**HTML**

What is HTML

HTML stands for Hypertext Markup Language, which is used to structure content on the web page.HTML uses tags to structure content on the web. HTML tags are enclosed in angle brackets and come in pairs: an opening tag and a closing tag. HTML is not a programming language, it's a markup language that uses tags to format text, images, audio, video on a webpage.

Here are some key points about HTML:

1) Markup Language:

HTML is a markup language, not a programming language. It uses tags to display text, images, and other content on a web page.

2) Structural Hierarchy:

HTML documents are organized hierarchically. They have a root element <html> that contains two childs sections: <head> (which contains meta-information and links to external resources) and <body> (which contains main content displayed in web browser).

3) Elements and Tags:

HTML documents are made up of elements. Elements are defined using tags. For example, a paragraph is defined using the <p> tag: <p>This is a paragraph.</p>. Tags typically come in pairs, with an opening tag <p> and a closing tag </p>. The content is placed between these tags.

4) Attributes:

Many HTML elements can have attributes that provide additional information about the element or modify its behavior. For example, the <img> element can have an "src" attribute to specify the image file source.

-Web developers use HTML to create the structure and content of web pages, and then they use CSS to style those pages and JavaScript to add interactivity and functionality.

Basic tags

!DOCTYPE>: Specifies the document type and version of HTML being used.

<html>: The root element that contains all other HTML elements on the page.

<head>: Contains metadata about the document, such as the title, and linked stylesheets or scripts and meta tags.

<title>: Defines the title of the webpage, which appears in the browser's title bar or tab.

<meta>: Provides metadata about document.

<link>: Links external resources such as stylesheets and icons.

<script>: Embeds or references JavaScript code/file.

<style>: Defines CSS for styling the page.

<body>: Contains the main content of the webpage, including text, images, audio, video and other elements.

meta tags

The <meta> tag defines metadata about an HTML document. Metadata is data (information) about data.

<meta> tags always go inside the <head> element, and are typically used to specify character set, page description, keywords, author of the document, and viewport settings.

Metadata will not be displayed on the page, but is machine parsable.

Metadata is used by browsers (how to display content or reload page), search engines (keywords), and other web services.

1) <meta charset="UTF-8">:

Specifies the character encoding for the document. UTF-8 is widely used for encoding text in various languages.To display an HTML page correctly, a web browser must know which character set to use.The HTML5 specification encourages web developers to use the UTF-8 character set.

UTF-8 covers almost all of the characters and symbols in the world!

2) <meta name="description" content="Your page description">:

Provides a brief summary or description of the web page's content. Search engines often use this information in search results.

3) <meta name="keywords" content="keyword1, keyword2, keyword3">:

Specifies a list of keywords or key phrases related to the page's content. However, this tag is no longer widely used by search engines to determine rankings.

4) <meta name="author" content="Author Name">:

Identifies the author of the web page's content.

5) <meta name="viewport" content="width=device-width, initial-scale=1.0">:

Sets the initial scale and width of the viewport, important for responsive design.

6) <meta http-equiv="refresh" content="20">:

Automatically redirects the browser to another page after a specified time interval (in this case, 20 seconds).

=> html tags vs html elements vs html attributes.

HTML Attributes

Html attributes are the properties of html element and it is used to give extra information about element to the browser. Every attribute will be in the key value pair.

Block-level and Inline elements

-A block-level element always starts on a new line and takes up the full width available.

An inline element does not start on a new line and it only takes up as much width as necessary.

HTML Elements

1) Heading elements

<h1>content</h1>

h1 to h6

- Heading tags are used to define headings or titles for sections of a web page.

- We have different levels of headings from h1 to h6.

- It is a block-level element, which means it starts in a new line and takes 100% width.

2) Pargraph element

<p>content</p>

- The <p> tag is used to define and format text as a paragraph.

- It is a block-level element, which means it starts on a new line and takes 100% width.

3) image element

<img src='' width='' height='' alt='' title=''/>

- The <img> tag is used to display images on a web page. It is a self-closing tag, which means it doesn't have a closing tag.

- It is inline element, It takes only required space for image.

- Html attributes are the properties of html element and it is used to give extra information about element to the browser. Every attribute will be in the key value pair.

Attributes:

src

alt

width

height

title

4) List elements:

<ul>: Defines an unordered (bulleted) list.

<ol>: Defines an ordered (numbered) list.

<li>: Represents a list item.

<dl>: Defines a description list.

<dt>: Represents a term in a description list.

<dd>: Provides the description for a term.

Attributes:

- type: we can use type attribute for only ordered and unordered elements.

- start and reversed attributes are only for ordered list.

- start will only work for numbers.

- These are only for ordered list

- Number (default)

- type = 'a/A/I/i'

- reversed

- start

- Alpha

lower case (a)

UPPER CASE (A)

- Roman

lower case (i)

UPPER CASE (I)

- These are only for unordered list

type:

- disc (default)

- circle

- square

- none

5) Anchor element

<a href='' target=''>content </a>

- The <a> tag (anchor tag) in HTML is used to create a links in the webpage.

- By using links we can navigate from one webpage to other web page(inter navigation).

- By using links we can navigate from one section to another section with in the same webpage

(intra navigation).

Navigation:

1) Inter navigation (navigation between documents)

2) Intra navigation (navigation between sections with in document)

Atttribute:

href => hyper reference

href = 'path of another webpage' or '#id value of section in same page'

target => Specifies where to open the linked document.

\_blank

\_parent

\_self (default)

\_top

\_blank => Opens the linked document in a new window or tab.

\_self => Opens the linked document in the same tab as it was clicked (default).

framename => Opens the linked document in the named iframe.

6) Button element

- The <button> tag in HTML is used to define the clickable button. <button> tag is used to submit the content. The images and text content can be used inside <button> tag.

Attributes

----------

disabled

autofocus

type

Following are the three values of type attribute:

Button: The button value is used for the clickable button.

Submit: The submit value is used for submitting the form-data.

Reset: The reset value is used to reset the values of a form to its initial values.

7) <br/>

- The <br> tag in HTML document is used to create a line break in a text. It is self closing.

- Don't use br tag for the margin between two paragraphs, use CSS margin property instead.

