INTRODUCING CSS

The contents and slides of this topic are based on:

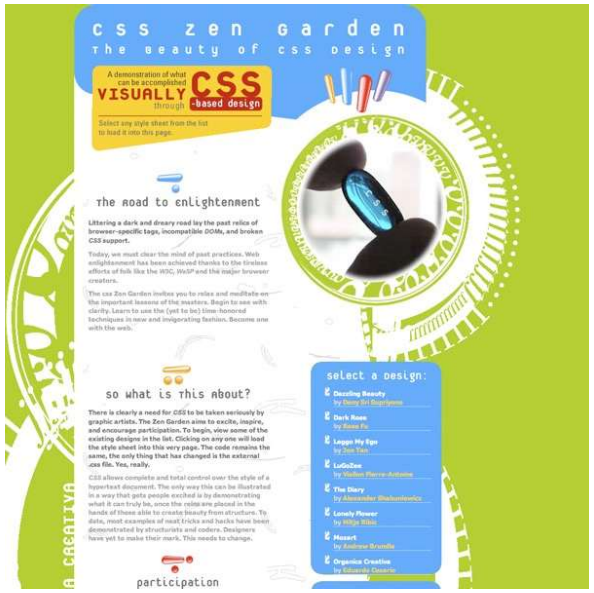
- Jennifer Robbins, Learning Web Design, O'Reilly, 5th edition, May 2018, ISBN 978-1-491-96020-2.
- Paul S. Wang, Dynamic Web programming and HTML5, Routledge, 1 edition, 2012, ISBN 1439871825.

CSS Defined:

- Short for "Cascading Style Sheets".
- Determines how the elements in our XHTML documents are displayed and formatted.
- Designed to separate the content of a web page from the presentation of that content.
- Enables us to make all pages of our website look similar and consistent (font, color, etc.).
- Allows us to make site-wide formatting changes from a single location (rather than having to edit each page individually).

Style Separate from Structure

- These pages have the exact same HTML source but different style sheets:



css zen garden
The Beauty of css Design

A demonstration of what can be accomplished VISUALLY through CSS-based design

Select any style sheet from the list to load it into this page.

The Road to enlightenment

Littering a dark and dreary road lay the past relics of browser-specific tags, incompatible *DOMs*, and broken *CSS* support.

Today, we must clear the mind of past practices. Web enlightenment has been achieved thanks to the tireless efforts of folk like the *W3C*, *WaSP* and the major browser creators.

The css Zen Garden invites you to relax and meditate on the important lessons of the masters. Begin to see with clarity. Learn to use the (yet to be) time-honored techniques in new and invigorating fashion. Become one with the web.

So what is This About?

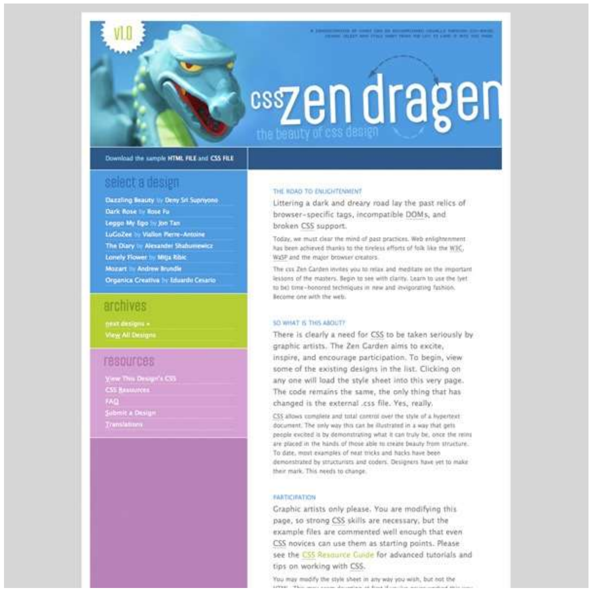
There is clearly a need for *CSS* to be taken seriously by graphic artists. The Zen Garden aims to excite, inspire, and encourage participation. To begin, view some of the existing designs in the list. Clicking on any one will load the style sheet into this very page. The code remains the same, the only thing that has changed is the external .css file. Yes, really.

CSS allows complete and total control over the style of a hypertext document. The only way this can be illustrated in a way that gets people excited is by demonstrating what it can truly be, once the reins are placed in the hands of those able to create beauty from structure. To date, most examples of neat tricks and hacks have been demonstrated by structurists and coders. Designers have yet to make their mark. This needs to change.

participation

select a Design:

- Dazzling Beauty by Deny Sri Supriyono
- Dark Rose by Rose Fu
- Leggo My Ego by Jon Tan
- LuGoZee by Viallon Pierre-Antoine
- The Diary by Alexander Shabuniewicz
- Lonely Flower by Mitja Ribic
- Mozart by Andrew Brundle
- Organica Creative by Eduardo Cesario



v1.0

css zen dragen
the beauty of css design

Download the sample HTML FILE and CSS FILE

select a design

- Dazzling Beauty by Deny Sri Supriyono
- Dark Rose by Rose Fu
- Leggo My Ego by Jon Tan
- LuGoZee by Viallon Pierre-Antoine
- The Diary by Alexander Shabuniewicz
- Lonely Flower by Mitja Ribic
- Mozart by Andrew Brundle
- Organica Creativa by Eduardo Cesario

archives

- next designs »
- View All Designs

resources

- View This Design's CSS
- CSS Resources
- FAQ
- Submit a Design
- Translations

THE ROAD TO ENLIGHTENMENT

Littering a dark and dreary road lay the past relics of browser-specific tags, incompatible DOMs, and broken CSS support.

Today, we must clear the mind of past practices. Web enlightenment has been achieved thanks to the tireless efforts of folk like the W3C, WaSP and the major browser creators.

The css Zen Garden invites you to relax and meditate on the important lessons of the masters. Begin to see with clarity. Learn to use the (yet to be) time-honored techniques in new and invigorating fashion. Become one with the web.

SO WHAT IS THIS ABOUT?

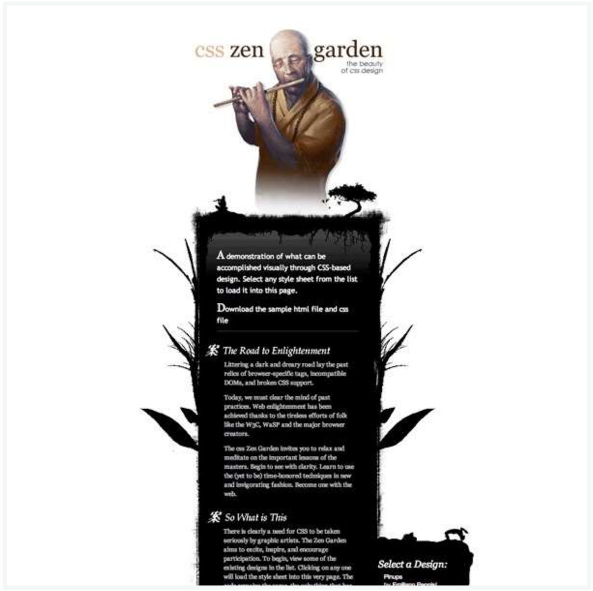
There is clearly a need for CSS to be taken seriously by graphic artists. The Zen Garden aims to excite, inspire, and encourage participation. To begin, view some of the existing designs in the list. Clicking on any one will load the style sheet into this very page. The code remains the same, the only thing that has changed is the external .css file. Yes, really.

CSS allows complete and total control over the style of a hypertext document. The only way this can be illustrated in a way that gets people excited is by demonstrating what it can truly be, once the reins are placed in the hands of those able to create beauty from structure. To date, most examples of neat tricks and hacks have been demonstrated by structurists and coders. Designers have yet to make their mark. This needs to change.

PARTICIPATION

Graphic artists only please. You are modifying this page, so strong CSS skills are necessary, but the example files are commented well enough that even CSS novices can use them as starting points. Please see the CSS Resource Guide for advanced tutorials and tips on working with CSS.

You may modify the style sheet in any way you wish, but not the



css zen garden
the beauty of css design

A demonstration of what can be accomplished visually through CSS-based design. Select any style sheet from the list to load it into this page.

Download the sample html file and css file

The Road to Enlightenment

Littering a dark and dreary road lay the past relics of browser-specific tags, incompatible DOMs, and broken CSS support.

Today, we must clear the mind of past practices. Web enlightenment has been achieved thanks to the tireless efforts of folk like the W3C, WaSP and the major browser creators.

The css Zen Garden invites you to relax and meditate on the important lessons of the masters. Begin to see with clarity. Learn to use the (yet to be) time-honored techniques in new and invigorating fashion. Become one with the web.

So What is This

There is clearly a need for CSS to be taken seriously by graphic artists. The Zen Garden aims to excite, inspire, and encourage participation. To begin, view some of the existing designs in the list. Clicking on any one will load the style sheet into this very page. The

Select a Design:

Pinups

(csszengarden.com)

How Style Sheets Work

1. Start with a marked up document (like HTML, but could be another XML markup language).
2. Write styles for how you want elements to look using CSS syntax.
3. Attach the styles to the document (there are a number of ways).
4. The browser uses your instructions when rendering the elements.

Three Ways to Use CSS:

1) Inline Style - CSS is placed directly into the HTML element.
2) Internal Style Sheet - CSS is placed into a separate area within the <head> section of a web page.
3) External Style Sheet - CSS is placed into a separate computer file and "connected" to a web page.

CSS Format Conflicts:

- It's possible for CSS formatting to be defined in all three locations at the same time.
- For example, a paragraph element could contain an inline style (color:red) but the internal style sheet (color:blue) and the external style sheet (color:green) give conflicting instructions to the web browser.
- Web browsers need a consistent way of "settling" this disagreement.
- Within this cascade of style declarations, the closest rule wins.
- An inline style overrules an internal style, which overrules an external style.

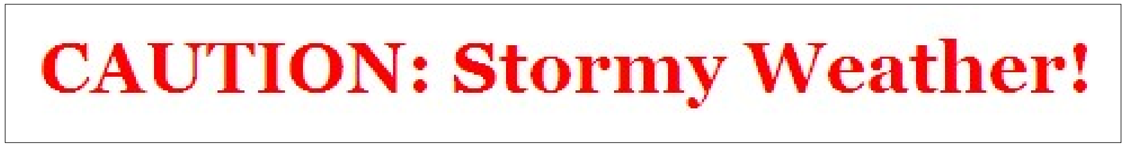
What is Meant by "Cascading"?

- We use the term cascading because there is an established order of priority to resolve these formatting conflicts:
  1) Inline style (highest priority)
  2) Internal style sheet (second priority)
  3) External style sheet (third priority)
  4) Web browser default (only if not defined elsewhere)

Example: Inline Style

```
<h2 style="font-family:georgia; color:red;">
CAUTION: Stormy Weather!
</h2>
```

PREVIEW:


CAUTION: Stormy Weather!

A semicolon must follow each style declaration.

Example: Internal Style Sheet

```
<head>
<style type="text/css">
h2 {font-family:georgia; color:red;}
</style>
</head>
```

- For internal style sheets, all formatting declarations are placed inside the <style> element within the <head> section of the document.
- An element is listed and all the styling information follows, surrounded by opening and closing curly brackets, **{ }**.
- A semicolon must still follow each style declaration.

