CSS

CSC 337, Fall 2013
The University of Arizona
William H. Mitchell
whm@cs

Big picture

HTML – Hypertext Markup Language
    Specifies structure and meaning

CSS – Cascading Style Sheets
    Specifies presentation

PHP – PHP: Hypertext Preprocessor
    One of many back-end options

JavaScript
    Specifies behavior

Introduction

A web without style

Open one of your favorite sites in Firefox and do this:

View>Page Style>No Style

psst! Tell me if you know how to do this in Chrome!

HTML 3.2 Visual Markup

```
<body bgcolor=silver>
    <p>Visual
    <center>
    <font size=2 color=maroon>Caesar says...<br>
    <font size=+2>
    Lorem<img src=caesar.jpg width=50
               border=10 align=middle>Ipsum
    </font></font>
    </center>
    <p align=right>Markup
</body>
```

vismark.html

What is CSS?

"CSS 2.1 is a style sheet language that allows authors and users to attach style (e.g., fonts and spacing) to structured documents. By separating the presentation style of documents from the content of documents, CSS 2.1 simplifies Web authoring and site maintenance." – W3C

[psst! csszengarden.com here!]

1996: CSS 1 REC (IE5 Mac: full support in 1999)
1997: HTML 4.0 -- deprecates visual markup
1998: CSS 2 REC
2011: CSS 2.1 REC ("CSS level 2 revision 1")

CSS 3

There is a one-document CSS 2.1 specification:

    http://www.w3.org/TR/CSS2

CSS 3 has "modules". Examples:

    Selectors Level 3
    CSS Backgrounds and Borders Module Level 3
    CSS Text Decoration Module Level 3
    CSS Basic User Interface Module Level 3

http://www.w3.org/Style/CSS/specs

    Descriptions of all CSS specifications (64 now)

CSS4 can be thought of as the modules at level 4

Resources

Cascading Style Sheets Level 2 Revision 1 (W3C REC)
    http://www.w3.org/TR/CSS2/

Full Property Index
    http://www.w3.org/TR/CSS2/propidx.html

CSS: The Definitive Guide, 3rd ed. by Eric Meyer
    On Safari

reference.sitepoint.com/css
    Well-organized and concise

HFHC and MDN

Others? (Post on Piazza in "resources" folder)

CSS Rules

CSS Rules

Here is a CSS *rule* (or *rule set*):

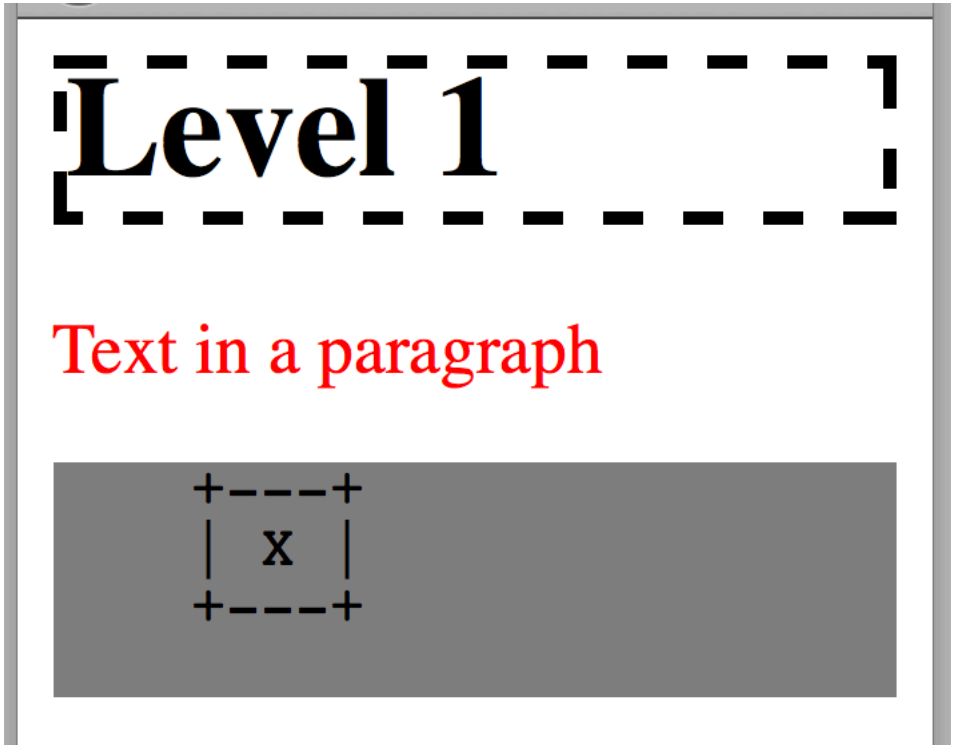
```
p { color: red; }
```

It says,
"Use red text in all paragraph elements."

What do these say?

```
h1 { border-style: dashed; }

pre { background-color: gray }
```

CSS Rules--Basics



Level 1

Text in a paragraph

```
+----+
| x  |
+----+
```

```
h1 { border-style:
        dashed; }

p { color: red; }

pre {
  background-color:
        gray }
```

cssbasic1.html

Applying rules

One way to apply CSS rules to HTML is to add a <style> element in <head>:

```
<head>
    ...
      <style>
          p { color: red; }
          h1 { border-style: dashed; }
          pre { background-color: gray }
      </style>
```

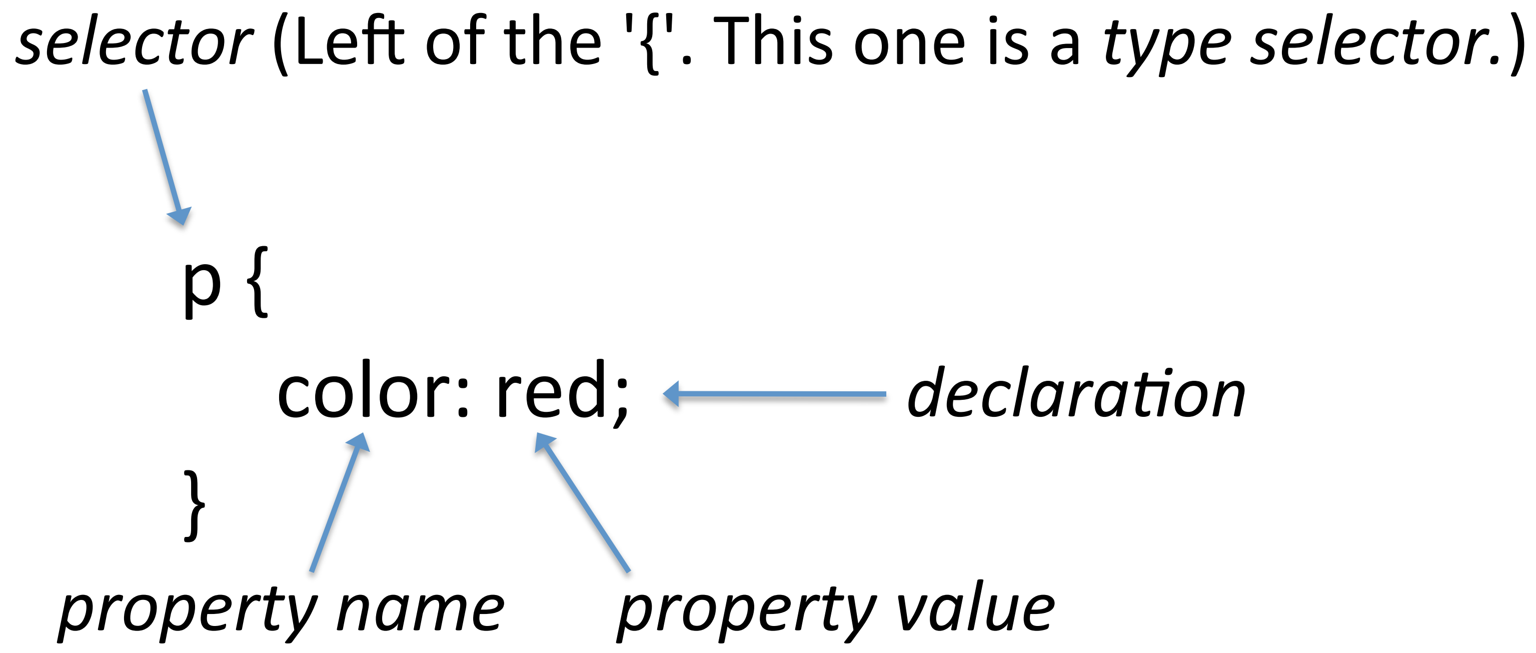
This is an *embedded style sheet*. The rules are applied only to elements on this page.

Hands-on with styles

Let's try...
- Multiple declarations
- Multiple selectors ("grouping")
- More properties:
  - font-family, border-radius, text-align, list-style-type, display

Chrome:
- Toggle properties
- Set colors
- Increment/decrement values
- Add a property
- Misspelled property
- Note source of rules

CSS Rule Anatomy


selector (Left of the '{'. This one is a *type selector.*)
declaration
property name
property value

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

- Selector and property names are case insensitive.
- The braces and the declaration(s) they enclose are the *declaration block*.

