Click Here

Share copy and redistribute the material in any medium or format for any purpose, even commercially. Adapt remix, transform, and build upon the material for any purpose, even commercially. The licensor cannot revoke these freedoms as long as you follow the license terms. Attribution You must give appropriate credit , provide a link to the license, and indicate if changes were made . You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use. ShareAlike If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original. No additional restrictions You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits. You do not have to comply with the license for elements of the material in the public domain or where your use is permitted by an applicable exception or limitation . No warranties are given. The license may not give you all of the permissions necessary for your intended use. For example, other rights such as publicity, privacy, or moral rights may limit how you use the material. In this example, we use the line-height property with a value that is equal to the height property to center the div element: I am vertically and horizontally centered. When the first CSS specification was published, all of CSS level 1 was contained in one document that defined CSS Level 1. CSS Level 2 was defined also by a single, multi-chapter document. However for CSS beyond Level 2, the CSS Working Group chose to adopt a modular approach, where each module defines a part of CSS, rather than to define a single manageable chunks and allows, meaningful, incremental improvement to CSS. Since different specifications are at different levels of stability, the CSS Working Group has chosen to publish this profile to define the current scope and state of Cascading Style Sheets as of 2024. Cascading Style Sheets (CSS) CSS is a language for writing style sheets, and is designed to describe the rendering of structured documents (such as HTML and XML) on a variety of media. CSS is used to describe the presentation of a source document, and usually does not change the underlying semantics expressed by its document language. Style sheet A set of rules that specify the presentation of a document. Style sheets are written by an Author, and interpreted by a User Agent, to present the document to the User. Source document The document to which one or more style sheets apply. A source documents structure and semantics are encoded using a document language (e.g., HTML, XHTML, or SVG). Author An author is a person who writes documents and associated style sheets. An authoring tool is a User Agent that generates style sheets. User A user is a person who interacts with a user agent to view, hear, or otherwise use the document. User Agent (UA) A user agent is any program that interprets a document and its associated style sheets on behalf of a user. A user agent may display a document, read it aloud, cause it to be printed, convert it to another format, etc. For the purposes of the CSS specifications, a User Agent is one that supports and interprets Cascading Style Sheets as defined in these specifications. This section is non-normative. In the W3C Process, a Recommendation-track document passes through three levels of stability, summarized below: Working Draft (WD) This is the design phase of a W3C spec. The WG iterates the spec in response to internal and external feedback. The first official Working Draft is designated the First Public Working Draft (FPWD). In the CSSWG, publishing FPWD indicates that the Working Group as a whole has agreed to work on the module, roughly as scoped out and proposed in the editors draft. The transition to the next stage is sometimes called Last Call Working Draft (LCWD) phase. The CSSWG transitions Working Drafts once we have resolved all known issues, and can make no further progress without feedback from building tests and implementations. This Last Call for Comments sets a deadline for reporting any outstanding issues, and requires the WG to specially track and address incoming feedback. The comment-tracking document is the Disposition of Comments (DoC). It is submitted along with an updated draft for the Directors approval, to demonstrate wide review and acceptance. Candidate Recommendation (CR) This is the testing phase of a W3C spec. Notably, this phase is about using tests and implementations to test the specification: it is not about testing the implementations. This process often reveals more problems with the spec, and so a Candidate Recommendation feedback, though usually less so than during the design phase (WD). Demonstration of two correct, independent implementations of each feature is required to exit CR, so in this phase the WG builds a test suite and generates implementation reports. The transition to the next stage is Proposed Recommendation (PR). During this phase the W3C Advisory Committee must approve the transition to REC. Recommendation (REC) This is the completed state of a W3C spec and represents a maintenance phase. At this point the WG only maintains an errata document and occasionally publishes an updated edition that incorporates the errata back into the spec. An Editors Draft is effectively a live copy of the editors working copy. It may or may not reflect Working Group consensus, and can at times be in a self-inconsistent state. (Because the publishing process at W3C is time-consuming and onerous, the Editors Draft is usually the best (most up-to-date) reference for a spec. Efforts are currently underway to reduce the friction of publishing, so that official drafts will be regularly up-to-date and Editors Drafts can return to their original function as scratch space.) A list of all CSS modules, stable and in-progress, and their statuses can be found at the CSS Current work page. This profile includes only specifications that we consider stable and for which we have enough implementation experience that we are sure of that stability. Note: This is not intended to be a CSS Desktop Browser Profile: inclusion in this profile is based on feature stability only and not on expected use or Web browser adoption. This profile defines CSS in its most complete form. As of 2024, Cascading Style Sheets (CSS) is defined by the following specifications. CSS Level 2, latest revision (including errata) [CSS2] This defines the core of CSS, parts of which are overridden by later specifications. We recommend in particular reading Chapter 2, which introduces some of the basic concepts of CSS and its design principles. CSS Syntax Level 3 [CSS-SYNTAX-3] Replaces CSS24.1, CSS24.2, CSS24.4, and CSS2G, redefining how CSS is parsed. CSS Style Attributes [CSS-STYLE-ATTR] Defines how CSS declarations can be embedded in markup attributes. Media Queries Level 3 [CSS-MEDIAQUERIES] Replaces CSS27.3 and expands on the syntax for media-specific styles. CSS Conditional Rules Level 3 [CSS-CONDITIONAL-3] Extends and supersedes CSS27.2, updating the definition of @media rules to allow nesting and introducing the @supports rule for feature-support queries. Selectors Level 3 [SELECTORS-3] Replaces CSS25 and CSS26.4.3, defining an extended range of selectors. CSS Namespaces [CSS3-NAMESPACE] Introduces an @namespace rule to allow namespace-prefixed selectors. CSS Cascading and Inheritance Level 4 [CSS-CASCADE-4] Extends and supersedes CSS21.4.3 and CSS26, as well as CSS6. Describes a way for authors to define a transformation that controls the rate of change of some value. Applied to animations, such transformations can be used to produce animations that mimic physical phenomena such as momentum or to cause the animation to move in discrete steps producing robot-like movement. CSS Counter Styles Level 3 [CSS-COUNTER-STYLES-3] Introduces the @counter-style rule, which allows authors to define their own custom counter styles for use with CSS list-marker and generated-content counters [CSS-LISTS-3]. It also predefines a set of common counter styles, including the ones present in CSS2 and CSS2.1. Note: Although we dont anticipate significant changes to the specifications that form this snapshot, their inclusion does not mean they are frozen. The Working Group will continue to address problems as they are found in these specs. Implementers should monitor www-style and/or the CSS Working Group Blog for any resulting changes, corrections, or clarifications. The following specifications are considered to be in a reliable state, meaning they have largely stable implementations and specifications, but are not yet at the Recommendation level due to minor issues or the need for additional implementation reports. Media Queries Level 4 [MEDIAQUERIES-4] Extends and supersedes [CSS-MEDIAQUERIES], expanding the syntax, deprecating most media types, and introducing new media features. CSS Scroll Snap Module Level 1 [CSS-SCROLL-SNAP-1] Defines properties for influence the visual styling of scrollbars, introducing controls for their color and width. CSS Grid Layout Module Level 1 [CSS-GRID-1] Introduces a two-dimensional grid-based layout system, optimized for user interface design. In the grid layout model, the children of a grid container can be positioned into arbitrary slots in a predefined flexible or fixed-size layout grid. CSS Grid Layout Module Level 2 [CSS-GRID-2] Extends and supersedes [CSS-GRID-1], introducing subgrids for managing nested markup in a shared grid framework. The following modules have completed design work, and are fairly stable, but have not received much testing and implementation experience yet. We hope to incorporate them into the official definition of CSS in a future snapshot. Media Queries Level 4 [MEDIAQUERIES-4] Extends and supersedes [CSS3-MEDIAQUERIES], expanding the syntax, deprecating most media types, and introducing new media features. CSS Display Module Level 3 [CSS-DISPLAY-3] Replaces CSS29.1.2, 9.2.1 (but not 9.2.1.1), 9.2.2 (but not 9.2.1.1), 9.2.3, and 9.2.4 (and lays the foundations for replacing 9.7), defining how the CSS formatting box tree is generated from the document element tree and defining the display property that controls it. CSS Writing Modes Level 4 [CSS-WRITING-MODES-4] Extends and supersedes [CSS-WRITING-MODES-3], adding more options for vertical writing. CSS Fragmentation Module Level 3 [CSS-BREAK-3] Describes the fragmentation model that partitions a flow into pages, columns, or regions and defines properties that control it. Extends and supersedes CSS213.3. CSS Box Alignment Module Level 3 [CSS-ALIGN-3] Introduces properties to control the alignment of boxes within their containers in the various CSS box layout models: block layout, table layout, flex layout, and grid layout. CSS Shapes Module Level 1 [CSS-SHAPES-1] Extends floats (CSS29.5) to effect non-rectangular wrapping shapes. CSS Text Module Level 3 [CSS-TEXT-3] Extends and supersedes CSS216 excepting 16.3, defining properties for text manipulation and specifying their processing model. It covers line breaking, justification and alignment, white space handling, and text transformation. CSS Text Decoration Module Level 3 [CSS-TEXT-DECOR-3] Extends and supersedes CSS216.3, providing more control over text decoration lines and adding the ability to specify text emphasis marks and text shadows. CSS Masking Module Level 1 [CSS-MASKING-1] Replaces CSS211.1.2 and introduces more powerful ways of clipping and masking content. CSS Scroll Snap Module Level 1 [CSS-SCROLL-SNAP-1] Contains features to control panning and scrolling behavior with snap positions. CSS Speech Module Level 1 [CSS-SPEECH-1] Replaces CSS2A, overhauling the (non-normative) speech rendering chapter. CSS View Transitions Module Level 1 [CSS-VIEW-TRANSITIONS-1] Defines the View Transition API, along with associated properties and pseudo-elements, which allows developers to create animated visual transitions representing changes in the document state. Although the following modules have been widely deployed with rough interoperability, their details are not fully worked out or sufficiently well-specified and they need more testing and bugfixing. We hope to incorporate them into the official definition of CSS in a future snapshot. CSS Transitions Level 1 [CSS-TRANSITIONS-1] and CSS Animations Level 1 [CSS-ANIMATIONS-1]. Introduces mechanisms for transitioning the computed values of CSS properties over time. CSS Will Change Level 1 [CSS-WILL-CHANGE-1] Introduces a performance hint property