Example: External Style Sheet

```
<head>
<link rel="stylesheet" type="text/css" href="style.css" />
</head>
```

style.css (separate file):

```
h2 {font-family:georgia; color:red;}
```

- For external style sheets, a <link> tag is placed at the beginning of the <head> section of the document specifying the external style sheet (with a .css extension) to be used for formatting.
- The external style sheet uses the same syntax as the internal style sheet when listing elements and their styling.
- Styles declared in an external style sheet will affect all matching elements on all web pages that link to the stylesheet.
- In this example, all <h2> elements on all pages using this style sheet will be displayed in Georgia font and in red color.

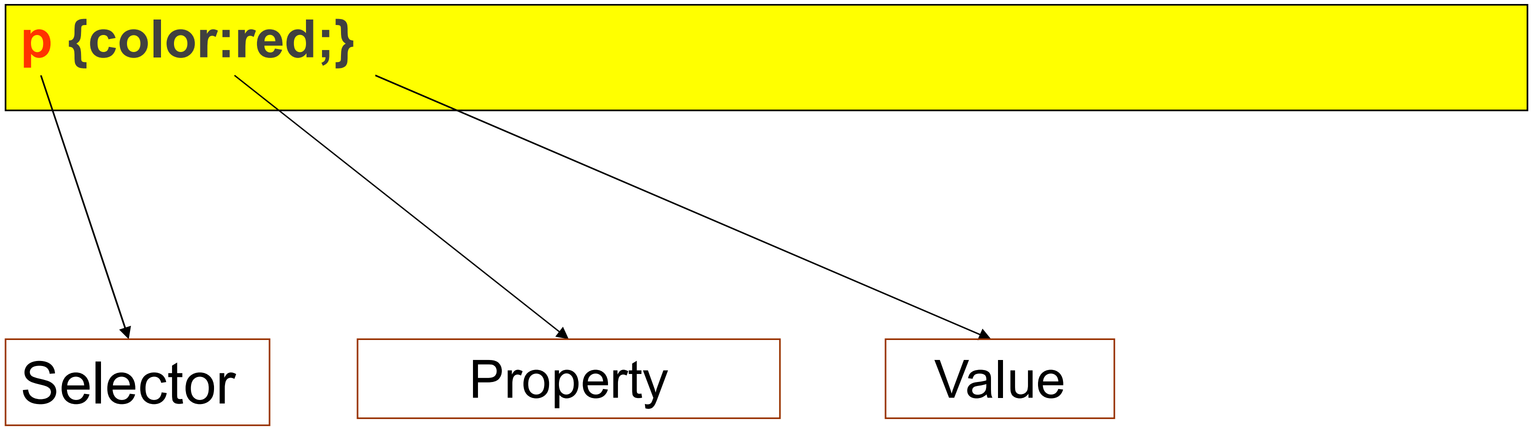
Internal vs. External Style Sheets

- Internal style sheets are appropriate for very small sites, especially those that have just one page.
- Internal style sheets might also make sense when each page of a site needs to have a completely different look.
- External style sheets are better for multi-page websites that need to have a uniform look and feel to all pages.
- External style sheets not only make for faster-loading sites (less redundant code) but also allow designers to make site-wide changes quickly and easily.

CSS Terminology and Syntax:

Correct syntax:

selector {property:value;}



```
p {color:red;}
```

Some Examples

Background Picture

```
body {
   background-image:url('picture.gif');
    background-repeat:repeat-x;
   background-color:red;
  }
```

Paragraph Properties

```
p {
   color:red;
   font-style:italic;
   text-align:center;
}
```

Style Rules

Each rule *selects* an element and *declares* how it should display.

```
h1 { color: green; }
```

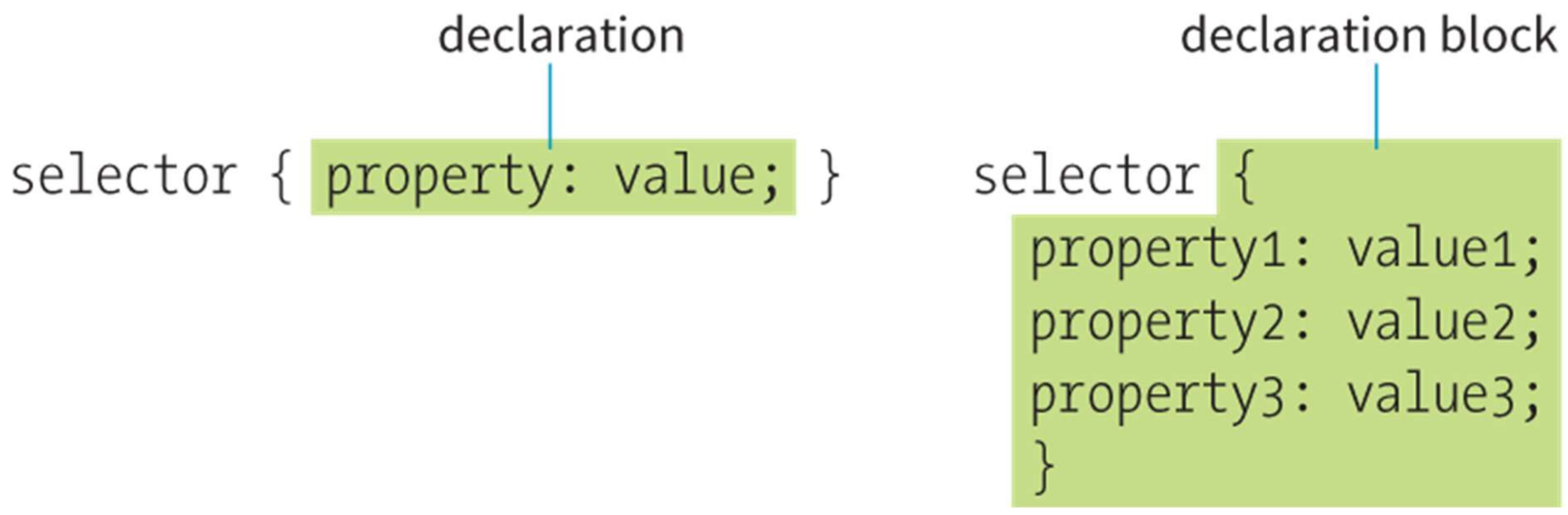
This rule selects all `h1` elements and declares that they should be green.

```
strong { color: red; font-style: italic; }
```

This rule selects all `strong` inline elements and declares that they should be red and in an italic font.

Style Rule Structure

- A style rule is made up of a **selector** a **declaration**.
- The declaration is one or more **property / value** pairs.



```
selector { property: value; }
```

```
selector {
  property1: value1;
  property2: value2;
  property3: value3;
  }
```

Selectors

There are many types of selectors. Here are just two examples:

```
p {property: value;}
```

Element type selector: Selects all elements of this type (**`p`**) in the document.

```
#intro {property: value}
```

ID selector (indicated by the # symbol) selects by ID value. In the example, an element with an id of “intro” would be selected.

Declarations

The **declaration** is made up of a **property/value pair** contained in curly brackets `{ }`:

```
selector { property: value; }
```

Example

```
h2 { color: red;
     font-size: 2em;
     margin-left: 30px;
     opacity: .5;
   }
```

Declarations (cont’d)

- End each declaration with a semicolon to keep it separate from the next declaration.
- White space is ignored, so you can stack declarations to make them easier to read.
- **Properties** are defined in the CSS specifications.
- **Values** are dependent on the type of property:
    - Measurements
    - Keywords
    - Color values
    - More

CSS Comments

```
/* comment goes here */
```

- Content between `/*` and `*/` will be ignored by the browser.
- Useful for leaving notes or section labels in the style sheet.
- Can be used within rules to temporarily hide style declarations in the design process.

Adding Styles to the Document

There are three ways to attach a style sheet to a document:

External style sheets
A separate, text-only *.css* file associated with the document with the `link` element or `@import` rule

Embedded style sheets
Styles are listed in the `head` of the HTML document in the `style` element.

Inline styles
Properties and values are added to an individual element with the `style` attribute.

External Style Sheets

The style rules are saved in a separate text-only *.css* file and attached via **`link`** or **`@import`**.

Via **`link`** element in HTML:

```
<head>
  <title>Titles are require</title>
  <link rel="stylesheet" href="/path/example.css">
</head>
```

Via **`@import`** rule in a style sheet:

```
<head>
  <title>Titles are required</title>
  <style>
    @import url("/path/example.css");
    p {font-face: Verdana;}
  </style>
</head>
```

Embedded Style Sheets

Embedded style sheets are placed in the `head` of the document via the `style` element:

```
<head>
  <title>Titles are required</title>
  <style>
    /* style rules go here */
  </style>
</head>
```

Inline Styles

Apply a style declaration to a single element with the **`style`** *attribute*:

```
<p style="font-size: large;">Paragraph text...</p>
```

To add multiple properties, separate them with semicolons:

```
<h3 style="color: red; margin-top: 30px;">Intro</h3>
```

Document Structure

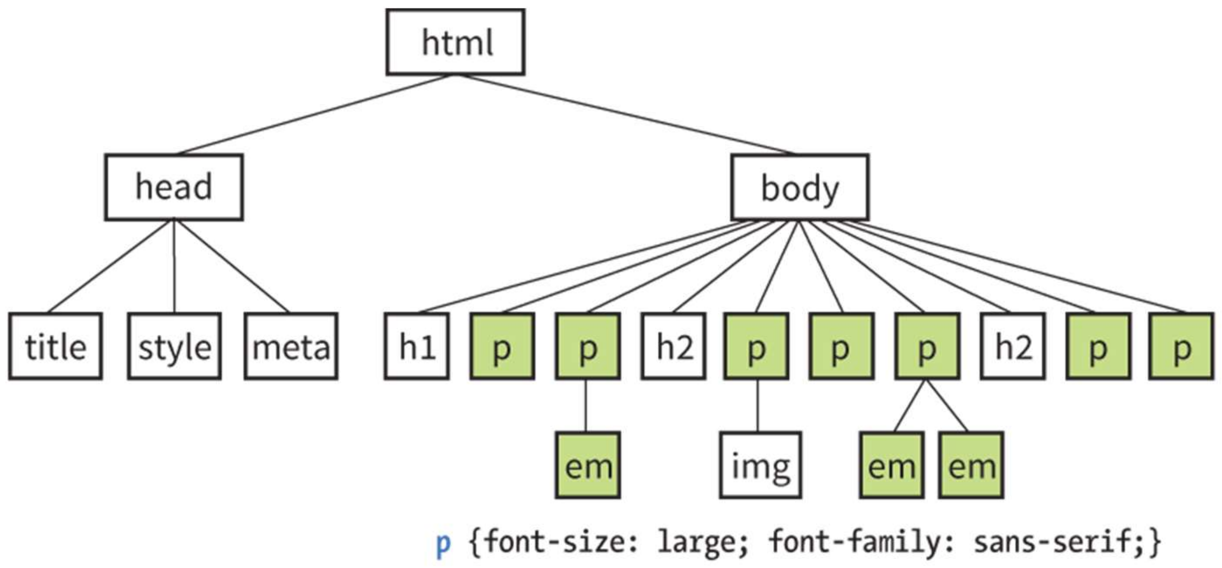
Documents have an implicit structure.