Grouping

If we have rules with identical declarations ...

```
h1 { font-family: cursive }
h2 { font-family: cursive }
h3 { font-family: cursive }
```

they can be *grouped*:

```
h1, h2, h3 { font-family: cursive }
```

An ineffective rule

None of the declarations in this CSS rule have any effect:

```
p {
    color: "red";
    font-family: "sans-serif";
    border-style: "dashed";
}                                    /* ineff.html */
```

What's the problem?

Does Chrome shed any light on the problem?

Does it validate at jigsaw.w3.org/css-validator?

A key to CSS: keywords

Here are some Java keywords*:
    if, while, for

Here are some CSS keywords:
    red, sans-serif, dashed

"red", "sans-serif", and 'dashed' are strings

Typically, keywords (not strings!) are used for non-numeric property values.

We'll see places to use strings later.

There are many CSS keywords!

*JFotD: true, false, and null are Java literals

What if...

Speculate: What happens if two rules have the same selector?

```
h1 {
    background-color: black;
    color: white;
    }

h1 {
    background-color: white;
    color: black;
}                                /* samesel.html */
```

HFHC p.266

An example from the text:

```
h1, h2 { font-family: sans-serif; color: grey;}
p { color: maroon; }  </style>

<h1>Welcome!</h1>
<p>Welcome to the HFL...
<h2>Directions</h2>
<p>Try Google Maps...             hfhc266.html
```

Problem: Add a nice-looking line under Welcome!

External Stylesheets

External stylesheets

The HTML `link` element can be used to reference an *external stylesheet*. Its content category is metadata. It is typically a child of `head`.

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

For stylesheets, use the `rel` and `type` values above.

A document can have any number of link elements.

What might external stylesheets let us do?

External stylesheet hands-on

Using csslink1.html...

Try each `link` in turn outside of the comment.

Try permutations of two and three.

See if `rel` and `type` attributes are really needed.

External stylesheets, continued

External stylesheets let us...

- Use the same collection of rules in many HTML files.
- Include rules from several stylesheets in a single file.

CSS also provides an *at-rule*, @import, which allows nesting, but read this post before using it:
http://www.stevesouders.com/blog/2009/04/09/dont-use-import/

Inheritance
(Take #2)

Inheritance

Some of an element's CSS property values are *inherited* from the element's parent element.

The following rule causes all text on a page to be in a typewriter-like font:

```
    body { font-family: monospace }
```

Look at cssinh1.html. How many children does `body` have? How many descendants?

Every descendant of body will have monospace text

Inheritance, continued

Adding the following rule causes text in paragraphs and all descendants of paragraphs to be in red:

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

What kinds of elements can be descendants of paragraphs?

How many descendants of paragraphs are in cssinh1.html?

Inheritance, continued

Adding the following rule causes text in list items and all descendants of list items to be in a cursive font:

```
    li { font-family: cursive }
```

What kinds of elements can be descendants of list items?

How many descendants of list items are in cssinh1.html?

Inheritance, continued

Problem: Predict the result of adding each of these, one at a time, to cssinh1.html:

```
ul { color: blue }
ol { color: green }
li { color: lime }
blockquote { color: blue }
```

Inheritance, continued

From *Selectors, Specificity, and the Cascade* by Meyer: (all after first sentence is just FYI for now)

> "Inheritance is the mechanism by which some property values are passed on from an element to its descendants. When determining which values should apply to an element, a user agent must consider not only inheritance but also the *specificity* of the declarations, as well as the origin of the declarations themselves. This process of consideration is what's known as the *cascade*. We will explore the interrelation between these three mechanisms—specificity, inheritance, and the cascade—in this chapter, but the difference between the latter two can be summed up this way: choosing the result of h1 {color: red; color: blue;} is the cascade; making a span inside the h1 blue is inheritance."

Inheritance, continued

Some properties inherit values from parent element values and some do not.

http://www.w3.org/TR/CSS2/propidx.html shows a full list of CSS 2.1 properties, including whether their values are inherited.

Let's look at it and see if we can discern a pattern to what's inherited and what isn't.

Inheritance, continued

Here are the common non-table, non-list, properties whose values are inherited.

color	quotes
font-family	text-align
font-size	text-indent
font-style	text-transform
font-variant	visibility
font-weight	white-space
letter-spacing	word-spacing
line-height	

What's the pattern?

Sidebar: handy pipeline

For those who like shell pipelines...

(1) Put the text of the CSS "Full property table" on the clipboard with C-A, C-C.

(2) Do one of the following with Cygwin, OSX, or Linux, respectively:

```
grep -v aural </dev/clipboard | grep yes | cut -f2 -d\'

pbpaste | grep -v aural | grep yes | cut -f2 -d\'

xsel | grep -v aural | grep yes | cut -f2 -d\'
```

Beware of false positives/negatives with this approximation!

Properties and more properties!

Go to Chrome DevTools, expand Computed Style, and enable Show inherited. How many?

Firefox: Try Firebug and Tools>Web Developer>Toggle Tools

Does the number of properties vary based on...

- The page?
- The element?
- The browser?

Vendor-specific extensions

Keywords and property names that start with dash or underscore are reserved for *vendor-specific extensions*.

Examples:

- -moz-binding
- -ms-fullscreen
- -webkit-mask
- -moz-initial (keyword)

Some vendor extensions are adopted. There's -o-tab-size, -moz-tab-size, and tab-size.

On a Mac, Dash is good for seeing lots of these.

Is background-color inherited?

From the last version of these slides...

The following rule causes all elements on a page to have a silver background:

body { background-color: silver } wrong1.html

Visually, that statement seems to be true but it's plain wrong!

Let's discuss!

Puzzle!

```
<style>
  foo { color: red }
  bar { font-family: monospace }
  baz { text-decoration: underline }
</style>
<foo>
 this is foo!
</foo>
<bar>this is bar!</bar>
<foo>
<bar>this is foo bar!
<baz>baz, too</baz>
</bar>
</foo>
```

How will it render?

What will the element tree look like?

Is the CSS valid? The HTML?

puzzle1.html

Sidebar: CSS vs. OOP Inheritance

How does CSS inheritance differ from inheritance in an object-oriented programming language like Java or Python?

What's being inherited in each?

Sidebar, continued

Java and Python: Subclasses inherit fields from superclasses.

```
abstract class Shape { // Assume all fields are "public ..."
    float xpos, ypos;
    Color color; }

class Circle extends Shape { float radius; }

class Figure8 extends Circle { Circle upper, lower; }
```

Instances of Circle have xpos, ypos, color, radius.

Instances of Figure8 have two Circles.

Java folks: Would assigning to `figure8.color` cause `figure8.{upper,lower}.color` to change?

Sidebar, continued

I'm not sure I like this analog but let's consider it.

```
<style>
    figure8 { color: red }
</style>
<figure8>
    <circle>UPPER</circle>
    <br>
    <circle>LOWER</circle>
</figure8>
```

figure8.html

What color will "UPPER" and "LOWER" be?

Sidebar, continued

Contrast in a nutshell:

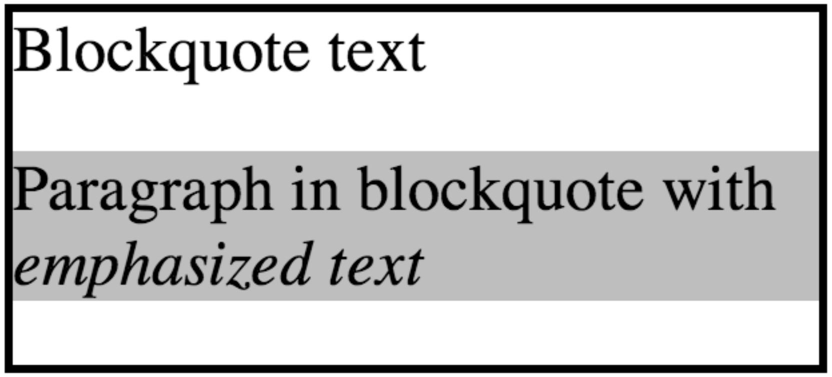
With languages like Java, Python, and C++, classes inherit fields from ancestor classes.

With CSS, elements inherit property values from ancestor elements.

The `inherit` keyword

The property value `inherit` causes a child to use its parent's value for that property.

```
<style>
    blockquote { border-style: solid }  /* not inherited! */
    p { background-color: silver }
    /*p { border-style: inherit }*/
    /*em { border-style: inherit }*/
</style>

<blockquote>
Blockquote text
<p>
Paragraph in blockquote
with <em>emphasized text
</em></p></blockquote>
```



Blockquote text

Paragraph in blockquote with *emphasized text*

inhkw1.html

Class selectors

Class selectors

Thus far we've seen only *type selectors*. They specify properties for elements of a certain type.

```
    p { color: red }
    h1, h2 { color: blue }
```

Class selectors look like this:

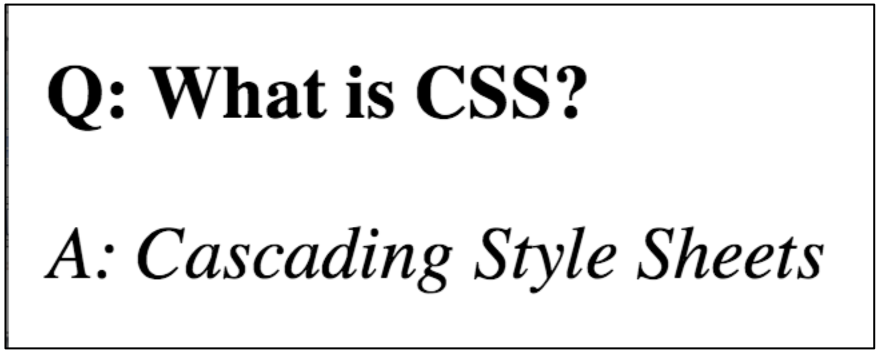
```
    .question { font-weight: bold }
    .answer {font-style: italic }
```

What's the syntactic difference?