called will-change. Filter Effects Module Level 1 [FILTER-EFFECTS-1] Introduces filter effects as a way of processing an elements rendering before it is displayed in the document. CSS Font Loading Module Level 3 [CSS-FONT-LOADING-3] Introduces events and interfaces used for dynamically loading font resources. CSS Box Sizing Level 3 [CSS-SIZING-3] Overlays and extends CSS10., expanding the value set of the sizing properties, introducing more precise sizing terminology, and defining with more precision and detail various automatic sizing concepts only vaguely defined in CSS2. CSS Transforms Level 2 [CSS-TRANSFORMS-2] builds upon [CSS-TRANSFORMS-1] to add new transform functions and properties for three-dimensional transforms, and convenience functions for simple transforms. CSS Lists and Counters Module Level 3 [CSS-LISTS-3] Contains CSS features related to list counters: styling them, positioning them, and manipulating their value. CSS Logical Properties and Values Level 1 [CSS-LOGICAL-1] Introduces logical properties and values that provide the author with the ability to control layout through logical, rather than physical, direction and dimension mappings. Also defines logical properties and values for the border-*, margin-*, and padding-* shorthand properties. These properties are writing-mode relative equivalents of their corresponding physical properties. CSS Positioned Layout Module Level 3 [CSS-POSITION-3] Contains defines coordinate-based positioning and offsetting schemes of CSS: relative positioning, sticky positioning, absolute positioning, and fixed positioning. Resize Observer [RESIZE-OBSERVER-1] This specification describes an API for observing changes to elements principal box size. CSS Fonts Module Level 4 [CSS-FONTS-4] Extends and supersedes CSS Fonts 3 and provides more control over font choice and feature selection, including support for OpenType variations. CSS Color Adjustment Module Level 1 [CSS-COLOR-ADJUST-1] This module introduces a model and controls over automatic color adjustment by the user agent to handle user preferences and device output optimizations. CSS Conditional Rules Module Level 4 [CSS-CONDITIONAL-4] Extends CSS Conditional 3 to allow testing for supported selectors. CSS Cascading and Inheritance Level 5 [CSS-CASCADE-5] Extends CSS Cascade 4 to add cascade layers. Motion Path Module Level 1 [MOTION-1] This module allows authors to position any graphical object and animate it along an author specified path. CSS Scroll Anchoring Module Level 1 [CSS-SCROLL-ANCHORING-1] This module aims to minimize content shifts by locking the scroll position of a scroll container to a particular anchor element. CSS Object Model (CSSOM) [CSSOM-1] This module defines APIs for parsing, serializing, and manipulating CSS, Media Queries, and Selectors. CSS Color Module Level 5 [CSS-COLOR-5] Extends CSS Color 4 to add color spaces and color modification functions. Selectors Level 4 [SELECTORS-4] Extends Selectors Level 3 by introducing new pseudo-classes, pseudo-elements, and combinators, enhancing the ability to select elements based on more complex criteria and states. Cascading Style Sheets does not have versions in the traditional sense; instead it has levels. Each level of CSS builds on the previous, refining definitions and adding features. The feature set of each higher level is a superset of any lower level, and the behavior allowed for a given feature in a higher level is a subset of that allowed in the lower levels. A user agent conforming to a higher level of CSS is thus also conformant to all lower levels. CSS Level 1 The CSS Working Group considers the CSS1 specification to be obsolete. CSS Level 1 is defined as all the features defined in the CSS1 specification (properties, values, at-rules, etc), but using the syntax and definitions in the CSS2.1 specification. CSS Style Attributes defines its inclusion in element-specific style attributes. CSS Level 2 Although the CSS2 specification is technically a W3C Recommendation, it passed into the Recommendation stage before the W3C had defined the Candidate Recommendation stage. Over time implementation experience and further review has brought to light many problems in the CSS2 specification, and therefore an already unwieldy errata list, the CSS Working Group chose to define CSS Level 2 Revision 1 (CSS2.1). In case of any conflict between the two specs CSS2.1 contains the definitive definition. Once CSS2.1 became Candidate Recommendation effectively though not officially the same level of stability as CSS2obsoleted the CSS2 Recommendation. Features in CSS2 that were dropped from CSS2.1 should be considered to be at the Candidate Recommendation stage, but note that many of these have been or will be pulled into a CSS Level 3 working draft, in which case that specification will, once it reaches CR, obsolete the definitions in CSS2. The CSS2.1 specification defines CSS Level 2 and the CSS Style Attributes specification defines its inclusion in element-specific style attributes. CSS Level 3 CSS Level 3 builds on the CSS2 module by module, using the CSS2.1 specification as its core. Each module adds functionality and/or replaces part of the CSS2.1 specification. The CSS Working Group intends that the new CSS modules will not contradict the CSS2.1 specification: only that they will add functionality and refine definitions. As each module is completed, it will be plugged in to the existing system of CSS2.1 plus previously-completed modules. From this level on modules are leveled independently: for example Selectors Level 4 may well be completed before CSS Line Module Level 3. Modules with no CSS Level 2 equivalent start at Level 1; modules that update features that existed in CSS Level 2 start at Level 3. CSS Level 4 and beyond There is no CSS Level 4. Independent modules can reach Level 4 or beyond, but CSS the language no longer has levels. (CSS Level 3 as a term is used only to differentiate it from the previous monolithic versions.) Not all implementations will implement all functionality defined in CSS. In the past, the Working Group published a few Profiles, which were meant to define the minimal subset of CSS that various classes of user agents were expected to support. This effort has been discontinued, as the Working Group was not finding it effective or useful, and the profiles previously defined are now unmaintained. Note: Partial implementations of CSS, even if that subset is an official profile, must follow the forward-compatible parsing rules for partial implementations. The following sections define several conformance requirements for responsibly, in a way that promotes interoperability in the present and future. So that authors can exploit the forward-compatible parsing rules to assign fallback values, CSS renderers must treat as invalid (and ignore as appropriate) any at-rules, properties, property values, keywords, and other syntactic constructs for which they have no usable level of support. In particular, user agents must not selectively ignore unsupported property values and honor supported values in a single multi-value property declaration: if any value is considered invalid (as unsupported values must be), CSS requires that the entire declaration be ignored. To avoid clashes with future CSS features, the CSS2.1 specification reserves a prefixed syntax [CSS2] for proprietary and experimental extensions to CSS. A CSS feature is a proprietary extension if it is meant for use in a closed environment accessible only to a single vendors user agent(s). A UA should support such proprietary extensions only through a vendor-prefixed syntax and not expose them to open (multi-UA) environments such as the World Wide Web. Why? The prefixing requirement allows shipping specialized features in closed environments without conflicting with future additions to standard CSS. The restriction on exposure to open systems is to prevent accidentally causing the public CSS environment to depend on an unstandardized proprietary extensions. For example, Firefoxs XUL-based UI, Apples iTunes UI, and Microsofts Universal Windows Platform app use extensions to CSS implemented by their respective UAs. So long as these UAs do not allow Web content to access these features, they do not provide an opportunity for such content to become dependent on their proprietary extensions. Even if a feature is intended to eventually be used in the Web, if it hasnt yet been standardized it should still not be exposed to the Web. If a feature is unstable (i.e. the spec has not yet stabilized, but at least three UAs implement the feature)or a UA has broken the other rules and shipped for broad usean unstable or otherwise non-standard feature in a production release), and the implementations have rough interoperability, and the CSS Working Group has recorded consensusthat this feature should exist and be released, implementers may ship that feature unprefixed in broad-release builds. Rough interoperability is satisfied by a subjective judgment that even though there may be differences, the implementations are sufficiently similar to be used in production websites for a substantial number of use cases. Note that the CSSWG must still be consulted to ensure coordination across vendors and to ensure coherency review by the CSS experts from each vendor. Note also that rough interoperability still usually means painful lack of interop in edge (or not-so-edge) cases, particularly because details have not been ironed out through the standards/bugfixing process. Why? If a feature is sufficiently popular that three or more browsers have implemented it before its finished standardization, this clause allows releasing the pressure to ship. Also, if a feature has already escaped into the wild and sites have started depending on it, pretending its still experimental doesnt help anyone. Allowing others to ship unprefixed recognizes that the feature is now de facto standardized and encourages authors to write cross-platform code. When exposing such a standards-track unstable feature to the Web in a production release, implementations should support both vendor-prefixed and unprefixed syntaxes for the feature. Once the feature has stabilized and the implementation is updated to match interoperable behavior, support for the vendor-prefixed syntax should be removed. Why? This is recommended so that authors can use the unprefixed syntax to target all implementations, but when necessary, can target specific implementations to work around incompatibilities among implementations as they get ironed out through the standards/bugfixing process. The lack of a phase where only the prefixed syntax is supported greatly reduces the risk of websites being written with only the vendor-prefixed syntax. This in turn allows UAs vendors to retire their prefixed syntax once the feature is stable, with a lower risk of breaking existing content. It also reduces the need occasionally felt by some vendors to support a feature with the prefix of another vendor, due to content depending on that syntax. Anyone promoting unstable features to authors should document them using their standard unprefixed syntax, and avoid encouraging the use of the vendor-prefixed syntax for any purpose other than working around implementation differences. In order to preserve the open nature of CSS as a technology, vendors should make it possible for other implementors to freely implement any features that they do ship. To this end, they should provide spec-editing and testing resources to complete standardization of such features, and avoid other obstacles (e.g., platform dependency, licensing restrictions) to their competitors shipping the feature. Once a specification reaches the Candidate Recommendation stage, implementers should release an unprefixed implementation of any CR-level feature they can demonstrate to be correctly implemented according to spec, and should avoid exposing a prefixed variant of that feature. To establish and maintain the interoperability of CSS across implementations, the CSS Working Group requests that non-experimental CSS renderers submit an implementation report (and, if necessary, the testcases used in that report) to the W3C before releasing an unprefixed implementation of any CSS features. Testcases submitted to W3C are subject to review and correction by the CSS Working Group. Further information on submitting testcases and implementation reports can be found from on the CSS Working Groups website at . Questions should be directed to the public-css-testsuite@w3.org mailing list. The following features have been explicitly and proactively cleared by the CSS Working Group for broad release prior to the spec reaching Candidate Recommendation. 