We give certain relationships names, as if they're a family:

- All the elements contained in a given element are its **descendents.**
- An element that is directly contained within another element is the **child** of that element.
- The containing element is the **parent** of the contained element.
- Two elements with the same parent are **siblings**.

Inheritance

- Many properties applied to elements are passed down to the elements they contain. This is called **inheritance**.
- For example, applying a sans-serif font to a **`p`** element causes the **`em`** element it contains to be sans-serif as well:



```
p {font-size: large; font-family: sans-serif;}
```

Inheritance (cont’d)

- **Some properties inherit; others do not.**
  Properties related to text usually inherit; properties related to layout generally don’t.
- Styles explicitly applied to specific elements override inherited styles.
- You’ll learn to use inheritance strategically to keep your style rules simple.

The Cascade

- The **cascade** refers to the system for resolving conflicts when several styles apply to the same element.
- Style information is passed down (it "cascades" down) until overwritten by a style rule with more **weight**.
- Weight is considered based on:
    - **Priority** of style rule source
    - **Specificity** of the selector
    - **Rule order**

The Cascade: Priority

- We use the term cascading because there is an established order of priority to resolve these formatting conflicts:
  1) Inline style (highest priority)
  2) Internal style sheet (second priority)
  3) External style sheet (third priority)
  4) Web browser default (only if not defined elsewhere)

The Cascade: Specificity

- When two rules in a single style sheet conflict, the **type of selector is used to determine which rule has more weight.**
- For example, ID selectors are more specific than general element selectors.

NOTE: Specificity will be discussed once we have covered more selector types.

The Cascade: Rule Order

- When two rules have equal weight, rule order is used. **Whichever rule appears last "wins."**

  ```
  <style>
    p {color: red;}
    p {color: blue;}
    p {color: green;}
  </style>
  ```

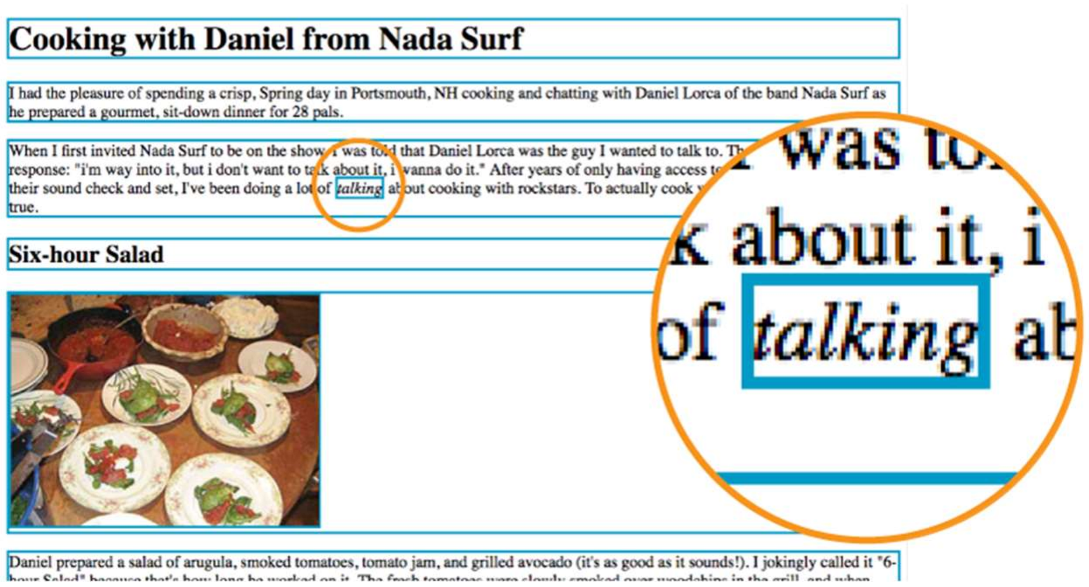
  In this example, paragraphs would be green.

- Styles may come in from external style sheets, embedded style rules, and inline styles. The style rule that gets parsed last (the one closest to the content) will apply.

The Box Model

Browsers see every element on the page as being contained in a little rectangular box. **Block elements and inline elements participate in the box model.**

In this example, a blue border is added to all elements.



Cooking with Daniel from Nada Surf

I had the pleasure of spending a crisp, Spring day in Portsmouth, NH cooking and chatting with Daniel Lorca of the band Nada Surf as he prepared a gourmet, sit-down dinner for 28 pals.

When I first invited Nada Surf to be on the show, I was told that Daniel Lorca was the guy I wanted to talk to. Th
response: "i'm way into it, but i don't want to talk about it, i wanna do it." After years of only having access to
their sound check and set, I've been doing a lot of *talking* about cooking with rockstars. To actually cook w
true.

Six-hour Salad

Daniel prepared a salad of arugula, smoked tomatoes, tomato jam, and grilled avocado (it's as good as it sounds!). I jokingly called it "6-

The Box Model (cont’d)

- The **box model** is the foundation of CSS page layout.
- Apply properties such as **borders**, **margins**, **padding**, and **backgrounds** to element boxes.
- Position, move, grow, and shrink boxes to create fixed or flexible page layouts.

CSS Units of Measurement

CSS provides a variety ways to specify measurements:

Absolute units
Have predefined meanings or real-world equivalents

Relative units
Based on the size of something else, such as the default text size or the size of the parent element

Percentages
Calculated relative to another value, such as the size of the parent element

Absolute Units

With the exception of pixels, absolute units are not appropriate for web design:

`px`	pixel
`in`	inches
`mm`	millimeters
`cm`	centimeters
`q`	1/4 millimeter
`pt`	points (1/72 inch)
`pc`	pica (1 pica = 12 points = 1/6 inch)

Relative Units

Relative units are based on the size of something else:

`em`	a unit equal to the current font size
`ex`	x-height, equal to the height of a lowercase *x*
`rem`	root em, equal to the font size of the `html` element
`ch`	zero width, equal to the width of a zero (0)
`vw`	viewport width unit (equal to 1/100 of viewport width)
`vh`	viewport height unit (1/100 of viewport height)
`vmin`	viewport minimum unit (value of `vh` or `vw`, whichever is smaller)
`vmax`	viewport maximum unit (value of `vh` or `vw`, whichever is larger)

RELATIVE UNITS

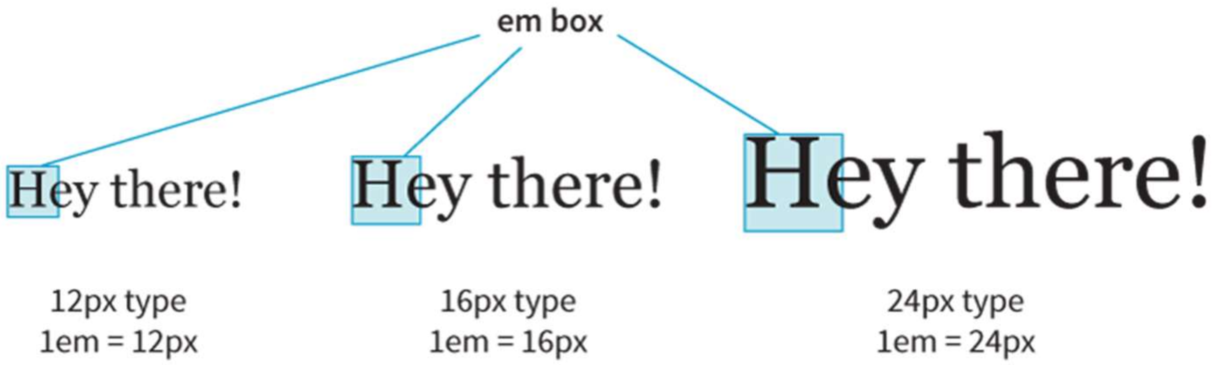
The rem Unit

- The **rem** (**root em**) unit is based on the font size of the **`html`** element, whatever that happens to be.
- **Default** in modern browsers: Root font size is 16 pixels, so a **rem = a 16-pixel unit**.
- If the root font size of the document changes, so does the size of a rem (and that's good for keeping elements proportional).

RELATIVE UNITS

The em Unit

- The **em** unit is traditionally based on the width of a capital letter *M* in the font.
- When the font size is 16 pixels,1em = 16 pixels, 2em = 32 pixels, and so on.



NOTE: Because they're based on the font size of the current element, the size of an em may not be consistent across a page.

RELATIVE UNITS

Viewport Percentage Lengths (vw/vh)

Viewport width (**`vw`**) and viewport height (**`vh`**) units are relative to the size of the viewport (browser window):

`vh` = 1/100th width of viewport

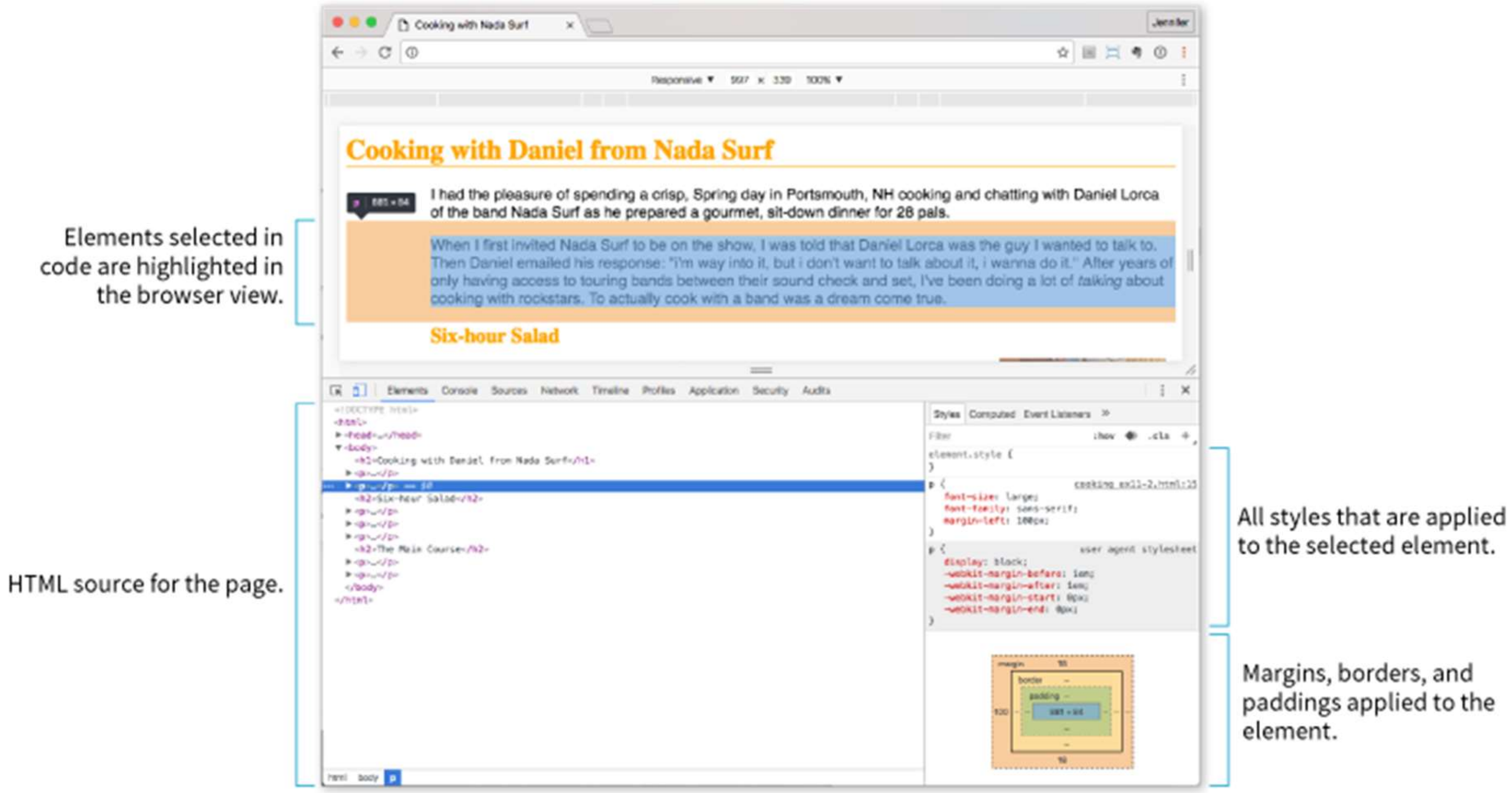
`vh` = 1/100th height of viewport

They're useful for making an element fill the viewport or a specified percentage of it. This image will be 50% the width and height of the viewport:

```
img { width: 50vw; height: 50vh; }
```

