# HTML & CSS Revision Guide

# Chapter 1: HTML

## Tables

A foundational element in HTML, the <table> element, along with its associated tags, is purpose-built for organizing data in a structured, two-dimensional grid of rows and columns.[1] This rigid structure is ideal for representing complex relationships between different types of data, such as a timetable, financial figures, or a list of planetary characteristics. For instance, a user can quickly and easily find a value, like the number of moons for a specific planet, by visually associating the relevant row and column headers.[1]

The correct application of tables is strictly limited to the display of tabular data. They should be used for content that is inherently a grid, like a list of products and their specifications, or statistical results. It is an outdated practice and a critical anti-pattern in modern web development to use tables for page layout. Historically, this approach was common before the widespread adoption of CSS, but it creates non-semantic markup that is difficult to maintain and, most importantly, is a significant barrier to accessibility. A screen reader, for example, will attempt to interpret the layout table as data, announcing rows and cells in a non-sensical order and making navigation and comprehension extremely difficult for visually impaired users. When implemented correctly, however, using semantic tags like <th> to denote headers, HTML tables are handled well by accessibility tools and enhance the user experience for everyone.[1]

For developers, understanding the nuances of table tags is essential. The <table> tag is the outer container for all table content, with <tr> defining each table row. The smallest container within a row is a table cell, created with the <td> element ("table data") or the <th> element ("table header").[1] To create more complex layouts within the table's data structure, the

colspan and rowspan attributes can be used. colspan allows a cell to span across multiple columns, while rowspan allows a cell to extend across multiple rows.[1]

| Tag | Purpose |
|---|---|
| <table> | Wraps all table content. |
| <tr> | Defines a row within the table. |
| <td> | Defines a data cell. |
| <th> | Defines a header cell for a row or column. |
| colspan | An attribute for <td> and <th> to make a cell span multiple columns. |
| rowspan | An attribute for <td> and <th> to make a cell span multiple rows. |

### Forms

An HTML form serves as a powerful mechanism for a website to interact with users, most commonly for collecting data.[2] A basic form consists of a

<form> container that wraps all of the other form elements, such as <input>, <label>, and <button>.[3] Forms are used for a wide range of tasks, from user registration and login to complex e-commerce payment checkouts and simple search bars.[3]

A fundamental principle of form development, and a key element of modern web standards, is the proper association of a <label> with its corresponding form control.[3] This link is crucial for accessibility and usability. An explicit association is created by setting the

<label>'s for attribute to match the form control's id. An implicit association is also possible by nesting the form control directly inside the <label> tag. This structural relationship ensures that assistive technologies, such as screen readers, can correctly announce the purpose of each input field to users with disabilities.[3] This practice is not merely for screen readers; for a user with limited motor control, clicking the larger area of a label to activate a small checkbox

provides a much-improved user experience.

The <form> element has key attributes that define its behavior, including action, which specifies the path to the page that processes the form data, and method, which determines the data transmission method, such as GET or POST.[3] Each input element uses the

type attribute to define the control's behavior, such as a text field, radio button, or checkbox. The name attribute gives the data a unique identifier, and the id attribute is used to link it to a <label>.[3] Client-side validation, achieved with attributes like

required, offers immediate feedback to the user and is useful for improving the user experience. However, a significant pitfall is relying solely on this type of validation for security. Client-side code can be easily bypassed by malicious users, making robust server-side validation a non-negotiable security requirement to ensure data integrity.[3]

| Common <input> Types | Purpose |
|---|---|
| text | A basic text field for any value. |
| password | A text field where the input characters are hidden. |
| email | A text field for an email address. |
| radio | A single choice from a group. |
| checkbox | A single on/off choice. |
| tel | A text field for a telephone number. |

# Chapter 2: CSS Fundamentals

### Background Properties

The background shorthand property and its individual components are used to control the visual "canvas" behind an element's content and border.[5] This powerful set of properties allows for the application of a solid color, a single image, multiple layered images, or even complex gradients.[5] Common applications include setting a theme color for a section of a page, applying a decorative pattern, or creating a dynamic effect, such as a parallax background image with

background-attachment: fixed.[5]

