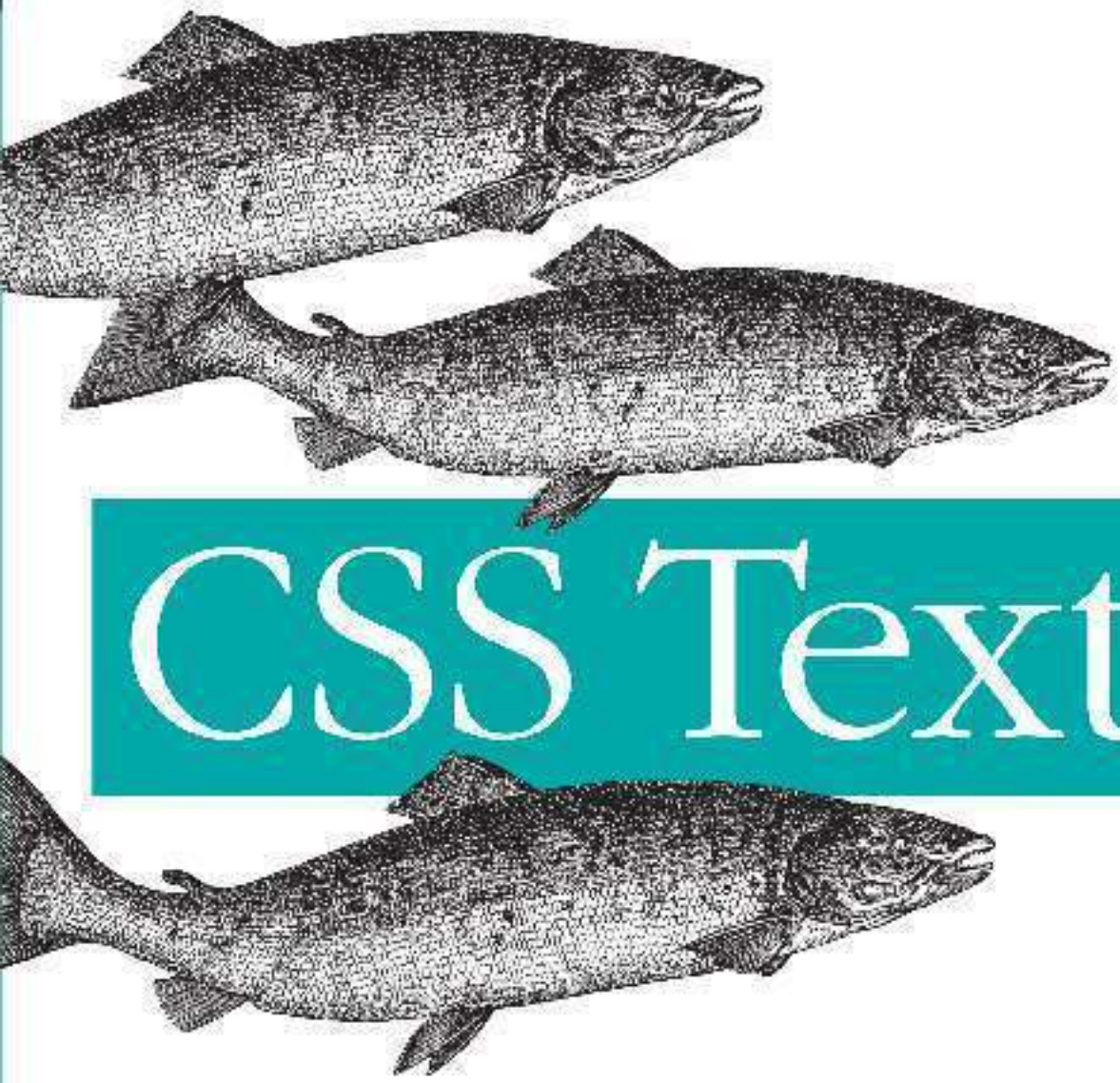
Styling Your Words



CSS Text

O'REILLY®

Eric A. Meyer

CSS Text

As a web designer, you probably spend more time working with text than any other element. With this concise guide, you'll learn CSS properties for changing the appearance of text without altering the font face—including horizontal and vertical alignment, text transformation, word and letter spacing, text wrapping, and the direction of text flow.

This book is an excerpt from the upcoming fourth edition of *CSS: The Definitive Guide*. When you purchase either the print or the ebook edition of *CSS Text*, you'll receive a discount on the entire *Definitive Guide* once it's released. Why wait, when you can start manipulating text on your pages right away?