Browser Developer Tools

Chrome DevTools (View > Developer > Developer Tools)


Elements selected in code are highlighted in the browser view.
HTML source for the page.
All styles that are applied to the selected element.
Margins, borders, and paddings applied to the element.
Cooking with Nada Surf
Responsive
Elements
Console
Sources
Network
Timeline
Profiles
Application
Security
Audits
Styles
Computed
Event Listeners
margin
border
padding

Cooking with Daniel from Nada Surf

I had the pleasure of spending a crisp, Spring day in Portsmouth, NH cooking and chatting with Daniel Lorca of the band Nada Surf as he prepared a gourmet, sit-down dinner for 28 pals.

When I first invited Nada Surf to be on the show, I was told that Daniel Lorca was the guy I wanted to talk to. Then Daniel emailed his response: "i'm way into it, but i don't want to talk about it, i wanna do it." After years of only having access to touring bands between their sound check and set, I've been doing a lot of *talking* about cooking with rockstars. To actually cook with a band was a dream come true.

Six-hour Salad

```
<!DOCTYPE html>
<html>
▶<head>…</head>
▼<body>
    <h1>Cooking with Daniel from Nada Surf</h1>
  ▶<p>…</p>
  ▶<p>…</p> == $0
    <h2>Six-hour Salad</h2>
  ▶<p>…</p>
  ▶<p>…</p>
  ▶<p>…</p>
    <h2>The Main Course</h2>
  ▶<p>…</p>
  ▶<p>…</p>
  </body>
</html>
```

```
element.style {
}
p {                                    cooking_ex11-2.html:15
  font-size: large;
  font-family: sans-serif;
  margin-left: 100px;
}
p {                                    user agent stylesheet
  display: block;
  -webkit-margin-before: 1em;
  -webkit-margin-after: 1em;
  -webkit-margin-start: 0px;
  -webkit-margin-end: 0px;
}
```

Firefox, Safari, Opera, and Microsoft Edge also have developer tools.

Formatting Text

- Font-related properties
- Text line settings
- Various text effects
- List style properties
- ID, class, and descendent selectors
- Specificity

Designing Text

Styling text on the web is tricky because you don’t have control over how the text displays for the user:

- They may not have the font you specify.
- They may have their text set larger or smaller than you designed it.

Best practices allow for flexibility in text specification.

Typesetting Terminology

- A **typeface** is a set of characters with a single design (example: Garamond).
- A **font** is a particular variation of the typeface with a specific weight, slant, or ornamentation (example: Garamond Bold Italic).
- In traditional metal type, each size was a separate font (example: 12-point Garamond Bold Italic).
- On a computer, fonts are generally stored in individual font files.

CSS Basic Font Properties

CSS font properties deal with specifying the shapes of the characters themselves:

- `font-family`
- `font-size`
- `font-weight`
- `font-style`
- `font-variant`
- `font` (a shorthand that includes settings for all of the above)

Specifying the Font Family

`font-family`

Values: One or more font family names, separated by commas

Example:

```
body { font-family: Arial; }
var { font-family: Courier, monospace; }
```

Specifying the Font Family (cont’d)

- Font names must be capitalized (except generic font families).
- Use commas to separate multiple font names.
- If the name has a character space, it must appear within quotation marks:

```
p { font-family: "Duru Sans", Verdana, sans-serif; }
```

Using Fonts on the Web

- The font must be available on the user’s machine for it to display.
- The best practice is to provide a list of options. The browser uses the first one that is available.
- Start with the font you want and then provide backup options ending with a generic font family, as shown here:

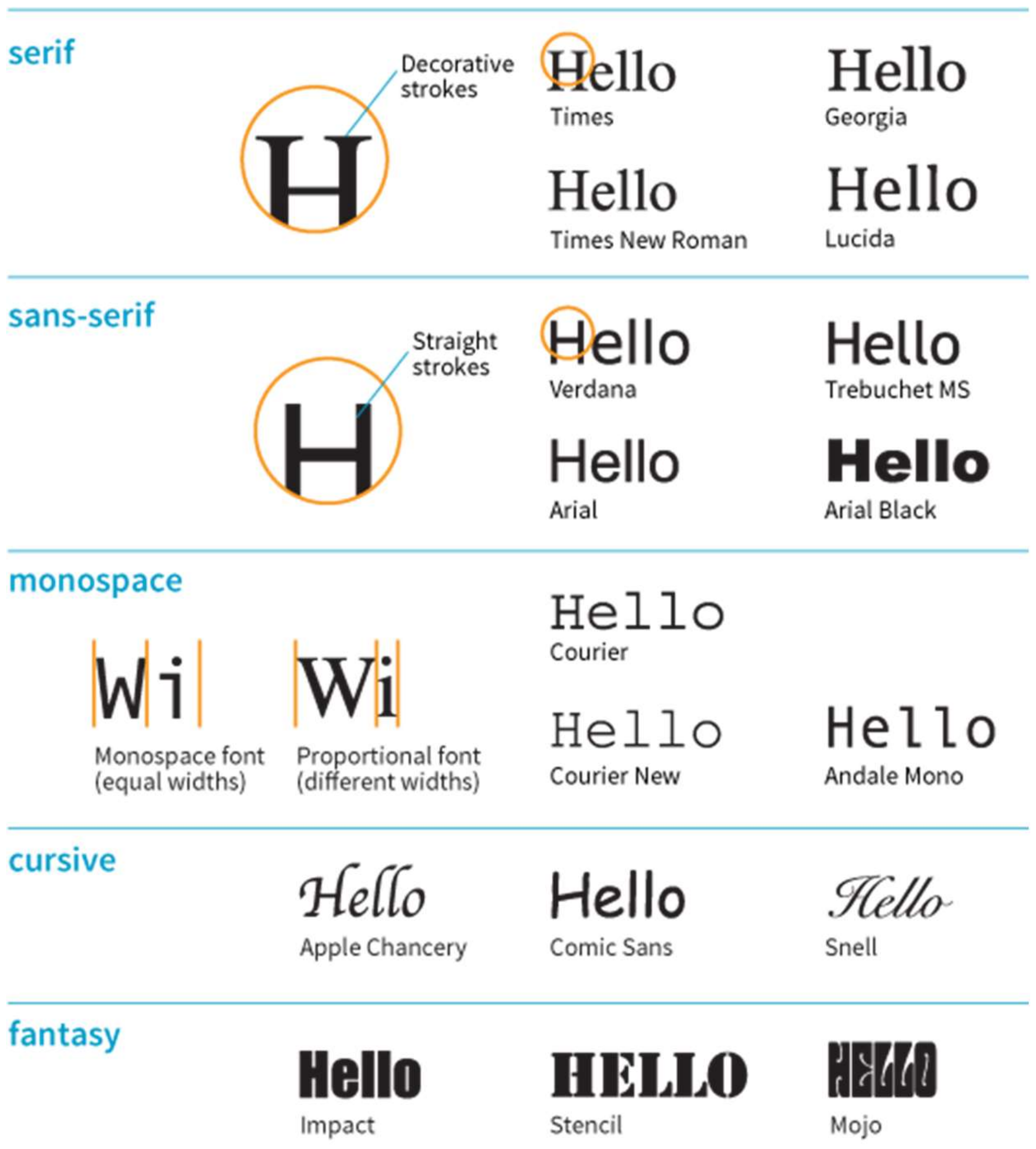
```
p { font-family: "Duru Sans", Verdana, sans-serif; }
```

- You can also download a web font with the page, but it adds to the download and display time.

Generic Font Families

- Generic font families instruct the browser to use an available font from one of five stylistic categories: **serif**, **sans-serif**, **monospace**, **cursive**, **fantasy**
- Generic font families are often used as the last backup option.

Generic Font Families (cont'd)



Specifying Font Size

`font-size`

Values:

- CSS length units
- Percentage value
- Absolute keywords (`xx-small`, `x-small`, `small`, `medium`, `large`, `x-large`, `xx-large`)
- Relative keywords (`larger`, `smaller`)

Example:

```
h1 { font-size: 2.5rem; }
```

Specifying Font Size (cont’d)

Most common sizing methods:

- **rem** and **em** units
- **percentages** (based on the inherited font size for that element)
- **pixels** (px) can be used, but they’re not flexible.

Font Size: rem Units

- The **rem** (**root em**) is equal to the font size of the **`html`** (root) element.
- In browsers, the **default** root size is 16 pixels, so:
  1 rem = 16 pixels.
- If the font size of the root is changed, rem measurements change too.
- !!! Old browsers *do not* support rem units (IE8 and earlier).

Font Size: em Units

- The **em unit** is based on the current font size of the element.
- The default font size is 16 pixels. **By default, 1em = 16 pixels.**
- But if you change the font size of the element, the size of its em unit changes too.
- Ems may be different sizes in different parts of the document and may compound larger or smaller when elements are nested.
- This makes ems a little tricky to use, although they are better supported than rem units.