The background shorthand is a highly efficient way to set multiple background properties—such as background-color, background-image, background-repeat, background-position, and background-size—in a single line.[5] It's important to remember that any property not explicitly set in the shorthand will revert to its default value.[5] A key aspect of background styling is the layering order. Background images are stacked one on top of the other in the order they are declared, with the first image appearing closest to the user. The

background-color is always drawn at the very bottom, beneath all images.[5] This layering behavior presents a crucial opportunity for web resilience. When a

background-image fails to load due to a network error or a broken URL, the background-color serves as an essential fallback. Without it, the element's background would be transparent, potentially rendering the foreground text unreadable. Therefore, always specifying a background-color is a fundamental practice for building robust and reliable web pages.[5]

When working with images, the background-size property offers two distinct and powerful keywords: cover and contain.[6] The

cover value scales the image to the smallest possible size that completely covers the container, preserving its aspect ratio. This often results in the image being cropped if its proportions do not match the container's. In contrast, contain scales the image to the largest size possible to fit entirely within its container without cropping or stretching it. If the image's proportions differ from the container's, this will leave empty space, which will be filled by the background-color.[6]

| Property | Purpose | Default Value |
|---|---|---|
| background-attachment | Controls if background scrolls with page. | scroll |
| background-clip | Defines the background painting area. | border-box |

| | | |
|---|---|---|
| background-color | Sets the background color. | transparent |
| background-image | Sets one or more background images. | none |
| background-origin | Specifies the origin of the background position. | padding-box |
| background-position | Sets the initial position of a background image. | 0% 0% |
| background-repeat | Controls how a background image repeats. | repeat |
| background-size | Sets the size of a background image. | auto |

### Text & Font Properties

The properties that control text and fonts are central to all web design, as they define the visual appearance, legibility, and readability of content.[7] These properties are used to establish a clear visual hierarchy by differentiating between headings, body text, and captions.[8] They encompass everything from the font family (

font-family) and size (font-size) to the weight (font-weight), style (font-style), and spacing between lines (line-height) and letters (letter-spacing).[7]

When setting the vertical spacing between lines of text with the line-height property, a subtle but crucial detail in how CSS inheritance works becomes apparent. While line-height can be defined using percentages, em units, or pixels, the recommended best practice is to use a unitless number (e.g., 1.5).[9] This is because a unitless value is inherited by child elements as a simple multiplier. The browser then computes the actual line height by multiplying this number by the child element's own

font-size. This prevents a common pitfall where a child element with a different font size inherits the parent's pre-computed pixel value for line height, which can lead to disproportionate and unreadable spacing.[9] The unitless value ensures that the spacing remains proportional to the text size throughout the document, creating a more robust and scalable stylesheet.[9]

Accessibility is a key consideration when styling text. For primary paragraph content, a line-height of at least 1.5 is recommended to aid users with low vision or cognitive concerns like dyslexia.[9] In a similar vein, large sections of text set in an

italic font face can be difficult to read and should be avoided.[11] Beyond the basic keywords like

normal and bold, the font-weight property can accept a numerical value from 1 to 1000, allowing for a more fine-grained control over the text's boldness. This is particularly relevant with variable fonts, which can support a continuous range of weights within a single file, giving designers a greater degree of stylistic control.[12]

| Property | Purpose | Values |
|---|---|---|
| font-family | Specifies a prioritized list of font families. | "Helvetica", sans-serif |
| font-size | Sets the size of the font. | 16px, 1em, 1.2rem |
| font-weight | Sets the boldness of the font. | normal, bold, 400, 700 |
| font-style | Sets a normal, italic, or oblique style. | normal, italic, oblique |
| line-height | Sets the height of a line box. | 1.5, 1.2em, 150% |
| text-align | Aligns text horizontally. | left, right, center, justify |

# Chapter 3: CSS Effects

### Transform

CSS transform is a powerful property that lets you alter the shape and position of an element in a 2D or 3D coordinate space.[13] The unique aspect of transforms is that they modify an element's visual representation without disrupting the normal document flow. The element still occupies its original space in the layout, and surrounding elements are not affected by the transformation.[14] This is a key reason why transforms, when combined with transitions and animations, are the preferred method for creating performant visual effects, as they avoid the computational cost of a full page re-layout.