- Use properties for indenting and aligning lines of text
- Control the leading between lines of text beyond the font's size
- Change the amount of space between words and individual characters
- Add underlines, overlines, strike-throughs, shadows, and other effects
- Instruct browsers to prioritize speed, legibility, or geometric precision when rendering text
- Learn how and when to suppress automatic hyphenation
- Determine the direction that text flows, including left-to-right and top-to-bottom

Get the ebook edition of this O'Reilly title at oreilly.com and receive free updates for the life of the edition. Our ebooks are optimized for several electronic formats, including PDF, EPUB, Mobi, and DAISY—all DRM free.

US $4.99 CAN $5.99
ISBN: 978-1-449-37374-0

twitter: goreillymedia
facebook.com/oreilly

O'REILLY®
oreilly.com

CSS Text

Eric A. Meyer

O'REILLY®
Beijing · Cambridge · Farnham · Köln · Sebastopol · Tokyo

CSS Text

by Eric A. Meyer

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://my.safaribooksonline.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Editors: Simon St. Laurent and Meghan Blanchette
Production Editor: Kara Ebrahim
Proofreader: Nicole Shelby
Cover Designer: Karen Montgomery
Interior Designer: David Futato
Illustrator: Rebecca Demarest

August 2013: First Edition

Revision History for the First Edition:

2013-08-21: First release

See *http://oreilly.com/catalog/errata.csp?isbn=9781449373740* for release details.

ISBN: 978-1-449-37374-0

[LSI]

Table of Contents

Preface

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic
: Indicates new terms, URLs, email addresses, filenames, and file extensions.

`Constant width`
: Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

`Constant width bold`
: Shows commands or other text that should be typed literally by the user.

`Constant width italic`
: Shows text that should be replaced with user-supplied values or by values determined by context.

This icon signifies a tip, suggestion, or general note.

This icon indicates a warning or caution.

Safari® Books Online

Safari Books Online (*www.safaribooksonline.com*) is an on-demand digital library that delivers expert content in both book and video form from the world's leading authors in technology and business.

Technology professionals, software developers, web designers, and business and creative professionals use Safari Books Online as their primary resource for research, problem solving, learning, and certification training.

Safari Books Online offers a range of product mixes and pricing programs for organizations, government agencies, and individuals. Subscribers have access to thousands of books, training videos, and prepublication manuscripts in one fully searchable database from publishers like O'Reilly Media, Prentice Hall Professional, Addison-Wesley Professional, Microsoft Press, Sams, Que, Peachpit Press, Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan Kaufmann, IBM Redbooks, Packt, Adobe Press, FT Press, Apress, Manning, New Riders, McGraw-Hill, Jones & Bartlett, Course Technology, and dozens more. For more information about Safari Books Online, please visit us online.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800 998 9938 (in the United States or Canada)
707-829-0515 (international or local)
707 829 0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at *http://oreil.ly/csstext-mirror*.

To comment or ask technical questions about this book, send email to *bookquestions@oreilly.com*

For more information about our books, courses, conferences, and news, see our website at *http://www.oreilly.com*.

Find us on Facebook: *http://facebook.com/oreilly*

Follow us on Twitter: *http://twitter.com/oreillymedia*

Watch us on YouTube: *http://www.youtube.com/oreillymedia*

Text Properties

Sure, a lot of web design involves picking the right colors and getting the coolest look for your pages, but when it comes right down to it, you probably spend more of your time worrying about where text will go and how it will look. Such concerns gave rise to HTML tags such as `<FONT>` and `<CENTER>`, which allow you some measure of control over the appearance and placement of text.

Because text is so important, there are many CSS properties that affect it in one way or another. What is the difference between text and fonts? Simply, text is the content, and fonts are used to display that content. Using text properties, you can affect the position of text in relation to the rest of the line, superscript it, underline it, and change the capitalization. You can even simulate, to a limited degree, the use of a typewriter's Tab key.

Indentation and Horizontal Alignment

Let's start with a discussion of how you can affect the horizontal positioning of text within a line. Think of these basic actions as the same types of steps you might take to create a newsletter or write a report.

Indenting Text

Indenting the first line of a paragraph on a web page is one of the most sought-after text-formatting effects. (Eliminating the blank line between paragraphs is a close second.) Some sites used to create the illusion of indented text by placing a small transparent image before the first letter in a paragraph, which shoves the text over. Thanks to CSS, there's a much better way to indent text, called `text-indent`.

text-indent

Values:
<*length*> | <*percentage*> | `inherit`

Initial value:
`0`

Applies to:
Block-level elements

Inherited:
Yes

Percentages:
Refer to the width of the containing block

Computed value:
For percentage values, as specified; for length values, the absolute length

Using `text-indent`, the first line of any element can be indented by a given length—even if that length is negative. The most common use for this property is, of course, to indent the first line of a paragraph:

```
p {text-indent: 3em;}
```

This rule will cause the first line of any paragraph to be indented three ems, as shown in Figure 1.

