FORMATTING TEXT

When you design a text document (for print or the web), one of the first things you do is specify a font. In CSS, fonts are specified using a set of font-related properties for **typeface, size, weight, font style,** and special characters. There are also shortcut properties that let you specify multiple font attributes in a single rule.

Specifying the Font Name

Choosing a typeface, or font family as it is called in CSS, for your text is a good place to start. Let's begin with the font-family property and its values.

font-family

Values: one or more font or generic font family names, separated by commas
Default: depends on the browser
Applies to: all elements Inherits: yes

Use the font-family property to specify a font or list of fonts (known as a **font stack**) by name, as shown in these examples:

```
body { font-family: Arial; }
var { font-family: Courier, monospace; }
p { font-family: "Duru Sans", Verdana, sans-serif; }
```

Here are some important syntax requirements:

- All font names, with the exception of generic font families, must be capitalized. For example, use **Arial** instead of **arial**.
- Use commas to separate multiple font names, as shown in the second and third examples.
- Notice that font names that contain a character space (such as Duru Sans in the third example) must appear within quotation marks.

Font Limitations

Browsers are limited to displaying fonts they have access to. Traditionally, that meant the fonts that were already installed on the user's computer. In 2010, however, there was a boom in browser support for embedded web fonts using the CSS @font-face rule, so it became possible for designers to provide their own fonts.

Even when you specify that the font should be Futura in a style rule, if the browser can't find it (for example, if that font is not installed on the user's computer or the provided web font fails to load), the browser uses its default font instead.

Fortunately, CSS allows us to provide a list of back-up fonts (that **font stack** mentioned above) should our first choice not be available. If the first specified font is not found, the browser tries the next one, and down through the list until it finds one that works. In the third font-family rule shown in the previous code example, if the browser does not find Duru Sans, it will use Verdana, and if Verdana is not available, it will substitute some other sans-serif font.

Generic Font Families

That last option, "some other sans-serif font," needs further discussion. "Sans-serif" is just one of five generic font families that you can specify with the font-family property. When you specify a generic font family, the browser chooses an available font from that stylistic category.
The figure below shows examples from each family.

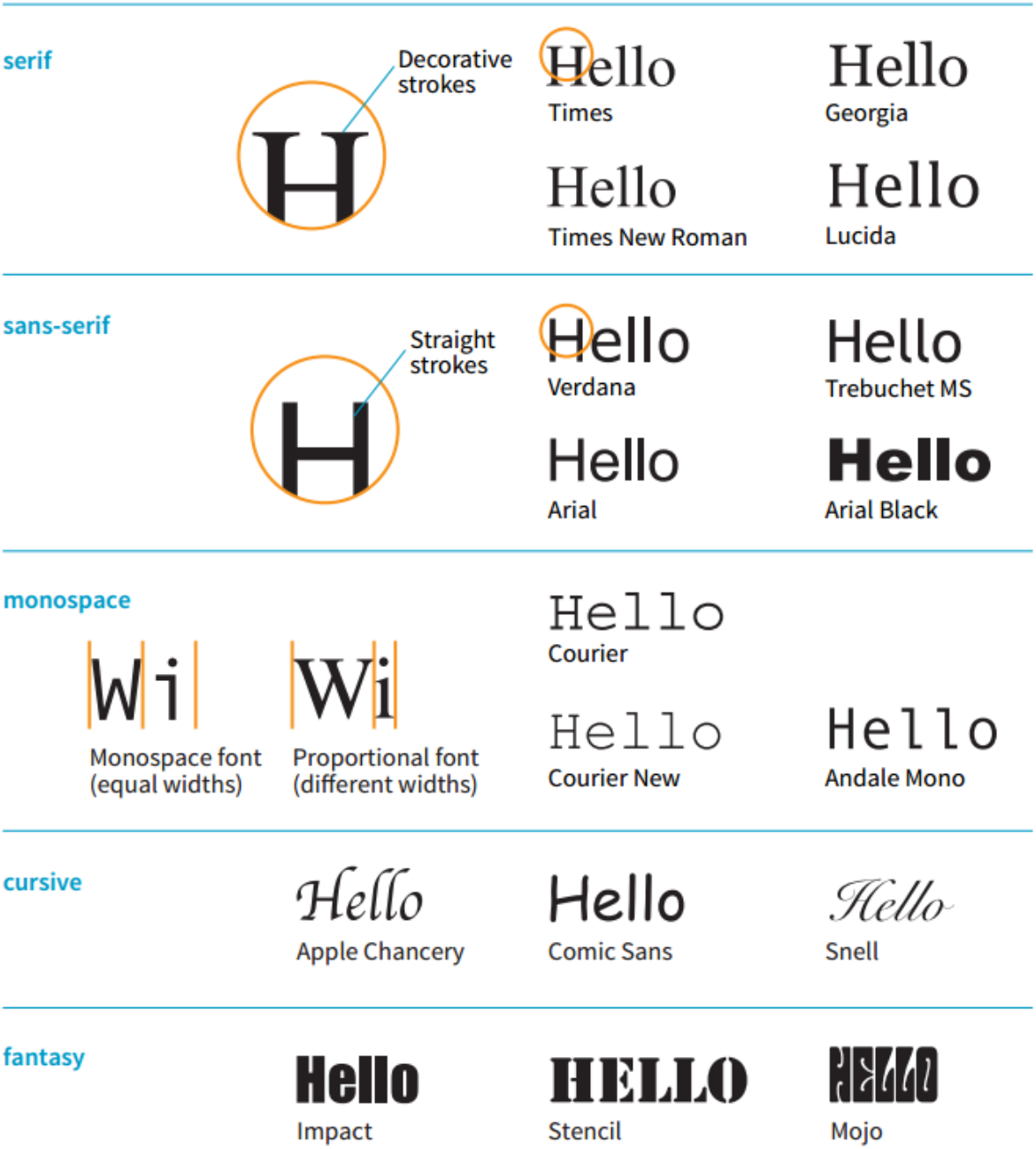


Fig. Examples of the five generic font families.