The transform property is typically set to one or more functions that define a specific manipulation.[13] The most common functions are:

translate() for moving an element, rotate() for rotating it around a fixed point, scale() for resizing it, and skew() for slanting it along the X and Y axes.[13] A critical aspect to understand is that the order of these functions matters. Functions are applied in the order they are declared, from left to right. For example,

transform: translateX(200px) rotate(135deg) will first move the element 200 pixels horizontally and then rotate it. In contrast, transform: rotate(135deg) translateX(200px) will first rotate the element and then translate it along its new, rotated axis, resulting in a completely different final position.[13]

An element with a transform value other than none creates a new stacking context.[13] This means it acts as a container for any

position: fixed or position: absolute elements it contains, affecting how they are positioned and layered within the document. It is also important to consider accessibility when using certain transforms. Scaling or zooming animations, in particular, can be a common trigger for certain types of migraines. Therefore, when implementing such effects, it is a best practice to provide a site-wide control that allows users to disable animations.[13]

| Function | Purpose | Example |
|---|---|---|
| translate() | Moves an element from its current position. | transform: translate(120px, 50%); |
| rotate() | Rotates an element around a fixed point. | transform: rotate(0.5turn); |

| | | |
|---|---|---|
| scale() | Scales an element up or down. | transform: scale(2, 0.5); |
| skew() | Skews an element along the X and Y axes. | transform: skew(30deg, 20deg); |
| matrix() | Combines multiple 2D transform functions. | transform: matrix(1, 2, 3, 4, 5, 6); |

### Transition

CSS transitions provide a way to create a smooth, implicit animation when a CSS property is changed. Rather than an abrupt shift in an element's state, a transition defines a duration and a timing function for the change to occur over a set period.[15] Transitions are ideal for simple, state-based animations, typically triggered by a user action like hovering over a link or focusing on a form field. They are perfect for creating subtle yet engaging UI feedback, such as a button's background color fading on hover or a menu item changing size.[15]

For managing all aspects of a transition, it is highly recommended to use the transition shorthand property. This shorthand combines transition-property, transition-duration, transition-timing-function, and transition-delay into a single declaration.[15] Using the shorthand prevents a common source of frustration where individual properties might get out of sync, making them difficult to debug. For instance,

transition: all 0.5s ease-out; will apply a smooth, half-second transition to all animatable properties on the element.[15]

A notable pitfall for beginners is attempting to animate the display property from none to a visible state. By default, this is a discrete, instantaneous change. To create a fade-in effect, for example, the opacity should be transitioned from 0 to 1. The research also suggests that a new CSS feature, @starting-style in combination with transition-behavior: allow-discrete, is necessary to properly animate a property like display on a new element.[15]

| Property | Purpose |
|---|---|
| transition-property | Specifies the names of the properties to be animated. |

| | |
|---|---|
| transition-duration | Defines how long the transition will last. |
| transition-timing-function | Specifies a function to control the speed curve of the transition. |
| transition-delay | Defines the delay before the transition begins. |

### Animation

CSS animations offer a more robust and powerful alternative to transitions, allowing for complex, multi-step, and repeatable effects.[16] They are composed of two primary components: the

@keyframes at-rule and the animation property. The @keyframes rule is where the animation's choreography is defined, detailing the appearance of the element at specific points in time, from a start (from or 0%) to an end (to or 100%) and any intermediate steps in between.[17] The

animation property is then used on an element to apply the @keyframes rule, controlling its duration, iteration count, and other timing details.[16]

Animations are particularly useful for effects that are not triggered by a simple state change, such as a continuous looping effect like a pulsing button or a complex sequence of movements and color changes.[17] The declarative nature of CSS animations is a key concept: the