Class selectors, continued

The `class` attribute can be used to specify that an element belongs to a particular class.

```
<style>
    .question { font-weight: bold }
    .answer {font-style: italic }
</style>


<p class="question">
Q: What is CSS?
<p class="answer">
A: Cascading Style Sheets
```



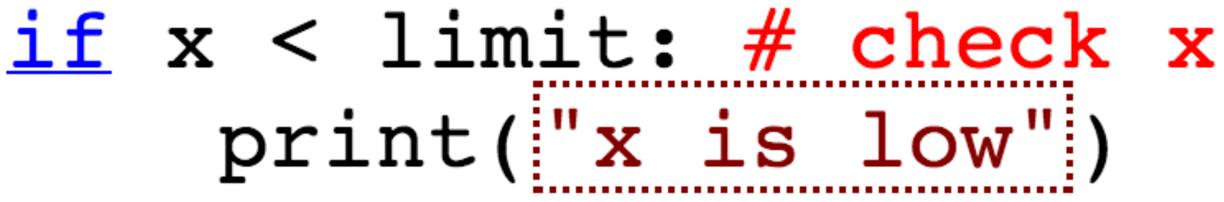
Q: What is CSS?

A: Cascading Style Sheets

csel1.html

Problem: syntax highlighting

The VP of Engineering says that from now on, all source code shown on web pages should have syntax highlighting. Example:



```
if x < limit: # check x
    print("x is low")
```

What specifications does the example imply?

Can we do this with what we already know?

syntax1.html

`<span>` to the rescue!

The `span` element is "a generic container for phrasing content, which does not inherently represent anything." It has inline display.

Here's how we might markup code with `span`:

```
<pre>
<span class=kw>if</span> x &lt; limit: <span
class=cmt># check x</span>
    print(<span class=str>"x is low"</span>)
</pre>
```

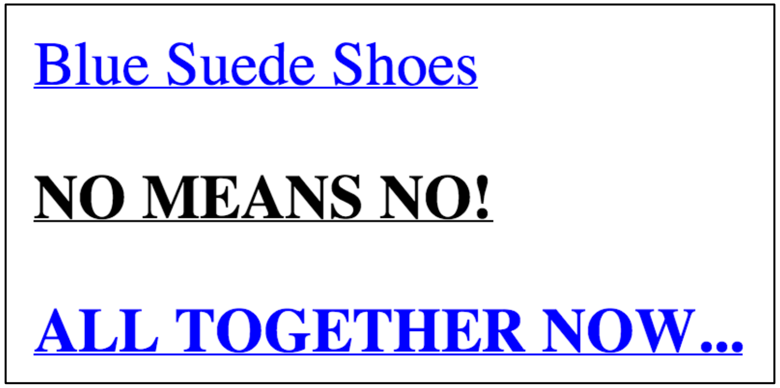
Note that the colored text is the source code.

Let's edit syntax1.html to produce the desired result!

Multiple classes for an element

The class attribute can specify any number of classes.

```
<style>
    .bluetext { color: blue }
    .uline { text-decoration: underline }
    .shout {text-transform: uppercase;
              font-weight: bold }
</style>
<p class="bluetext uline">
Blue Suede Shoes
<p class="uline shout">
no means no!
<p class="uline shout bluetext">
All together now...
```



Blue Suede Shoes

NO MEANS NO!

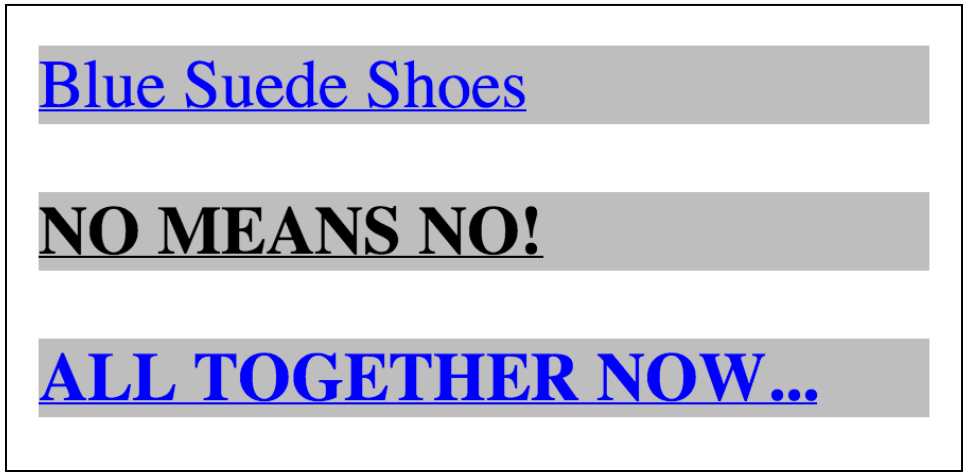
ALL TOGETHER NOW...

mclass1.html

Class selectors can be grouped

Let's extend the example from the last slide and say that we want a silver background on any element whose class is bluetext, shout, or uline:

```
.bluetext, .shout, .uline
   { background-color: silver }
```



Blue Suede Shoes

NO MEANS NO!

ALL TOGETHER NOW...

Remember that *grouping* is a shorthand for multiple rules with the same declaration block.

The truth about class selectors

In fact, class selectors are a specialized form of *attribute selectors*, which we may discuss in more detail later.

FYI, here's an equivalent set of selectors that don't use the `.class` notation:

```
[class~=bluetext] { color: blue }
[class~=uline] { text-decoration: underline }
[class~=shout]
    { text-transform: uppercase; font-weight: bold }
[class~=bluetext], /* note grouping... */
[class~=uline],
[class~=shout] { background-color: silver }
```

See 5.8.3 in the CSS 2.1 REC for more on this.

Combining selectors

Combining selectors

Selectors can be combined to create a selector that expresses multiple criteria. Example:

```
    p.sans { font-family: sans-serif }
```

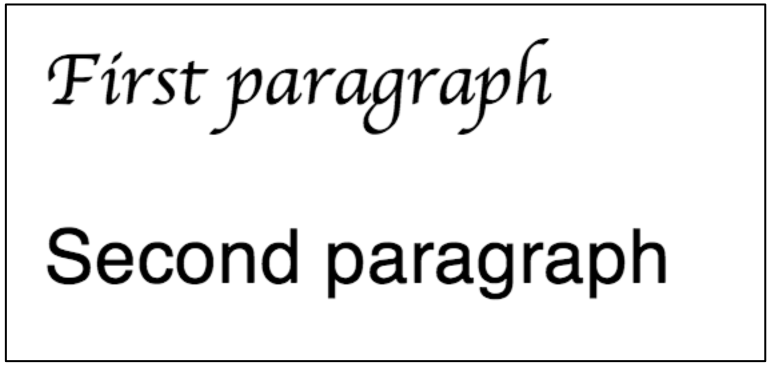
The selector `p.sans` selects <u>paragraphs</u> with `class="... sans ..."`.

Those who have had 335 may be reminded of the Composite pattern—a selector can be composed of selectors.

Combining selectors, continued

Full example:

```
<style>
    p { font-family: cursive }
    p.sans { font-family: sans-serif }
</style>
<p>
First paragraph
<p class=sans>
Second paragraph
```


First paragraph

Second paragraph

How would we express these two rules in english?

Would switching their order change the result?

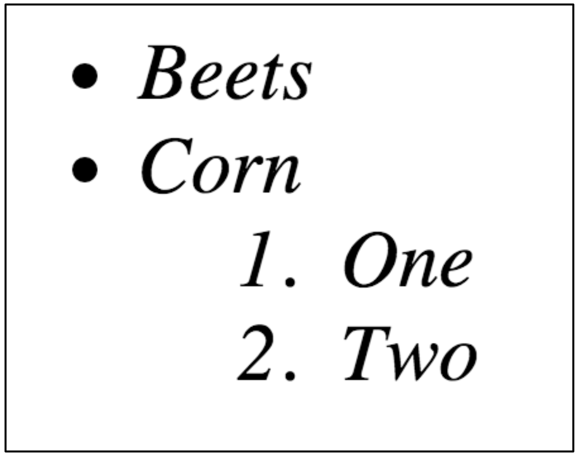
Descendant selectors

Here is an example of a *descendant selector*:

```
ul li { font-style: italic }
```

It means, "Use italics for every list item that is a descendant of an unordered list."

```
<ul>
    <li>Beets
    <li>Corn
    <ol>
        <li>One
        <li>Two
    </ol>
</ul>
```



- *Beets*
- *Corn*
    1. *One*
    2. *Two*

Any surprises?

dsel1.html

Descendant selectors, continued

Problem: Predict the result of this:

```
<style>
    ul li { font-style: italic }
    li em { text-decoration: underline }
    ol li em { color: red }  /*Note the triple */
</style>
<ul>
    <li>Beets
    <li>Corn
</ul>
<ol>
    <li>One <em>Mississippi</em>
    <li>Two <em>Mississippi</em>
</ol>
```

dsel2.html

Child selectors

A descendant selector like `"ul li"` will match at *any* distance in the tree. A *child selector* limits the second selector to children of the first.

```
<style>
    ul > li { border-style: dotted  } /* note the > */
</style>
<ul>
    <li>Beets
    <li>Corn
        <ol>
            <li>One
            <li>Two
        </ol>
</ul>
```

Try it with chsel1.html

Why would color or background-color make for a bad demonstration of this? (A key point!)