serif
Examples: Times, Times New Roman, Georgia
Serif typefaces have decorative slab-like appendages (serifs) on the ends of certain letter strokes.

sans-serif
Examples: Arial, Arial Black, Verdana, Trebuchet MS, Helvetica, Geneva Sans-serif typefaces have straight letter strokes that do not end in serifs.

monospace
Examples: Courier, Courier New, and Andale Mono

In monospace (also called constant width) typefaces, all characters take up the same amount of space on a line. For example, a capital W will be no wider than a lowercase i. Compare this to proportional typefaces (such as the one you're reading now) that allot different widths to different characters.

cursive
Examples: Apple Chancery, Zapf-Chancery, and Comic Sans Cursive fonts emulate a script or handwritten appearance.

fantasy
Examples: Impact, Western, or other decorative font
Fantasy fonts are purely decorative and would be appropriate for headlines and other display type.

The best practice for specifying fonts for web pages is to start with your first choice, provide some similar alternatives, and then end with a generic font family that at least gets users in the right stylistic ballpark. For example, if you want an upright, sans-serif font, you might start with a web font if you are providing one (Oswald), list a few that are more common (Univers, Tahoma, Geneva), and finish with the generic sans-serif. There is no limit to the number of fonts you can include, but many designers strive to keep it under 10.

```
font-family: Oswald, Univers, Tahoma, Geneva, sans-serif;
```

Specifying Font Size

Use the *font-size* property to specify the size of the text.

font-size
Values: length unit | percentage | xx-small | x-small | small | medium | large | x-large | xx-large | smaller | larger
Default: medium
Applies to: all elements
Inherits: yes

You can specify text size in several ways:

- Using one of the CSS length units, as shown here:
  ```
  h1 { font-size: 1.5em; }
  ```
  When specifying a number of units, be sure the unit abbreviation immediately follows the number, with no extra character space in between.

- As a percentage value, sized up or down from the element's inherited font size:
  ```
  h1 { font-size: 150%; }
  ```

- Using one of the absolute keywords (**xx-small, x-small, small, medium, large, x-large, xx-large**). On most current browsers, medium corresponds to the default font size.

  ```
  h1 { font-size: x-large; }
  ```
- Using a relative keyword (**larger** or **smaller**) to nudge the text larger or smaller than the surrounding text:
  ```
  strong { font-size: larger; }
  ```

Despite all these options, the preferred values for font-size in contemporary web design are the relative length units **em** and **rem**, as well as percentage values

Sizing Text with Relative Values

The best practice for setting the font size of web page elements is to do it in a way that respects the user's preference. Relative sizing values **%**, rem, and **em** allow you to use the default font size as the basis for proportional sizing of other text elements. It's usually not important that the headlines are exactly 24 pixels; it is important that they are 1.5 times larger than the main text so they stand out. If the user changes their preferences to make their default font size larger, the headlines appear larger, too.

To maintain the browser's default size, set the font-size of the root element to 100%:

```
html {
 font-size: 100%;
}
```

That sets the basis for relative sizing. Because the default font size for all modern browsers is 16 pixels. Keep that fact in mind going forward.

Rem Values

The **rem** unit, which stands for "**root em**," is always relative to the size of the root (html) element. If the root size is 16 pixels, then a **rem** equals 16 pixels. What's nice about rem units is, because they are always relative to the same element, they are the same size wherever you use them throughout the document. In that way, they work like an absolute unit. However, should the root size be something other than 16 pixels, elements specified in rem values will resize accordingly and proportionally.

Here is that same heading sized with rem values:

h1 { font-size: 1.5rem; } /* 1.5 x 16 = 24 */

em Measurements

Em units are based on the font size of the current element. When you specify **font-size** in ems, it will be relative to the inherited size for that element. Once the **em** is calculated for an element, it can be used for other measurements as well, such as margins, padding, element widths, and any other setting you want to always be relative to the size of the font.

Here we've used **em** units to specify the size of an h1 that has inherited the default 16-pixel font size from the root:

```
h1 { font-size: 1.5em; } /* 1.5 x 16 = 24 */
```

There are a few problems in working with ems. One is that because of rounding errors, there is some inconsistency in how browsers and platforms render text set in ems.

The other tricky aspect to using ems is that they are based on the inherited size of the element, which means that their size is based on the context in which they are applied.

The **h1** in the previous example was based on an inherited size of 16 pixels. But if this **h1** had appeared in an article element that had its font size set to 14 pixels, it would inherit the 14-pixel size, and its resulting size would be just 21 pixels (1.5 × 14 = 21).