Font Weight (Boldness)

`font-weight`

Values: `normal`, `bold`, `bolder`, `lighter`, `100`, `200`, `300`, `400`, `500`, `600`, `700`, `800`, `900`

Example:

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

span.new { font-weight: bold; }
```

- Most common values are normal and bold.
- Numerical values are useful when using a font with multiple weights.

Font Style (Italics)

`font-style`

Values: `normal`, `italic`, `oblique`

Example:

```
cite { font-style: italic; }
```

- Makes text italic, normal, or oblique (slanted, but generally the same as italics).

Small Caps

`font-variant`

Values (in CSS2.1): `normal`, `small-caps`

Example:

```
abbr { font-variant: small-caps; }
```

- Small caps are a separate font design that uses small uppercase characters in place of lowercase letters.
- They help acronyms and other strings of capital letters blend in with the weight of the surrounding text.

Condensed and Extended Text

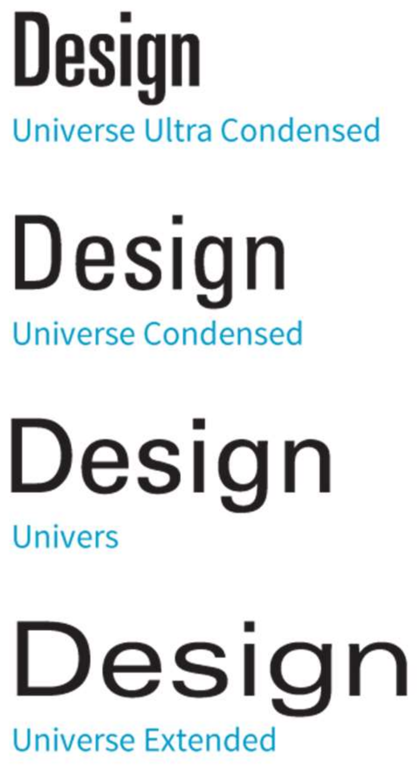
`font-stretch`

Values (in CSS2.1): `normal`, `ultra-condensed`, `extra-condensed`, `condensed`, `semi-condensed`, `semi-expanded`, `expanded`, `extra-expanded`, `ultra-expanded`

Example:

```
abbr { font-variant: small-caps; }
```

- Tells the browser to select a normal, condensed, or extended font variation from a typeface if it is available



The Shortcut font Property

`font`

Values (in CSS2.1): A list of values for all the individual properties, in this order:

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

At minimum, it must contain **`font-size`** and **`font-family`**, in that order. Other values are optional and may appear in any order prior to **`font-size`**.

Example:

```
 p { font: 1em sans-serif; }
 h3 { font: oblique bold small-caps 1.5em Verdana, sans-serif;
}
```

Advanced Typography

The **CSS3 Font Module** offers properties for fine-tuned typography control, including:

- Ligatures
- Superscript and subscript
- Alternate characters (such as a swash design for an *S*)
- Proportional font sizing using x-height
- Kerning
- OpenType font features

Text Line Treatments

Some properties control whole lines of text:

- Line height (`line-height`)
- Indents (`text-indent`)
- Horizontal alignment (`text-align`)

Line Height

`line-height`

Values: *Number*, *length*, *percentage*, `normal`

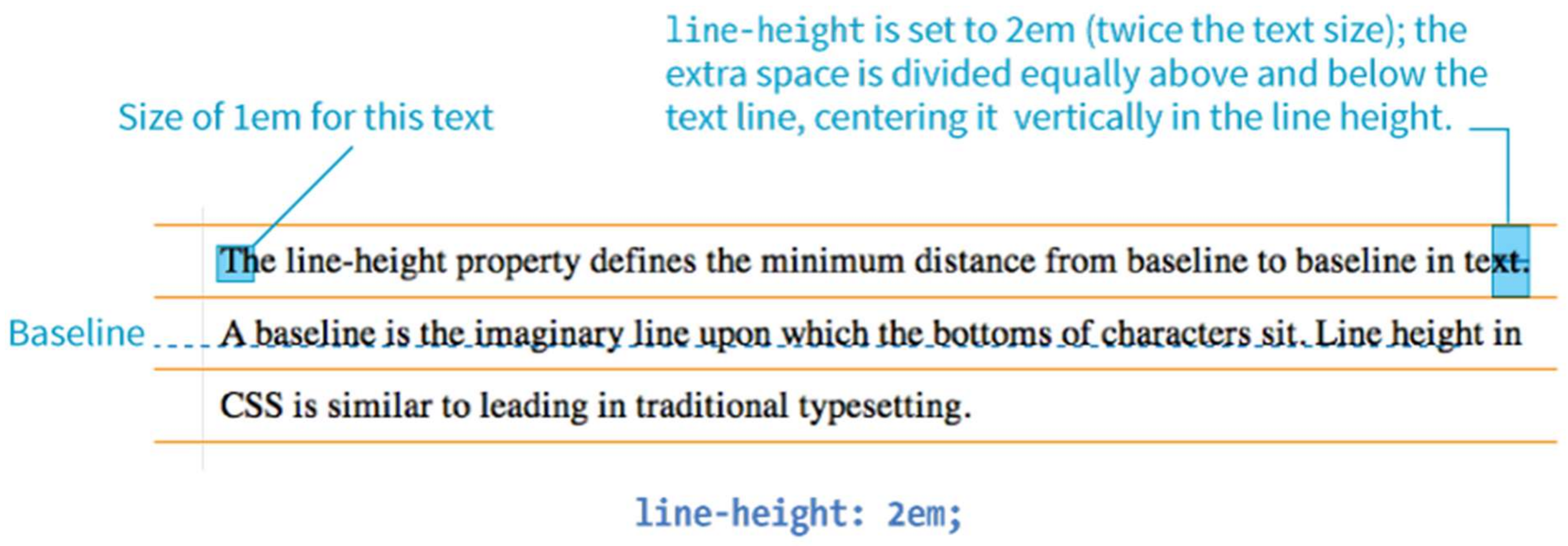
Example:

```
p { line-height: 1.4em; }
```

- Line height defines the minimum distance from baseline to baseline in text.

Line Height (cont’d.)

- The **baseline** is the imaginary line upon which the bottoms of characters sit.
- If a large character or image is on a line, the line height expands to accommodate it.



Size of 1em for this text
line-height is set to 2em (twice the text size); the extra space is divided equally above and below the text line, centering it vertically in the line height.
Baseline
line-height: 2em;

The line-height property defines the minimum distance from baseline to baseline in text. A baseline is the imaginary line upon which the bottoms of characters sit. Line height in CSS is similar to leading in traditional typesetting.

Indents

`text-indent`

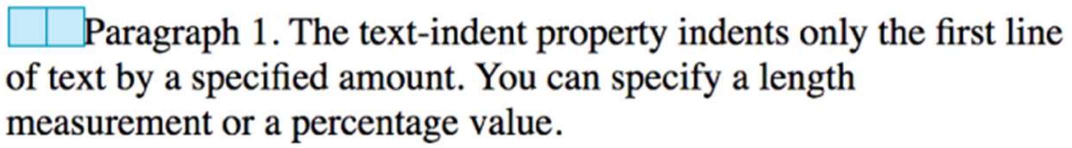
Values: *Length*, *percentage*

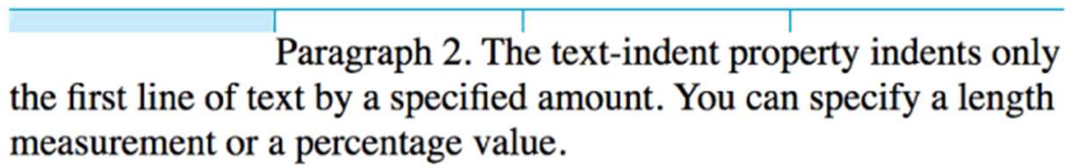
Examples:

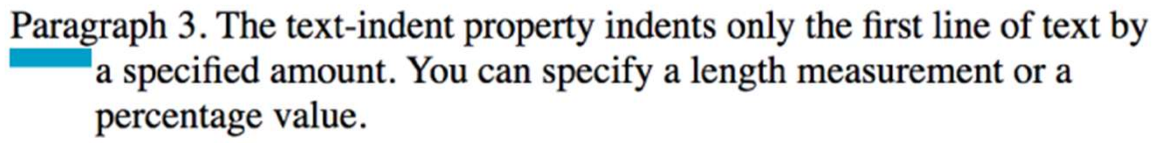
```
p {text-ident: 2em;}
```

```
p {text-ident: 25%;}
```

```
p {text-ident: -35px;}
```


Paragraph 1. The text-indent property indents only the first line of text by a specified amount. You can specify a length measurement or a percentage value.


Paragraph 2. The text-indent property indents only the first line of text by a specified amount. You can specify a length measurement or a percentage value.

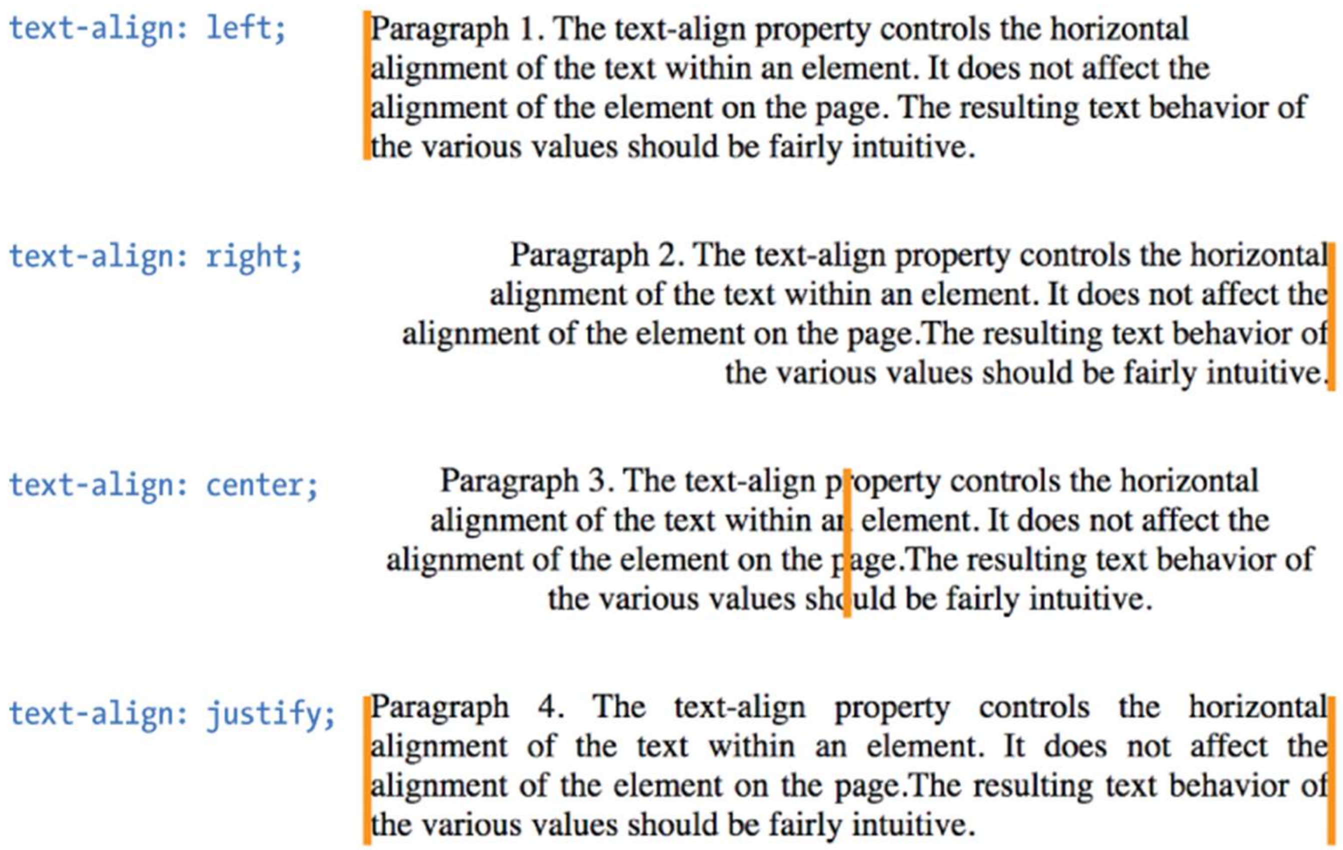

Paragraph 3. The text-indent property indents only the first line of text by a specified amount. You can specify a length measurement or a percentage value.

Horizontal Text Alignment

`text-align`

Values: `left, right, center, justify, start, end`

Examples:



```
text-align: left;
```

Paragraph 1. The text-align property controls the horizontal alignment of the text within an element. It does not affect the alignment of the element on the page. The resulting text behavior of the various values should be fairly intuitive.