ID selectors

An *ID selector* can be used to match only an element with the specified `id` attribute.

```
<style>
    #disclaimer { text-transform: uppercase }
    #disclaimer em {     font-weight: bold;
                         text-decoration: underline;
                         background-color: pink}
</style>
<p id=disclaimer>
Nothing on this page is intended to cause you to take or
not take any action whatsoever.
<em>We mean it!</em>
```

idsel1.html

ID selectors, continued

Recall that id attribute values are expected to be unique on a page.

Proper use: "Some pages might not have a disclaimer but a page will never have more than one disclaimer."

Improper use: "I might want more like it later but right now there's only one element to style like this."

HFHC has an excellent discussion of appropriate use of ID selectors on pp. 394-397.

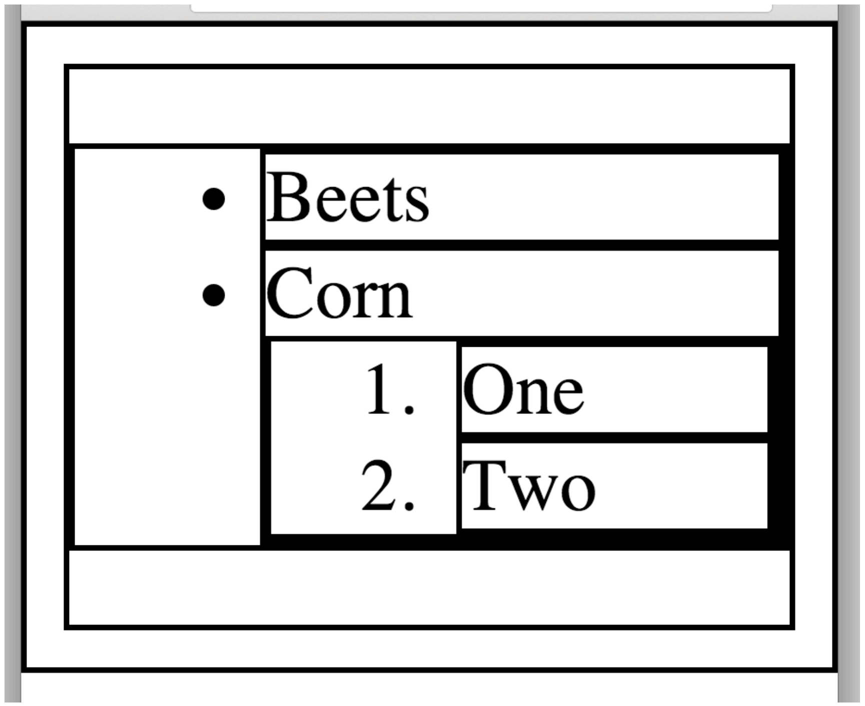
The universal selector

The *universal selector* matches every element.
It is specified as an asterisk.

```
* { border-style: solid }
```

unisel1.html

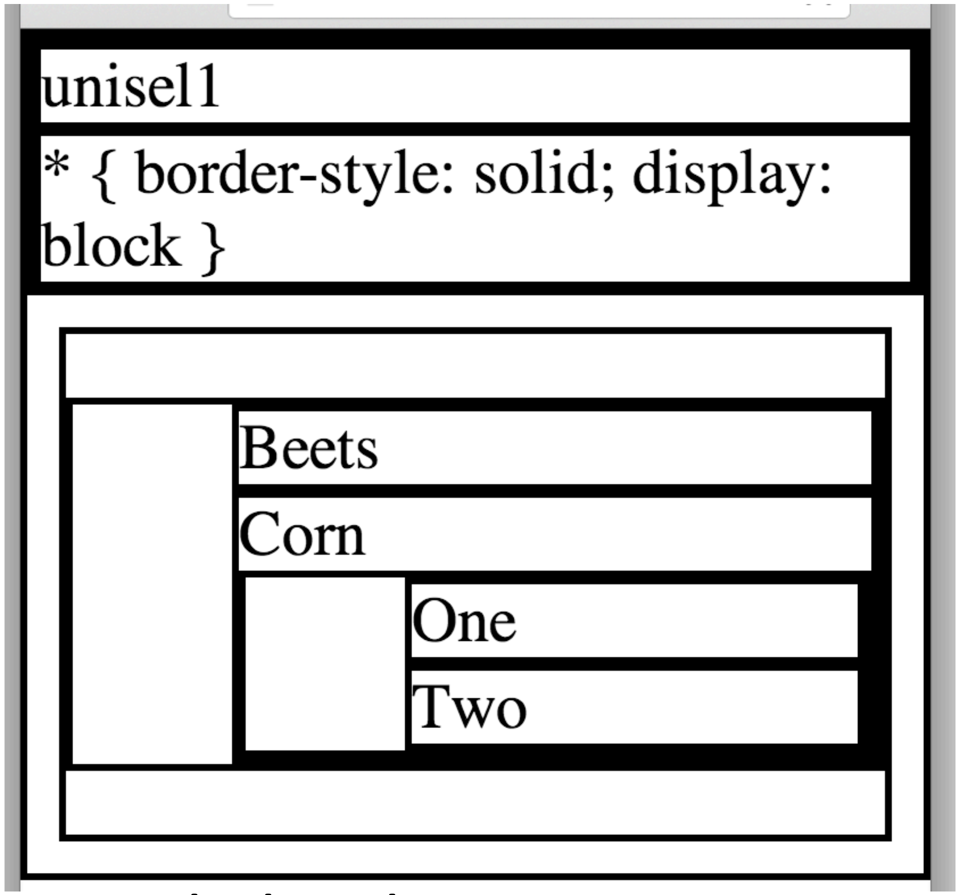
Here's the result with chsel1.html:



The universal selector, continued

Explain this:

```
* { border-style: solid;
    display: block }
```



unisel1

* { border-style: solid; display: block }

Beets

Corn

One

Two

unisel2.html

The universal selector, continued

The CSS 2.1 REC allows the universal selector to be omitted in some cases.

Examples:

*.bluetext { ... }	can be	.bluetext { ... }
*#disclaimer { ... }	can be	#disclaimer {...}
*[align] { ... }	can be	[align] { ... }

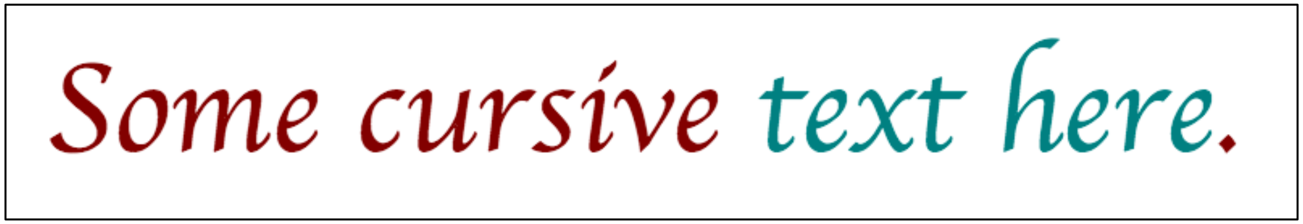
(unisel3.html)

We've been using universal selection without knowing it!

Inline styles

An *inline style* is a set of declarations specified as the value of the `style` attribute of an element.

```
<p style="font-family: cursive; color: maroon">
Some cursive
<span style="color:teal">text here</span>.
```


Some cursive text here.

Note: No curly braces! Are quotes required?

inline1{,a}.html

Inline styles, continued

Inline styles are very handy for experimenting and learning.

Typical practice for production pages is to use inline styles rarely if ever.

An inline style might be used as a "band-aid" until a change to an external stylesheet can be made.

What can be done with an inline style that couldn't be done with selectors?