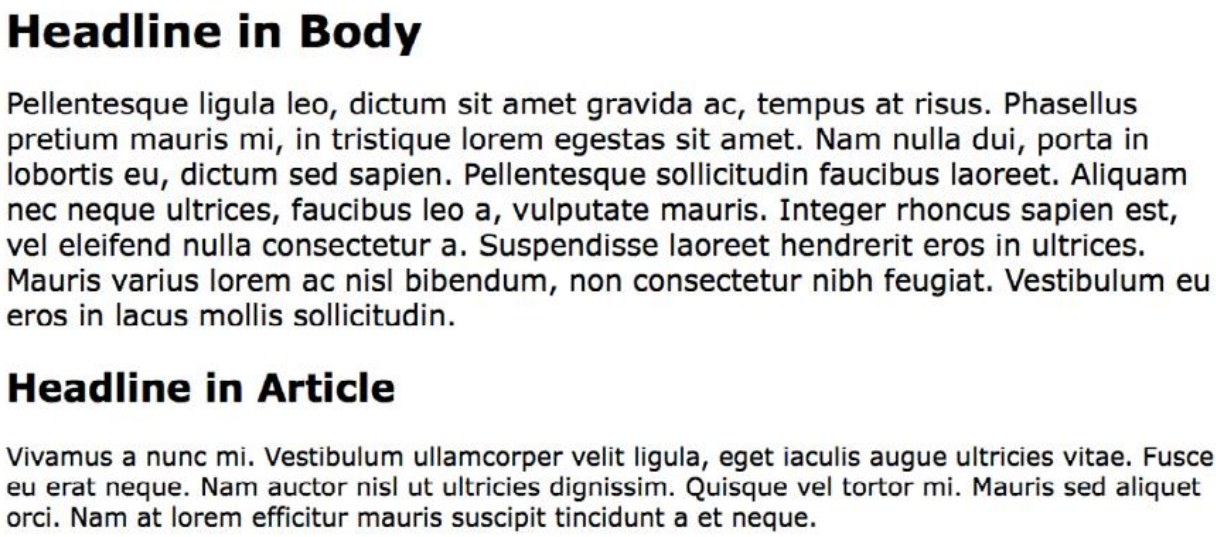
Example:

The Markup

```
<h1>Headline in Body</h1>
<p>Pellentesque ligula leo,…</p>
<article>
 <h1>Headline in Article</h1>
 <p>Vivamus …</p>
</article>
```

The Styles

```
h1 {
 font-size: 1.5em; /* sets all h1s to 1.5em */
}
article {
 font-size: .875em /* 14 pixels based on 16px default */
}
```



From this example, you can see that an element set in ems might appear at different sizes in different parts of the document. If you wanted the h1 in the article to be 24 pixels as well, you could calculate the em value by dividing the target size by its context: 24 / 14 = 1.71428571 em (there is need to round that figure down…the browser knows what to do with it.).

Percentage Values

We saw a percentage value (100%) used to preserve the default font size, but you can use percentage values for any element. They are straightforward.

In this example, the **h1** inherits the default 16px size from the html element, and applying the 150% value multiplies that inherited value, resulting in an h1 that is 24 pixels:

```
h1 { font-size: 150%; }
```

Specifying Font-size with Keywords

An alternative way to specify font-size is by using one of the predefined absolute keywords**: xx-small, x-small, small, medium, large, x-large, and xx-large**. The keywords do not correspond to particular measurements, but rather are scaled consistently in relation to one another. The default size is medium in current browsers.

The relative keywords, **larger** and **smaller**, are used to shift the size of text relative to the size of the parent element text. The exact amount of the size change is determined by each browser and is out of your control.

```
body {
 font-family: Verdana, sans-serif;
 font-size: x-large;
}
P {
```

```
  Font-size: smaller;
}
```

Font Weight (Boldness)

After font families and size, the remaining font properties are straightforward. For example, if you want a text element to appear in bold, use the **font-weight** property to adjust the boldness of type.

font-weight

Values:	normal | bold | bolder | lighter | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800| 900
Default:	normal
Applies to:	all elements
Inherits:	yes

As you can see, the font-weight property has many predefined values, including descriptive terms (normal, bold, bolder, and lighter) and nine numeric values (100 to 900) for targeting various weights of a font if they are available.

Because most fonts commonly used on the web have only two weights, normal (or Roman) and bold, the only font weight value you will use in most cases is bold. You may also use normal to make text that would otherwise appear in bold (such as strong text or headlines) appear at a normal weight.

Font Style (Italics)

The font-style property affects the posture of the text—that is, whether the letter shapes are vertical (**normal**) or slanted (**italic** and **oblique**).

font-style

Values:	normal | italic | oblique
Default:	normal
Applies to:	all elements
Inherits:	yes

Use the **font-style** property to make text **italic**. Another common use is to make text that is italicized in the browser's default styles (such as emphasized text) display as **normal**. There is an **oblique** value that specifies a slanted version of the font; however, browsers generally display **oblique** exactly the same as **italic**.

Font Variant (Small Caps)

font-variant

Values: normal | small-caps
Default: normal
Applies to: all elements
Inherits: yes

Some typefaces come in a "small caps" variant. This is a separate font design that uses small uppercase-style letters in place of lowercase letters. Small caps characters are designed to match the size and density of lowercase text so they blend in.