```
text-align: right;
```

Paragraph 2. The text-align property controls the horizontal alignment of the text within an element. It does not affect the alignment of the element on the page.The resulting text behavior of the various values should be fairly intuitive.

```
text-align: center;
```

Paragraph 3. The text-align property controls the horizontal alignment of the text within an element. It does not affect the alignment of the element on the page.The resulting text behavior of the various values should be fairly intuitive.

```
text-align: justify;
```

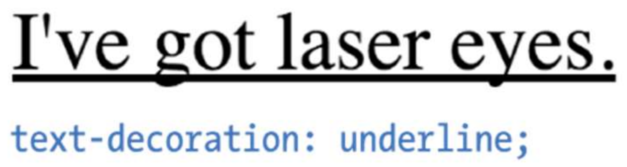
Paragraph 4. The text-align property controls the horizontal alignment of the text within an element. It does not affect the alignment of the element on the page.The resulting text behavior of the various values should be fairly intuitive.

Underlines (Text Decoration)

`text-decoration`

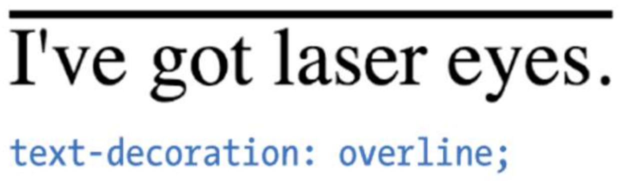
Values: `none`, `underline`, `overline`, `line-through`, `blink`

Examples:



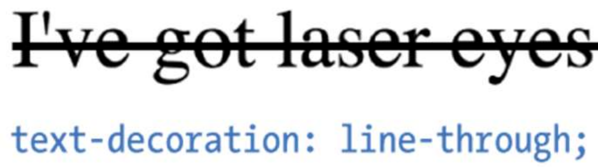
I've got laser eyes.

```
text-decoration: underline;
```



I've got laser eyes.

```
text-decoration: overline;
```



~~I've got laser eyes.~~

```
text-decoration: line-through;
```

NOTE: `text-decoration` is often used to turn **off** underlines under links:

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

Text Decoration Tips

- If you turn off underlines under links, be sure there is another visual cue to compensate.
- Underlining text that is not a link may be misleading. Consider italics instead.
- Don’t use `blink`. Browsers don’t support it anyway.

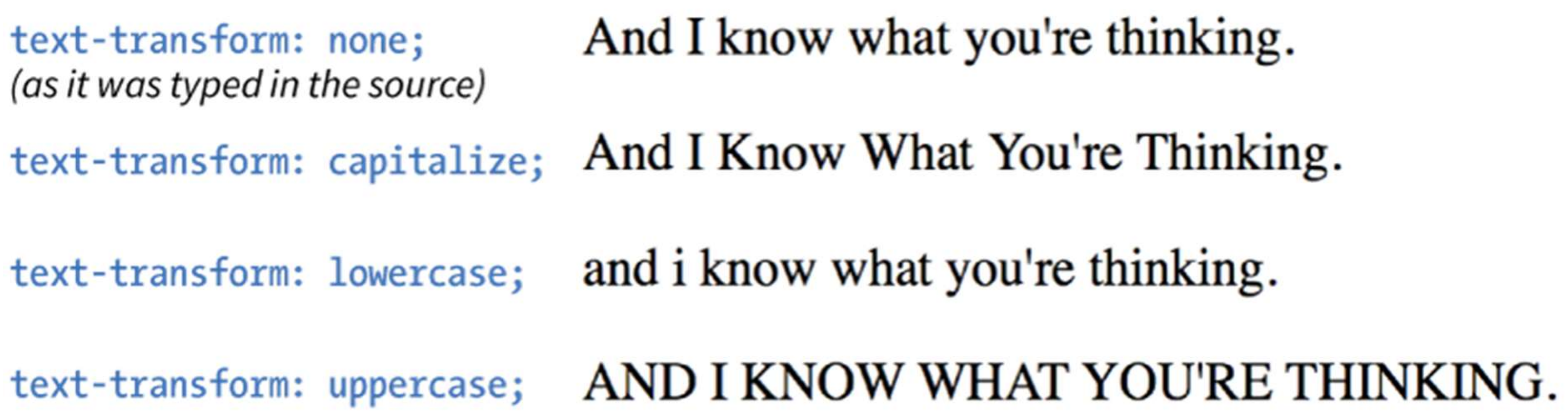
Capitalization

`text-transform`

Values:

`none, capitalize, lowercase, uppercase, full-width`

Examples:



`text-transform: none;` *(as it was typed in the source)*	And I know what you're thinking.
`text-transform: capitalize;`	And I Know What You're Thinking.
`text-transform: lowercase;`	and i know what you're thinking.
`text-transform: uppercase;`	AND I KNOW WHAT YOU'RE THINKING.

Spacing

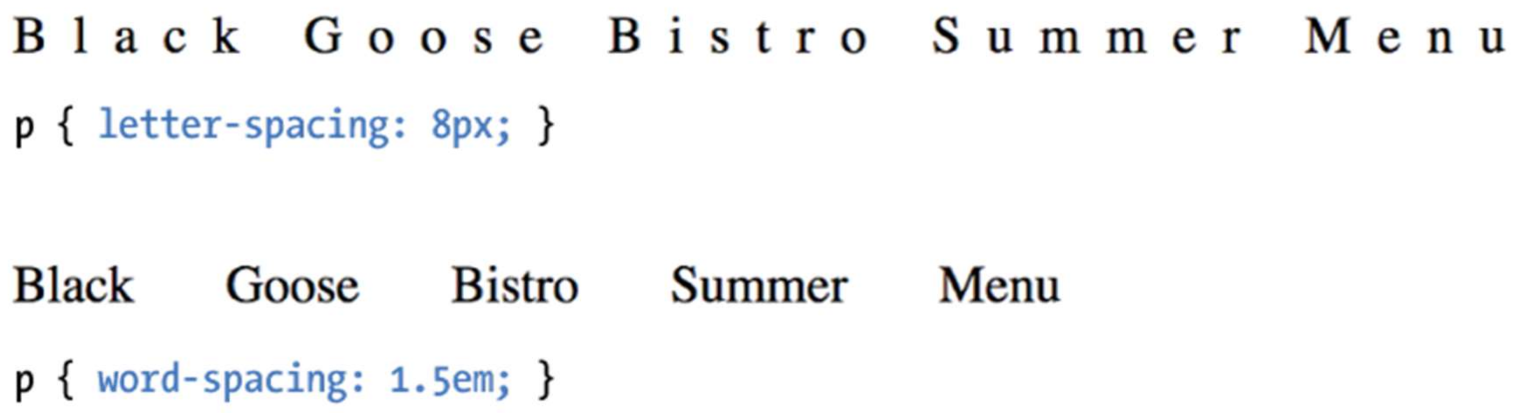
`letter-spacing`

Values: *length*, `normal`

`word-spacing`

Values: *length*, `normal`

Examples:



B l a c k   G o o s e   B i s t r o   S u m m e r   M e n u

```
p { letter-spacing: 8px; }
```

Black   Goose   Bistro   Summer   Menu

```
p { word-spacing: 1.5em; }
```

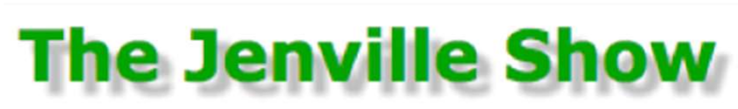
Text Shadow

`text-shadow`

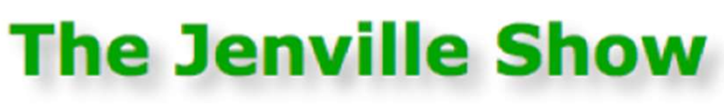
Values: *'horizontal-offset' 'vertical-offset' 'blur-radius' 'color'*, `none`

The value is two offset measurements, an optional blur radius, and a color value (with no commas between).

Example:



```
text-shadow: .2em .2em .1em silver;
```



```
text-shadow: .2em .2em .3em silver;
```

List Style Properties

There are three properties for affecting the display of lists:

- **`list-style-type`**
  Chooses the type of list marker
- **`list-style-position`**
  Sets the position of the marker relative to the list element box
- **`list-style-image`**
  Allows you to specify your own image for use as a bullet

LIST STYLES

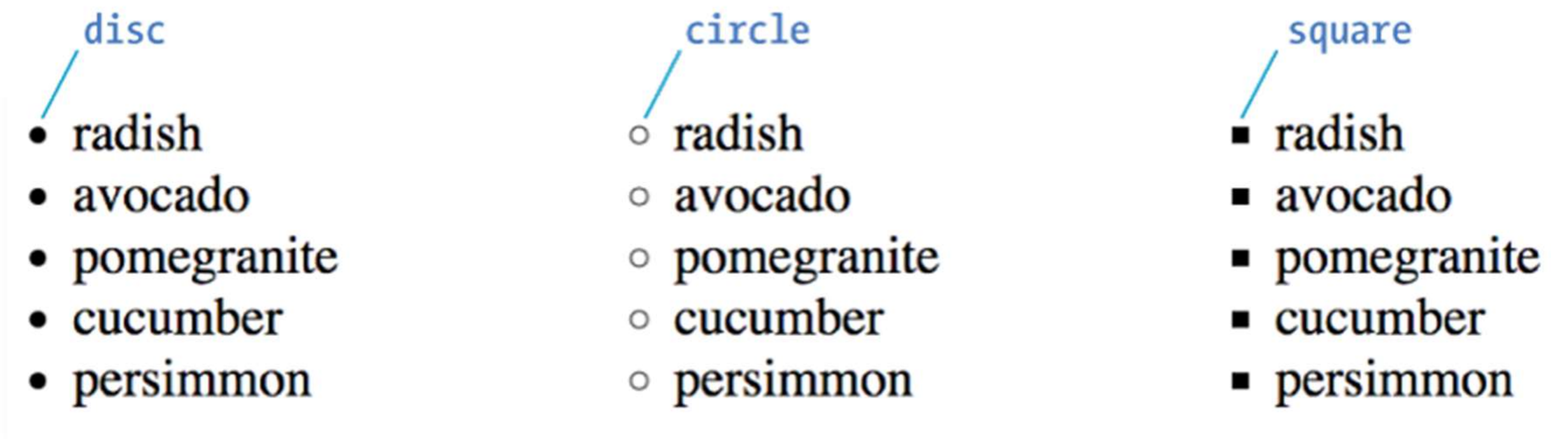
Choosing a Marker

`list-style-type`

Values:

`none, disc, circle, square, decimal, decimal-leading-zero, lower-alpha, upper-alpha, lower-latin, upper-latin, lower-roman, upper-roman, lower-greek`

Unordered lists: `ul { list-style-type: keyword; }`



● radish
● avocado
● pomegranite
● cucumber
● persimmon

○ radish
○ avocado
○ pomegranite
○ cucumber
○ persimmon

■ radish
■ avocado
■ pomegranite
■ cucumber
■ persimmon

LIST STYLES

Choosing a Marker (cont'd)

Ordered lists: `ol { list-style-type: keyword; }`

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...

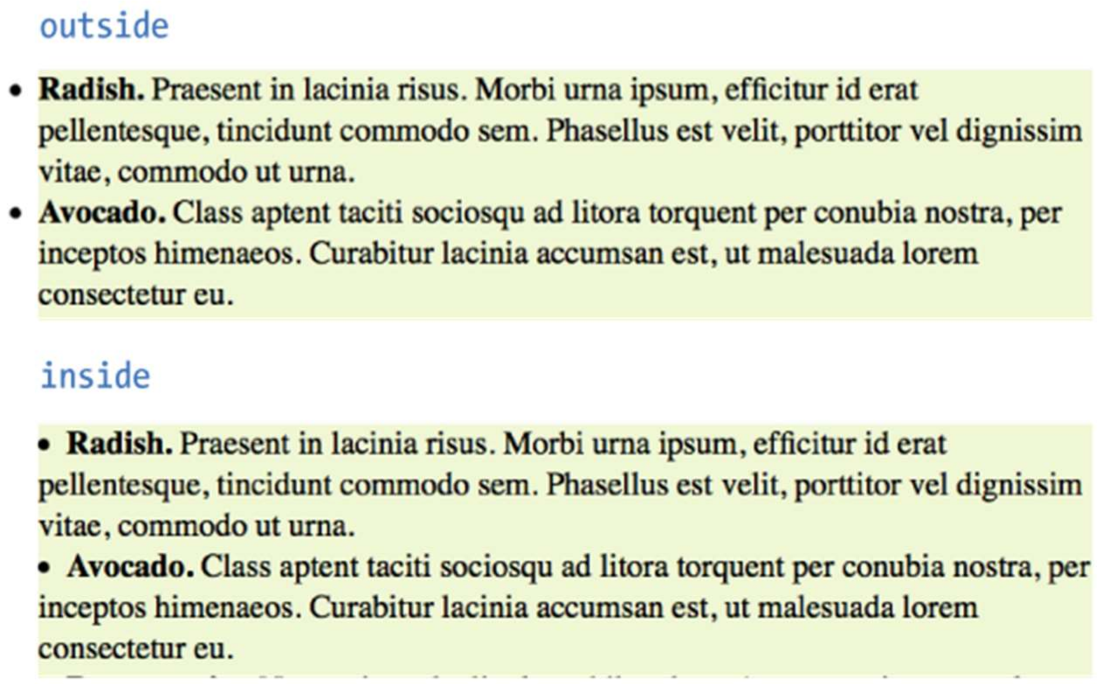
LIST STYLES

Marker Position

`list-style-position`

Values: `inside`, `outside`, `hanging`

Positions the marker relative to the content area:



`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.

`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.

LIST STYLES

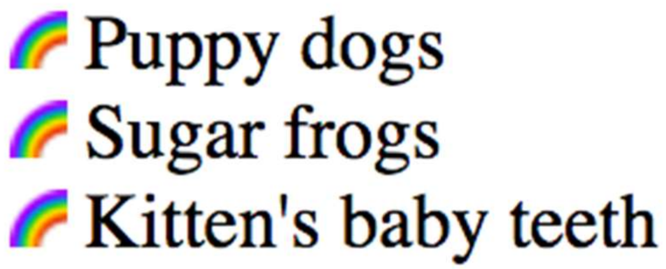
Custom Bullets

`list-style-image`

Values: `url(`*location*`)`, `none`

Example:

```
ul {
      list-style-type: disc;
      list-style-image: url(/images/rainbow.gif);
      list-style-position: outside;
   }
```



Puppy dogs
Sugar frogs
Kitten's baby teeth

More Selector Types

- Descendent selectors
- ID selectors
- Class selectors
- Universal selector

Descendent Selectors

A **descendent selector** targets elements contained in another element.

It’s a kind of **contextual selector** (it selects based on relationship to another element).

It’s indicated in **a list separated by a character space**.

```
ol a {font-weight: bold;}
```

(only the links (`a`) in ordered lists (`ol`) would be bold)

```
h1 em {color: red;}
```

(only emphasized text in h1s would be red)

Descendent Selectors (cont'd)

They can appear as part of a grouped selector:

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

(only emphasized text in `h1`, `h2`, and `h3` elements)

They can be several layers deep:

```
ol a em {font-variant: small-caps;}
```

(only emphasized text in links in ordered lists)

ID Selectors

ID selectors (indicated by a # symbol) target elements based on the value of their ID attributes:

```
<li id="primary">Primary color t-shirt</li>
```

To target just that item:

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

To omit the element name:

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

It can be used as part of a compound or contextual selector:

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

Class Selectors

Class selectors (indicated by a `.` symbol) select elements based on the value of their class attributes:

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

(All paragraphs with the class name "special" would be orange.)

To target *all* element types that share a class name, omit the element name in the selector:

```
.hilight { background-color: yellow;}
```

(All elements with the class “hilight” would have a yellow background.)

Universal Selector

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

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

(puts a 1-pixel gray border around every element in the document)

Can be used as part of contextual selectors:

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

(selects all elements contained within an element with the ID **`intro`**)

Specificity Basics

Specificity refers to a system for sorting out which selectors have more weight when resolving style rule conflicts.

More specific selectors have more weight.

In simplified terms, it works like this:

- **Inline styles** with the `style` attribute are more specific than (and will override…)
- **ID selectors**, which are more specific than (and will override…)
- **Class selectors**, which are more specific than (and will override…)
- **Individual element selectors**

Calculating Specificity

There is a system used to calculate specificity. Start by drawing three boxes:

[ ] [ ] [ ]

For each style rule:

1. Count the **IDs** in the selector and put that number in the first box.

2. Count the **class** and pseudo-class selectors and put the number in the second box.

3. Count the **element** names and put the number in the third box

[ID] [class] [elements]

4. **The first box that is not a tie determines which selector wins.**

Calculating Specificity (cont’d)

Example:

```
h1 { color: red;}                 [0] [0] [1]

h1.special { color: lime; }   [0] [1] [1]
```

The second one has a class selector and the first one doesn’t, therefore the second one is more specific and has more weight.

The lime color applies to `h1`s when they have the class name “special.”

Using Specificity

Use specificity strategically to take advantage of overrides:

```
p { line-height: 1.2em; }                [0]  [0]  [1]
```

(sets the line-height for all paragraphs)

```
blockquote p { line-height: 1em; }  [0]  [0]  [2]
```

(more specific selector changes line-height when the paragraph is in a blockquote)

```
p.intro { line-height: 2em; }            [0]  [1]  [1]
```

(paragraphs with the class “intro” have a line-height of 2em, even when they’re in a blockquote. A class name in the selector has more weight than two element names.)

Exercises

EXERCISE 11-2. Your first style sheet

Open *cooking.html* in a text editor. In the **head** of the document you will find that I have set up a **`style`** element for you to type the rules into. The **`style`** element is used to embed a style sheet in an HTML document. To begin, we'll simply add the small style sheet that we just looked at in this section. Type the following rules into the document, just as you see them here:

```
<style>
h1 {
  color: green;
}
p {
  font-size: large;
  font-family: sans-serif;
}
</style>
```

Save the file, and take a look at it in the browser. You should notice some changes (if your browser already uses a sans-serif font, you may see only a size change). If not, go back and check that you included both the opening and closing curly bracket and semicolons. It's easy to accidentally omit these characters, causing the style sheet not to work.

Now we'll edit the style sheet to see how easy it is to write rules and see the effects of the changes. Here are a few things to try.

IMPORTANT: Remember that you need to save the document after each change in order for the changes to be visible when you reload it in the browser.

- Make the **h1** element "gray" and take a look at it in the browser. Then make it "blue". Finally, make it "orange". (We'll run through the complete list of available color names in **Chapter 13, Colors and Backgrounds**.)
- Add a new rule that makes the **h2** elements orange as well.
- Add a 100-pixel left margin to paragraph (**p**) elements by using this declaration:

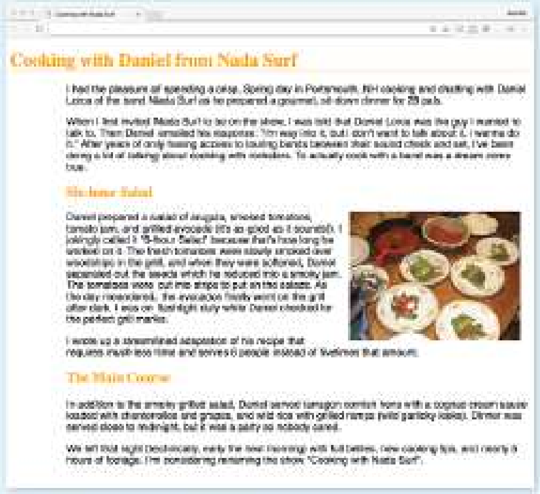
  ```
  margin-left: 100px;
  ```

  Remember that you can add this new declaration to the existing rule for **p** elements.
- Add a 100-pixel left margin to the **h2** headings as well.
- Add an orange, 1-pixel border to the bottom of the **h1** element by using this declaration:

  ```
  border-bottom: 1px solid orange;
  ```
- Move the image to the right margin, and allow text to flow around it with the **`float`** property. The shorthand **`margin`** property shown in this rule adds zero pixels of space on the top and bottom of the image and 12 pixels of space on the left and right of the image (the values are mirrored in a manner explained in **Chapter 14, Thinking Inside the Box**):