3.2.1 Experimentation and Unstable Features. The following features have been explicitly and retroactively cleared by the CSS Working Group for broad release prior to the spec reaching Candidate Recommendation. Everything in CSS Animations Level 1 and CSS Transitions Level 1. These sections are non-normative. = == 1st 2d matrix 2nd 3rd 4th absolute length unit absolutely positioned abstract dimensions :active activeborder activecaption active duration active (pseudo-class) actual value in css-cascade-4 in css21 additive tuple adjoining margins advance measure :after after:after-change style alighdoe alignment baseline alignment content alignment content self-alignment subject 'all' media group:alphabetic baseline ambiguous image url an+b ancestor anchor anchor unit animation origin animation-composition annotation automatic position automatic name atomic inline box atomic inline-level box at-rule attr() attribute 'audio' media group auditory icon augmented grid aural box model authoring tool author origin author presentational hint origin author style sheet automatic column position automatic column numbering automatic placement automatic position automatic row position auto-placement cursor available fill forces available grid space avoid break values axis-lock axis value azure backdrop background color background image layer background positioning area backslash escapes baseline baseline alignment preference baseline content-alignment baseline self-alignment baseline box baseline-sharing group baseline table base size bearing angle :before before before-change style before flag beige bfc bidi formatting characters bidi-isolate bidi-isolated bidi isolation bidi paragraph bidirectionality bidirectional (bidi) bi-orientational bi-orientational transform bisque 'bitmap' media group blanchedalmond (l-block (l-block block in css-display-3 in css21 {}-block block at-rule block axis block-axis block box in css-display-3 in css21 block container block container in css-display-3 in css21 block container block-level element block-level content block level element block-start block-start blue blueviolet blur radius boolean context border box border color border image border image region border:of a box border radius border style border width bottom box alignment properties box::border box::content box::content height box::content width box-corner box fragment box::margin box::overflow box::padding box tree box break burlywood buttonface buttonhighlight buttonshadow buttontext cadetblue cancel canonical unit canvas canvas surface captiontext captures snap positions cascade in css-cascade-4 in css21 cascade-dependent keyword cascaded independently cascaded value cascade origin central baseline character character encoding character map "@charset" chartreuse check if three code points would start a unicode-range check if two code points are a valid escape child child combinator clipping path clipping region in css-masking-1 in css21 closest-side clustered scripts collapse collapsed collapsed flex item collapsed grid track collapsed gutter collapsed track collapse through collapsing margin color color stop color transition hint column row color line column-span width combinator combinator units complete computed transition component value composite value computed computed computed track list computed value in css-cascade-4 in css21 concrete object size conditional group rule conditionally conformance consecutive constraint containing block content container container query conditional imported conditionally has conformance consecutive rectangle consume a block consume a component's values consume a declaration consume a list of declarations consume a list of rules consume an escaped code point consume an ident-like token consume a number consume a numeric token consume a qualified rule consume a simple block consume a string token consume a style block's contents consume a stylesheet's contents consume a unicode-range token consume the next input token containing block in css-display-3 in css21 containing block content consume the remnants of a bad declaration consume the remnants of an invalid url content content box content edge content language content::of a box content-box content distribution content-distribution properties content edge content language content::of a box content edge content::min content distribution offset content track sizing continuous media media query media feature medium suggestion in css-flexbox-1 in css-grid-1 content writing system continuous media 'continuous' media group convert a string to a number coordinated self-alignment count counter corner counter style symbol corner constraint crimson cross axis cross-axis baseline set cross size cross start cross-size cross-start cross-size cross size bracketed range notation css-connected css feature definition css font face current cursive cursive script custom property cyan darkblue darkcyan darkgoldenrod darkgray darkgreen darkgrey darkkhaki darkmagenta darkolivegreen darkorange darkorchid darkred darksalmon darkseagreen darkslateblue darkslategray darkslategrey darkturquoise darkviolet declaration in css-syntax-3, for CSS in css21 declaration declared declared value decode bytes decorating box deeppink deepskyblue default face default namespace default object size default style sheet definite column position definite column span definite row position definite row span descendant descendant-selectors descriptor descriptor declaration destination dimma the fallback encoding device pixel dice digit dimension dimgray dimgrey directional embedding direction render discard a mark discard a token discard whitespace display type distributed alignment document order document tree document width space characters dodgerblue dominant baseline easing function effective character map element in css-display-3, for CSS in css21 element::following element::preceding element tree emoji enterprise participating code points empty in css-syntax-3, for token stream in css21 en (unit) encapsulation contexts end ending point ending shape ending token endmost token end node endmost token end node end value environment encoding of code point escaping establish an independent formatting context establish an orthogonal flow established an independent formatting context establish an independent formatting context establish a new formatting context establishing an independent formatting context exactly maching expanded name empty (null) fallback alignment in the negative range fartheststyle false in fetch an @import fictional tag sequence filter code points filter primitive filter primitive attributes filter primitive filter primitive filter subregion filter primitive region find the matching font faces first a font baseline first baseline alignment first-baseline alignment set first-baseline content-alignment first baseline first-baseline first baseline set first baseline self-alignment set first-line first-line first-line first main-axis baseline set first symbol value fixed sizing function flex case sizing flex direction flex factor sum flex direction flexible length in css-flexbox-1 in css-grid-1 flexible sizing function flexible tracks flex item flex layout flex-line flex line flex shrink factor float area float rules floralwhite flow layout flow of an element flow-relative flow-relative::direction focus focus (pseudo-class) following element font block period font download timer font failure period font feature value declaration font-feature-value-type font-fragmentation containment fragmentation root fragmented box free space fuchsia full-size full-size kana full-width fully inflexible function function fallback function::center gainsboro generate a counter representation generate content ghostwhite go gold goldenrod gradient-average-color gradient box gradient center gradient function gradient line grapheme cluster gray graytext green greenyellow grey grid grid area grid cell grid column grid column line grid column span grid container grid formatting context grid item grid item placement algorithm grid layout grid layout 'grid' media group grid-modified document order grid order grid placement grid-placement property grid position grid row grid row line grid row line grid span grid track grid item placement grid item limit guaranteed-invalid value gutter half-width hang hanging glyph hebrt hex digit highlight highlighttext honeydew horizontal axis horizontal flow horizontal script horizontal sizing css-flexbox-1 in css-grid-1 horizontal start point start point identity transform hsl hue hyphenation honeydew horizontal axis horizontal flow horizontal script horizontal sizing css-flexbox-1 in css-grid-1 horizontal start point start point ignored identity transform ignored ignore most declarations implicit anchor element implicitly-assigned line name implicit-named area implicit grid implicitly-assigned line name important conditions inactive inactiveborder inactivecaption inactivecaptiontext indefinite indefinite size independent formatting context index indianred indigo infinitely growable in flow in-flow in css-display-3 in css21 infobackground infotext inherit in css-cascade-4 in css-cascade-4, for CSS in css21 inheritance in css-cascade-4 in css21 inherited value inherited value inherited value initial free space initial representation for the counter value initial value in css-cascade-4 in css21 inline axis inline-axis inline-axis inline base direction inline block inline-block inline box inline box in css-display-3 in css21 inline dimension inline end inline-end inline formatting context inline-level inline-level box inline-level box in css-display-3 in css21 inline-level content inline level element inline size inline start inhibition inlinity inner box inner display type installed font fallback integer intended direction intended direction end point intended direction intended direction end point intended direction in css-display-3 in css21 interaction media group 'interactive' media group interpret in css-namespaces-3 in css-style-attr interpreter in css-namespaces-3 in css-style-attr dimensions intrinsic sizing function invalid invalid at computed-value time invalid url invalid url isolated sequence isolated sequence item baseline set item cross size item main size ivory japanese justification opportunity keyword khaki korean :lang (pseudo-class) last-baseline alignment last-baseline content-alignment last baselines last-baseline self-alignment last baseline set last cross-axis baseline set last main-axis baseline set lavender lavenderblush lawngreen laying out in-place layout containment layout containment box layout-internal::left left leftover space legacy name alias legacy shorthand legacy value alias lemonchiffon letter in css-speech-1 in css-text-3 lightblue lightcoral lightcyan lightgoldenrodyellow lightgray lightgreen lightgrey lightpink lightsalmon lightseagreen lightskyblue lightslategray lightslategrey lightsteelblue lightyellow lime limegreen limited max-content contribution limited min-content contribution linear easing function linear timing function line box line break in css-break-3 in css-text-3 line breaking line breaking process line-left item name line item orientation line-over line-relative line-relative direction line-relative::under ::link (link (pseudo-class)) list list arguments list item list markers main axis main-axis baseline set main size main start main-start main-start main-start cross-size lavender lawngreen laying out in-place layout containment layout containment box layout-internal::left left leftover space legacy name alias legacy shorthand legacy value alias lemonchiffon letter in css-speech-1 in css-text-3 lightblue lightcoral lightcyan lightgoldenrodyellow light system local url flag logical height logical width longhand longhand property lowercase letter magenta main axis main-axis main-axis main size main-axis baseline set main box main dimension main end main-end main flex main size main-size main-start cross-size main start main-start main-start mark mask layer mask painting area mask-position area mask-size match matching