Small caps should be used for strings of three or more capital letters appear ing in the flow of text, such as acronyms and abbreviations, that may look jarring as full-sized capitals. Compare NASA and USA in the standard font to nasa and usa in small caps. Small caps are also recommended for times, like 1AM or 2017AD.

Font Stretch (Condensed and Extended)

font-stretch

Values: normal | ultra-condensed | extra-condensed | condensed | semi-condensed | semi-expanded | expanded | extra-expanded | ultra-expanded
Default: normal
Applies to: all elements
Inherits: yes
The CSS3 font-stretch property tells the browser to select a normal, condensed, or extended font in the font family.

Fig. Examples of condensed, normal, and extended versions of the Universe typeface.

If the browser cannot find a matching font, it will not try to synthesize the width by stretching or squeezing text; it may just substitute a font of a different width

The Shortcut font Property
Specifying multiple font properties for each text element can get repetitive and lengthy, so the creators of CSS provided the shorthand font property, which compiles all the font-related properties into one rule.

Values: font-style font-weight font-variant font-stretch font-size/line-height font-family status-bar
Default: depends on default value for each property listed
Applies to: all elements
Inherits: yes
The value of the font property is a list of values for all the font properties separated by character spaces. In this property, the order of the values is important:

```
{ font: style weight stretch variant size/line-height font-family; }
```

CHANGING TEXT COLOR
You change the color of text with the color property.
color
Values: color value (name or numeric)
Default: depends on the browser and user's preferences
Applies to: all elements
Inherits: yes

Using the color property is very straightforward. The value of the color property can be a predefined color name or a numeric value describing a specific RGB color. Here are a few examples, all of which make the h1 elements in a document gray:

```
h1 { color: gray; }
h1 { color: #666666; }
h1 { color: #666; }
h1 { color: rgb(102,102,102); }
```

Color is inherited, so you can change the color of all the text in a document by applying the color property to the body element, as shown here:

```
body { color: fuchsia; }
```

The color property is not strictly a text-related property. It can also be used to change the foreground (as opposed to the background) color of an element. The foreground of an element consists of both the text it contains as well as its border. So, when you apply a color to an element (including image elements), know that color will be used for the border as well, unless there is a specific **border-color** property that overrides it

SELECTOR TYPES

So far, we've been using element names as selectors. We have seen how to group selectors together in a comma-separated list so you can apply properties to several elements at once. Here are examples of the selectors that we have already seen.

Element selector `p { color: navy; }`

Grouped selectors `p, ul, td, th { color: navy; }`

The disadvantage of selecting elements this way, of course, is that the property (in this case, navy blue text) is applied to every paragraph and other listed elements in the document. Sometimes you want to apply a rule to a particular paragraph or paragraphs.

We now look at three selector types that allow us to do just that: descendant selectors, ID selectors, and class selectors.

Contextual Selectors

Contextual selectors selects an element based on its context or relation to another element. Following is a discussion of the contextual selectors available in CSS.

i. Descendant Selectors

A descendant selector targets elements that are contained within (and therefore are descendants of) another element. It is an example of a contextual selector because it selects the element based on its context or relation to another element.

Descendant selectors are indicated in a list separated by a character space. This example targets emphasized text (**em**) elements, but only when they appear in list items (**li**). Emphasized text in paragraphs and other elements would be unaffected.

```
li em { color: olive; }
```

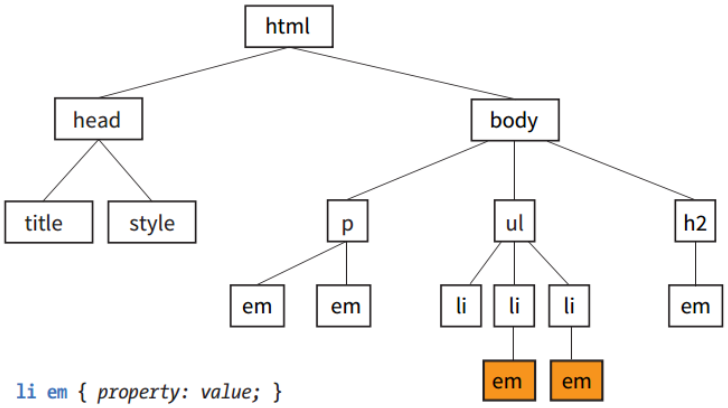


Fig. Only **em** elements within **li** elements are selected. The other **em** elements are unaffected.