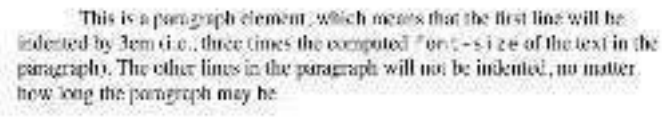


Figure 1. Text indenting

In general, you can apply `text-indent` to any block-level element. You can't apply it to inline elements or on replaced elements such as images. However, if you have an image within the first line of a block-level element, like a paragraph, it will be shifted over with the rest of the text in the line.

If you want to "indent" the first line of an inline element, you can create the effect with left padding or margin.

You can also set negative values for `text-indent`, a technique that leads to a number of interesting effects. The most common use is a "hanging indent," where the first line hangs out to the left of the rest of the element:

```
p {text-indent: -4em;}
```

Be careful when setting a negative value for `text-indent`; the first three words ("This is a") may be chopped off by the left edge of the browser window. To avoid display problems, I recommend you use a margin or some padding to accommodate the negative indentation:

```
p {text-indent: -4em; padding-left: 4em;}
```

Negative indents can, however, be used to your advantage. Consider the following example, demonstrated in Figure 2, which adds a floated image to the mix:

```
p.hang {text-indent: -25px;}

<img src="star.gif" style="width: 60px; height: 60px;
float: left;" alt="An image of a five-pointed star."/>
<p class="hang"> This paragraph has a negatively indented first
line, which overlaps the floated image that precedes the text.  Subsequent
lines do not overlap the image, since they are not indented in any way.</p>
```

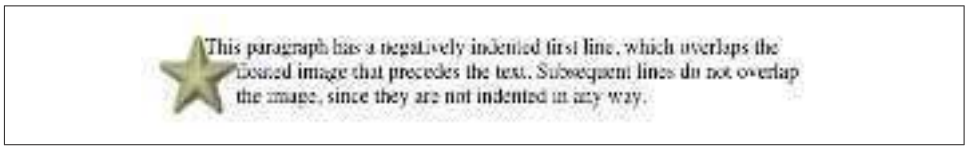


Figure 2. A floated image and negative text indenting

A variety of interesting designs can be achieved using this simple technique.

Any unit of length, including percentage values, may be used with `text-indent`. In the following case, the percentage refers to the width of the parent element of the element being indented. In other words, if you set the indent value to `10%`, the first line of an affected element will be indented by 10 percent of its parent element's width, as shown in Figure 3:

```
div {width: 400px;}
p {text-indent: 10%;}

<div>
<p>This paragraph is contained inside a DIV, which is 400px wide, so the
first line of the paragraph is indented 40px (400 * 10% = 40).  This is
```

```
because percentages are computed with respect to the width of the element.</p>
</div>
```

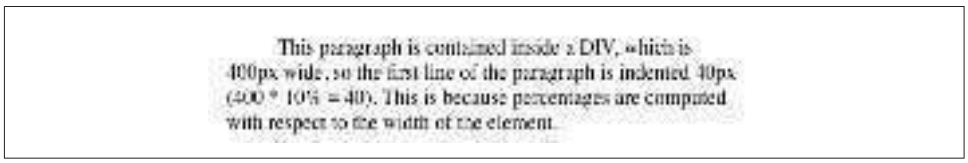


Figure 3. Text indenting with percentages

Note that this indentation only applies to the first line of an element, even if you insert line breaks. The interesting part about `text-indent` is that because it's inherited, it can have unexpected effects. For example, consider the following markup, which is illustrated in Figure 4:

```
div#outer {width: 500px;}
div#inner {text-indent: 10%;}
p {width: 200px;}
```

```
<div id="outer">
<div id="inner">
This first line of the DIV is indented by 50 pixels.
<p>
This paragraph is 200px wide, and the first line of the paragraph
is indented 50px.  This is because computed values for 'text-indent'
are inherited, instead of the declared values.
</p>
</div>
</div>
```

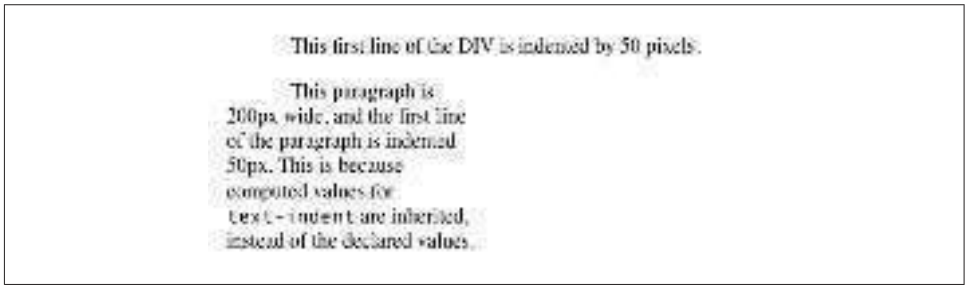


Figure 4. Inherited text indenting

Horizontal Alignment

Even more basic than `text-indent` is the property `text-align`, which affects how the lines of text in an element are aligned with respect to one another.