8) <hr>

The <hr> tag in HTML stands for horizontal rule and is used to insert a horizontal rule to divide or separate document sections. The <hr> tag is an empty tag, and it does not require an end tag.

Attributes

width,

color,

size,

align.

9) Center element

- The <center> HTML element is a block-level element that displays its block-level or inline contents centered horizontally within its containing element.

- This tag is not supported in HTML5. CSS’s property is used to set the alignment of the element instead of the center tag in HTML5.

10) HTML entities

=> &nbsp; A commonly used HTML entity is the non-breaking space.

=> &copy; It will show copy right symbol.

=> &reg; It will show trademark symbol.

11) Division element

<div> </div>

- It is used to create divisions or sections inside the html documents.

- The <div> tag is used as a container for HTML elements.

- Div tag is Block level tag.

- It is a generic container tag.

- It is used to group various tags of HTML so that sections can be created.

- As we know Div tag is block-level tag, the div tag contains entire width.

Hence, every div tag will be started from a new line, and not the same line.

12) Span element

<span> </span>

- The <span> tag is an inline container used to mark up a part of a text, or a part of document.

Note:

Div and Span Tags: These are generic containers that can be used to group and style content. <div> is a block-level element, and <span> is an inline element.

- <div>This is a block-level container. </div>

- <span>This is an inline container. </span>

13) Formatting Elements

Formatting elements were designed to display special types of text with css styles:

Bold text: <b> or <strong>

HTML5 provides two tags to make text bold: <b> and <strong>.

Both tags have the same effect of making the enclosed text bold, but <strong> is preferred because it indicates that the text is important for semantic purposes.

Italicized text: <i> or <em>

HTML5 provides two tags to make text italic: <i> and <em>. Both tags have the same effect of making the enclosed text italic, but <em> is preferred because it indicates that the text has emphasis for semantic purposes.

Strike-through text: <del> or <s>

HTML5 provides two tags to make text strike-through: <s> and <del>. Both tags have the same effect of making the enclosed text strike-through, but <del> is preferred because it indicates that the text has been deleted for semantic purposes.

Superscript and subscript text: <sup> or <sub>

HTML5 provides two tags to make text superscript and subscript: <sup> and <sub>, respectively.

u and ins

HTML5 provides two tags u and ins. Both tags have the same effect of making underline.

<ins> is used for semantic purpose. The <ins> tags means content inserted after it was first published. The <u> tag is simply for underlining and has no meaning.

small vs big

In HTML, the <small> tag is used to emphasize small text or information that is of secondary significance, such as copyright and legal disclaimer.

The <small> tag can be used to mark up a copyright notice or license agreement.

HTML <big> tag was used to increase the text font size one level bigger than the document's base font size.

<b> - Bold text

<strong> - Important text

<i> - Italic text

<em> - Emphasized text

<mark> - Highlighted Text

<del> - Deleted text

<s> - text strick

<ins> - Inserted text

<u> - Underline text

<sub> - Subscript text

<sup> - Superscript text

<small> - Smaller text

<big> tag - Big Text

14) Table elements

<table> - Defines a table

<caption> - Defines a table caption

<thead> - Groups the header content in a table

<tbody> - Groups the body content in a table

<tfoot> - Groups the footer content in a table

<tr> - Defines a row in a table (rows)

<th> - Defines a header cell in a table (columns)

<td> - Defines a data cell in a table (columns)

Attributes:

1) border = '1' | '5' | '10' => if we use border then no need of frame and rules.

frame = box | border | above | below | hsides | vsides | lhs | rhs | void

rules = all | rows | cols | groups | none

2) width => It will effect the table header & table body & table footer.

height => It will give effect only to table body but not effect the table header(thead) & (tfoot) footer.

3) cellspacing => It gives space between all cells.

cellpadding => It gives space with in cells.

4) colspan => merging columns in specific row.

rowspan => merging rows in specific column.

5) text-align => text align horizontally.

vertical-align => text align vertically.

6) align = left | center | right

7) bgcolor => bg color to table.

Colspan

Colspan is an attribute which assigns multiple columns to a cell of a table. The number of columns depends on the value entered by you in colspan="" attribute.

rowspan

Rowspan in table, works similar to the clospan for columns, but here, we assign multiple rows to a cell using an attribute rowspan=""

horizontal alignment (text in table)

The text-align property sets the horizontal alignment (like left, right, or center) of the content in <th> or <td>.

By default, the content of <th> elements are center-aligned and the content of <td> elements are left-aligned.

vertical alignment (text in table)

The vertical-align property sets the vertical alignment (like top, bottom, or middle) of the content in <th> or <td>.

By default, the vertical alignment of the content in a table is middle (for both <th> and <td> elements).

15) Form element

Form element is used to take input from an user to send data to server and then db.

The <form> element is a container for different types of input elements, such as: text fields, checkboxes, radio buttons, submit buttons, etc.

Form Elements:

The HTML <form> element can contain one or more of the following form elements.

<form> => Defines an HTML form for user input.

<input> => <input> element can be displayed in many ways, depending on the type attribute.

<label> => Defines a label for an <input> element.

The for attribute of the <label> tag should be equal to the id attribute of the <input> element to bind them together.

<textarea> => Defines a multiline input control (text area).

The <textarea> element is often used in a form, to collect user inputs like comments or reviews.The size of a text area is specified by the cols and rows attributes.

<button> => Defines a clickable button.

<fieldset> => Groups related elements in a form.

<legend> => Defines a caption for a <fieldset> element.

<select> => Defines a drop-down list.

<option> => Defines an option in a drop-down list.(The <option> tags inside <select> element define the available options in the drop-down list).(selected attribute)

<optgroup> => Defines a group of related options in a drop-down list.

<datalist> => Specifies a list of pre-defined options for input controls.

-The <datalist> tag is used to provide an "autocomplete" feature for <input> elements. Users will see a drop-down list of pre-defined options as they input data.

-The <datalist> element's id attribute must be equal to the <input> element's list attribute (this binds them together).

Form attributes:

-autocomplete: 'on' (in form opening tag).

-novalidate => form-data (input) should not be validated when submitted.

-onSubmit: form data will be submited when click on submit button.

Input element Attributes

Input element attributes

1.type: text | password | email | button | radio | checkbox | number | color | date | reset | submit | tel | range | time | month

The type attribute is used to specify the type of the input element. Its default value is text.