Here's another example that shows how contextual selectors can be grouped in a comma-separated list, just as we saw earlier. This rule targets em elements, but only when they appear in h1, h2, and h3 headings:

```
h1 em, h2 em, h3 em { color: red; }
```

It is also possible to nest descendant selectors several layers deep. This example targets em elements that appear in anchors (a) in ordered lists (ol):

```
ol a em { font-size: x-small; }
```

Descendant selectors are one of four types of contextual selectors (also called combinators).
The other three are child selectors, next-sibling selectors, and subsequent-sibling selectors.

ii. Child selector

A child selector is similar to a descendant selector, but it targets only the direct children of a given element. There may be no other hierarchical levels in between. They are indicated with the greater-than symbol (**>**). The following rule affects emphasized text, but only when it is directly contained in a **p** element. An **em** element inside a link (**a**) within the paragraph would not be affected.

```
p > em {font-weight: bold;}
```

iii. Next-sibling selector

A next-sibling selector targets an element that comes directly after another element with the same parent. It is indicated with a plus (**+**) sign. This rule gives special treatment to paragraphs that follow an **h1**. Other paragraphs are unaffected.

```
h1 + p {font-style: italic;}
```

iv. Subsequent-sibling selectors

A subsequent-sibling selector selects an element that shares a parent with the specified element and occurs after it in the source order. They do not need to follow one another directly. This type of selector is new in CSS3 and is not supported by Internet Explorer 8 and earlier. The following rule selects any **h2** that both shares a parent element (such as a section or article) with an **h1** and appears after it in the document.

```
h1 ~ h2 {font-weight: normal;}
```

ID Selectors

As seen previously, the **id** attribute gives an element a unique identifying name (its id reference). The **id** attribute can be used with any element, and it is commonly used to give meaning to the generic **div** and **span** elements. ID selectors allow you to target elements by their **id** values. The symbol that identifies ID selectors is the octothorpe (#), also known as a hash or pound symbol.

Here is an example of a list item with an id reference:

```
<li id="compSci">Computer Science</li>
```

Now you can write a style rule just for that list item using an ID selector as shown below (notice the # preceding the id reference):

```
li#compSci {color: olive;}
```

Because id values must be unique in the document, it is acceptable to omit the element name. The following rule is equivalent to the last one:

```
#compSci {color: olive;}
```

You can also use an ID selector as part of a contextual selector. In this example, a style is applied only to **a** elements that appear within the element identified as "resources." In this way, you can treat links in the element named "resources" differently than all the other links on the page without any additional markup.

```
#resources a {text-decoration: none; }
```

Class Selectors

The class identifier, used to classify elements into a conceptual group. Unlike the **id** attribute, multiple elements may share a class name. Not only that, but an element may belong to **more than one** class.

You can target elements belonging to the same class with a class selector. Class names are indicated with a period (**.**) at the beginning of the selector. For example, to select all paragraphs with **class="special",** use this selector (the period indicates the following word is a class selector):

```
p.special { color: orange; }
```

To apply a property to all elements of the same class, omit the element name in the selector (be sure to leave the period; it's the character that indicates a class). This example targets all paragraphs and any other element that has been marked up with class="special":

```
.special { color: orange; }
```

The Universal Selector

The universal element selector (*) matches any element , like a wildcard in programming languages. The style rule

```
* {border: 1px solid gray; }
```

puts a 1-pixel gray border around every element in the document. It is also useful as a contextual selector, as shown in this example that selects all elements in an "intro" section:

```
#intro * {color: gray; }
```

Be aware that every element will be selected with the universal selector, including some that you might not be expecting to style. For example, some styles might mess up your form controls, so if your page contains form inputs, the safest bet is to avoid the universal selector.

TEXT LINE ADJUSTMENTS

The next batch of text properties has to do with the treatment of whole lines of text rather than the shapes of characters. They allow web authors to for-mat web text with indents, extra space between lines (leading), and different horizontal alignments, similar to print.

a. Line Height

line-height

Values: number | length measurement | percentage | normal

Default: normal

Applies to: all elements
Inherits: yes

The **line-height** property defines the minimum distance from baseline to baseline in text. The line-height property is said to specify a "minimum" distance because if you put a tall image or large characters on a line, the height of that line expands to accommodate it.

A **baseline** is the imaginary line upon which the bottoms of characters sit. Setting a line height in CSS is similar to adding leading in traditional type-setting; however, instead of space being added between lines, the extra space is split above and below the text. The result is that line-height defines the height of a **line-box** in which the text line is vertically centered.

The possible values are: number, length measurement, percentage, normal

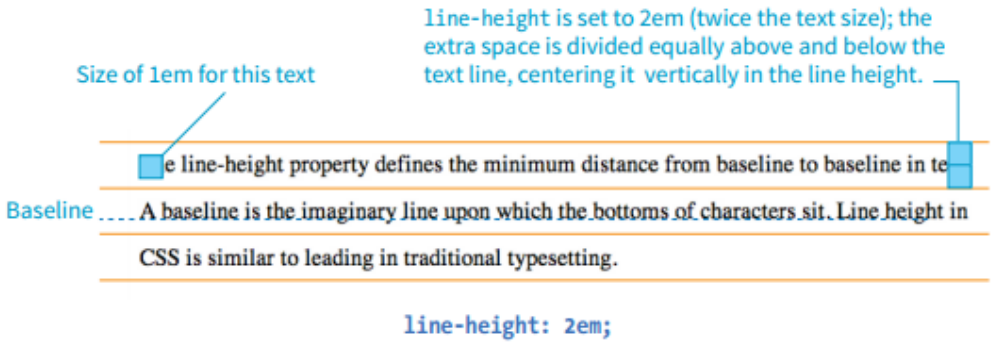


Fig. Text lines are centered vertically in the line height.