text-align

CSS 2.1 values:
`left` | `center` | `right` | `justify` | `inherit`

CSS3 values:
[[`start` | `end` | `left` | `right` | `center`] || <*string*>] | `justify` | `match-parent` | `start end` | `inherit`

Initial value:
In CSS3, `start`; in CSS 2.1, user agent-specific, likely depending on writing direction

Applies to:
Block-level elements

Inherited:
Yes

Computed value:
As specified, except in the case of `match-parent`

Note:
CSS2 included a <*string*> value that was dropped from CSS 2.1 due to a lack of implementation

The quickest way to understand how these values work is to examine Figure 5, which sticks with three of the CSS 2.1 values for the moment.

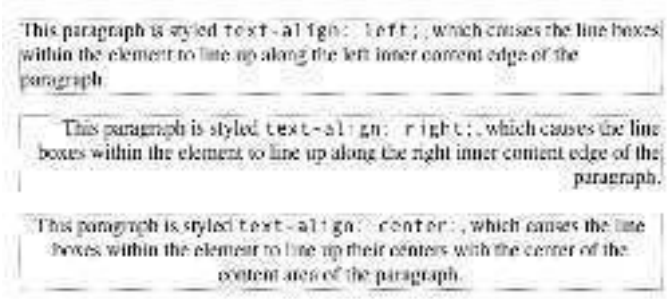


Figure 5. Selected behaviors of the text-align property

Obviously, the values `left`, `right`, and `center` cause the text within elements to be aligned exactly as described. Because `text-align` applies only to block-level elements,

such as paragraphs, there's no way to center an anchor within its line without aligning the rest of the line (nor would you want to, since that would likely cause text overlap).

Historically, which is to say under CSS 2.1 rules, the default value of `text-align` is `left` in left-to-right languages, and `right` in right-to-left languages. (CSS 2.1 had no notion of vertical writing modes.) In CSS3, `left` and `right` are mapped to the start or end edge, respectively, of a vertical language. This is illustrated in Figure 6.

As you no doubt expect, `center` causes each line of text to be centered within the element. Although you may be tempted to believe that `text-align: center` is the same as the `<CENTER>` element, it's actually quite different. `<CENTER>` affected not only text, but also centered whole elements, such as tables. `text-align` does not control the alignment of elements, only their inline content. Figures 5 and 6 illustrate this clearly in various writing directions.

Figure 6. Left, right, and center in vertical writing modes

Start and end alignment

CSS3 (which is to say, the "CSS Text Module Level 3" specification) added a number of new values to `text-align`, and even changed the default property value as compared to CSS 2.1.

The new default value of `start` means that the text is aligned to the start edge of its line box. In left-to-right languages like English, that's the left edge; in right-to-left languages such as Hebrew, it's the right edge. In vertical languages, for that matter, it will be the top or bottom, depending on the writing direction. The upshot is that the default value is much more aware of the document's language direction while leaving the default behavior the same in the vast majority of existing cases.

In a like manner, `end` aligns text with the end edge of each line box—the right edge in LTR languages, the left edge in RTL languages, and so forth. The effects of these values are shown in Figure 7.

Figure 7. Start and end alignment

And then there's the value `start end`, which is declared like so:

```
p {text-align: start end;}
```

This is a fascinating, and so far unsupported, value of `text-align`. The effect is that the first line of the element, as well as any line that immediately follows a line break, uses `start` alignment. The rest of the lines in the element use `end` alignment. (It may be dropped from the CSS3 module if nobody starts supporting it.)

Justified text

An often-overlooked alignment property is `justify`, which raises some issues of its own. In justified text, both ends of a line of text are placed at the inner edge of the parent element, as Figure 8 shows. Then, the spacing between words and letters is adjusted so that each line is precisely the same length. Justified text is common in the print world (for example, in this book), but under CSS, a few extra considerations come into play.