The box model

Element boxes

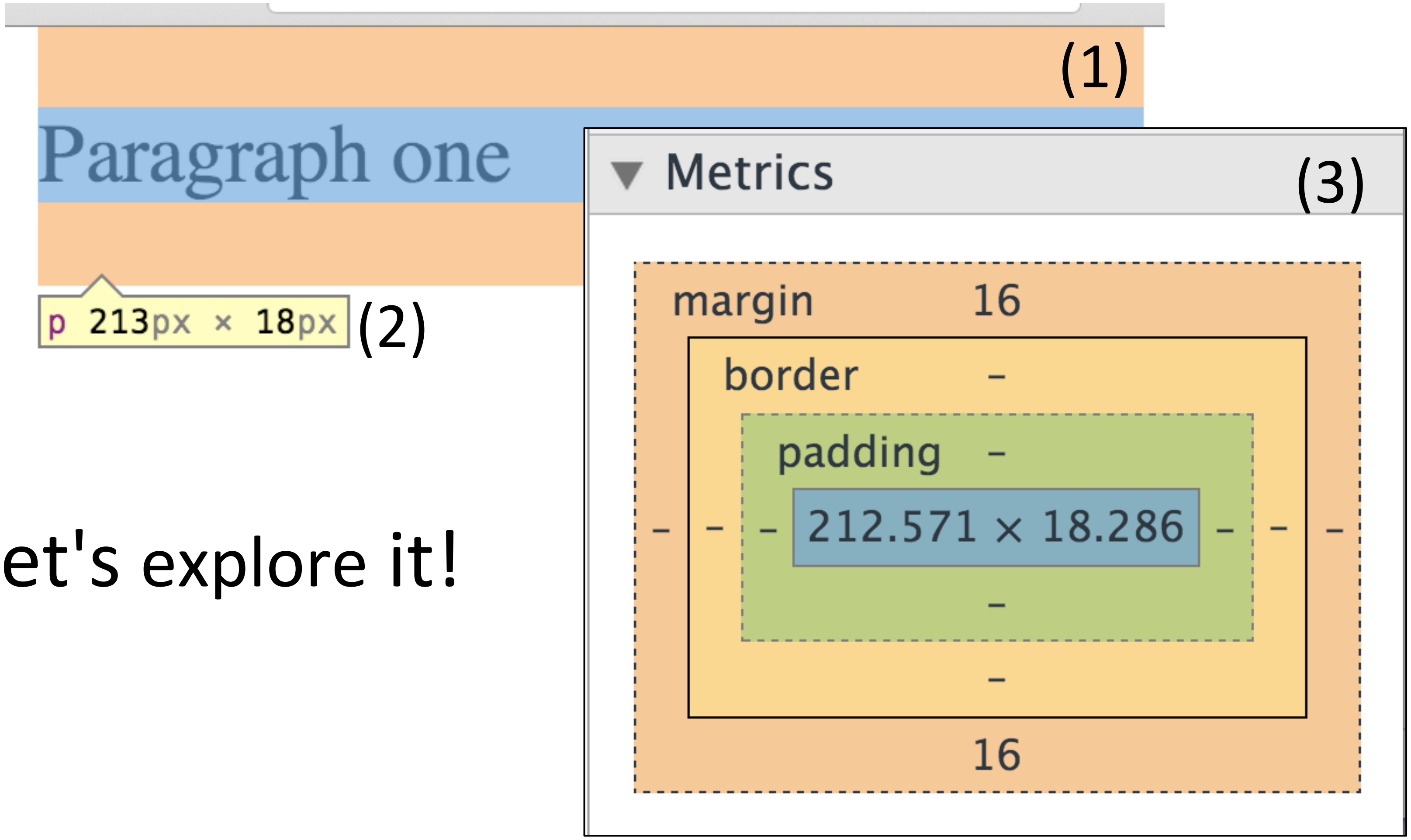
CSS considers every element to have a rectangular *element box*.

The element's *content area* is surrounded by *margins*, *borders*, and *padding*.

"The CSS *box model* describes the rectangular boxes that are generated for elements in the document tree and laid out according to the visual formatting model."—CSS 2.1 REC

Chrome's display of the box model

For a selected paragraph we might see this:



Let's explore it!

box1.html

Chrome's display, continued

From the previous slide:

- The element box (1) for the paragraph is shown with the orange/blue/orange bands.
- The "tab" (2) shows the <u>content area</u> to be 213 pixels wide and 18 pixels high.
- The Metrics panel (3) shows the top and bottom margins to be 16 pixels. Left and right margins are zero; indicated by dashes instead of numbers.
- Metrics also shows there are no borders or padding.
- The colors in (1) and (3) correspond!
- Hover over an area in Metrics to color only that area.

Margins

There are four properties for setting margins individually:

```
p {  margin-top:       30px;
     margin-right:     2.3in;
     margin-bottom:    10cm;
     margin-left:      3em;
     }
```

"4.3 Values" in the CSS 2.1 REC talks about units.

Note the WHAT-WHICH pattern of property naming

Common puzzler: Forgetting the units!

margin1.html

The `margin` shorthand property

Shorthand properties specify values for multiple underlying properties with a single declaration.

We can use the *shorthand property* `margin` to specify all four margins with a single value:

```
p {
    margin: 30px
    }
```

What does Chrome show in this case?

Speculate:
What would { margin: 20px 30px } mean?

margin2.html

`margin`, continued

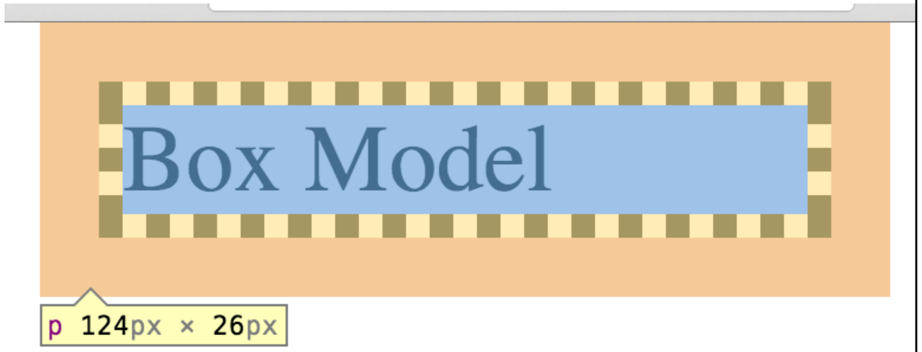
One, two, three or four values can be specified for margin:

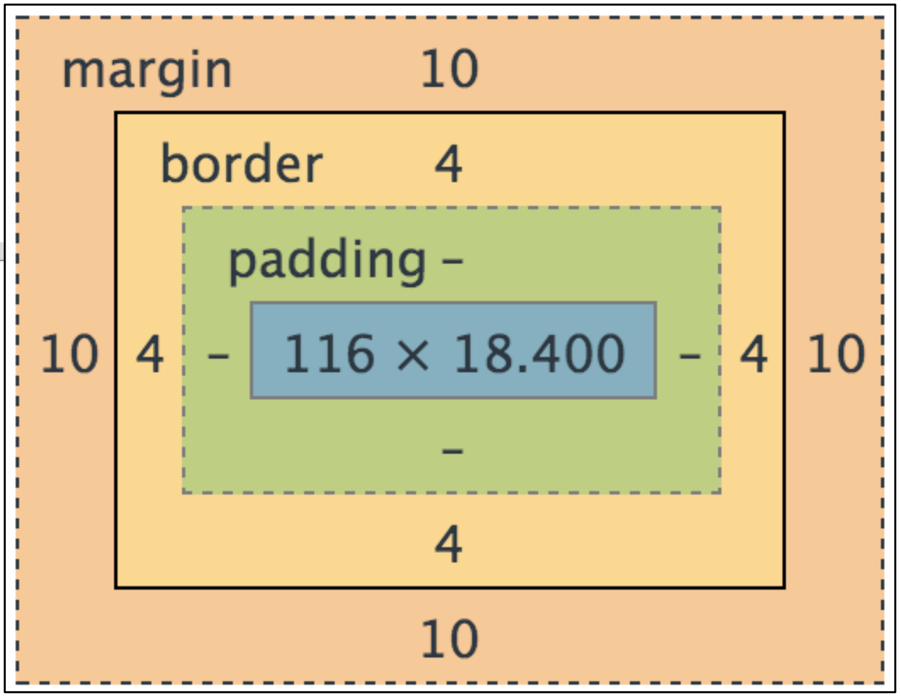
```
margin: 10px           /* 10 px for all */

margin: 10px 20px  /* vertical horizontal */

margin: 5px 10px 20px /* top horiz bottom */

margin: 3px 5px 7px 9px /* T R B L */
```

Borders

Including shorthand properties, CSS 2.1 has 24 border properties. Here are two in action:

```
p { border-style: dotted;
    border-width: 4px;
    margin: 10px; }
```



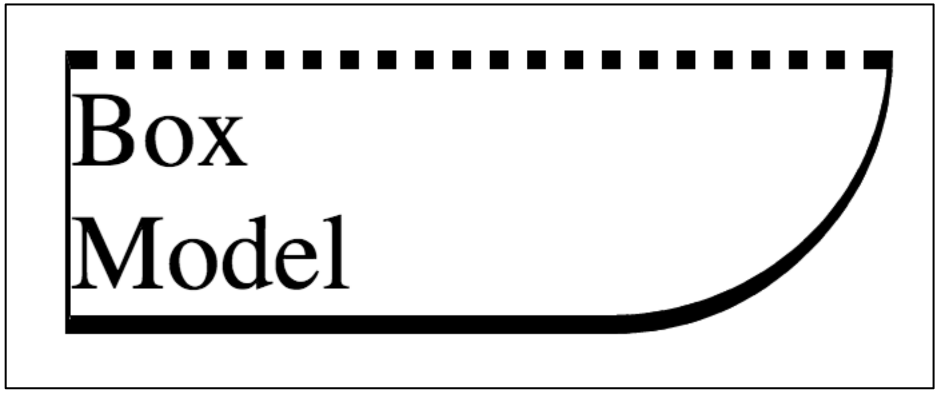


border1.html

Borders, continued

border-style, border-width and border-radius are among the shorthand properties for borders.

```
p { border-style: dotted solid;
    border-width: 3px 1px;
    border-bottom-right-radius: 50px;
    margin: 10px; }
```



Let's fiddle with that radius!

border2.html

The `border` shorthand property

The `border` property can be used to set the width, style, and/or color of all four borders at once.