These examples show three different ways to make the line height twice the height of the font size:

```
p { line-height: 2; }
p { line-height: 2em; }
p { line-height: 200%; }
```

b. Indents
text-indent
Values: length measurement | percentage
Default: 0
Applies to: block containers
Inherits: yes

The **text-indent** property indents the first line of text by a specified amount. You can specify a length measurement or a percentage value for **text-indent**.
Example:

```
p#1 { text-indent: 2em; }
p#2 { text-indent: 25%; }
p#3 { text-indent: -35px; }
```

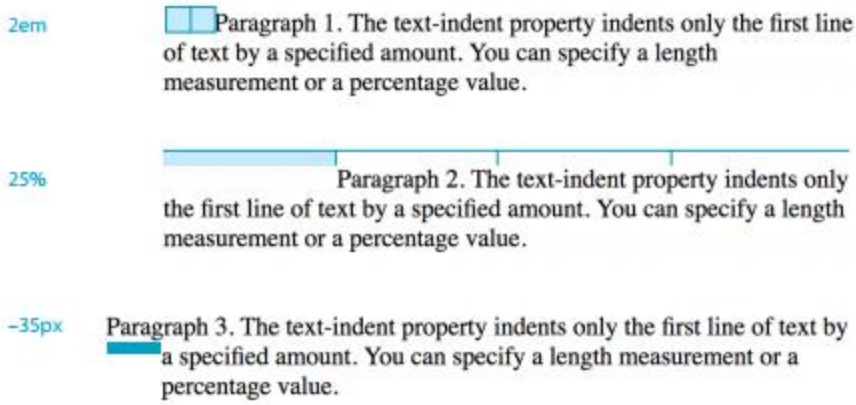
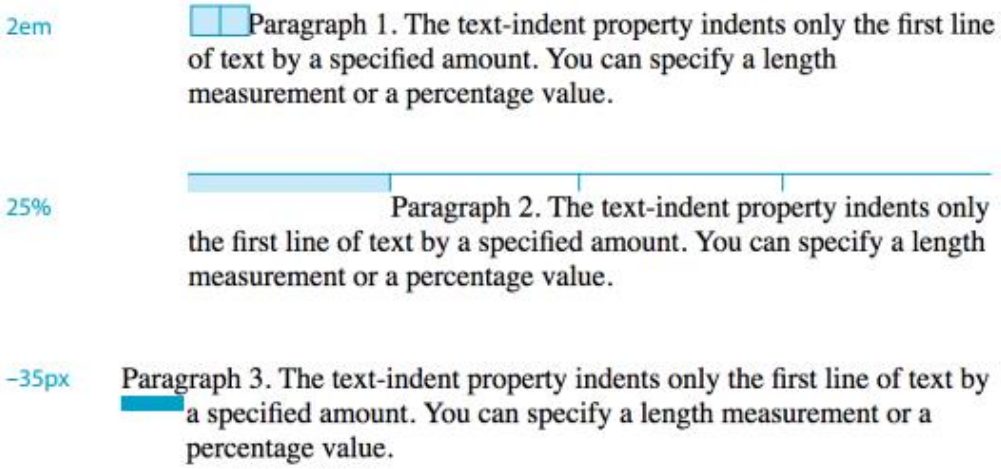


Fig. Examples of the text-indent property.

Percentage values are calculated based on the width of the parent element, and they are passed down to their descendant elements as percentage values (not calculated values). So if a div has a text-indent of 10%, so will all of its descendants.

In the third example, notice that a negative value was specified, and that's just fine. It will cause the first line of text to hang out to the left of the left text edge.



c. Horizontal Text Alignment
text-align

Values: left | right | center | justify | start | end
Default: start
Applies to: block containers
Inherits: yes

You can align text for web pages just as you would in a word processing or desktop publishing program with the **text-align** property. The possible values for this property are: Values: **left , right, center, justify, start,** and **end.**

Examples:

`text-align: left` Aligns text on the left margin

```
text-align: right     Aligns text on the right margin
text-align: center    Centers the text in the text block
text-align: justify   Aligns text on both right and left margins
```

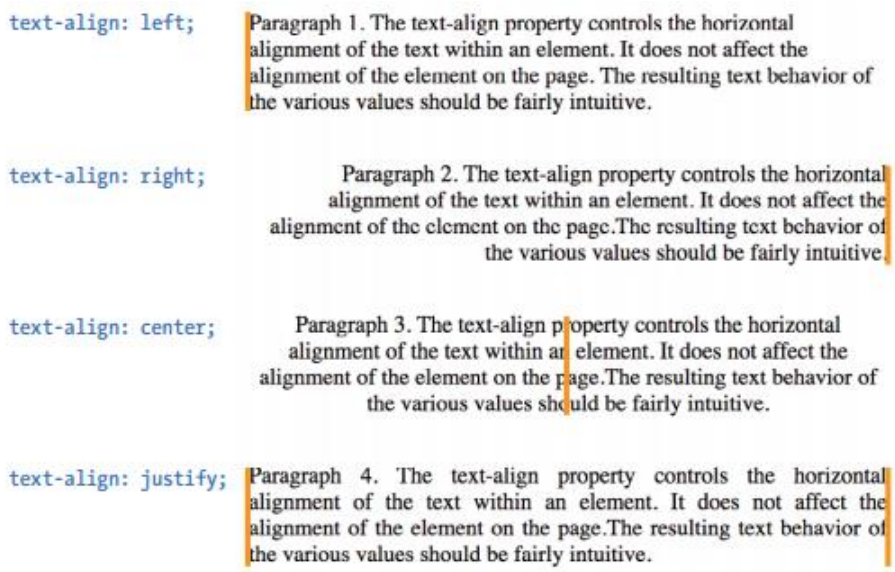


Fig. Examples of CSS2.1 text-align values.