transition duration matching transition timing function max cross size max cross size property maximum allowed code point max inner height max inner width max outer height max inner width max main size max outer size property max track sizing function may media media condition media-dependent import media feature media group media query list media query modifier media type mediumaquamarine mediumblue mediumorchid mediumpurple mediumseagreen mediumslateblue mediumspringgreen mediumturquoise mediumvioletred menu menutext message entity midnightblue mincss mint mince in css-text-3 min cross size in css-sizes size property mintcream min track sizing function moccasin monochrome multicol container multicol element multi-column container multi-column spanning element multi-line flex container multiple declarations multiply must must not named cell area named grid area named grid row natural aspect ratio natural dimension natural height natural natural size natural width navajowhite navy nearest neighbor newline next input code point next input token next-token next-token next token non-ascii code point non-ascii ident code point 'none'::as display value non-overwrite counter-style names non-printable code point non-replaced non-replaced element normal normalize into a token stream null cell token number numeric data types objects object size negotiation occupied oldlace olive olivedrab opacity operating coordinate space optimal viewing region optional orange orangered orchid order-modified document order origin origin in css-display-3 in css21 orthogonal flow orthogonal orthogonal box other display type outer box shadow outer display type outer edge outline out of flow in css-display-3 in css21 out-of-flow output of the cascade output progress value overflow overflow alignment overflow columns padding box padding::of a box @page page area page box break page-context paged media paged 'paged' media group paletor palevioletred papayawhip parent box parse parse a blocks contents parse a comma-separated list according to a css grammar parse a comma-separated list of component values parse a component value parse a declaration parse a list of component values parse a list of declarations parse a list of rules a parse a rule a style block's contents parse a stylesheet's contents parse error parse something according to a list participations in baseline alignment pass through filter peachpuff pending on the environment pending-substitution value percentage peru physical direction bottom physical dimensions physical direction physical left physical right physical top physical unit pink pixel pixel unit plum positional alignment positioned element/box positioning scheme post-multiplied post-multiply powderblue preceding element pre-multiplied pre-multiply preserved white space primary filter primitive tree principal block-level box principal box principal writing mode process propagation propagate proper table child proper table row parent property in css-cascade-4, for CSS in css21 property declarations pseudo-classes::focus pseudo-classes::active pseudo-classes::focus pseudo-classes::visited pseudo-class-:: pseudo-element pseudo-elements pseudo-elements::after pseudo-elements::before pseudo-elements::before pseudo-elements::first-letter pseudo-elements::first-line purple quad width :qualified rule range rendered content render with a fallback font face render with an invisible fallback font face replaced element in css-display-3 in css21 representation required reset explicitly reset-only sub-property re-snap resolved type restore a mark reversing-adjusted start value reversing shortening factor :right right root root element row row group box row group line run-in in css-display-3 in css21 run-in box run-in sequence running translation saddlebrown salmon sandybrown sans-serif scaled flex shrink factor scope screen reader scroll bar scroll snap scroll snap area scroll snap container scroll support scroll snap positions seashell segment break separate set main axis second point context sheet shorthand shorthand property in css-cascade-4 in css21 should should be simple selector in css-syntax-3, for parsing in css-speech-1 sienna silver single lateral right media:skyblue slategray slategrey small-caps snow soft wrap break soft wrap opportunity source source document spaces space-separated matching space to fill span count specified size specified size suggestion in css-flexbox-1 in css-grid-1 specified value in css-cascade-4 in css21 'speech' media group 'speech' media group steelblue step easing function step position steps stop or comma stretched strictness value stroke box structural pseudo-classes strut stuck on the environment style attribute style change event style rule style sheet in css-namespaces-3 in css-speech-1 in css-style-3 subject of the selector sub-property subsequent subtract subjects of the selector subpixel subsequent subtract suggest on target main size text suggestion in css-flexbox-1 in css-grid-1 transformable element subsequent transformable element transform subtree support table face support table caption box support table cell support table column support table column group support table row group support table row tab size thistle threedshadow timing function tokenization tokenizer tokens token stream tomato top tracking track list track section track sizing algorithm track sizing function track sizing function transfer function element transferable element functions transformable element attributes transferred size suggestion transferable transformable element transform transition transitionable transition origin transparent trash token triangle turquoise type selector typeset typeset sideways typesetting typesetting upright typographic character typographic character unit type tzpical letter unit typographic letter unit type typographic letter unit typographic character unit unforced break universal selector unknown unoccupied upper-alpha-legal uppercase letter upright typesetting url use a negative sign used value in css-cascade-4 in css21 user agent origin user-agent style sheet user coordinate system user origin user-origin user style sheet uses a used value user agent origin value definition syntax var() variable vertical axis vertical block flow vertical dimension vertical offset vertical-only vertical script typographic mode vertical writing mode viewport viewport-percentage lengths violet :visited visited (pseudo-class) visual angle unit visual formatting model 'visual' media group volume wheat white whitesmoke white space whitespace white space characters window windowframe windowtext word separator word-wrap script writing-mode writing mode writing system x-axis x-height y-axis yellow yellowgreen absolute in css-scroll-snap-1, for voice-pitch in css-speech-1, for voice-range add additive alias all allow-end all-petite-caps all-small-caps alpha alphabetic alternate alternate-reverse always in css-scroll-snap-1, for scroll-snap-stop in css-speech-1 for speak annotation() anywhere in css-text-3, for line-break in css-text-3, for overflow-wrap arabic-indic armenian in css-counter-styles-3, for @counter-style in css21 auto in css21 auto-fit auto-flow available baseline behind below in css-speech-1, for @counter-style in css21 auto in css21 auto-fit auto-flow available baseline behind below in css-speech-1 for border-image-width in css-backgrounds-3, for border-image-width breakable break-all break-spaces break-word in css-text-3, for overflow-wrap in css-text-3, for word-wrap bullets center center in css-align-3, for justify-self in css-scroll-snap-1, for scroll-snap-align in css-text-3, for text-align in css-transforms-1, for transform-origin ch character-variant(#) child cjk unit circle cjk-decimal cjk-earthly-branch cjk-heavenly-stem cjk-ideographic clip clone close-quote closest-corner closest-side cm coarse collapse color color-burn color-dodge column column-reverse common-ligatures condensed contain in css-backgrounds-3, for background-size in css-images-3, for object-fit contain in css-flexbox-1, for flex-basis content-box content-box content-box content-box context-menu contextual copy cover in css-backgrounds-3, for background-size in css-images-3, for object-fit crop crosshair cursive cyclic dark in css-speech-1, for voice-rate in mediaqueries-4, for @media/update darken dashed decimal decimal-leading-zero in css-counter-styles-3, for @counter-style in css21 disclosure-closed disclosure-open discretionary-ligatures distribute dot dotted double double-circle dpcm dpi dppx e each-line ease ease-in ease-in-out ease-out ellipse em embed embossed emoji em unit end endexpanded ethiopic-numeric evenodd ew-resize ex exclude exclusion expanded expanded extends expanded extra-condensed extra-expanded ex unit fallback fantasy farthest-corner farthest-side fast in css-speech-1, for voice-rate in mediaqueries-4, for @media/update female fill fill-box filled first first-baseline alignment first baseline set first-baseline content-alignment first baseline set first baseline self-alignment set first-line first child first cross-axis baseline set first formatted line :first-letter first-letter :first-line first-line first main-axis baseline set first symbol value fixed sizing flex base size flex basis flex direction flex factor sum flex direction flexible length in css-flexbox-1 in css-grid-1 flexible sizing function flexible tracks flex item flex layout flex-line flex line flex shrink factor float area float rules fit-content in css-sizing-3, for width in css-grid-1, for grid-template-columns, grid-template-rows in css-images-3, for image-rendering fixed in css-backgrounds-3, for background-attachment in css-position-3, for position flex flex-end flex-start font-fragmentation footnote for force-end forwards in css-scroll-snap-1, for scroll-padding in css-scroll-snap-1, for scroll-padding-inline in css-scroll-snap-1, for scroll-padding-block in css-scroll-snap-1, for scroll-padding-block-end in css-scroll-snap-1, for scroll-padding-block-start full-size-kana full-width in css-text-3, for text-transform in css-fonts-3, for font-variant-east-asian female fit-content full-width fit content-box underline-position in css-3, for caret-color in css-3, for cursor in css-will-change-1, for will-change in css-filter-effects-3, for filter in css-text-decoration-3, for text-underline-position auto-fill auto-fit (auto-flow & dense?) ? / avoid avoid-column avoid-page avoid-region backwards balance balance-all break-spaces break-word in css-text-3, for word-break in css-text-3, for overflow-wrap break-word breakable capitalize caption center in css-align-3, for justify-content, align-content in css-backgrounds-3, for background-position in css-flexbox-1, for align-items, align-self in css-flexbox-1, for justify-content in css-scroll-snap-1, for scroll-snap-align in css-speech-1, for voice-balance in css-text-3, for text-align in css-transforms-3, for transform-origin in character-variant(#) child ch unit circle cjk-decimal cjk-earthly-branch cjk-heavenly-stem cjk-ideographic clip clone close-quote closest-corner color color-burn color-dodge column column-reverse common-ligatures condensed contain context-menu contextual copy cover crop crosshair cursive cyclic dashed decimal decimal-leading-zero disc disclosure-closed disclosure-open discretionary-ligatures distribute dot dotted double double-circle dpcm dpi dppx each-line ease ease-in ease-in-out ease-out ellipse em embed emoji end ethiopic-numeric evenodd ew-resize exclude exclusion expanded extra-condensed extra-expanded fallback fantasy fast female fill fill-box filled first first-letter first-line fixed flex flex-end flex-start footnote for force-end forwards full-size-kana full-width generic(fangsong) generic(khmer-mul) generic(nastaliq) georgian in css-counter-styles-3, for @counter-style in css-display-3, for display, in css-grid-1, for display, / auto-flow & dense?) ? / groove gujarati gurmukhi handheld hanging hard-light hebrew help hidden high in css-speech-1, for voice-pitch high in css-speech-1, for voice-range high-quality khaki horizontal horizontal-tb hover hsl hsla hue iroha historical-forms historical-ligatures horizontal-tb hover hue ic in inherit initial inline inline-block inline-flex inline-grid inline-table inset inset in css-display-3, for display, in css-grid-1, for display, / auto-flow & dense? in css-display-3, for display, in css-grid-1, for display, in css-grid-1, for display / auto-flow & dense isolate isolate-override italic japanese-formal japanese-informal jis04 jis78 jis83 jis90 jump-both jump-end jump-none jump-start justify justify justify-all kannada katakana katakana-iroha keep-all khmer khz korean-hanja-formal korean-hanja-informal landscape lao last baseline landscape last lang last-baseline alignment set last-baseline content-alignment set last-baseline self-alignment set lavender left leftwards legacy {2} light in css-fonts-4, for base-palette in css-fonts-4, for font-palette lighter linear linearrgb (? ? ? ?)