@keyframes rule is like a script, and the animation property acts as a director, telling the element when and how to run that script. A common pitfall is forgetting to link the animation-name property to the @keyframes rule, which will prevent the animation from ever running.[16] When using the

animation shorthand, the order of values is important, especially for the two time values. The first time value is always the animation-duration, and the second is the animation-delay.[16]

The distinction between a transition and an animation is crucial. A transition is for simple, single-stage, event-triggered changes (like a hover effect), whereas an animation is for complex, multi-stage, or looping effects defined in @keyframes.[16] The performance benefits of using both

transform and opacity are significant, as they avoid causing a reflow or repaint of the entire page on every frame, leading to more fluid and performant visual effects.

# Chapter 4: CSS Layout

## Display

The display property is arguably the most fundamental property in CSS, as it dictates how an element is rendered and how it participates in the document's flow and layout.[18] It controls both the element's outer display type (

block or inline), which determines how it interacts with its siblings, and its inner display type (flow, flex, grid), which defines the layout method for its children.[18]

Each display value has a specific purpose and set of behaviors. block elements, such as headings and paragraphs, generate a block box that creates line breaks both before and after the element, taking up the full width of its container by default.[18] In contrast,

inline elements, like links and <span> tags, do not create line breaks and flow with the surrounding text, taking up only as much space as their content requires.[18] The

inline-block value is a legacy option that merges these two behaviors, allowing an element to flow with text while also accepting dimensions like width and height.[18]

For modern layouts, the most powerful values are flex and grid. The display: flex value is set on a parent element to enable the flexbox model, which is an efficient, one-dimensional layout system for aligning and distributing space among items in a single row or column.[19] The

display: grid value is set on a parent element to enable the grid model, a two-dimensional layout system that excels at dividing a page into major regions and aligning elements into both columns and rows.[20]

A key concept to master is the distinction between display: none and visibility: hidden. The former completely removes an element from the document flow, causing other elements to fill its space as if it never existed. The latter simply hides the element visually, but it continues to occupy its original space in the layout, which must be accounted for.[18] Understanding the distinction between

flex (one-dimensional) and grid (two-dimensional) is the foundation of modern CSS layout. A developer should choose flexbox for tasks like a simple navigation bar or evenly spaced gallery items, and grid for more complex, two-dimensional structures like a full page layout with headers, sidebars, and main content areas.

| Value | Behavior | Use Case |
|---|---|---|
| block | Takes up the full width, creates a new line. | Headings, paragraphs. |
| inline | Flows with text, takes up content width. | Links, <span> tags. |
| inline-block | Flows with text but accepts width/height. | Icons, button groups. |
| flex | A one-dimensional layout for items in a row or column. | Navigation bars, form inputs. |
| grid | A two-dimensional layout for columns and rows. | Full page layout, complex sections. |
| none | The element is not displayed, removed from flow. | Hiding elements. |

## Position

The position property defines how an element is positioned on the page, in conjunction with the top, right, bottom, and left properties.[21] It allows for fine-grained control over an element's placement relative to its normal position, a parent element, or the viewport.

Each of the five position values has a distinct behavior and use case:

- static: This is the default value. The element is positioned according to the normal flow of the document, and the top, right, bottom, and left properties have no effect.[21]
- relative: A relatively positioned element is offset from its normal position. This offset does not affect the position of other elements, so the space it occupies in the layout remains the same.[21] This value is most commonly used to establish a positioning context for absolutely positioned children.
- absolute: An element with position: absolute is removed from the normal document flow and positioned relative to its closest positioned ancestor (any ancestor with a position value other than static). If no such ancestor exists, it is positioned relative to the initial containing block (the viewport).[21] This is ideal for elements that need to be layered on top of the content, such as tooltips, dropdown menus, or modals.
- fixed: A fixed element is also removed from the normal document flow but is positioned relative to the viewport itself.[21] Its position is determined by
  top, right, bottom, and left, and it remains in that location even when the user scrolls the page. This is the perfect choice for a fixed header, a sticky sidebar, or a "back to top" button.[21]
- sticky: This modern value acts like relative until a specified scroll offset is reached, at which point it becomes fixed.[21] This is a common and highly effective pattern for list headers that "stick" to the top of the viewport as the user scrolls, or for side navigation that stays in view.

