flex;**, you can use additional properties to control the layout and behavior of its children (flex items). Some of the most common properties used with flexbox include:

* **justify-content**: This property aligns flex items along the main axis of the flex container. It can distribute space between items or around them, aligning them towards the start, end, or center of the container.
* **align-items**: This property aligns flex items along the cross axis of the flex container. It determines how the items are positioned in the container if they do not take up all the available space along the cross axis.
* **flex-direction**: This property establishes the main axis of the flex container and defines the direction in which flex items are placed. It can be set to **row** (default), **row-reverse**, **column**, or **column-reverse**.
* **flex-wrap**: This property specifies whether flex items are forced onto a single line or can wrap onto multiple lines within the flex container if there is not enough space along the main axis.
* **align-content**: This property aligns a flex container's lines within the flex container when there is extra space in the cross-axis.

These properties, along with others, give you precise control over the layout and alignment of elements within a flex container, making it a powerful tool for creating responsive and dynamic web layouts.