+ {)? line-through loading main-size line left in css-speech-1, for voice-range css21 lower-roman in css-counter-styles-3, in css-21 lr luminance luminosity malayalam male mandatory manual margin-box match match-parent match-source math max-content medium in css-backgrounds-3, for border-width, border-top-width, border-bottom-width, border-right-width, border-left-width, border-right-width, border-bottom-width, border-right-width, border-left-width mathematical in css-backgrounds-3, for border-style in css-backgrounds-3, for border-style in css-backgrounds-3, for border-top-style, border-right-style, border-bottom-style, border-left-style, border-block-start-style, border-block-end-style, border-inline-start-style, border-inline-end-style after in css-speech-1, for rest-after in css-speech-1, for voice-pitch in css-speech-1, for rest-before, rest-after in css-speech-1, for voice-range in css-speech-1, for voice-volume menu message-box min-content minmax() mixed mm moderate move ms multiply myanmar ne-resize nearest neither nesw-resize never no-clip no-close-quote no-common-ligatures no-drop no-shadow in css-contain-1, for contain in css-display-3, for display, in css-flexbox-1, for flex in css-fonts-4, for font-kerning in css-fonts-3, for font-optical-sizing in css-fonts-4, for font-optical-sizing in css-fonts-4, for font-synthesis-small-caps in css-fonts-4, for font-synthesis-style in css-fonts-4, for font-synthesis-weight in css-fonts-4, for font-variant in css-fonts-4, for font-variant-alternates in css-grid-1, for grid-template in css-grid-1, for grid-template-areas in css-grid-1, for grid-template-columns in css-images-3, for image-orientation in css-images-3, for object-fit in css-multicol-1, for column-span in css-scroll-snap-1, for scroll-snap-type in css-snap-align in css-speech-1, for pause-before, pause-after in css-speech-1, for voice-stress in css-text-3, for hanging-punctuation in css-text-3, for text-align in css-text-3, for text-align-last in css-text-decor-3, for text-decoration-line in css-text-decor-3, for text-emphasis-style in css-transitions-1, for transition-property in css-ui-3, for cursor in css-writing-modes-4, for text-combine-upright in mediaqueries-4, for @media/hover in mediaqueries-4, for @media/overflow-block in mediaqueries-4, for @media/overflow-inline in mediaqueries-4, for @media/pointer in mediaqueries-4, for @media/update 'none'::as border style nonzero no-open-quote no-punctuation no-repeat normal in compositing-1, for in css-align-3, for align-self in css-align-3, for justify-content, align-content in css-align-3, for justify-self in css-align-3, for row-gap, column-gap, gap in css-animations-1, for animation-direction in

font-feature-settings in css-fonts-4, font-kerning in css-fonts-4, font-language override in css-fonts-4, font-palette in css-fonts-4, font-size in css-fonts-4, font-style in css-fonts-4, font-variant in css-fonts-4, font-variant-caps in css-fonts-4, font-variant-east-asian in css-fonts-4, font-variant-emoji in css-fonts-4, font-feature-ligatures in css-fonts-4, font-variant-numeric in css-fonts-4, font-variant-position in css-fonts-4, font-weight in css-fonts-4, font-width in css-fonts-4, css-scroll-snap-1, for scroll-snap-type in css-speech-1, for speak-as in css-speech-1, for voice-balance in css-speech-1, for voice-duration in css-speech-1, for line-break in css-text-3, for overflow-wrap in css-text-3, for white-space in css-text-3, for word-break in css-text-3, for word-spacing in css-writing-modes-4, for unicode-bidi not set allowed nowrap in css-flexbox-1, for flex-wrap in css-text-3, for white-space in css-text-3 ...

[Note: This page consists almost entirely of densely concatenated, machine-extracted text from a Wikipedia-style article on Cascading Style Sheets (CSS), including specification names, prose on CSS syntax, selectors, the cascade, history, and an extensive bibliography/reference section. The text is run together without clear paragraph or structural boundaries and is not cleanly readable as discrete content.]

...Archived from the original on 2014-10-20. Retrieved 2014-10-16. ^ "W3C CSS validation service". World Wide Web Consortium. ^ "Selectors". Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification. W3C. Archived from the original on 2006-04-23. ^ "Selectors Level 3". W3C. Archived from the original on 2014-06-03. Retrieved 2014-05-30. ^ "CSS Syntax Module Level 3". W3C. Archived from the original on 2014-03-30. Retrieved 2014-04-05. ^ "6.1 Length units". Archived from the original on 2024-03-27. Retrieved 2019. ^ "6.1 Length units". ... "HTML 5. A vocabulary and associated APIs for HTML and XHTML". World Wide Web Consortium. ... ^ a b c d e f Lie, Hkon Wium; Bos, Bert (1999). Cascading Style Sheets, designing for the Web. Addison Wesley. ISBN 0-201-59625-3. Retrieved 23 June 2010. ... Scott Furman; Donna Converse; Troy Chevalier (22 August 1996). "JavaScript-Based Style Sheets". W3C. Archived from the original on 27 May 2010. Retrieved 23 June 2010. ...

may make their text look pretty or my favourite forbidden word in web design, clean. But in reality, this is another example of people sacrificing user experiencing (UX) for user interface design (UI). Its time to put a stop to this! Does text alignment matter for accessibility? Lets jump in. The 4 types of alignment in graphic, UI, and web design are.In terms of specifically web design, these types of alignment are used in different ways.Left alignment is by the most popular text alignment, and also the default one. Left alignment is commonly associated with lengthy paragraph text, which enables it to be as easy as possible to read.Center alignment is used more sparingly and for visual appeal like these common 3 or 4 column layouts on homepages. I also often see it at the bottom of pages as call to actions, giving less lengthy content better symmetry.Right alignment is rarely used, if ever. But if it is used, its to align text up to another element for more visual flare.And lastly justified alignment is used more commonly with minimal or luxurious design styles to give it some visual taste and elegance.Left is the most popular and default text alignment. Its the best for readability and user experience because of the way our eyes read.Right alignment is probably the most uncommon alignment used, and when it is used its typically used in small quantities.A great example of right alignment being used effectively is in the website navigation. With right alignment, you can include a CTA button In the top right and have the main links align with it nicely making the navigation very scannable.Right-aligned navigationRight alignment should be used sparingly at best, as right alignment isnt good for user experience and readability.There are some rare cases that it can actually make sense if used in low uses, like in this modern business letterhead example.As a rule of thumb, right alignment should be used if you are aligning text with more than 5 words at a time. This keeps the user experience in check while still adding in some rebellious formatting.In the example to the right, the right-aligned text on has a maximum of 3 words per line which limits the negative user experience to be insignificant at worst making it a good use case.I should mention that right-aligned text is different than right-aligned elements. Right-aligned elements work well in most cases, and can actually make use of otherwise unused space. In the letterhead picture, its okay that the content area is right-aligned, its the right text-align in that content area where my critiques lay.Center alignment can look great in small doses, but it can lead to problems when people overuse it.The reason why center text alignment is horrible for user experience is that with each new line the user reads, there is a brief moment where the user has to find where the next line begins decreasing the users reading speed.Intentionally handicapping your users ease of reading? Hell yes! This is what I think goes through peoples heads when they decide to center a large paragraph.What youll find is that in most cases centered text makes it worse for user experience. However there are some limits and exceptions of center alignment.Why oh why are we doing this. Oh yes, because its pretty. Centered text has become such a common occurrence, especially in website design.In my opinion centered paragraphs are only acceptable up to a point, 3 lines of text to be specific. Anymore, it becomes too displeasing to read each line after.Here are some examples:This is an acceptable length of text in a paragraph to center align. No more than three lines of text.This is a much longer paragraph with a length that is displeasing to read because of the number of times you have to find the beginning of the next line. Centered paragraph text should be limited and used sparingly so it doesnt make the user annoyed to read your text content. Notice all the content on this blog is left aligned? Thats called user experience, and dont you forget it!You can just see from the second example the paragraph is flat-out annoying to read. This only gets worse on mobile devices. This is why left alignment should be used 95% of the time.Primary page titles should be okay centered as they tend to not have as many words and therefore lines of text. Most page titles arent long enough that text alignment becomes an issue with usability. However, with secondary titles (h2s) and anything under should always be left-aligned to match its paragraph text.An example of google title alignment is by the most popular text alignment, and also the default one. Left alignment is commonly associated with lengthy paragraph text, and also the default one. At the title section at the top, I use left alignment because some of the titles of some of my articles may get long enough that centering the text would become annoying to read and so I use left alignment.Justified text looks like it cant already together. Justified text makes your paragraphs look like blocks and defined have sides.It works by changing the spacing between each word depending on how many words it best uses can fit onto one line. The spacing between the words in the same for each line, but changes for the next. The different spacings between each word because of justified-aligned textThe idea behind using justified text is that it looks more visually appealing than if left-justified alignment was used. This creates for a bad user experience, making the text hard to read.This might happen without you knowing, if text is scaled/resized in such a way to force the text like that.When justified alignment is used in books and written material.What can very easily happen with justified alignment is in narrow columns or lines with a lot of long words, there can be massive awkward spacings between each word. Sometimes its so bad where it splits up words into syllables. This creates for a bad user experience, making the text hard to read.This might happen without you knowing, if text is scaled/resized in such a way to force the text like that.When justified alignment is used in books and written material. The spacing between the words in the same for each line. Its common use in books and written material.What can very easily happen with justified alignment is in narrow columns or lines with a lot of long words, there can be massive awkward spacings between each word because of justified-aligned textThe idea behind using justified text is that it looks more visually appealing than if left-justified alignment was used. Hence its common use in books and written material.What can very easily happen with justified alignment is in narrow columns or lines with a lot of long words, there can be massive awkward spacings between each word because of justified-aligned textThe idea behind using justified