2.placeholder: Placeholder attribute is used to specify hint that describes the expected value of an input field.

3.value: The value attribute is used to specify the value of the input element.

4.name: The name attribute is used to specify the name of the input element.

5.maxlength: This property is used to specifies the maximum number of characters allowed in an <input> element

6.size: This property is used to specifies the width of <input> element(default 20 chars)

7.required: The required attribute specifies that an input field must be filled out before submitting the form.

8.disabled: The disabled attribute specifies that the element should be disabled. The disabled attribute can be set to keep a user from using the < input > element until some other condition has been fulfilled.

9.readonly: The readonly attribute specifies that an input field is read-only. A read-only input field cannot be modified. A form will still submit an input field that is readonly, but will not submit an input field that is disabled.

Note:

The default width of an input field is 20 characters.

- The for attribute of the <label> tag should be equal to the id attribute of the <input> element to bind them together.

- Name attribute should be same value for all radio buttons to select any one of them.

- The <input type="submit"> defines button for submitting the form data to form-handler(function).

16) marquee element

The <marquee> tag is a container for creating scrollable text or images within a web page from either left to right or vice versa, or top to bottom or vice versa. But this tag has been deprecated in the new version of HTML, i.e., HTML 5

Attributes

width

height

direction : left (default right to left) | right | up | down

behaviour : slide | scrolling (defalut) | alternate

scrolldelay : default 85 (to make slow)

scrollamount : default 6 (to make speed)

bgcolor

loop : times of loop (The default value of loop is INFINITE)

17) symantic elements

Semantic elements = elements with a meaning.

A semantic element clearly describes its meaning to both the browser and the developer.

Examples of non-semantic elements: <div> and <span> - Tells nothing about its content.

Examples of semantic elements: <form>, <table>, and <article> - Clearly defines its content.

<table> : Defines table.

<form> : Defines form.

<details> : Defines additional details that the user can view or hide.

<summary> : Defines a visible heading for a <details> element.

<header> : Defines a header for a document or section.

<nav> : Defines navigation links (nav contains <a> tag).

<footer> : Defines a footer for a document or section.

<main> : Specifies the main content of a document.

<section> : The <section> element defines a section in a document.

<article> : Defines an article. Examples of where the <article> element can be used.

(Forum posts, Blog posts, User comments, Product cards, Newspaper articles)

<aside> : Defines content aside from the page content. The <aside> content should be indirectly related to the surrounding content.

<figure> : Specifies self-contained content, like illustrations, diagrams, photos etc.

<figcaption> : Defines a caption for a <figure> element.

Note: All semantic tags contains no default styles as like div.

18) iframe

The iframe in HTML stands for Inline Frame.

- Use the <iframe> tag to embed another document within the current HTML document.

- The HTML iframe "name" attribute is used to specify a reference for an <Iframe> element.

- The iframe is basically used to show a webpage inside the current web page.

- The "src" attribute is used to specify the URL of the document that occupies the iframe.

Attributes

src

width

height

name

frameBorder

-An iframe can be used as the target frame for a link. The target attribute of the link must refer to the name attribute of the iframe.

-Bydefault iframe has border, we can remove border by using css also.

19) Media Elements

- HTML5 introduces media elements like <audio> and <video> for displaying and playing audio

and video content.

- The <video> tag is used to embed video content in a document, such as some videos.

- The <audio> tag is used to embed sound content in a document, such as some audios.

<audio controls autoplay muted loop>

<source src="">

</audio>

<video width="320" height="240" controls autoplay muted loop>

<source src="">

</video>

Attributes

width

height

controls

autoplay

muted

poster

loop

- The <source> element allows you to specify alternative audio files.

- controls: attribute adds audio controls, like play, pause, and volume.

- autoplay: it will work with muted for videos.

- loop: Specifies that the video will start over again, every time it is finished.

- muted: Specifies that the audio output of the video should be muted.

- poster: Specifies an image to be shown while the video is downloading, or until the user hits the play button.

20) Web storage

=> Local storage

=> Session storage

With web storage, web 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.

It is used to store data on the client side. It has no expiration time, so the data in the LocalStorage exists always till the user manually deletes it.

HTML web storage provides two objects for storing data on the client:

window.localStorage - stores data with no expiration date

window.sessionStorage - stores data for one session (data is lost when the browser tab is closed)

=> Store

localStorage.setItem("name", "Hari");

=> Retrieve

document.getElementById("result").innerHTML = localStorage.getItem("name");

=> remove

localStorage.removeItem('name')

localStorage.clear()

22. Block-level vs Inline

Block

|  |  |  |  |
| --- | --- | --- | --- |
| <address> | <hr> | <abbr> | <map> |
| <article> | <li> | <acronym> | <object> |
| <aside> | <main> | <b> | <output> |
| <blockquote> | <nav> | <bdo> | <q> |
| <canvas> | <noscript> | <big> | <samp> |
| <dd> | <ol> | <br> | <script> |
| <div> | <p> | <button> | <select> |
| <dl> | <pre> | <cite> | <small> |
| <dt> | <section> | <code> | <span> |
| <fieldset> | <table> | <dfn> | <strong> |
| <figcaption> | <tfoot> | <em> | <sub> |
| <figure> | <ul> | <i> | <sup> |
| <footer> | <video> | <img> | <textarea> |
| <form> | Inline | <input> | <time> |
| <h1>-<h6> | <a> | <kbd> | <tt> |
| <header> |  | <label> | <var> |

**CSS CONTENT**

1. What is CSS

CSS stands for Cascading Style Sheet. It is a styling language, by using CSS we can style the html content or elements on webpage by using CSS properties. CSS properties will be in key value pair.

2. How many ways we can add css to html elements

1) Inline

- By using style attribute we can apply css directly to that html element. By using inline style we can apply only css to only one element at a time.

2) Internal

- By using style tag in head tag we can apply css to many html elements in the same html file at a time html elements.

3) Extrenal

- By using external style sheet we can apply css to many html elements in different html files at a time. We can link external css file to html file by using link tag in head tag.

3. CSS selectors

CSS selectors are used to select the HTML elements you want to style.