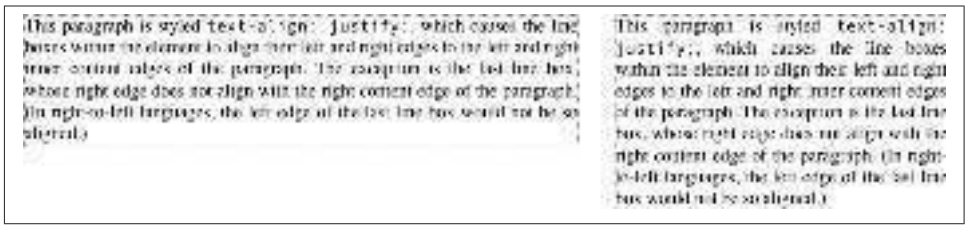


Figure 8. Justified text

The user agent—and not CSS, at least as of mid-2013—determines how justified text should be stretched to fill the space between the left and right edges of the parent. Some browsers, for example, might add extra space only between words, while others might distribute the extra space between letters (although the CSS specification states that "user agents may not further increase or decrease the inter-character space" if the property `letter-spacing` has been assigned a length value). Other user agents may reduce space on some lines, thus mashing the text together a bit more than usual. All of these possibilities will affect the appearance of an element, and may even change its height, depending on how many lines of text result from the user agent's justification choices.

String alignment

String alignment is an interesting case, one that has been lurking throughout the history of CSS without actually gaining traction. The easiest way to explain it is to show some code and the results (Figure 9):

```
table[summary="Daily Sales"] th {text-align: "/";}
table[summary="Daily Sales"] td {text-align: ".";}

<table summary="Daily Sales">
<tr><th>Mon 4/8</th><td>$12,122.13</td></tr>
<tr><th>Tue 4/9</th><td>$13,729.10</td></tr>
<tr><th>Wed 4/10</th><td>$9,447</td></tr>
<tr><th>Thu 4/11</th><td>$10,308.76</td></tr>
<tr><th>Fri 4/12</th><td>$999.99</td></tr>
</table>
```

Figure 9. String alignment

As Figure 9 shows, all the cells' contents are aligned such that the "." characters line up along a vertical line (since this is a horizontal language). Had the CSS actually been `text-align: ","` then the cells would all have lined up on the commas.

The problem, as always, is that string alignment still isn't supported by browsers, and may be once again dropped from the specification. (Figure 9 was faked, I'm sorry to say.)

Parent matching

There's one more value to be covered, which is `match-parent`. This isn't supported by browsers, but its intent is mostly covered by `inherit` anyway. The idea is, if you declare `text-align: match-parent`, the alignment of the element will match the alignment of its parent. So far, that sounds exactly like `inherit`, but there's a difference: if the parent's alignment value is `start` or `end`, the result of `match-parent` is to assign a computed value of `left` or `right` to the element. That wouldn't happen with `inherit`, which would simply apply start or end to the element with no changes.

Aligning the Last Line

There may be times when you want to align the text in the very last line of an element differently than you did the rest of the content. For example, you might left-align the last line of an otherwise fully justified block of text, or choose to swap from left to center alignment. For those situations, there is `text-align-last`.

text-align-last

Values:
`auto` | `start` | `end` | `left` | `right` | `center` | `justify`

Initial value:
`auto`

Applies to:
Block-level elements

Inherited:
Yes

Computed value:
As specified

As with `text-align`, the quickest way to understand how these values work is to examine Figure 10.

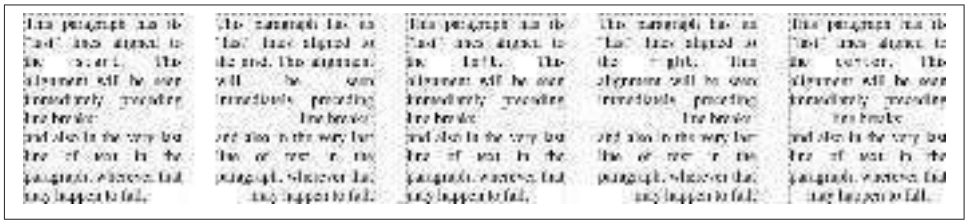


Figure 10. Differently aligned last lines

As you can see, the last lines of the elements are aligned independently of the rest of the elements, according to the elements' `text-align-last` values.

A close study of Figure 10 will reveal that there's more at play than just the last lines of block-level elements. In fact, `text-align-last` applies to any line of text that immediately precedes a forced line break, whether or not said line break is triggered by the end of an element. Thus, a line break occasioned by a `<br>` tag will make the line of text immediately before that break use the value of `text-align-last`. So too will the last

line of text in a block-level element, since a line break is generated by the element's closure.

There are two more interesting wrinkles in `text-align-last`. The first is that if the first line of text in an element is also the last line of text in the element, then the value of `text-align-last` takes precedence over the value of `text-align`. Thus, the following styles will result in a centered paragraph, not a start-aligned paragraph:

```
p {text-align: start; text-align-last: center;}

<p>A paragraph.</p>
```

The second, closely related, wrinkle is that `text-align-last` is ignored if the element's `text-align` value is `start end`. In other words, in the following case, the text will *not* be centered:

```
p {text-align: START end; text-align-last: center;}

<p>A paragraph.</p>
```

As of mid-2013, support for `text-align-last` was limited to a prefixed Gecko property, `-moz-text-align-last`.