text is that it looks more visually appealing than if left-justified alignment was used. Sometimes its so bad where it splits up words into syllables. This creates for a bad user experience, making the text hard to read.This might happen without you knowing, if text is scaled/resized in such a way to force the text like that.When justified alignment is used in books or magazines, there are people that go through each page and manually adjust the spacing to get rid of any of those awkward spacings and even add hyphens if necessary. This is completely unreasonable to do on a website, there are just too many different screen sizes, screen resolutions, and zoom levels to have justified alignment for at least someone.Books and magazines are by far the most common source of justified text out there. They love using justified text because it looks more visually appealing, and looks more professional.It can even save on printing costs due to the pages saved in making sure each line of text is used to its full width.Each line of a book or magazine usually have enough words so that the problems of justified alignment dont happen. Each line of text in a book typically has around 60 characters per line, which is about 10 words per line. This enables books to have the visual appeal of justified text, and the user experience of left-justified text.In the case where justified text doesnt work, left alignment would be used to make word spacing less distracting. Its common practice for book designers and book formatters to choose what alignment is best for each book or written material.Overall justified text has its uses. It can definitely add to the visual appeal of paragraphs on a page, but if not used with caution it can cause the text to look fragmented detracting from a positive user experience.Left justified text is where its at. Its familiar, its fast, and its reliable. Left text alignment should be used in 95% of cases to help your readers read at an optimal, undiminished reading speed.You can use center alignment in small doses like main page headings without detracting from the users experience. Paragraph text can get away with being center-aligned if it has 3 or fewer lines of text, any more then left-aligned should be used as the text becomes too notably annoying to read.Only use justified text for mediums where its commonplace like material books or e-books.Rarely you can get away with justified text on a website, but left alignment will always be better for usability. If you are going to use justified alignment, make sure there are enough words per line so that the problems of justified alignment dont happen.Each line of text in a book typically has around 60 characters per line, which is about 10 words per line.The only commonplace acceptable use of right alignment is navigations on websites. To horizontally center a block element (like div), use margin: auto; Nearly all browsers nowadays support CSS and many other applications do, too. To write CSS, you dont need more than a text editor, but there are many tools available that make it even easier. Of course, all software has bugs, even after several updates. And some programs are further ahead implementing the latest CSS modules than others. Various sites describe bugs and work-arounds. More For beginners, Starting with HTML + CSS teaches how to create a style sheet. For a quick introduction to CSS, try chapter 2 of Lie & Bos or Dave Raggett's intro to CSS. Or see examples of styling XML and CSS tips & tricks. Another page also has some books, mailing lists and similar fora, and links to other directories. The history of CSS is described in chapter 20 of the book Cascading Style Sheets, designing for the Web, by Hkon Wium Lie and Bert Bos (2nd ed., 1999, Addison Wesley, ISBN 0-201-59625-3) More CSS inspired quayjn to write the song CSS is OK. Site navigation I see it time and time again with UI and web design people love centering, justifying, and right aligning text. People think this may make their text look pretty or my favourite forbidden word in web design, clean. But in reality, this is another example of people sacrificing user experiencing (UX) for user interface design (UI). Its time to put a stop to this! Does text alignment matter for accessibility? Lets jump in. The 4 types of alignment in graphic, UI, and web design are.In terms of specifically web design, these types of alignment are used in different ways. Left alignment is commonly associated with lengthy paragraph text, which enables it to be as easy as possible to read. Center alignment is used more sparingly and for visual appeal like these common 3 or 4 column layouts on homepages. I also often see it at the bottom of pages as call to actions, giving less lengthy content better symmetry. Right alignment is rarely used, if ever. But if it is used, its to align text up to another element for more visual flare.And lastly justified alignment is used more commonly with minimal or luxurious design styles to give it some visual taste and elegance.Left is the most popular and default text alignment. Its the best for readability and user experience because of the way our eyes read.Right alignment is probably the most uncommon alignment used, and when it is used its typically used in small quantities.A great example of right alignment being used effectively is in the website navigation. With right alignment, you can include a CTA button In the top right and have the main links align with it nicely making the navigation very scannable.Right-aligned navigationRight alignment should be used sparingly at best, as right alignment isnt good for user experience and readability.There are some rare cases that it can actually make sense if used in low uses, like in this modern business letterhead example.As a rule of thumb, right alignment should be used if you are aligning text with more than 5 words at a time. This keeps the user experience in check while still adding in some rebellious formatting.In the example to the right, the right-aligned text on has a maximum of 3 words per line which limits the negative user experience to be insignificant at worst making it a good use case.I should mention that right-aligned text is different than right-aligned elements. Right-aligned elements work well in most cases, and can actually make use of otherwise unused space. In the letterhead picture, its okay that the content area is right-aligned, its the right text-align in that content area where my critiques lay.Center alignment can look great in small doses, but it can lead to problems when people overuse it.The reason why center text alignment is horrible for user experience is that with each new line the user reads, there is a brief moment where the user has to find where the next line begins decreasing the users reading speed.Intentionally handicapping your users ease of reading? Hell yes! This is what I think goes through peoples heads when they decide to center a large paragraph.What youll find is that in most cases centered text makes it worse for user experience. However there are some limits and exceptions of center alignment.Why oh why are we doing this. Oh yes, because its pretty. Centered text has become such a common occurrence, especially in website design.In my opinion centered paragraphs are only acceptable up to a point, 3 lines of text to be specific. Anymore, it becomes too displeasing to read each line after.Here are some examples:This is an acceptable length of text in a paragraph to center align. No more than three lines of text.This is a much longer paragraph with a length that is displeasing to read because of the number of times you have to find the beginning of the next line. Centered paragraph text should be limited and used sparingly so it doesnt make the user annoyed to read your text content. Notice all the content on this blog is left aligned? Thats called user experience, and dont you forget it!You can just see from the second example the paragraph is flat-out annoying to read. This only gets worse on mobile devices. This is why left alignment should be used 95% of the time.Primary page titles should be okay centered as they tend to not have as many words and therefore lines of text. Most page titles arent long enough that centering the text would become annoying to read and so I use left alignment.Justified text looks like it center alignment and left alignment had a child together. Justified text makes your paragraphs look like blocks and defined have sides.It works by changing the spacing between each word depending on how many words it best uses can fit onto one line. The spacing between the words in the same for each line, but changes for the next. The different spacings between each word because of justified-aligned textThe idea behind using justified text is that it looks more visually appealing than if left-justified alignment was used. Hence its common use in books and written material.What can very easily happen with justified alignment is in narrow columns or lines with a lot of long words, there can be massive awkward spacings between each word. Sometimes its so bad where it splits up words into syllables. This creates for a bad user experience, making the text hard to read.This might happen without you knowing, if text is scaled/resized in such a way to force the text like that.When justified alignment is used in books or magazines, there are people that go through each page and manually adjust the spacing to get rid of any of those awkward spacings and even add hyphens if necessary. This is completely unreasonable to do on a website, there are just too many different screen sizes, screen resolutions, and zoom levels to have justified text on a website without having awkward alignment for at least someone.Books and magazines are by far the most common source of justified text out there. They love using justified text because it looks more visually appealing, and looks more professional.It can even save on printing costs due to the pages saved in making sure each line of text is used to its full width.Each line of a book or magazine usually have enough words so that the problems of justified alignment dont happen. Each line of text in a book typically has around 60 characters per line, which is about 10 words per line.Overall justified text has its uses. It can definitely add to the visual appeal of paragraphs on a page, but if not used with caution it can cause the text to look fragmented detracting from a positive user experience.Left justified text is where its at. Its familiar, its fast, and its reliable. Left text alignment should be used in 95% of cases to help your readers read at an optimal, undiminished reading speed.You can use center alignment in small doses like main page headings without detracting from the users experience. Paragraph text can get away with being center-aligned if it has 3 or fewer lines of text, any more then left-aligned should be used as the text becomes too notably annoying to read.Only use justified text for mediums where its commonplace like material books or e-books.Rarely you can get away with justified text on a website, but left alignment will always be better for usability. If you are going to use justified alignment, make sure there are enough words per line so that the problems of justified alignment dont happen.The only commonplace acceptable use of right alignment is navigations on websites. Get ready to learn how to approach the age old question faced by many a CSS practitioner: "How do I center a div?" The holy grail: vertical and horizontal centering, aka centering along both the x-axis and the y-axis. The most modern and easiest way is with the following two lines of CSS: CSS for "XY Grid Solution".grid { display: grid; place-content: center; } We are centered!GotchasCollapse of child grid using auto-fit or auto-fillGiven a child grid that uses the following styles-grid-template-columns: repeat(auto-fit or auto-fill), minmax(10ch, 1fr));The child grid will collapse in on itself, in this case down to the min part of minmax, due to the justify-content set in the place-content shorthand. CSS for "auto-fit fixed".grid-autofit { display: grid; grid-template-columns: repeat(auto-fit, minmax(10ch, 1fr)); } The fix is two-fold: switch the grid centering technique to use place-items instead of place-content, and then to specifically define that the child grid