A key challenge with position: absolute is the concept of a "positioned ancestor." An element will be positioned relative to the first parent element it finds with a position value that is not static.[21] A common pitfall for new developers is failing to set a

position on the parent, leading to the absolute element positioning itself unexpectedly relative to the entire document. Both absolute and fixed elements are removed from the document flow, meaning they no longer take up space and can cause surrounding elements to reflow, which is an important consequence to manage in your layout.[21]

| Value | Behavior | Use Case |
|---|---|---|
| static | Normal flow. | Default positioning. |
| relative | Offset from normal position. | Container for absolute children. |
| absolute | Removed from flow, relative to a positioned ancestor. | Tooltips, dropdown menus. |

| | | |
|---|---|---|
| fixed | Removed from flow, relative to the viewport. | Fixed headers, sticky footers. |
| sticky | Initially relative, then becomes fixed on scroll. | Sticky list headers. |

# Chapter 5: Responsive Design

## Media Queries

CSS media queries are the cornerstone of responsive web design. They provide a mechanism to apply styles conditionally based on the characteristics of a user's device or browser environment.[22] This allows a website's layout and appearance to adapt gracefully to different screen sizes, resolutions, orientations, and other user preferences.[24] A media query uses the

@media rule, a media type (screen, print), and one or more expressions or conditions to determine if the enclosed CSS rules should be applied.[22]

The mobile-first approach is a widely accepted best practice for building responsive websites today.[24] This methodology involves defining a minimal set of base styles for mobile devices first, and then using

min-width media queries to progressively add more complex layouts and styling for larger screens, such as tablets and desktops.[24] This approach is not only more efficient, as it sends less CSS to resource-constrained mobile devices, but it also promotes a more thoughtful design process by forcing a developer to prioritize content and layout from the start.[24] The reverse, a desktop-first approach with

max-width queries, can often lead to more complex and brittle stylesheets as a developer must override a heavier set of styles for smaller viewports.

The conditions within a media query are known as "media features" and can be combined using logical operators.[24] The

and operator is used to combine multiple conditions, ensuring all must be true for the styles to apply. The comma , acts as a logical OR, allowing styles to be applied if any of the specified conditions are met. Common media features include min-width and max-width, which are essential for defining a website's breakpoints.[22] These breakpoints, which define when a layout change occurs, should be chosen strategically based on where the content itself breaks or becomes unreadable, rather than being tied to arbitrary standard device sizes.[24] Other valuable features include

orientation (for portrait or landscape mode) and prefers-color-scheme (for adapting to a user's dark or light mode preference).[24]

| Media Feature | Purpose |
|---|---|
| width | The width of the viewport. |
| min-width | The minimum width of the viewport. |
| max-width | The maximum width of the viewport. |
| orientation | The orientation of the device (portrait or landscape). |
| resolution | The pixel density of the screen. |

| Logical Operator | Purpose |
|---|---|
| and | Combines multiple conditions; all must be true. |
| , (Comma) | Acts as an OR; styles apply if any condition is true. |
| not | Negates a media query, excluding a condition. |
| only | Prevents older browsers from applying styles. |

# Chapter 6: Quick Cheat Sheet

**HTML**

- <table>: Tabular data.
- <tr>: Table row.
- <td>: Table data cell.
- <th>: Table header cell.
- colspan: Span multiple columns.
- rowspan: Span multiple rows.
- <form>: User input container.
- <input>: Form control element.
- <label>: Label for input.
- <button>: Form submission.
- <fieldset>: Group form controls.

**CSS Fundamentals**

- background: All background properties.
- background-image: Set background image.
- background-size: Resize background image.
- background-repeat: Tile background image.
- background-color: Set background color.
- font-family: Font type.
- font-size: Font size.
- font-weight: Font boldness.
- font-style: Normal, italic, oblique.
- line-height: Vertical line spacing.
- text-align: Horizontal text alignment.