Vertical Alignment

Now that we've covered horizontal alignment, let's move on to vertical alignment. Since the construction of lines is a very complex topic that merits its own small book, I'll just stick to a quick overview here.

The Height of Lines

The `line-height` property refers to the distance between the baselines of lines of text rather than the size of the font, and it determines the amount by which the height of each element's box is increased or decreased. In the most basic cases, specifying `line-height` is a way to increase (or decrease) the vertical space between lines of text, but this is a misleadingly simple way of looking at how `line-height` works. `line-height` controls the *leading*, which is the extra space between lines of text above and beyond the font's size. In other words, the difference between the value of `line-height` and the size of the font is the leading.

line-height

Values:
<*length*> | <*percentage*> | <*number*> | `normal` | `inherit`

Initial value:
`normal`

Applies to:
All elements (but see text regarding replaced and block-level elements)

Inherited:
Yes

Percentages:
Relative to the font size of the element

Computed value:
For length and percentage values, the absolute value; otherwise, as specified

When applied to a block-level element, `line-height` defines the minimum distance between text baselines within that element. Note that it defines a minimum, not an absolute value, and baselines of text can wind up being pushed further apart than the value of `line-height`. `line-height` does not affect layout for replaced elements, but it still applies to them.

Constructing a line

Every element in a line of text generates a *content area*, which is determined by the size of the font. This content area in turn generates an *inline box* that is, in the absence of any other factors, exactly equal to the content area. The leading generated by `line-height` is one of the factors that increases or decreases the height of each inline box.

To determine the leading for a given element, simply subtract the computed value of `font-size` from the computed value of `line-height`. That value is the total amount of leading. And remember, it can be a negative number. The leading is then divided in half, and each half-leading is applied to the top and bottom of the content area. The result is the inline box for that element.

As an example, let's say the `font-size` (and therefore the content area) is 14 pixels tall, and the `line-height` is computed to 18 pixels. The difference (4 pixels) is divided in half, and each half is applied to the top and bottom of the content area. This creates an inline box that is 18 pixels tall, with 2 extra pixels above and below the content area. This sounds like a roundabout way to describe how `line-height` works, but there are excellent reasons for the description.

Once all of the inline boxes have been generated for a given line of content, they are then considered in the construction of the line box. A line box is exactly as tall as needed to enclose the top of the tallest inline box and the bottom of the lowest inline box. Figure 11 shows a diagram of this process.

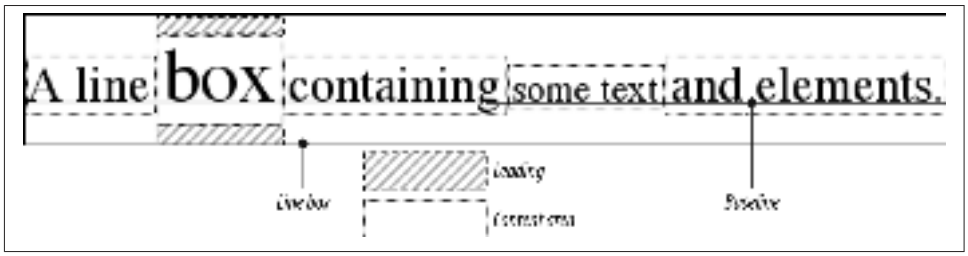


Figure 11. Line box diagram

Assigning values to line-height

Let's now consider the possible values of `line-height`. If you use the default value of `normal`, the user agent must calculate the vertical space between lines. Values can vary by user agent, but they're generally 1.2 times the size of the font, which makes line boxes taller than the value of `font-size` for a given element.

Most values are simple length measures (e.g., `18px` or `2em`). Be aware that even if you use a valid length measurement, such as `4cm`, the browser (or the operating system) may be using an incorrect metric for real-world measurements, so the line height may not show up as exactly four centimeters on your monitor.