should be width: 100% or whatever youd prefer as a width value to create space for the grid columns. CSS for "auto-fit collapse".grid { display: grid; place-items: center; } .grid-autofit { display: grid; grid-template-columns: repeat(auto-fit, minmax(10ch, 1fr)); width: 80%;} Alternatively, you can use the very slightly more verbose Flexbox version: CSS for "XY Flex Solution".flex { display: flex; align-items: center; justify-content: center; } GotchasFlexbox has a slightly different behavior when a second item is added since flex items default to placement along the x-axis: CSS for "XY Flex Gotcha".flex { display: flex; align-items: center; justify-content: center; } We are centered! ...Sort of!One way to resolve this is by adding flex-direction: column. CSS for "XY Flex Gotcha fixed".flex { display: flex; flex-direction: column; align-items: center; justify-content: center; } Alternatively, wrap the children in a single element, especially if you don't want them to be affected by the outer flexbox alignment.If a child element uses grid with auto-fit or auto-fill it will encounter the same issue as when the parent container is grid as described previously. margin: auto is unique for flexbox and grid, and in the case you have only one child item, you can do the following for either flex or grid. CSS for "XY Alternative Flexbox Solution".flex { display: flex; } .only-child { margin: auto;}The auto behavior for flex children, unlike children of block elements, can also be applied vertically which allows this solution to work. If you are unable to switch to grid or flexbox layout, here's a modern solution to this classic problem.Ensure the child elements are wrapped in a containing element for the following to work: CSS for "XY Centering for Block Elements".parent { position: relative;} .child-wrapper { position: absolute; top: 50%; left: 50%; transform: translate(-50%, -50%);} This combo works because when a percentage value is supplied to a translate definition, it based the percentage on the computed width (translateX) or height (translateY). In this example, we use shorthand to apply both x and y values to translate().Absolute positioning takes an element out of normal document flow after which we can apply precise values (as needed) to control its positioning in the document, or in this case, relative to the parent with the required position: relative on the parent element.After absolutely positioning the child from the top 50% and left 50%, which is 50% of the parent's height and width, respectively, we then use translate(-50%, -50%) to pull the child back up 50% of its own height and back left 50% of its own width. This results in a centered appearance that scales with the content.GotchasBecause we've used absolute positioning, there's a chance the content will grow to overflow the parent, even if like in the demo the parent has a min-width which typically grows with the content except for absolute children.The fix for this is: use grid or flexbox :) Or prepare to create #allthemediaqueries. Solutions for centering vertically, aka on the y-axis. We only need one property to vertically align in grid: CSS for "Y Grid Solution".grid { display: grid; align-content: center; } I am centered vertically! Me too! Flexbox items can be centered vertically, but this requires a little more property to set than grid-auto-flow: column. CSS for "Y Grid Solution for columns".grid { display: grid; align-content: center; grid-auto-flow: column; } I am centered vertically! Me too! Flexbox items can be centered vertically. CSS for "Y Flexbox Solution".flex { display: flex; align-items: center; flex-direction: column; } I am centered vertically! Me too! Use of align-content is scalable for multiple child elements.It also works if we switch the default grid axis to x with grid-auto-flow: column. CSS for "Y Grid Solution for columns".grid { display: grid; align-content: center; grid-auto-flow: column; } I am centered vertically! Me too! Flexbox items can be centered. CSS for "Y Flexbox Gotcha for flex-direction: column".flex { display: flex; flex-direction: column; } I am centered vertically! Me too!GotchasIf you switch the default axis by adding flex-direction: column, the solution fails. CSS for "Y Flexbox Gotcha for flex-direction: column".flex { display: flex; flex-direction: column; align-items: center; } Now I am centered... horizontally? Hmm, me too...A huge culprit of issues when dealing with flexbox is missing that flipping the default axis flips the associated properties.For column, or y-axis flex layout, instead of align-items we now need to use: CSS for "Y Flexbox Gotcha fix for flex-direction: column".flex { display: flex; flex-direction: column; justify-content: center; } I am centered vertically! Me too! If possible, switch the layout model and use flex or grid.Otherwise, switch the XY solution, we'll use absolute positioning and transform, but only apply to translateY to move the child 50% of its height. CSS for "Y Flexbox Gotcha fix for the flex-direction: column".parent { position: relative;} .child-wrapper { position: absolute; top: 50%; transform: translateY(-50%);} I am centered vertically! Me too!Again, this holds up if we switch the default axis with: CSS for "X Grid Solution for columns".grid { display: grid; justify-content: center; grid-auto-flow: column;} I am centered horizontally! Me too! To center along the x-axis, which is the default flexbox axis for child item alignment, use: CSS for "X Flexbox Solution for flex-direction: column".flex { display: flex; flex-direction: column; align-items: center;} I am centered horizontally! Me too! This is the classic solution of using auto margins, although it must be placed on each element you wish to center individually.For the demo, I've also set a width since by default block elements take up the full-width of their container, which visually opposes the centering. CSS for "X Centering for Block Elements".block div { margin-left: auto; margin-right: auto; width: 60%;} I am centered horizontally! Me too! The use case here is for components like dropdown menus or tooltips when there is a requirement for centering of items of dynamic/unknown width relative to the associated trigger.We'll use a method similar to Y Centering for Block Elements, but using left and translateX properties: CSS for "X Centering for Dynamically Positioned Elements".parent { position: relative;} .child-wrapper { position: absolute; left: 50%; transform: translateX(-50%);} I am centered horizontally! Me too! On: 8 Feb 2013 By: Joshua Johnson Category: Layouts Length: 6 min read Today were going to go back to basics with a design 101 discussion on alignment. Centered alignments are an easy place to go wrong and if you dont know how to wield them properly, the result is a very poorly structured page.Join us as we take a look at why centered alignments tend to be weak, where you should avoid them and how you should be using them. Envato gives you unlimited access to 22+ million pro design resources, themes, templates, photos, graphics and more. Everything you'll ever need in your design resource toolkit. See More Centered AlignmentsOne of the first things that youll learn in any basic design layout class is that centered alignments are weak. Now, thats a blanket statement that might immediately cause you to protest, but well explore this more in a minute.Centered alignments by no means something only used by beginners and non-designers, but they do in fact tend to be the go-to option for these groups. The reason for this is complicated. For some reason, we instinctively feel that things should be centered. We know that good balance and centered alignments are nothing if not balanced, so they make sense. Secretaries, CEOs, teachers and all other manner of other professionals that dip their toe into design almost always use bots take a look at an example. Lets say youre making little flyers to hand out for an upcoming party. The result that a non-designer will come up with will almost always look something like this:This isnt a hideous flyer, in fact Ive seen far worse in the real world. However, for a number of reasons, its definitely not what you would call a strong design. If you handed me the flyer above and told me to improve it as much as possible in only three minutes while keeping the overall identify and how to fix these problems is a major step in your early design education. The Problem with Centered Alignmentsits difficult to explain the faults of pure centered alignments with almost always how so lets take a look at an example. Lets say youre making little flyers to hand out for an upcoming party. The result that a non-designer will come up with will almost always look something like this:The answer here is pretty simple. In fact, theyre usually subtler than these circumstances. If you implemented a left alignment instead of a centered alignment. The answer here is pretty simple. In fact, theyre usually subtler than these circumstances. If you implemented a left alignment instead of a centered alignment personality, this is what I would give back to you:This design simply feels better, and not just because I refuse to ever use the Party LET typeface, especially when typing the word party. I also split up the content a little better added some emphasis points and most importantly, changed to a left alignment. Notice how the left alignment feels very anchored. The text on the example below. The edge all the way from top to bottom. Despite the fact that the first flyer feels balanced, it doesnt feel anywhere near as structured. When To Stay Away from Centered AlignmentsAll right, so you get the point: left alignments are good, centered alignments are bad, so what? The reality is a little more complicated than that. Theres nothing inherently bad about a centered alignment, you just have to know how to wield them properly, the result is a very poorly structured page.Join us as we take a look at why centered alignments tend to be weak, where you should avoid them and how you should be using them. As we saw in the last example, centered alignments are pretty poor for large blocks of text. The lack of a hard edge makes reading difficult and erratic. The same goes for relying on a centered layout for your page as a whole, most of the time, its simply not the best idea. The website wireframe below is quite problematic from a layout standpoint.Now, dont read that as advising against centering your content on the page. Theres a difference between building a website that uses a centered container and a website that uses a center alignment. You can easily have left or right aligned content that is still centered in the browser window using margin: 0 auto. Thats another conversation entirely.AestheticsAlso, dont confuse this as a matter of aesthetics. As the page below shows, you can create beautiful sites that rely heavily on centered alignments. Its very important in any design to analyze your goals. If a significant degree of readability is one of them, and it often should be, then aesthetic appeal is often a consideration all on its own, but you have to know how to use the tool to be successful with it.There are a million different ways to successfully implement a centered layout so dont imagine that the following examples are exhaustive. However, they can serve as a way for you to start to get a feel for a solid layout versus a weak one.Very Little ContentOne of the first places you can start to think about implementing a completely centered layout is when you have very little content. Check out the example below. The main problem with centered alignments is that they dont hold up well with a lot of content, then theyll likely be just fine if youre going to implement it with any amount of success.The first thing you need to learn is when not to implement a centered alignment. The answer here is pretty simple. separate or even directly at odds with this goal. The trick is to find the balance between the two.The end result is almost always going to be completely subjective. The designer above decided that this chunk of content was small enough to be center aligned. I may think its pushing the limit a little but that doesnt mean Im right and he/she is wrong, it just means we draw the line in different places. How to Use Centered AlignmentsNow, with all of that said, there is no reason that you should abandon using centered