Here's what the REC says:

'border'

Value:

[<border-width> || <border-style> || <'border-top-color'>] | inherit

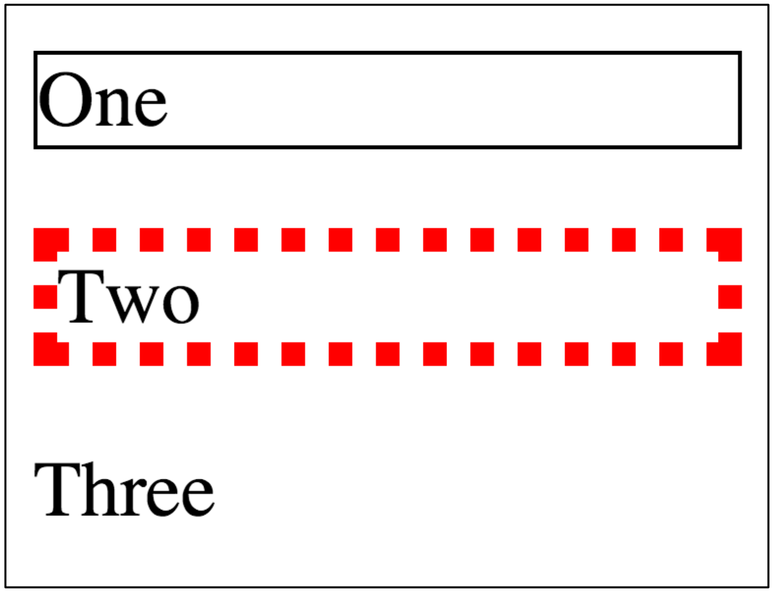
Inherited: no

What does it mean?

`border`, continued

Example:

```
<p style="border: solid 1px">
One
<p style="border: 5px red dotted">
Two
<p style="border: blue 10">
Three
```



One

Two

Three

Any questions?

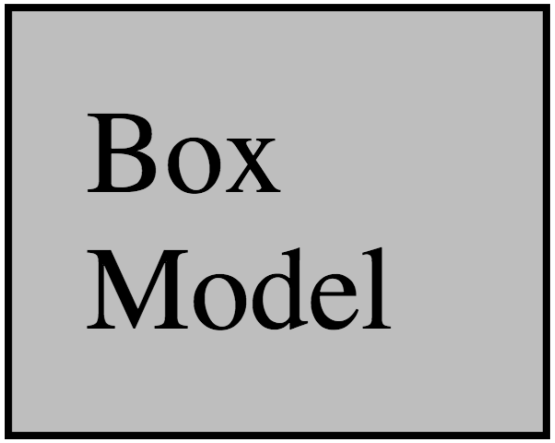
border3.html

Padding

The padding area lies between the border and the content. An example with the `padding` shorthand:

```
p {  padding: 10px;
     border: 1px solid black;
     background-color: silver;
     margin: 5px;
     }
```

What's the relationship between the background color and the padding?

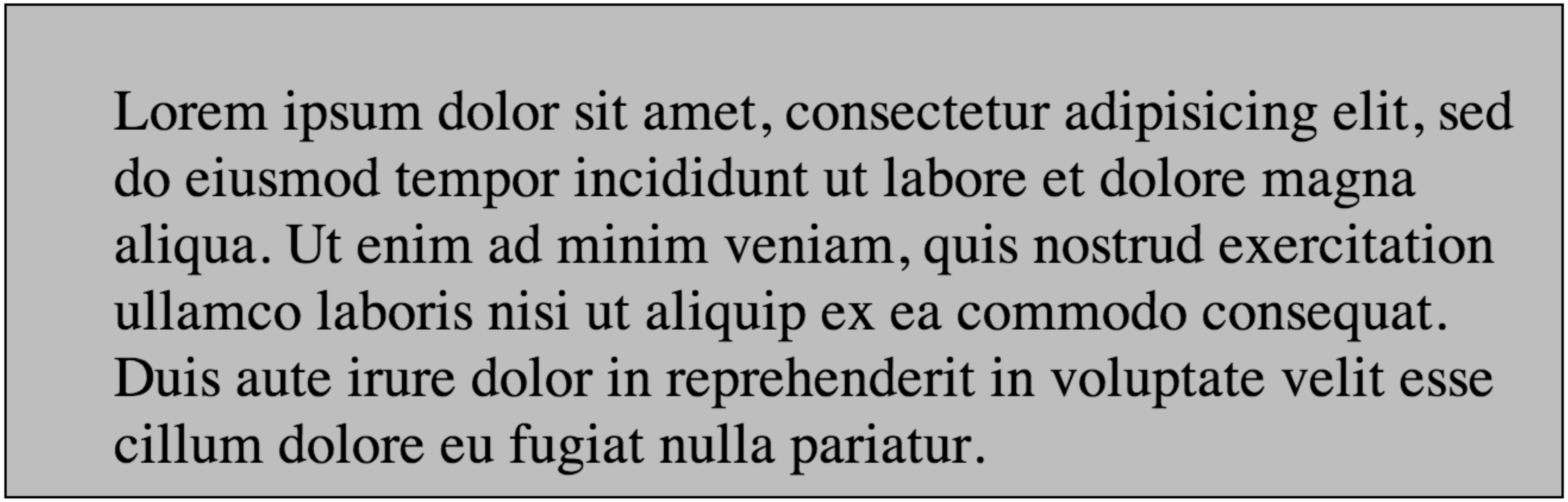


padding1.html

Padding, continued

Specifying different amounts padding for each side can produce a variety of looks.

```
p { padding: 20px 5px 5px 30px; ... }
```



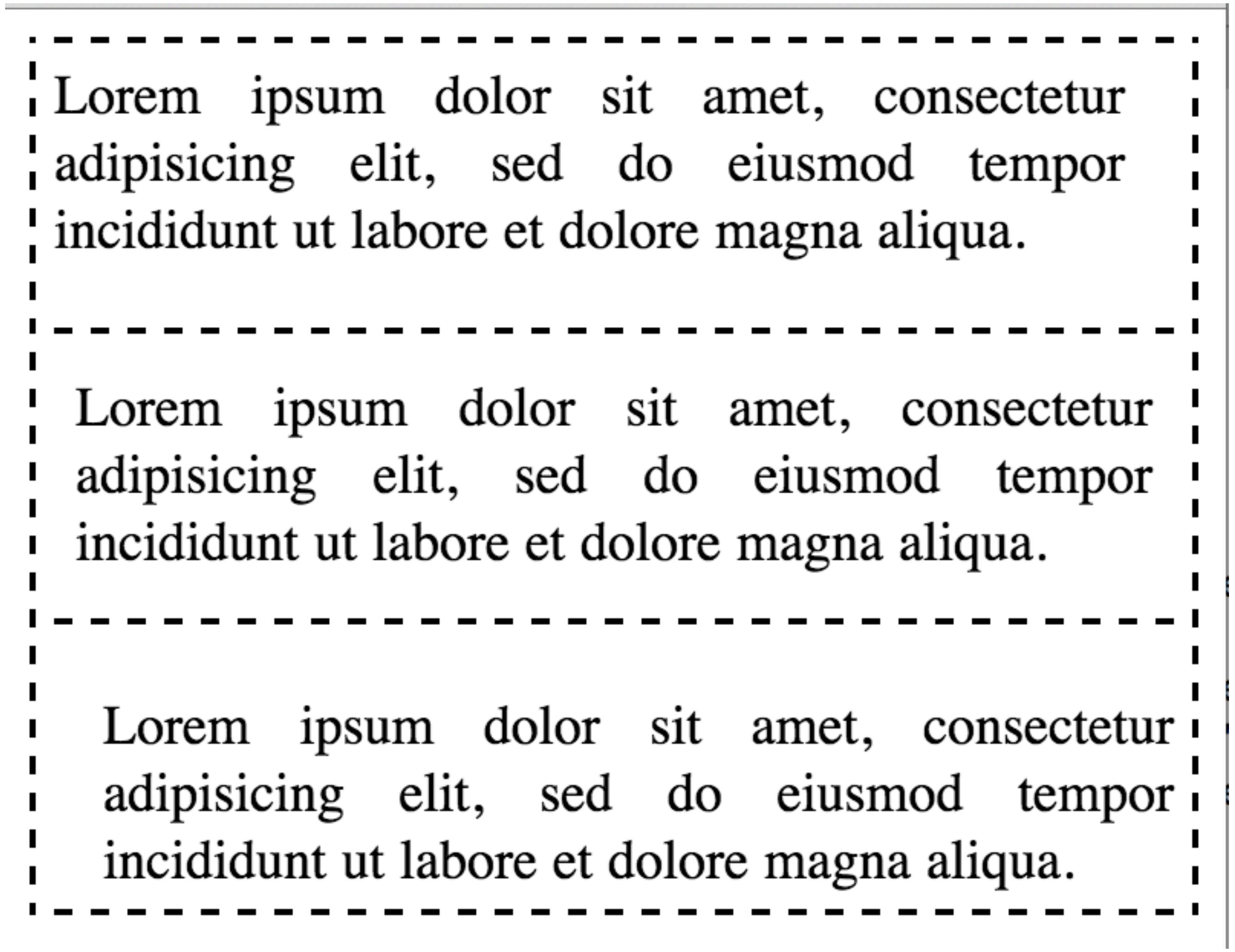
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.