- We can divide CSS selectors into several categories:

1. Simple selectors (select elements based on tag, id, class) (3 selectors)

2. Combinator selectors (select elements based on a specific relationship between them)

a) Descendent selector => div p {} - all childs

b) Child selector => div>p {} - direct childs

c) Adjacent sibling => div+p {} - next one (sibling)

d) General sibling => div~p {} - next all siblings

3. Pseudo-class selectors (select elements based on a certain state)

4. Pseudo-elements selectors (select and style a part of an element)

5. Attribute selectors (select elements based on an attribute or attribute value) (4 selectors)

6. Universal Selector (The universal selector (\*) selects all HTML elements on the page)

7. Grouping Selector (To group selectors, separate each selector with a comma)

Allows you to apply the same styles to multiple selectors.

Example: h1, h2, h3 { font-family: Arial, sans-serif; }

1.Type (Element) Selector:

Selects elements by their HTML tag name.

Example: p { color: blue; }

2.Class Selector:

Selects elements by their class attribute.

Example: .my-class { font-weight: bold; }

3.ID Selector:

Selects a single element by its unique ID attribute.

Example: #my-id { background-color: yellow; }

4.Universal Selector:

Selects all elements on the page.

Example: \* { margin: 0; }

5.Descendant Selector (all childs):

Selects an element that is a descendant of another element.

Example: ul li { list-style: square; }

6.Child Selector (direct child):

Selects an element that is a direct child of another element.

Example: ul > li { font-style: italic; }

7.Adjacent Sibling Selector (next sibling):

Selects an element that is immediately preceded by another element.

Example: h2 + p { color: red; }

8.General Sibling Selector (next all siblings):

Selects elements that are siblings of a specified element.

Example: h2 ~ p { font-size: 18px; }

9.Pseudo-Class Selector:

Selects elements based on a specific state or interaction, such as :hover, :active, and :focus.

Example: a:hover { text-decoration: underline; }

10.Pseudo-Element Selector:

Selects and styles a part of an element's content, such as

=> ::first-letter ::first-line and ::before and ::after

Example: p::before { content: "Note: "; }

Pseudo-elements in CSS are used to style a specific part of an element. They are denoted by double colons (::) and are often used with the ::before and ::after pseudo-elements to insert content before or after an element. Here's an example of using the ::before pseudo-element to add content before an element:

.myclass::before {

content: "This is content";

font-weight: bold;

color: red;

}

.myclass::after {

content: "This is content";

font-weight: bold;

color: white;

}

11.Grouping Selector:

Allows you to apply the same styles to multiple selectors.

Example: h1, h2, h3 { font-family: Arial, sans-serif; }

12.Attribute Selector:

Selects elements based on the value of their attributes.

Example: input[type="text"] { border: 1px solid #ccc; }

13.Attribute Value Starts With Selector:

Selects elements whose attribute values start with a specified string.

Example: [href^="https://"] { color: blue; }

14.Attribute Value Ends With Selector:

Selects elements whose attribute values end with a specified string.

Example: [src$=".jpg"] { border: 1px solid #ccc; }

15.Attribute Value Contains Selector:

Selects elements whose attribute values contain a specified string.

Example: [alt\*="cat"] { border: 2px solid orange; }

5. Text Properties

CSS (Cascading Style Sheets) provides a wide range of text properties that allow you to control the appearance and layout of text within HTML elements. A list of common text properties in CSS.

1.color:

Sets the color of the text.

Example: color: red;

2.direction:

Sets the text direction, such as ltr (left-to-right) or rtl (right-to-left).

Example: direction: rtl;

3.text-transform:

Modifies the capitalization of text, e.g., uppercase or lowercase or capitalize.

Example: text-transform: uppercase;

4.text-shadow:

It gives shadow to text.

text-shadow: x-axis y-axis blur color

text-shadow: 10px 0px 10px red

5.text-align:

Aligns the text horizontally within its container.

Example: text-align: center;

6.text-decoration:

Controls decorations like underline, overline, and line-through.

Example: text-decoration: underline;

7.text-indent:

Sets the indentation of the first line of text within an element.

Example: text-indent: 20px;

8.font-family:

Specifies the font family for the text.

Example: font-family: Arial, sans-serif;

9.font-size:

Sets the size of the text.

Example: font-size: 16px;

10.font-weight:

Defines the thickness or boldness of the text.

Example: font-weight: bold;

11.font-style:

Specifies the style of the text, such as italic.

Example: font-style: italic;

12.line-height:

Sets the height of a line of text, controlling spacing between lines.

Example: line-height: 1.5;

13.letter-spacing:

Adjusts the space between individual characters.

Example: letter-spacing: 2px;

14.word-spacing:

Controls the spacing between words in a block of text.

Example: word-spacing: 4px;

These are some of the essential text properties in CSS. By using these properties, you can customize the appearance and layout of text on your web pages, making it an integral part of web design and typography.

6. Background Properties

1.background-color:

<style>

.example {

background-color: #F0F0F0;

}

</style>

<div class="example">

This is a div with a colored background.

</div>

2.background-image:

<style>

.example {

background-image: url('image.jpg');

}

</style>

<div class="example">

This is a div with a background image.

</div>

3.background-repeat: ( repeat (default) | repeat-x | repeat-y | no-repeat )

=> Repeating horizontally and vertically

.example-repeat {

background-image: url('repeating-pattern.png');

background-repeat: repeat;

}

=> Repeating horizontally only

.example-repeat-x {

background-image: url('horizontal-pattern.png');

background-repeat: repeat-x;

}

=> Repeating vertically only

.example-repeat-y {

background-image: url('vertical-pattern.png');

background-repeat: repeat-y;

}

=> No repetition

.example-no-repeat {

background-image: url('single-image.jpg');

background-repeat: no-repeat;

}

ex:

<style>

.example {

background-image: url('image.jpg');

background-repeat: no-repeat;

}

</style>

<div class="example">

This is a div with a non-repeating background image.

</div>

4.background-position: left top | left center | left bottom | right top | right center |

------------------ right bottom | center top | center center | center bottom

<style>

.example {

background-image: url('image.jpg');

background-position: center top;

}

</style>

<div class="example">

This is a div with a centered top-aligned background image.