`em`, `ex`, and percentage values are calculated with respect to the `font-size` of the element. The markup is relatively straightforward, and the results are shown in Figure 12:

```
body {line-height: 18px; font-size: 16px;}
p.cl1 {line-height: 1.5em;}
p.cl2 {font-size: 10px; line-height: 150%;}
p.cl3 {line-height: 0.33in;}

<p>This paragraph inherits a 'line-height' of 14px from the body, as well as
a 'font-size' of 13px.</p>
<p class="cl1">This paragraph has a 'line-height' of 27px(18 * 1.5), so
it will have slightly more line-height than usual.</p>
<p class="cl2">This paragraph has a 'line-height' of 15px (10 * 150%), so
it will have slightly more line-height than usual.</p>
<p class="cl3">This paragraph has a 'line-height' of 0.33in, so it will have
slightly more line-height than usual.</p>
```

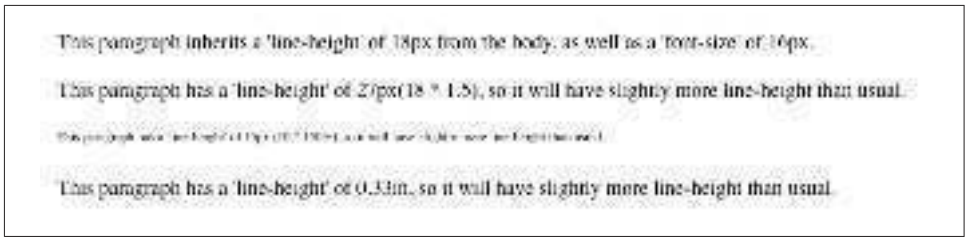


Figure 12. Simple calculations with the line-height property

Line-height and inheritance

When the `line-height` is inherited by one block-level element from another, things get a bit trickier. `line-height` values inherit from the parent element as computed from the parent, not the child. The results of the following markup are shown in Figure 13. It probably wasn't what the author had in mind:

```
body {font-size: 10px;}
div {line-height: 1em;}  /* computes to '10px' */
p {font-size: 18px;}

<div>
<p>This paragraph's 'font-size' is 18px, but the inherited 'line-height'
value is only 10px.  This may cause the lines of text to overlap each
other by a small amount.</p>
</div>
```

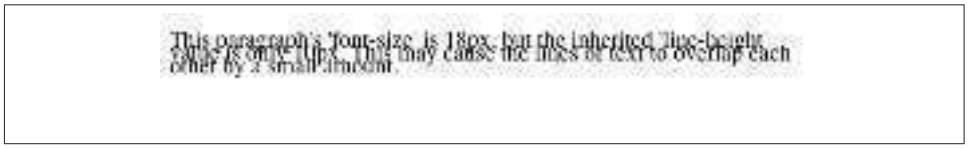


Figure 13. Small line-height, large font-size, slight problem

Why are the lines so close together? Because the computed `line-height` value of `10px` was inherited by the paragraph from its parent `div`. One solution to the small `line-height` problem depicted in Figure 13 is to set an explicit `line-height` for every element, but that's not very practical. A better alternative is to specify a number, which actually sets a scaling factor:

```
body {font-size: 10px;}
div {line-height: 1;}
p {font-size: 18px;}
```

When you specify a number, you cause the scaling factor to be an inherited value instead of a computed value. The number will be applied to the element and all of its child elements, so that each element has a `line-height` calculated with respect to its own `font-size` (see Figure 14):

```
div {line-height: 1.5;}
p {font-size: 18px;}

<div>
<p>This paragraph's 'font-size' is 18px, and since the 'line-height'
set for the parent div is 1.5, the 'line-height' for this paragraph
is 27px (18 * 1.5).</p>
</div>
```

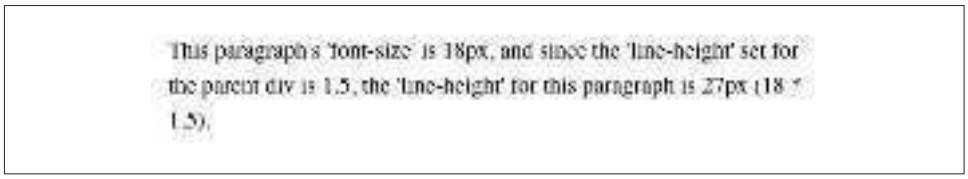


Figure 14. Using line-height factors to overcome inheritance problems

Though it seems like `line-height` distributes extra space both above and below each line of text, it actually adds (or subtracts) a certain amount from the top and bottom of an inline element's content area to create an inline box. Assume that the default `font-size` of a paragraph is `12pt` and consider the following:

```
p {line-height: 16pt;}
```

Since the "inherent" line height of 12-point text is 12 points, the preceding rule will place an extra 4 points of space around each line of text in the paragraph. This extra amount is divided in two, with half going above each line and the other half below. You now have 16 points between the baselines, which is an indirect result of how the extra space is apportioned.

If you specify the value `inherit`, then the element will use the computed value for its parent element. This isn't really any different than allowing the value to inherit naturally, except in terms of specificity and cascade resolution.

Now that you have a basic grasp of how lines are constructed, let's talk about vertically aligning elements relative to the line box.