Let's adjust it with Chrome!

padding2.html

Simple problem

Create this:



Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

padding3start.html

`span`'s blocky buddy: `div`

We saw `span` used to mark-up pieces of phrasing content to allow styling them individually.

`div` serves the same purpose but for flow content.

"The `div` element is the generic container for flow content, which does not inherently represent anything." – MDN

"The div element has no special meaning at all. It represents its children. It can be used with the `class`, `lang`, and `title` attributes to mark up semantics common to a group of consecutive elements." —W3C

Remember that `span` and `div` are HTML elements!

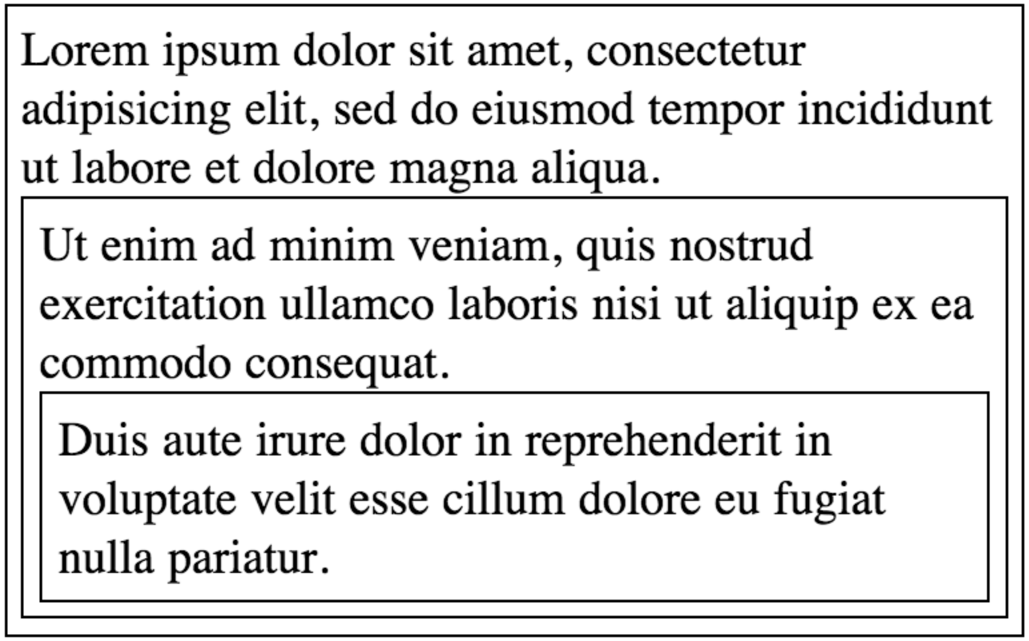
`div`, continued

Example:

```
<div>
 Lorem ipsum...
 <div>
  Ut enim...
  <div>
   Duis aute...
  </div>
 </div>
</div>
```

Rendering, with

```
div { border: solid 1px;
      padding: 5px }
```



Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

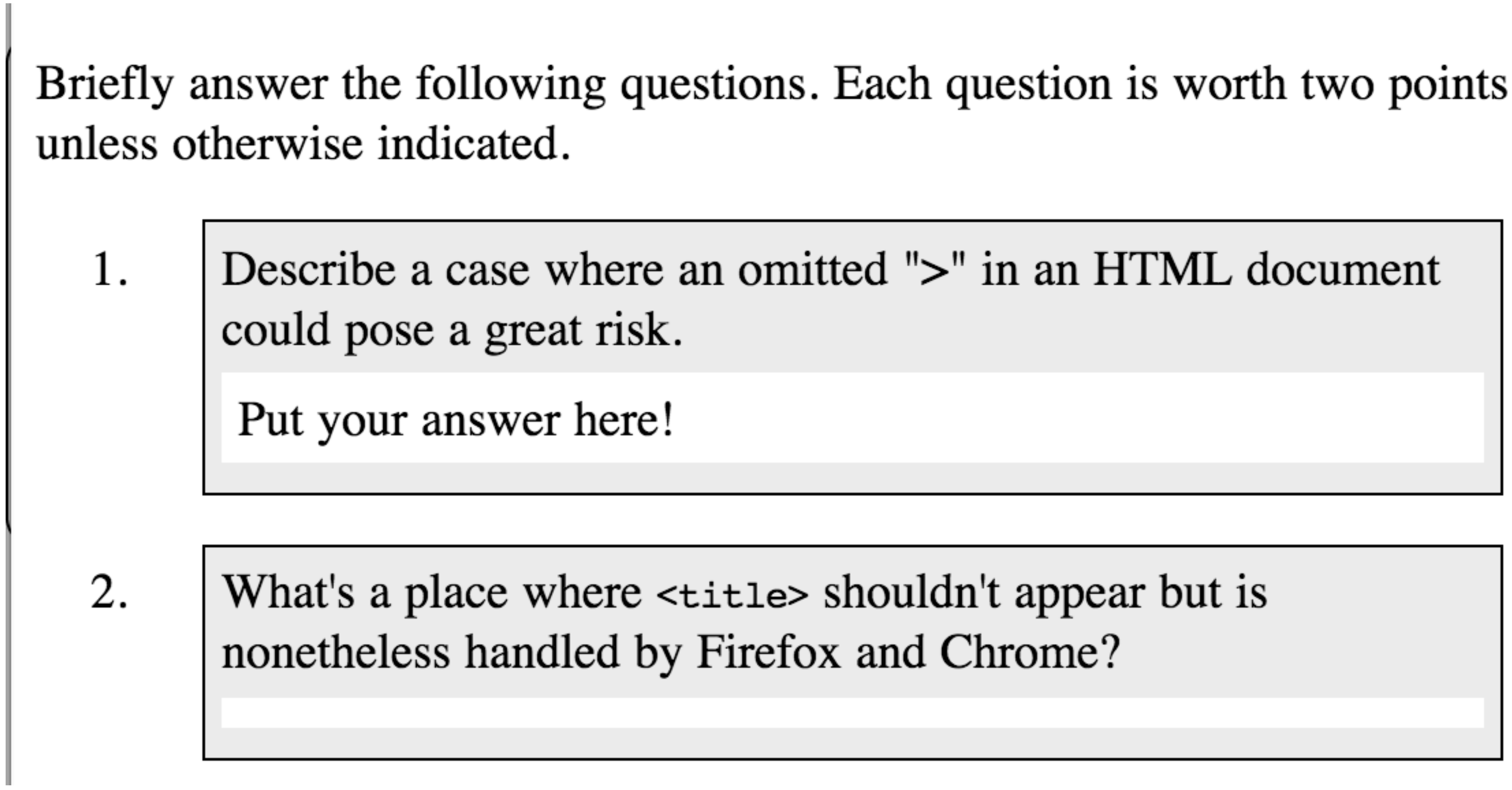
Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.

What are some things we can see about divs?

div1.html

Problem: from a1

How can we produce this layout?



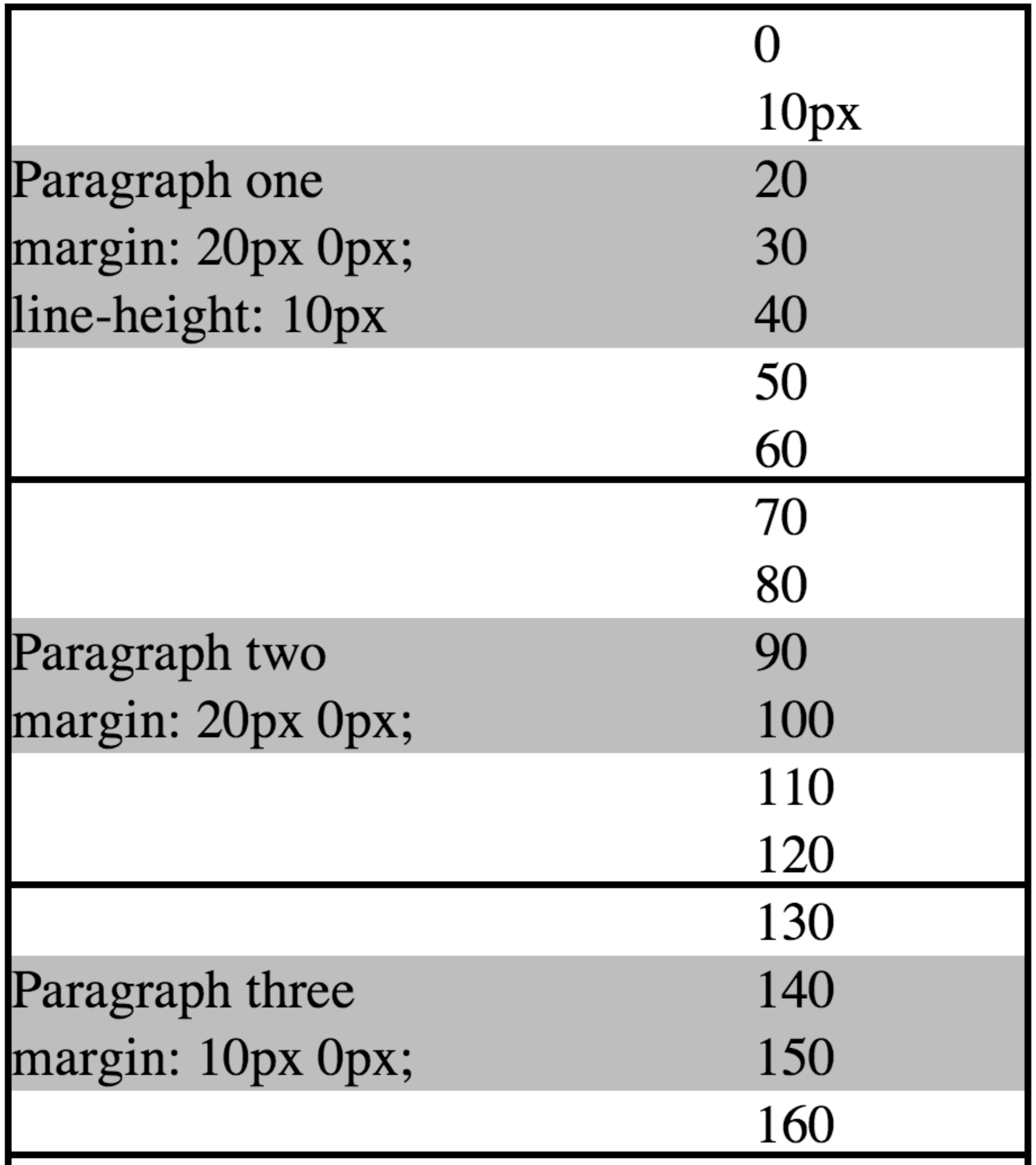
Briefly answer the following questions. Each question is worth two points unless otherwise indicated.