</div>

5.background-size:

=> Original size (auto)

.example-auto {

background-image: url('large-image.jpg');

background-size: auto;

}

=> Scale to cover the content area

.example-cover {

background-image: url('image.jpg');

background-size: cover;

}

=> Scale to fit within the content area

.example-contain {

background-image: url('image.jpg');

background-size: contain;

}

=> Specific size in pixels

.example-specific-size {

background-image: url('image.jpg');

background-size: 200px 150px;

}

=> Relative size with percentages

.example-percent-size {

background-image: url('image.jpg');

background-size: 50% 75%;

}

Ex:

<style>

.example {

background-image: url('image.jpg');

background-size: cover;

}

</style>

<div class="example">

This is a div with a background image that covers the entire element.

</div>

Note:

The 'cover' tells the browser to cover the whole area of an element.

The 'contain' tells the browser to show the whole image without loosing image.

6.background-attachment: (scroll | fixed)

=> Background image scrolls with content (default)

.example-scroll {

background-image: url('scrolling-background.jpg');

background-attachment: scroll;

}

=> Background image remains fixed in place

.example-fixed {

background-image: url('fixed-background.jpg');

background-attachment: fixed;

}

Ex:

<style>

.example {

background-image: url('image.jpg');

background-attachment: fixed;

}

</style>

<div class="example">

This is a div with a fixed background image.

</div>

These are just a few examples of how you can use CSS background properties to style and enhance the backgrounds of HTML elements. You can combine these properties and adjust their values to achieve the desired visual effects in your web pages.

7. Border properties

In CSS (Cascading Style Sheets), you can use various properties to control the appearance of borders around elements such as text, images, and containers. Here are some common border properties in CSS:

1)border-width:

This property sets the width of the border. You can specify the width in pixels, ems, rems, percentages, or other units.

border-width: 2px;

2)border-color:

This property sets the color of the border. You can use color names, hexadecimal values, RGB values, or other color notations.

border-color: #FF0000;

3)border-style:

This property sets the style of the border, and it can take values like "solid," "dashed," "dotted," "double," "none,".

border-style: solid;

4)border-radius:

This property sets the radius of the corners of an element, creating rounded corners. You can specify different values for each corner or use a single value for all corners.

The border-radius property is actually a shorthand property for the

border-top-left-radius,

border-top-right-radius,

border-bottom-right-radius

border-bottom-left-radius

border-radius: 10px;

5)border-image:

This property allows you to use an image as a border instead of a solid color. It's a more advanced property and requires defining an image to use as a border.

border-image: url(border-image.png) 27 27 27 27 round round;

border-image-source: url('./102.PNG');

border-image-slice: 30%;

border-image-width: 20px;

border-image-repeat: repeated, stretched(default).

border-image-outset: 20px

border-image is a shorthand property that lets you use an image or CSS gradient as the border of an element.

The border-image property can be applied to any element, except internal table elements (e.g. tr, th, td) when border-collapse is set to collapse.

The border-image-outset property specifies the amount by which the border image area extends beyond the border box

border:

This is a shorthand property that combines border-width, border-style, and border-color into a single declaration.

border: (width) 1px (style)solid (color) #000;

8. Margin and Padding properties

margin

The CSS margin property is used to give space around an element on a web page. It defines the area outside the element's border. Margins are used to create space and separation between elements.

1)margin-top: 10px;

2)margin-right: 20px;

3)margin-bottom: 15px;

4)margin-left: 25px;

5)margin: 10px 20px 15px 25px; (top, right, bottom, left)

6)margin: 10px 20px (Top and bottom have 10px margin, right and left have 20px margin)

7)margin: 10px 20px 15px (Top has 10px margin, right and left have 20px margin, bottom has 15px margin)

8)margin: 10px (all sides)

9)margin: auto

You can set the margin property to auto to horizontally center the element within its container.

The element will then take up the specified width, and the remaining space will be split equally between the left and right margins.

Margin Collapse

Top and bottom margins of elements are sometimes collapsed into a single margin that is equal to the largest of the two margins.

This does not happen on left and right margins! Only top and bottom margins!

padding

Padding is used to create space within an element, between the element's content and its border.

1)padding-top: 10px;

2)padding-right: 20px;

3)padding-bottom: 15px;

4)padding-left: 25px;

5)padding: 10px 20px 15px 25px; /\* top, right, bottom, left \*/

6)padding: 10px 20px

7)padding: 10px 20px 15px

8)padding: 10px (all sides)

margin vs padding

In CSS, both margin and padding are properties used to control spacing, but they affect the layout of elements in different ways:

Margin:

-Margin controls space outside an element, between the element's border and neighboring elements.

Margins create separation between elements.

-Margin values can be positive (adding space) or negative (reducing space).

-Margin do not have a background color or visible area, they only affect the spacing between elements.

-Margins can "collapse" under certain conditions, meaning that if two adjacent elements have margins, the larger of the two margins is used, and they don't stack.

Ex:

div {

margin: 10px;

}

In this example, a div element will have a 10-pixel margin around it, creating space between it and neighboring elements.

Margin collapse

The concept of "margin collapsing" in CSS refers to how margins between adjacent elements are computed when they are touching or close to each other. When two or more vertical margins meet or overlap under specific conditions, they don't simply add up or stack on top of each other as you might expect. Instead, they collapse, and the larger of the two margins is used to create the space between the elements. Here's a more detailed explanation:

=> Margins Collapsing Conditions:

Margins can collapse in the following situations:

When two or more block-level elements are vertically adjacent, meaning they are stacked on top of each other in the document flow.

When there is no padding, border, or inline content separating the elements.

When the margins are of the same type (e.g., both are top margins or both are bottom margins).

=> Result of Margin Collapsing:

When margins collapse, the larger of the margins is used, and the smaller margin effectively disappears in terms of creating spacing.

This means that if one element has a margin of 10px, and the adjacent element has a margin of 20px, the effective spacing between them will be 20px, not 30px.

=> Preventing Margin Collapsing:

You can prevent margin collapsing by introducing a non-collapsible element between the two elements. For example, adding padding, a border, or inline content between the elements will stop margin collapsing.