**CSS Effects**

- transform: Change shape, position.
- translate(): Move element.
- rotate(): Rotate element.
- scale(): Resize element.
- skew(): Slant element.
- transition: Smooth property changes.
- transition-duration: Length of transition.
- @keyframes: Animation sequence.
- animation: Apply keyframes.

- animation-duration: Length of animation.

### CSS Layout

- display: Element layout type.
- block: Takes full width.
- inline: Flows with text.
- inline-block: Inline, with box model.
- flex: 1D layout.
- grid: 2D layout.
- position: Positioning method.
- static: Normal flow.
- relative: Relative to self.
- absolute: Relative to positioned parent.
- fixed: Relative to viewport.
- sticky: Sticky based on scroll.

### Responsive Design

- @media: Conditional styles.
- min-width: Min viewport width.
- max-width: Max viewport width.
- orientation: Portrait/landscape.

## Works cited

1. HTML table basics - Learn web development | MDN, accessed on September 23, 2025, https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Structuring_content/HTML_table_basics
2. Structuring content with HTML - Learn web development | MDN - Mozilla, accessed on September 23, 2025, https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Structuring_content
3. Forms and buttons in HTML - Learn web development | MDN, accessed on September 23, 2025, https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Structuring_content/HTML_forms
4. How to structure a web form - Learn web development - MDN - Mozilla, accessed on September 23, 2025, https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Forms/How_to_structure_a_web_form
5. background - CSS | MDN, accessed on September 23, 2025, https://developer.mozilla.org/en-US/docs/Web/CSS/background
6. background-size - CSS | MDN - Mozilla, accessed on September 23, 2025, https://developer.mozilla.org/en-US/docs/Web/CSS/background-size

7. CSS fonts - MDN - Mozilla, accessed on September 23, 2025, https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_fonts
8. CSS text - CSS | MDN, accessed on September 23, 2025, https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_text
9. line-height - CSS | MDN, accessed on September 23, 2025, https://developer.mozilla.org/en-US/docs/Web/CSS/line-height
10. How CSS line-height works and best practices - DEV Community, accessed on September 23, 2025, https://dev.to/lampewebdev/css-line-height-jjp
11. font-style - CSS | MDN - Mozilla, accessed on September 23, 2025, https://developer.mozilla.org/en-US/docs/Web/CSS/font-style
12. font-weight - CSS | MDN - Mozilla, accessed on September 23, 2025, https://developer.mozilla.org/en-US/docs/Web/CSS/@font-face/font-weight
13. transform - CSS | MDN, accessed on September 23, 2025, https://developer.mozilla.org/en-US/docs/Web/CSS/transform
14. Using CSS transforms - MDN - Mozilla, accessed on September 23, 2025, https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_transforms/Using_CSS_transforms
15. Using CSS transitions - CSS | MDN, accessed on September 23, 2025, https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_transitions/Using_CSS_transitions
16. animation - CSS | MDN, accessed on September 23, 2025, https://developer.mozilla.org/en-US/docs/Web/CSS/animation
17. Using CSS animations - CSS | MDN, accessed on September 23, 2025, https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_animations/Using_CSS_animations
18. display - CSS | MDN, accessed on September 23, 2025, https://developer.mozilla.org/en-US/docs/Web/CSS/display
19. Flexbox - Learn web development | MDN, accessed on September 23, 2025, https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/CSS_layout/Flexbox
20. CSS grid layout - CSS | MDN, accessed on September 23, 2025, https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_grid_layout
21. position - CSS | MDN, accessed on September 23, 2025, https://developer.mozilla.org/en-US/docs/Web/CSS/position
22. Media query fundamentals - Learn web development - MDN - Mozilla, accessed on September 23, 2025, https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/CSS_layout/Media_queries
23. CSS media queries - MDN, accessed on September 23, 2025, https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_media_queries
24. A complete guide to CSS Media Query | BrowserStack, accessed on September 23, 2025, https://www.browserstack.com/guide/css-media-query