<u>Underlines And Other "Decorations"</u>

If you want to put a line under, over, or through text, or if you'd like to turn of the underline under links, then **text-decoration** is the property you use. It has the possible values **none, underline, overline, line-through, blink.**

text-decoration
Values: none | underline | overline | line-through | blink
Default: none
Applies to: all elements
Inherits: no, but since lines are drawn across child elements, they may look like they are "decorated" too

Examples:

```
text-decoration: underline     Underlines the element
text-decoration: overline      Draws a line over the text
text-decoration: line-through  Draws a line through the text
```

Fig. Examples of text-decoration values.

The most popular use of the text-decoration property is turning off the underlines that appear automatically under linked text, as shown below.

```
a { text-decoration: none; }
```

Changing Capitalization

CSS provides the **text-transform** property for changing the capitalization of text as it is rendered. The values for the property are as follows:

`none`	As it is typed in the source
`capitalize`	Capitalizes the first letter of each word
`lowercase`	Makes all letters lowercase
`uppercase`	Makes all letters uppercase
`full-width`	Chooses a "full-width" version of a character if one exists (not well supported)

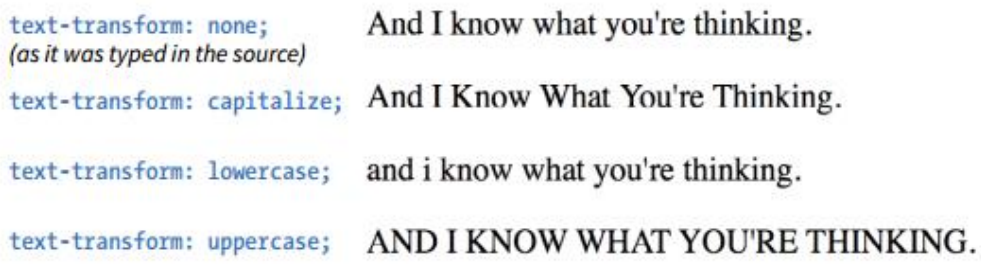


Fig. The text-transform property changes the capitalization of characters when they are displayed, regardless of how they are typed in the source.

Spacing

The next two text properties are used to insert space between letters (**letter-spacing**) or words (**word-spacing)** when the text is displayed.

The **letter-spacing** and **word-spacing** properties do what they say: add space between the letters of the text or words in a line, respectively.

The possible values for both properties are length measurement (**px, em, rem** etc) and **normal**.

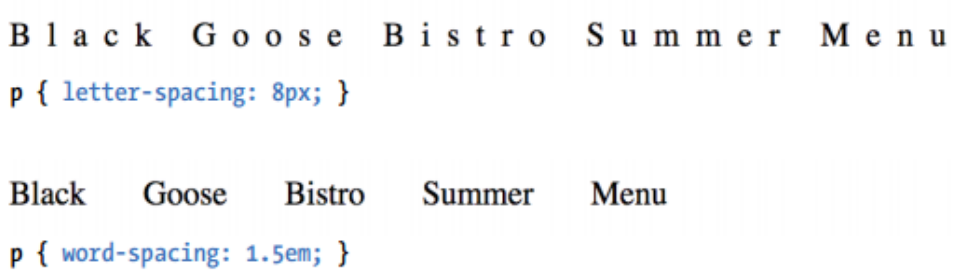


Fig. **letter-spacing** (top) and **word-spacing** (bottom).

Text Shadow

The text-shadow property adds a "shadow" below your text that makes it seem to hover or pop out above the page. Since flat-color design has become the fashion, drop shadows have gone out of style, but they can still be a useful visual tool, particularly when your text is in front of a patterned or photographic background.

Text shadows are drawn behind the text but in front of the background and border if there is one. Text shadows are supported by all current browsers. Internet Explorer versions 9 and earlier lack support.

Possible values for this property are **'horizontal offset' 'vertical offset' 'blur radius' 'color'** and **none**.
The value for the text-shadow property is two or three measurements (a horizontal offset, vertical offset, and an optional blur radius) and a color.

Example:

```
h1 {
 color: darkgreen;
 text-shadow: .2em .2em silver;
}
h1 {
 color: darkgreen;
 text-shadow: -.3em -.3em silver;
}
```

Fig. A minimal text drop shadow.

The first value is a horizontal offset that positions the shadow to the right of the text (a negative value pulls the shadow to the left of the text). The second measurement is a vertical offset that moves the

shadow down by the specified amount (a negative value moves the shadow up). The declaration ends with the color specification (silver). If the color is omitted, the text color will be used.

That should give you an idea for how the first two measurements work, but that sharp shadow doesn't look very…well…shadowy. What it needs is a blur radius measurement. Zero (0) is no blur, and the blur gets softer with higher values (Fig. below). Usually, you just have to fiddle with values until you get the effect you want.

Fig. Adding a blur radius to a text drop shadow.

It is possible to apply several text shadows to the same element. If you vary the position and blur amounts, you can give the text the appearance of multiple light sources.