Example of margin collapsing:

p {

margin: 10px 0;

}

div {

margin: 20px 0;

}

In this example, if a <p> element and a <div> element are placed next to each other, the effective vertical space between them will be 20px, not 30px, because the larger of the two margins (20px) takes precedence

Padding:

-Padding controls the space inside an element, between the element's content and its border.

-Padding is used to create space within an element, affecting the content's position within the element's boundaries.

-Padding values can be positive (adding space) and are never negative.

-Padding has a background color and visible area. It extends the background of the element.

EX:

div {

padding: 10px;

}

In this example, a div element will have a 10-pixel padding inside it, pushing the content away from the border.

Note: Inline elements vertical padding

CSS, inline elements, by default, do not respect vertical padding in the same way that block-level elements do. Vertical padding on inline elements will not push away adjacent inline elements. Instead, it may cause overlapping or wrapping of the content within the inline element.

When you apply vertical padding to an inline element, such as <span> or a text within a paragraph, it increases the height of the element but doesn't create space around it like it does with block-level elements. This means that if you have multiple inline elements with padding, they might overlap or wrap to the next line depending on the available space within their container.

If you want to create space around inline elements, you can use margin instead of padding, or you can change the display property of the elements to inline-block or inline-flex, which will make them behave more like block-level elements, respecting vertical padding and margin.

<p>

<span class="inline">This is some text.</span>

<span class="inline">This is some text with padding.</span>

</p>

CSS:

span.inline {

padding: 10px;

margin: 10px;

}

In this example, the padding will increase the height of each inline element, but the adjacent inline elements won't be pushed away. They may overlap or wrap to the next line depending on the available width of the containing element. If you want to create space between these inline elements, you would need to use margin or consider changing the display property to make them behave like block-level elements.

9. Width and height properties

Note: width and height in % => depends on parent's width and height.

- width = Width will be always fixed irrespective of screen size.

- min-width = min-width is fixed for given size and beyong min-width it depends on screen size.

- max-width = max-width is fixed for given size and below max-width it depends on screen size.

- height = Height is fixed it will not be depend on screen size.

- min-height = min-height is fixed for given size and beyond min-height depends on screen size.

- max-height = max-height is fixed for given size and below max-height it depends on screen size.

Ex:

In CSS, height, min-height, and max-height are properties used to control the height of an element, such as a container or a block-level element. They serve different purposes and have distinct effects:

height:

The height property sets the exact height of an element to a specific value in pixels, ems, rems, percentages, or other length units. This value is often fixed and doesn't change unless you explicitly modify it.

If the content inside the element exceeds the specified height, it may overflow and be hidden or affect the layout of surrounding elements unless you use additional CSS properties to manage overflow, like overflow: hidden or overflow: scroll.

.element {

height: 200px;

}

min-height:

The min-height property sets the minimum height that an element should have. It ensures that the element is at least as tall as the specified value, but it can expand beyond that height if needed to accommodate content or other factors.

If the content within the element is taller than the specified min-height, the element's height will adjust to fit the content.

.element {

min-height: 100px;

}

max-height:

The max-height property sets the maximum height that an element can have. It restricts the height to not exceed the specified value. If the content or other factors would cause the element to be taller than the max-height, it will be limited to that height, and scrollbars may appear to allow users to access the overflow content.

.element {

max-height: 300px;

}

You can use these properties individually or in combination to control the height of elements in your web page layout. They are particularly useful when dealing with responsive web design, ensuring that elements adapt to various screen sizes and content while maintaining a defined range of height.

10. Display Property

In CSS, the display property is used to control how an HTML element is displayed in the web browser. It defines the type of box an element generates, which in turn affects its layout and rendering on the web page. There are various display property values that determine the element's behavior. Here are some common display property values:

1.block:

The element is displayed as a block-level element.

It takes up the full width available in its parent container and starts on a new line.

Examples of block-level elements include <div>, <p>, and <h1>.

2.inline:

The element is displayed as an inline-level element.

It takes up only as much width as necessary and does not start on a new line.

Examples of inline-level elements include <span>, <a>, and <strong>.

3.inline-block:

Combines characteristics of both inline and block.

The element is inline, but it can have its own width and height, as well as margin and padding.

Elements with this display type can be positioned inline, but they can have block-level styling.

Note:

=> inline The element doesn’t start on a new line and only occupy just the width it requires. You can’t set the width or height.

=> inline-block It’s formatted just like the inline element, where it doesn’t start on a new line. BUT, you can set width and height values.

=> block element will start on a new line and occupy the full width available. And you can set width and height values.

4.none:

The element is not displayed at all, effectively making it invisible.

It doesn't occupy any space in the layout.

5.table:

The element is displayed as a table element.

It can have child elements with table-row, table-cell, and other table-related display values.

Float and Clear

float: left | right | none

It is used to float an element either to the left or to the right in its parent container.

float: left

float an element to the left in its parent container.

=> Block level element will be converted into inline block element.

=> Remaining childs will be pushed up in container.

=> Height of parent container will be reduced.

float: right

float an element to the right in its parent container.

float: none

It is used to remove float effect from an already floated element.

clear: left | right | both

It is used to clear the effect of floating on the parent container.

clear: left

It clears the effect of floating left on its parent container.

clear: right

It clears the effect of floating left on its parent container.

clear: both

It clears the effect of floating left and right on its parent container.

parent to be normal

elementselector::after{

content: " ",

display: block,

clear: left

}

6.flex:

The element becomes a flex container, and its children become flex items.

It is used to create flexible and responsive layouts, allowing elements to adjust their sizes based on available space.

7.grid:

The element becomes a grid container, and its children become grid items.

It is used to create grid-based layouts, providing fine control over the arrangement of items.

display: none vs visibility: hidden vs opacity: 0

In CSS, display: none, visibility: hidden, and opacity: 0 are three different ways to hide elements on a web page, and they have distinct behaviors and use cases.

display: none

When you apply display: none to an element, the element is completely removed from the layout of the page. It takes up no space, and the document flows as if the element does not exist.

Child elements of the hidden element are also hidden, and they do not affect the layout.

Events and interactions (e.g., clicking, hovering) cannot target hidden elements.