alignments altogether. To do so would be to remove a key element from your layout toolbox. Once again, you just have to know how to use the tool to be successful with it.There are a million different ways to successfully implement a centered layout so dont imagine that the following examples are exhaustive. However, there are very specific portions that revert to a centered alignment. The headline is the most natural fit and could even be given a little more emphasis through a slightly bigger alignment to be implemented. Once again, each of these blocks contains a fairly small amount of content so the result isnt overwhelming. Quick Tip: Wrap It UpOne of the simplest and most useful tricks for implementing centered text alignments within a larger layout is shown in the example below.First, look at this page as a series of blocks. Notice that all of the have a fairly justified alignment. With the exception of the left-aligned navigation, theres both a hard left and a hard right edge that defines the content. Now, within that framework I have a little more freedom to play around. Notice that Box One and Box Two actually contain centered content, but the containment devices keep them looking perfect within the overall layout.Anytime you have some center-aligned content that just doesnt feel like its working, try wrapping it in a box that conforms to the overall page layout to see if the design improves.ConclusionTo sum up, centered alignments are weak, but not bad. Theres a very important distinction there that you really need to grasp to be a good designer. Weak implies that they can easily be taken too far and are easy to abuse. However, you can and should be using centered alignments in your designs.Make sure that, if your entire layout is built on a centered alignment, you have a very simple design with only a few items. Once you start adding big blocks of text and lots of images, the centered alignments starts looking messy. Also, try building a solid left, justified or even right alignment for your page as a CSS practitioner: "How do I center a div?" The holy grail: vertical and horizontal centering, aka centering along both the x-axis and the y-axis. The most modern and easiest way is with the following two lines of CSS: CSS for "X Centering with Flexbox, just apply display: flex and justify-content: center to the parent element: .container { font-family: arial; font-size: 24px; margin: 25px; width: 350px; height: 200px; outline: dashed 1px black; display: flex; justify-content: center;} To center a box inside another box, first turn the containing box into a flex container by setting its display property to flex. Then set align-items to center for vertical centering (on the block axis) and justify-content to center for horizontal centering (on the inline axis). And that's all it takes to center one box inside another! I am centered!div { border: solid 3px; padding: 1em; max-width: 75%;} .item { border: 2px solid rgb(75 70 74); border-radius: 0.5em; padding: 10em;} .container { height: 8em; border: 2px solid rgb(75 70 74); border-radius: 0.5em; font: 1.2em sans-serif; height: 200px; display: flex; align-items: center; justify-content: center;}To center a box inside another box, first turn the containing box into a flex container by setting its display property to flex. Then set align-items to center for vertical centering (on the block axis) and justify-content to center for horizontal centering (on the inline axis). And that's all it takes to center one box inside another! I am centered!div { border: solid 3px; padding: 1em; max-width: 75%;} .item { border: 2px solid rgb(75 70 74); border-radius: 0.5em; padding: 8em; border: 2px solid rgb(75 70 74); border-radius: 0.5em; font: 1.2em sans-serif; display: grid; place-items: center; on the container, you can also vertically center the inner item by setting align-self to center on the inner item itself.Another method you can use for centering one box inside another is to first make the containing box a grid container and then set its place-items property to center to center align its items on both the block and inline axes. I am centered!div { border: solid 3px; padding: 1em; max-width: 75%;} .item { border: 2px solid rgb(95 97 110); border-radius: 0.5em; padding: 8em; border: 2px solid rgb(75 70 74); border-radius: 0.5em; font: 1.2em sans-serif; display: grid; place-items: center; on the container, you can achieve the same centering by setting place-content: center; on the container or by applying either place-self: center or margin: auto; on the inner item itself. Centering things is one of the most difficult aspects of CSS. The methods themselves usually aren't difficult to understand. Instead, it's more due to the fact that there are so many ways to center things. The method you use can vary depending on the HTML element you're trying to center, or whether you're centering it horizontally or vertically.In this tutorial, we'll go over how to use different elements horizontally, vertically, and both vertically and horizontally. Centering elements horizontally is generally easier than centering them vertically. Here are some common elements you may want to center horizontally and different ways to do it.How to Center Text with the CSS Text-Align Center PropertyTo center text or links horizontally, just use the text-align property with the value center: Hello, (centered) World!.container { font-family: arial; font-size: 24px; margin: 25px; width: 350px; height: 200px; outline: dashed 1px black;} p { text-align: center;} Use the shorthand margin property with the value 0 auto to center block-level elements like a div horizontally: .container { font-family: arial; font-size: 24px; margin: 25px; width: 350px; height: 200px; outline: dashed 1px black;} .child { width: 50px; height: 50px; background-color: red; margin: 0 auto;} Flexbox is the most modern way to center things on the page, and makes designing responsive layouts much easier than it used to be. However, it's not fully supported in some legacy browsers like Internet Explorer.To center an element horizontally with Flexbox, just apply display: flex and justify-content: center to the parent element: .container { font-family: arial; font-size: 24px; margin: 25px; width: 350px; height: 200px; outline: dashed 1px black; display: flex; justify-content: center;} Centering elements vertically without modern methods like Flexbox can be a real chore. Here we'll go over some of the older methods to center things vertically first, then show you how to do it with Flexbox. For a long time this was the go-to way to center things vertically. For this method you must know the height of the element you want to center.First, set the position property of the parent element to relative. Then for the child element, set the position property to absolute. You can set the top property to 50%.But that really just vertically centers the top of the child element.To truly center the child element, set the margin-top property to -(half the child element's height).container { font-family: arial; font-size: 24px; margin: 25px; width: 350px; height: 200px; outline: dashed 1px black; position: relative;} .child { width: 50px; height: 50px; background-color: red; position: absolute; top: 50%; margin-top: -25px; } If you don't know the height of the element you want to center (or even if you do), this method is a nifty trick.This method is very similar to the negative margins method above. Set the position property of the parent element to relative. For the child element, set the position property to absolute and set top to 50%. Now instead of using a negative margin to truly center the child element, just use transform: translate(0, -50%):.container { font-family: arial; font-size: 24px; margin: 25px; width: 350px; height: 200px; outline: dashed 1px black; position: relative;} .child { width: 50px; height: 50px; background-color: red; position: absolute; top: 50%; transform: translate(0, -50%);}Note that translate(0, -50%) is shorthand for translateX(0) and translateY(-50%). You could also write transform: translateY(-50%) to center the child element vertically. Like centering things horizontally, Flexbox is super easy to center things vertically.To center an element vertically, apply display: flex and align-items: center to the parent element: .container { font-family: arial; font-size: 24px; margin: 25px; width: 350px; height: 200px; outline: dashed 1px black; display: flex; align-items: center;} .child { width: 50px; height: 50px; background-color: red;} This is where I'll show you how to center elements both vertically and horizontally with the older methods and Flexbox.For this method, you must know the width and height of the element you want to center.Set the position property of the parent element to relative. Then set the child's position property to absolute, top to 50%, and left to 50%. This just centers the top left corner of the child element vertically and horizontally.To truly center the child element, apply a negative top margin set to half the child element's height, and a negative left margin set to half the child element's width.container { font-family: arial; font-size: 24px; margin: 25px; width: 350px; height: 200px; outline: dashed 1px black; position: relative;} .child { width: 50px; height: 50px; background-color: red; position: absolute; top: 50%; left: 50%; margin-top: -25px; margin-left: -25px;} Use this method to center an element vertically and horizontally if you don't know its exact dimensions and can't use Flexbox.First, set the position property of the parent element to relative. Next, set the child element's position property to absolute, top to 50%, and left to 50%. This just centers the top left corner of the child element vertically and horizontally. Finally, use transform: translate(-50%, -50%) to truly center the child element: .container { font-family: arial; font-size: 24px; margin: 25px; width: 350px; height: 200px; outline: dashed 1px black; position: relative;} .child { width: 50px; height: 50px; background-color: red; position: absolute; top: 50%; left: 50%; transform: translate(-50%, -50%);} Flexbox is the easiest way to center an element both vertically and horizontally. This is really just a combination of the two previous Flexbox methods. To center the child element(s) horizontally and vertically, apply justify-content: center and align-items: center to the parent element: .container { font-family: arial; font-size: 24px; margin: 25px; width: 350px; height: 200px; outline: dashed 1px black; display: flex; justify-content: center; align-items: center;} .child { width: 50px; height: 50px; background-color: red;} That's everything you need to know to center things with CSS. Now go forth and center all the things. A combinator is something that explains the relationship between the selectors.A CSS selector can contain more than one simple selector. Between the simple selectors, we can include a combinator. There are four different combinators in CSS:Descendant combinator (space)Child combinator (>)Next sibling combinator (+)Subsequent-sibling combinator (~) Descendant CombinatorThe descendant combinator matches all elements that are descendants of a specified element.The following example selects all elements inside elements div p {background-color: yellow;}Try it Yourself Child Combinator (>)The child combinator selects all elements that are the children of a specified element.The following example selects all elements that are children of a element:div > p {background-color: yellow;}Try it Yourself The next sibling combinator is used to select an element that is directly after another specific element.Sibling elements must have the same parent element, and "adjacent" means "immediately following".The following example selects the first element that are placed immediately after elements:div + p {background-color: yellow;}Try it Yourself Subsequent-sibling Combinator (~)The subsequent-sibling combinator selects all elements that are next siblings of a specified element.The following example selects all elements that are next siblings of elements:div ~ p {background-color: yellow;}Try it Yourself

When to use alignas. When to do alignment. When to do balancing and alignment. When should alignment be done. When is the best time to use center alignment. Alignment alignment.center. Alignment at. When to use centered text.