Changing List Bullets & Numbers

Let's now look at few tweaks you can make to bulleted and numbered lists. As you know, browsers automatically insert bullets before unordered list items, and numbers before items in ordered lists (the list markers). For the most part, the rendering of these markers is determined by the browser. However, CSS provides a few properties that allow authors to choose the type and position of the marker, or turn them off entirely.

Choosing a Marker

Apply the **list-style-type** property to the **ul**, **ol**, or **li** element select the type of marker that appears before each list item.

Possible values: none, disc, circle, square, decimal, decimal-leading-zero, lower-alpha, upper-alpha, lower-latin , upper-latin , lower-roman , upper-roman , lower-greek.

Default value: disc

More often than not, developers use the list-style-type property with its value set to none to remove bullets or numbers altogether. This is handy when you're using list markup as the foundation for a **horizontal navigation menu** or the entries in a **web form**. You can keep the semantics but get rid of the markers.

The disc, circle, and square values generate bullet shapes just as browsers have been doing since the beginning of the web itself (Fig. below). Unfortunately, there is no way to change the appearance (size, color, etc.) of generated bullets, so you're stuck with the browser's default rendering.

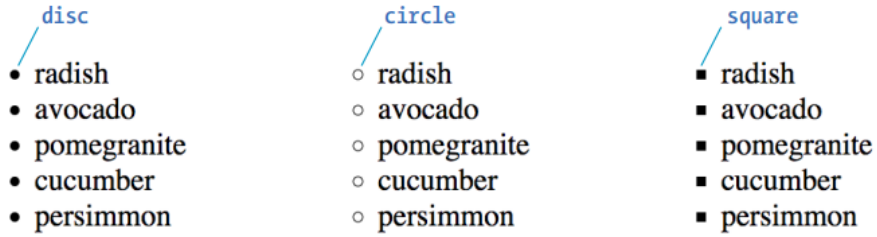


Fig. The list-style-type values disc, circle, and square.

The rest of the keywords (Table below) specify various numbering and lettering styles for use with ordered lists.

Table: Lettering and numbering system

Keyword	System
decimal	1, 2, 3, 4, 5...
decimal-leading-zero	01, 02, 03, 04, 05...
lower-alpha	a, b, c, d, e...
upper-alpha	A, B, C, D, E...
lower-latin	a, b, c, d, e... (same as lower-alpha)
upper-latin	A, B, C, D, E... (same as upper-alpha)
lower-roman	i, ii, iii, iv, v...
upper-roman	I, II, III, IV, V...
lower-greek	α, β, γ, δ, ε...

Marker Position

By default, the marker hangs outside the content area for the list item, displaying as a hanging indent. The **list-style-position** property allows you to pull the bullet inside the content area so it runs into the list content.

The possible values for this property are: **inside, outside and hanging.**

Example:

```
li {background-color: #F99;}
ul#outside {list-style-position: outside;}
ul#inside {list-style-position: inside;}
```

outside

Radish. Praesent in lacinia risus. Morbi urna ipsum, efficitur id erat pellentesque, tincidunt commodo sem. Phasellus est velit, porttitor vel dignissim vitae, commodo ut urna.
Avocado. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Curabitur lacinia accumsan est, ut malesuada lorem consectetur eu.
Pomegranite. Nam euismod a ligula ac bibendum. Aenean ac justo eget lorem dapibus aliquet. Vestibulum vitae luctus orci, id tincidunt nunc. In a mauris odio. Duis convallis enim nunc.

inside

- **Radish.** Praesent in lacinia risus. Morbi urna ipsum, efficitur id erat pellentesque, tincidunt commodo sem. Phasellus est velit, porttitor vel dignissim vitae, commodo ut urna.
- **Avocado.** Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Curabitur lacinia accumsan est, ut malesuada lorem consectetur eu.
- **Pomegranite.** Nam euismod a ligula ac bibendum. Aenean ac justo eget lorem dapibus aliquet. Vestibulum vitae luctus orci, id tincidunt nunc. In a mauris odio. Duis convallis enim nunc.

Fig. The list-style-position property.

You can see that when the position is set to outside (top), the markers fall outside the content area. When it is set to inside (bottom), the markers are tucked into the content area.

CSS3 adds the hanging value for list-style-position. it is similar to inside, but the markers appear outside and abutting the left edge of the shaded area.

Making Your Own Bullets

You can also use your own image as a bullet by using the **list-style-image** property.
The value of the list-style-image property is the URL of the image you want to use as a marker. The list-style-type is set to disc as a backup in case the image does not display or the property isn't supported by the browser or other user agent.

```
ul {
    list-style-type: circle;
    list-style-image: url(/images/favicon.ico);
    list-style-position: outside;
}
```

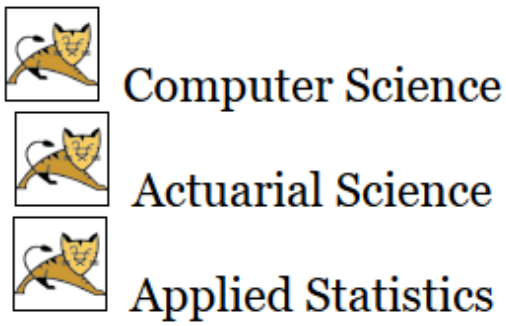


Fig. Using an image as a marker.