Vertically Aligning Text

If you've ever used the elements `sup` and `sub` (the superscript and subscript elements), or used an image with markup such as `<img src="foo.gif" align="middle">`, then you've done some rudimentary vertical alignment. In CSS, the `vertical-align` property applies only to inline elements and replaced elements such as images and form inputs. `vertical-align` is not an inherited property.

vertical-align

Values:
`baseline` | `sub` | `super` | `top` | `text-top` | `middle` | `bottom` | `text-bottom` | <*percentage*> | <*length*> | `inherit`

Initial value:
`baseline`

Applies to:
Inline elements and table cells

Inherited:
No

Percentages:
Refer to the value of `line-height` for the element

Computed value:
For percentage and length values, the absolute length; otherwise, as specified

Note:
When applied to table cells, only the values `baseline`, `top`, `middle`, and `bottom` are recognized

`vertical-align` accepts any one of eight keywords, a percentage value, or a length value. The keywords are a mix of the familiar and unfamiliar: `baseline` (the default value), `sub`, `super`, `bottom`, `text-bottom`, `middle`, `top`, and `text-top`. We'll examine how each keyword works in relation to inline elements.

Remember: `vertical-align` does *not* affect the alignment of content within a block-level element. You can, however, use it to affect the vertical alignment of elements within table cells.

Baseline alignment

`vertical-align: baseline` forces the baseline of an element to align with the baseline of its parent. Browsers, for the most part, do this anyway, since you'd obviously expect the bottoms of all text elements in a line to be aligned.

If a vertically aligned element doesn't have a baseline—that is, if it's an image, a form input, or another replaced element—then the bottom of the element is aligned with the baseline of its parent, as Figure 15 shows:

```
img {vertical-align: baseline;}

<p>The image found in this paragraph <img src="dot.gif" alt="A dot" /> has its
bottom edge aligned with the baseline of the text in the paragraph.</p>
```

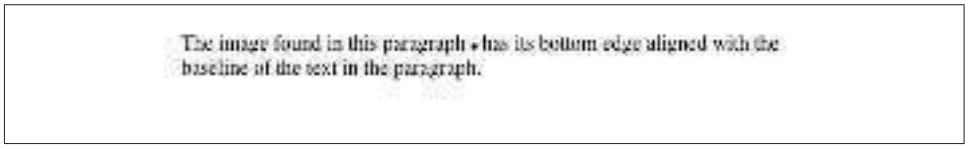


Figure 15. Baseline alignment of an image

This alignment rule is important because it causes some web browsers to always put a replaced element's bottom edge on the baseline, even if there is no other text in the line. For example, let's say you have an image in a table cell all by itself. The image may actually be on a baseline, but in some browsers, the space below the baseline causes a gap to appear beneath the image. Other browsers will "shrink-wrap" the image with the table cell, and no gap will appear. The gap behavior is correct, according to the CSS Working Group, despite its lack of appeal to most authors.

See the aged and yet still relevant article "Images, Tables, and Mysterious Gaps" for a more detailed explanation of gap behavior and ways to work around it.

Superscripting and subscripting

The declaration `vertical-align: sub` causes an element to be subscripted, meaning that its baseline (or bottom, if it's a replaced element) is lowered with respect to its parent's baseline. The specification doesn't define the distance the element is lowered, so it may vary depending on the user agent.

`super` is the opposite of `sub`; it raises the element's baseline (or bottom of a replaced element) with respect to the parent's baseline. Again, the distance the text is raised depends on the user agent.

Note that the values `sub` and `super` do *not* change the element's font size, so subscripted or superscripted text will not become smaller (or larger). Instead, any text in the sub- or superscripted element should be, by default, the same size as text in the parent element, as illustrated by Figure 16:

```
span.raise {vertical-align: super;}
span.lower {vertical-align: sub;}

<p>This paragraph contains <span class="raise">superscripted</span>
and <span class="lower">subscripted</span> text.</P>
```

sample content of CSS Text

- read online Candle 79 Cookbook: Modern Vegan Classics from New York's Premier Sustainable Restaurant book
- *download online Medium Raw: A Bloody Valentine to the World of Food and the People Who Cook book*
- Attaining the Way: A Guide to the Practice of Chan Buddhism online
- Moving Mars (Queen of Angels, Book 3) pdf, azw (kindle)
- click United States Catholic Catechism for Adults

- http://schrolf.de/books/Candle-79-Cookbook--Modern-Vegan-Classics-from-New-York-s-Premier-Sustainable-Restaurant.pdf
- http://patrickvincitore.com/?ebooks/Thin-Air--Shetland--Book-6-.pdf
- http://pittiger.com/lib/Total-Film--UK---January-2011-.pdf
- http://berttrotman.com/library/Better-Homes-and-Gardens-Skinny-Dinners.pdf
- http://patrickvincitore.com/?ebooks/Eragon--Inheritance-Cycle-Series--Book-1-.pdf