Good for situations where you want to completely remove an element from view and layout.

element {

display: none;

}

visibility: hidden

When you apply visibility: hidden to an element, the element is still part of the layout, but it is not visible. It still takes up space on the page, as if it were still visible. Child elements of the hidden element are also hidden, but they still affect the layout. Events and interactions can still target hidden elements. Useful when you want to hide an element but maintain its space in the layout.

element {

visibility: hidden;

}

opacity: 0

When you set opacity: 0, the element becomes fully transparent, but it still occupies the same space in the layout.

Child elements of the transparent element remain visible and affect the layout.

Events and interactions can target the transparent elements.

Useful when you want to hide an element while maintaining its layout position and possibly animate the element's visibility.

element {

opacity: 0;

}

In summary, the choice of which property to use (display, visibility, or opacity) depends on your specific requirements. If you want to completely remove the element and its space from the layout, display: none is suitable. If you want to hide the element but maintain its space in the layout, visibility: hidden is appropriate. If you want to hide the element but keep its space and potentially use transitions or animations, opacity: 0 is a good choice.

11) Block level vs Inline elements default behaviour

In HTML, content placement within block-level and inline elements follows specific rules that determine how elements are displayed and how they interact with the surrounding content.

Block-Level Elements:

Block-level elements create distinct blocks or containers within the layout. They naturally start on a new line and take up the full available width of their parent container. Content within block-level elements is displayed in a "block" format.

Here's how content is typically placed within block-level elements:

On New Lines:

Block-level elements start on a new line, and subsequent block-level elements are displayed below the previous one. This means each block-level element creates a new "block" in the layout.

Full Width:

By default, block-level elements span the full width of their parent container, unless specified otherwise using CSS properties like width.

Vertical Stacking:

Block-level elements stack vertically, creating a clear separation between different sections or blocks of content. Common examples of block-level elements include paragraphs (<p>), headings (<h1>, <h2>, etc.), lists (<ul>, <ol>), and div containers (<div>).

<div>This is a block-level container.</div>

<div>Another block-level container.</div>

-Block-level elements naturally create distinct "blocks" within the layout. Content placed within block-level elements is organized into separate, vertically stacked sections.

-Block-level elements extend the full width of their parent container by default. This means that they create new lines for each block-level element, and each block-level element takes up the entire width available.

-Block-level elements are ideal for structuring and organizing the content into different sections or divisions of a web page.

Inline Elements:

Inline elements, flow within the content of their parent block-level elements. They don't create new lines or block-level structures themselves.

Here's how content is typically placed within inline elements:

Within Text Flow:

Inline elements flow within the text content of their parent block-level element. They do not create new lines, and their width adjusts to fit the content they contain.

No Line Breaks:

Inline elements do not introduce line breaks, and they wrap to the next line only if there is not enough horizontal space within their parent block-level element.

Horizontal Flow:

Content within inline elements flows from left to right (in left-to-right writing systems) or from right to left (in right-to-left writing systems).

<p>This is a <span>inline element</span> within a paragraph.</p>

In this example, the <span> element is an inline element and flows within the text of the paragraph. It doesn't introduce new lines or separate blocks of content.

-Inline elements are designed to flow within the content of a block-level element. They do not create distinct blocks or new lines by default.

-Inline elements are often used for applying formatting and styling to specific parts of text or for embedding elements within a paragraph of text.

-By default, inline elements do not create a new block formatting context and flow within the content of their parent block-level element. They only take up as much horizontal space as necessary to contain their content.

-Inline elements do not accept width or height properties. The margin-top and margin-bottom properties do not create vertical spacing between inline elements in the same way they do for block-level elements.

To control the placement of content within both block-level and inline elements, you can use various HTML and CSS properties, such as display, position, and others, to modify their default behaviors and achieve the desired layout and structure in your web pages.

Content vs width vs height vs margin vs padding vs border

The behavior of content, width, height, margin, padding, and border for inline and block-level elements in HTML and CSS differs due to their default display characteristics.

Content:

Block-Level Elements:

Block-level elements create separate blocks or containers within the layout. They naturally start on new lines and contain their content within distinct blocks. Block-level elements can hold other elements and text content.

Inline Elements:

Inline elements flow within the text content of their parent block-level element. They do not create new lines or separate blocks and are typically used for styling portions of text or inline content.

Width and Height:

Block-Level Elements:

Block-level elements, by default, extend across the full width of their parent container. You can explicitly set their width using CSS to control the element's horizontal size, and you can set their height to control their vertical size.

Inline Elements:

Inline elements take up only as much horizontal space as necessary to contain their content. Applying a width or height property to inline elements doesn't typically have the same effect as it does for block-level elements. Inline elements are usually self-sizing based on their content.

Margin:

Block-Level Elements:

Block-level elements use the margin property to create spacing around them. Margins apply both vertically and horizontally, creating space outside the element's border. You can set margins for all four sides individually.

Inline Elements:

Inline elements can have horizontal margins (left and right), but their vertical margins (top and bottom) may not create the same spacing effects as they do for block-level elements. Margins often affect the horizontal layout and the positioning of the inline elements in relation to surrounding content.

Padding:

Block-Level Elements:

Block-level elements use the padding property to create spacing within the element, affecting the space between the element's content and its border. Padding properties apply both vertically and horizontally and can be set for all four sides individually.

Inline Elements: Inline elements can have horizontal padding (left and right), but vertical padding (top and bottom) may not have the same impact. Padding for inline elements often affects the horizontal spacing and the positioning of the content within the element.

Border:

Block-Level Elements:

Block-level elements can have a border defined using the border property. The border surrounds the entire block and is often used for creating visual boundaries around content.

Inline Elements:

Inline elements can also have borders, but the effect might differ. Borders may not consistently create a clear block-like boundary around inline elements due to their inline flow. The border can be applied to individual sides (e.g., border-left, border-right) to control how it appears in relation to the inline content.

In summary, block-level elements are typically used to create distinct blocks of content with more control over their width, height, margins, padding, and borders. Inline elements, on the other hand, are used for styling inline content within a block-level element and may not have the same level of control over these properties due to their inline flow. However, you can modify the default behavior of elements using CSS to achieve the desired layout and styling.