  ```
  img {
    float: right;
    margin: 0 12px;
  }
  ```

When you are done, the document should look something like the one shown in FIGURE 11-4.



Cooking with Daniel from Nada Surf

I had the pleasure of spending a crisp, Spring day in Portsmouth, NH cooking and chatting with Daniel Lorca of the band Nada Surf as he prepared a gourmet, sit-down dinner for 28 pals.

When I first invited Nada Surf to be on the show, I was told that Daniel Lorca was the guy I wanted to talk to. Then Daniel emailed his response: "I'm way into it, but I don't want to talk about it, I wanna do it." After years of only having access to touring bands between their sound check and set, I've been doing a lot of talking about cooking with musicians. To actually cook with a band was a dream come true.

Six-hour Salad

Daniel prepared a salad of arugula, smoked tomatoes, tomato jam, and grilled avocado (it's as good as it sounds!). I jokingly called it "6-hour Salad" because that's how long he worked on it. The fresh tomatoes were slowly smoked over woodchips in the grill, and when they were softened, Daniel separated out the seeds which he reduced into a smoky jam. The tomatoes were cut into strips to put on the salads. As the day progressed, the avocados finally went on the grill after dark. I was on flashlight duty while Daniel checked for the perfect grill marks.

I wrote up a streamlined adaptation of his recipe that requires much less time and serves 6 people instead of [illegible] that amount.

The Main Course

In addition to the smoky grilled salad, Daniel served tarragon cornish hens with a cognac cream sauce loaded with chanterelles and grapes, and wild rice with grilled ramps (wild garlicky leeks). Dinner was served close to midnight, but it was a party so nobody cared.

We left that night (technically, early the next morning) with full bellies, new cooking tips, and nearly [illegible] hours of footage. I'm considering renaming the show "Cooking with Nada Surf".

FIGURE 11-4. The article after we add a small style sheet. Not beautiful—just different.

EXERCISE 11-3. Applying an inline style

Open the article *cooking.html* in whatever state you last left it in EXERCISE 11-2. If you worked to the end of the exercise, you will have a rule that makes the **h2** elements orange.

Write an inline style that makes the second **h2** gray. We'll do that right in the opening **h2** tag by using the **`style`** attribute, as shown here:

```
<h2 style="color: gray">The
Main Course</h2>
```

Note that it must be gray-with-an-a (not grey-with-an-e) because that is the way the color is defined in the spec.

Save the file and open it in a browser. Now the second heading is gray, overriding the orange color set in the embedded style sheet. The other **h2** heading is unaffected.

Exercises

EXERCISE 12-5. Using selectors

This time, we'll add a few more style rules using descendant, ID, and class selectors combined with the `font` and `color` properties we've learned about so far.

1. I'd like to add some attention-getting color to the "new item!" elements next to certain menu item names. They are marked up as `strong`, so we can apply the `color` property to the `strong` element. Add this rule to the embedded style sheet, save the file, and reload it in the browser:

```
strong {
   font-style: italic;
   color: tomato;
}
```

   That worked, but now the `strong` element "Very spicy" in the description is "tomato" red too, and that's not what I want. The solution is to use a contextual selector that targets only the `strong` elements that appear in `dt` elements. Remove the `color` declaration you just wrote from the `strong` rule, and create a new rule that targets only the `strong` elements within definition list terms:

```
dt strong { color: tomato; }
```

2. Look at the document source, and you will see that the content has been divided into three unique `div`s: `info`, `appetizers`, and `entrees`. We can use these to our advantage when it comes to styling. For now, let's do something simple and apply a teal color to the text in the `div` with the ID "info". Because color inherits, we need to apply the property only to the `div` and it will be passed down to the `h1` and `p`:

```
#info { color: teal; }
```

3. Now let's get a little fancier and make the paragraph inside the "info" section italic in a way that doesn't affect the other paragraphs on the page. Again, a contextual selector is the answer. This rule selects only paragraphs contained within the `info` section of the document:

```
#info p { font-style: italic; }
```

4. I want to give special treatment to all of the prices on the menu. Fortunately, they have all been marked up with `span` elements:

```
<span class="price">$3.95</span>
```

   So now all we have to do is write a rule using a class selector to change the font to Georgia or some serif font, make the prices italic, and gray them back:

```
.price {
   font-family: Georgia, serif;
   font-style: italic;
   color: gray;
}
```

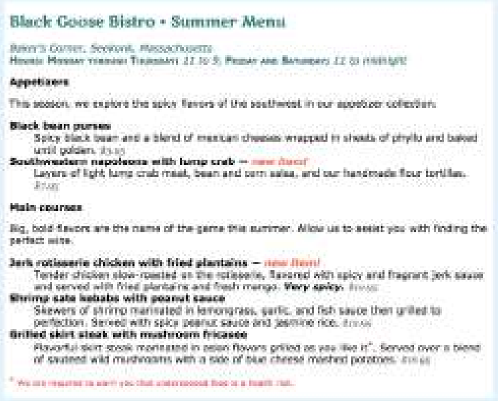
5. Similarly, in the "info" `div`, I can change the appearance of the spans that have been marked up as belonging to the "label" class to make the labels stand out:

```
.label {
   font-weight: bold;
   font-variant: small-caps;
   font-style: normal;
}
```

6. Finally, there is a warning at the bottom of the page that I want to make small and red. It has been given the class "warning," so I can use that as a selector to target just that paragraph for styling. While I'm at it, I'm going to apply the same style to the `sup` element (the footnote asterisk) earlier on the page so they match. Note that I've used a grouped selector, so I don't need to write a separate rule.

```
p.warning, sup {
   font-size: small;
   color: red;
}
```

FIGURE 12-11 shows the results of all these changes. We now have some touches of color and special typography treatments.



Black Goose Bistro • Summer Menu

Baker's Corner, Seekonk, Massachusetts
Hours: Monday through Thursday: 11 to 9; Friday and Saturday: 11 to midnight

Appetizers

This season, we explore the spicy flavors of the southwest in our appetizer collection.

Black bean purses
Spicy black bean and a blend of mexican cheeses wrapped in sheets of phyllo and baked until golden. [illegible]
Southwestern napoleons with lump crab — new item!
Layers of light lump crab meat, bean and corn salsa, and our handmade flour tortillas. [illegible]

Main courses

Big, bold flavors are the name of the game this summer. Allow us to assist you with finding the perfect wine.

Jerk rotisserie chicken with fried plantains — new item!
Tender chicken slow-roasted on the rotisserie, flavored with spicy and fragrant jerk sauce and served with fried plantains and fresh mango. **Very spicy.** [illegible]
Shrimp sate kebabs with peanut sauce
Skewers of shrimp marinated in lemongrass, garlic, and fish sauce then grilled to perfection. Served with spicy peanut sauce and jasmine rice. [illegible]
Grilled skirt steak with mushroom fricasee
Flavorful skirt steak marinated in asian flavors grilled as you like it*. Served over a blend of sauteed wild mushrooms with a side of blue cheese mashed potatoes. [illegible]

* We are required to warn you that undercooked food is a health risk.

FIGURE 12-11. The current state of the bistro menu.

EXERCISE 12-6. Finishing touches

Let's add a few finishing touches to the online menu, *menu.html*. It might be useful to save the file and look at it in the browser after each step to see the effect of your edits and to make sure you're on track. The finished style sheet is provided in the *materials* folder for this chapter.

1. First, I have a few global changes to the `body` element in mind. I've had a change of heart about the `font-family`. I think that a serif font such as Georgia would be more sophisticated and appropriate for a bistro menu. Let's also use the `line-height` property to open up the text lines and make them easier to read. Make these updates to the `body` style rule, as shown:

```
body {
   font-family: Georgia, serif;
   font-size: small;
   line-height: 1.75em;
}
```

2. I also want to redesign the "info" section of the document. Remove the teal color setting by deleting that whole rule. Once that is done, make the `h1` olive green and the paragraph in the header gray. Add color declarations to the existing rules:

```
~~#info { color: teal; }~~  /* delete */
h1 {
   font: bold 1.5em "Marko One", Georgia, serif;
   color: olive;}
#info p {
   font-style: italic;
   color: gray;}
```

3. Next, to imitate a fancy restaurant menu, I'm going to center a few key elements on the page with the `text-align` property. Write a rule with a grouped selector to center the headings and the "info" section.

```
h1, h2, #info {
   text-align: center;}
```

4. I want to make the "Appetizer" and "Main Courses" `h2` headings more eye-catching. Instead of large, bold type, I'm going to use all uppercase letters, extra letter spacing, and color to call attention to the headings. Here's the new rule for `h2` elements that includes all of these changes:

```
h2 {
   font-size: 1em;
   text-transform: uppercase;
   letter-spacing: .5em;
   color: olive;}
```

5. We're really close now; just a few more tweaks to those paragraphs right after the `h2` headings. Let's center those too and make them italic.

```
h2 + p {
   text-align: center;
   font-style: italic;}
```

   Note that I've used a next-sibling selector (`h2 + p`) to select any paragraph that follows an `h2`.

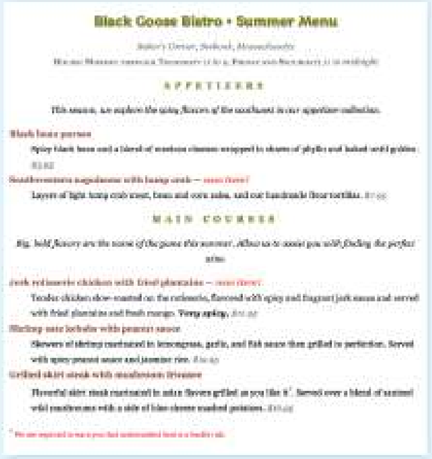
6. Next, add a softer color to the menu item names (in `dt` elements). I've chosen "sienna," one of the names from the CSS3 color module. Note that the `strong` elements in those `dt` elements stay "tomato" red because the color applied to the `strong` elements overrides the color inherited by their parents.

```
dt {
   font-weight: bold;
   color: sienna;}
```

7. Finally, for kicks, add a drop shadow under the `h1` heading. You can play around with the values a little to see how it works. I find it to look a little clunky against a white background, but when you have a patterned background image, sometimes a drop shadow provides the little punch you need to make the text stand out. Notice how small the shadow values are—a little goes a long way!

```
h1 {
   font: bold 1.5em "Marko One", Georgia, serif;
   color: olive;
   text-shadow: .05em .05em .1em lightslategray;}
```

And we're done! FIGURE 12-20 shows how the menu looks now—an improvement over the unstyled version, and we used only text and color properties to do it. Notice that we didn't touch a single character of the document markup in the process. That's the beauty of keeping style separate from structure.



Black Goose Bistro • Summer Menu

Baker's Corner, Seekonk, Massachusetts
Hours: Monday through Thursday: 11 to 9; Friday and Saturday: 11 to midnight

APPETIZERS

This season, we explore the spicy flavors of the southwest in our appetizer collection.

Black bean purses
Spicy black bean and a blend of mexican cheeses wrapped in sheets of phyllo and baked until golden. [illegible]
Southwestern napoleons with lump crab — new item!
Layers of light lump crab meat, bean and corn salsa, and our handmade flour tortillas. [illegible]

MAIN COURSES

Big, bold flavors are the name of the game this summer. Allow us to assist you with finding the perfect wine.

Jerk rotisserie chicken with fried plantains — new item!
Tender chicken slow-roasted on the rotisserie, flavored with spicy and fragrant jerk sauce and served with fried plantains and fresh mango. **Very spicy.** [illegible]
Shrimp sate kebabs with peanut sauce
Skewers of shrimp marinated in lemongrass, garlic, and fish sauce then grilled to perfection. Served with spicy peanut sauce and jasmine rice. [illegible]
Grilled skirt steak with mushroom fricasee
Flavorful skirt steak marinated in asian flavors grilled as you like it*. Served over a blend of sauteed wild mushrooms with a side of blue cheese mashed potatoes. [illegible]

* We are required to warn you that undercooked food is a health risk.

FIGURE 12-20. The formatted Black Goose Bistro menu.