Inline elements

Horizontal behaviour:

width: no effect

padding: work correctly

border: work corectly

margin: work correctly

Vertical behaviour:

height: no effect

margin: no effect

padding: go into block level element

border: go into block level element

12. Box model

The CSS box model is a container that contains multiple properties including borders, margins, padding, and the content itself. It is used to create the design and layout of web pages. It can be used as a toolkit for customizing the layout of different elements. The web browser renders every element as a rectangular box according to the CSS box model.

The CSS box-model and box-sizing are related concepts, but they serve different purposes when it comes to controlling how the dimensions of elements are calculated. Let me explain the difference between them:

The CSS box model, defines how an element's dimensions are calculated, taking into account the content, padding, border, and margin.

By default, when you set the width and height properties of an element, you are specifying the dimensions of the content area. The padding, border, and margin are added to these dimensions.

.box {

width: 200px;

height: 100px;

padding: 20px;

border: 2px solid #000;

margin: 10px;

}

The total space the .box element occupies on the page is calculated by summing the content width, padding, border, and margin, as described in the previous answer.

Box-Sizing Property:

The box-sizing property allows you to control how an element's dimensions are calculated, specifically how width and height are interpreted.

The box-sizing property has two possible values:

content-box (default):

This is the default behavior, where width and height represent the dimensions of the content area. Padding, border, and margin are added to these dimensions.

border-box:

With this value, width and height include the content area, padding, and border. The margin is still outside of these dimensions. If you set an element's box-sizing property to border-box, the specified width and height values will include the padding and border. This can be helpful for creating more predictable layouts, as you can work with the total space an element occupies.

Here's an example of how to use the box-sizing property:

.box {

width: 200px;

height: 100px;

padding: 20px;

border: 2px solid #000;

margin: 10px;

box-sizing: border-box; /\* Include padding and border in width and height \*/

}

In this case, the specified width and height values will include the padding and border, making it easier to control the overall dimensions of the element while ensuring that the content area remains a consistent size.

In summary, the CSS box-model defines the standard model for calculating element dimensions, and the box-sizing property allows you to change how width and height are interpreted, either as content dimensions or including padding and border.

13) Positions:

In CSS position property is used to position HTML element in HTML document.

While using position property, we take help of left, right, top, bottom and z-index properties to position HTML element.

1) static :

It is default position for all html elements we can not move element's position using top, left, right, bottom.

2) relative :

Element will be displayed as per normal flow of document.

It considers its own position as a reference point to move.

3) absolute :

Element will not be displayed as per normal flow of document.

It considers its positioned relative parent element as a reference point to move.

If no positioned parent then it takes top most parent(html/viewport) as reference point.

4) fixed :

Element will not be displayed as per normal flow of document.

It always considers html element(viewport) as a reference point to move,

So it stays (fixed) in the same position even when the page is scrolled.

5) sticky :

It is used to position and stick an HTML element to top edge of viewport.

Sticky positioning is a hybrid between relative and fixed positioning.

It will work initially as a relative (it will move certain extent) after that it will be fixed.

z-index:

The z-index property is typically used with positioned elements, such as relative,absolute,fixed. It does not have any effect on non-positioned (static) elements.

- Controls the stacking order of elements.

- Elements with higher z-index values appear in front of elements with lower values.

- If multiple elements have the same z-index, their stacking order is determined by their order in the HTML document. The element that appears later in the document will be on top.

- The z-index property only works on elements with a position value other than static. So, you need to set element's position property to relative, absolute,fixed for z-index to have effect.

- Negative values for z-index are allowed, and elements with negative z-index values will be placed behind elements with positive values.

15. Responsive Design

Media queries

-Media queries are essential for creating responsive web designs that adapt to various devices and screen sizes, providing a better user experience.

Media query is a technic of applying different styles to an element based on certain condition.

-They are commonly used to provide different styles for different devices by their features like type, width, height, orientaion.

Syntax:

1)@media

Media queries use the @media rule and have a specific syntax.

@media screen and (max-width: 600px) {

/\* CSS rules for screens with a width of 600px or less \*/

}

In this example, the media query targets screens with a width of 600 pixels or less.

2) Media Types:

You can specify different media types (screen, print, speech) to apply styles in different contexts. The most commonly used media type is screen.

-screen

-print

-speech

3) Media Features:

Media features like max-width, min-width, min-height, max-height, orientation,and many others are used to specify the conditions under which the CSS rules should be applied.

-min-width

-max-width

-min-height

-max-height

-orientation (landscape, portrait)

Here are some common examples of media queries:

Max-width:

To apply styles when the viewport width is at most a certain size.

@media screen and (max-width: 768px) {

/\* CSS rules for screens with a width of 768px or less \*/

}

Min-width:

To apply styles when the viewport width is at least a certain size.

@media screen and (min-width: 1024px) {

/\* CSS rules for screens with a width of 1024px or more \*/

}

Orientation:

To apply styles based on the device's orientation (portrait or landscape).

@media screen and (orientation: landscape) {

/\* CSS rules for landscape orientation \*/

}

4) Operators (Combining Conditions)

You can combine multiple conditions using and, or, not, only to create more complex media queries.

@media screen and (min-width: 768px) and (max-width: 1024px) {

/\* CSS rules for screens between 768px and 1024px wide \*/

}

Media queries are essential for creating responsive web designs that adapt to various devices and screen sizes, providing a better user experience.

16. CSS Animations

Animation is a process of changing elements style in a given duration.

we can apply multiple css styles to an element during given duration by using @keyframe.

syntax

@keyframe animation-name{

keyFrame-selector{

}

- keyFrame-selectors

"from" and "to" and we can give % also.

- You should configure the animation name and animation duration on an element.

properties on an element

1)animation-name: (compulsory)

2)animation-duration:(compulsory) (total animation time)

3)animation-delay: starts animation after some delay of time. (animation starts time)

4)animation-iteration-count: how many times animation need.

5)animation-direction: normal|reverse|alternate|alternate-reverse

6)animation-fill-mode: backwards|forwards|none|both (before & after animation styles)

7)animation-play-state: paused | running

8)animation-timing-function: linear|ease-in|ease-out|ease-in-out (speed of animation)

=> ease: Default value. The animation has a slow start, then fast, before it ends slowly.

=> linear: Play an animation with the same speed from beginning to end.

=> ease-in: The animation has a slow start.

=> ease-out: The animation has a slow end.

=> ease-in-out: The animation has both a slow start and a slow end.