1. Describe a case where an omitted ">" in an HTML document could pose a great risk.

   Put your answer here!

2. What's a place where `<title>` shouldn't appear but is nonetheless handled by Firefox and Chrome?

A marginal experiment


0
10px
Paragraph one
margin: 20px 0px;
line-height: 10px
20
30
40
50
60
70
80
Paragraph two
margin: 20px 0px;
90
100
110
120
130
Paragraph three
margin: 10px 0px;
140
150
160

Paragraphs have margins as shown.

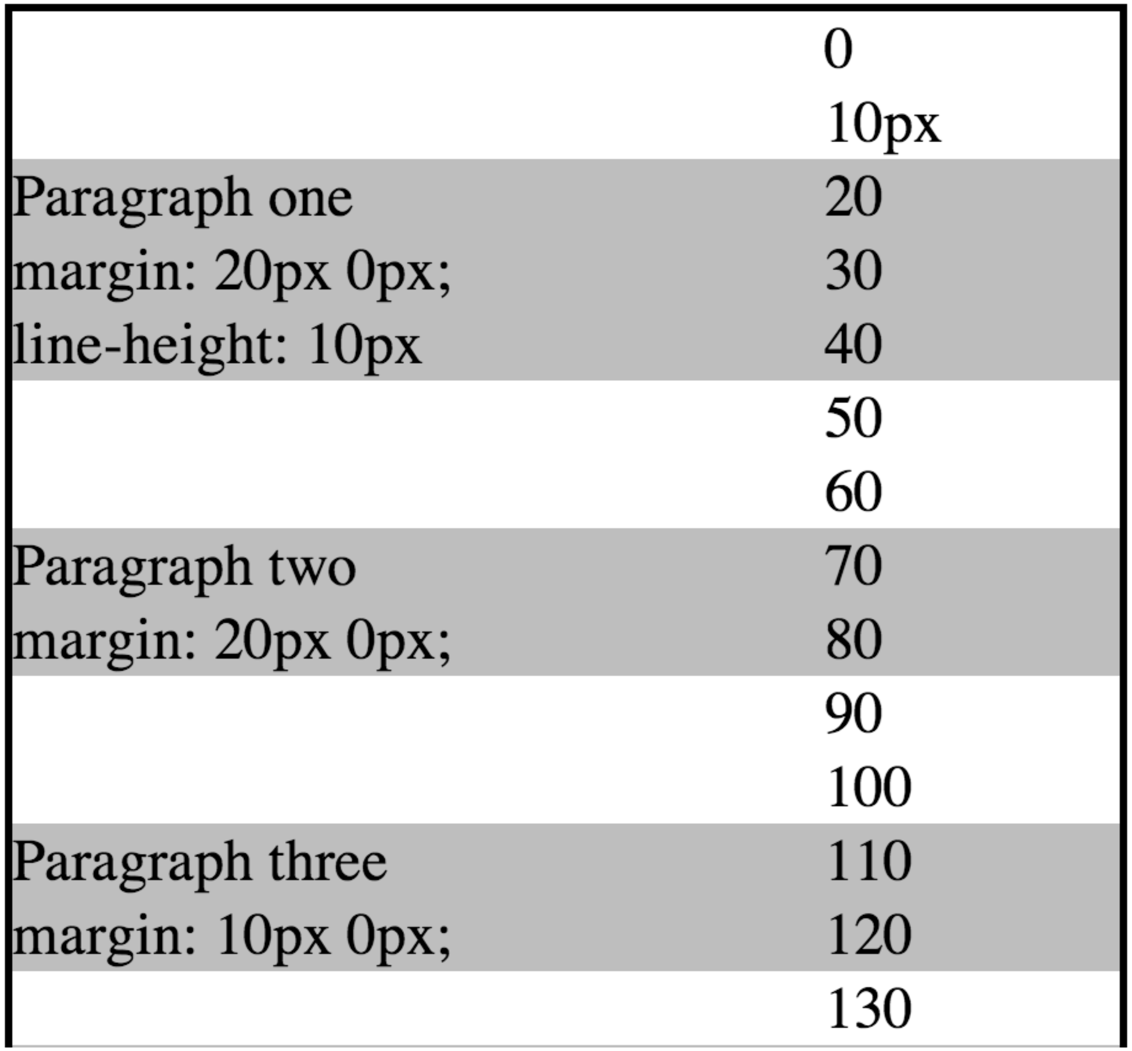
Note scale on right.

1px high divs separate paragraphs.

marexp1.html

(Scale is produced with a "float".)

Experiment, continued



0
10px
Paragraph one 20
margin: 20px 0px; 30
line-height: 10px 40
50
60
Paragraph two 70
margin: 20px 0px; 80
90
100
Paragraph three 110
margin: 10px 0px; 120
130

Same document but with `display` of separator `div`s set to `none`.

What's different?

Experiment, continued

"In CSS, the adjoining margins of two or more boxes can combine to form a single margin. Margins that combine this way are said to collapse, and the resulting combined margin is called a *collapsed margin*." (CSS 2.1 REC, 8.3.1)

There are lots of rules about collapsing margins.

Horizontal margins never collapse.

➔Examine the experiment's CSS machinery.

Experiment further: Use `outline` instead of `border` for the separating divs.

(Just a placeholder so whm doesn't forget to do Quiz 6a, some examples on the board, and then Quiz 6b!)

Styling Text

`font-size`

We've seen examples with `font-size` expressed in pixels (px) but ***physical units*** can be used, too:

- in: inches
- cm: centimeters
- mm: millimeters
- pt: points—1/72 of an inch
- pc: picas—1/6 of an inch (12 points)

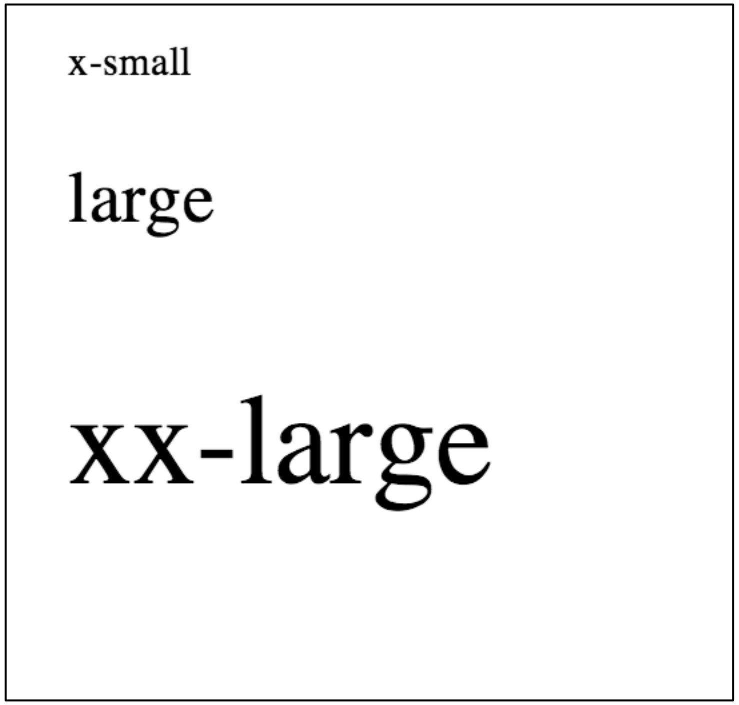
Demo: fontsize1.html

There are lots of details with *anchoring* and the *reference pixel*. See 4.3.2 in the REC.

`font-size`, continued

`font-size` can be specified with keywords that specify absolute sizes.

<p style="font-size: **<u>x-small</u>**">
x-small
<p style="font-size: **<u>large</u>**">
large
<p style="font-size: **<u>xx-large</u>**">
xx-large


x-small
large
xx-large

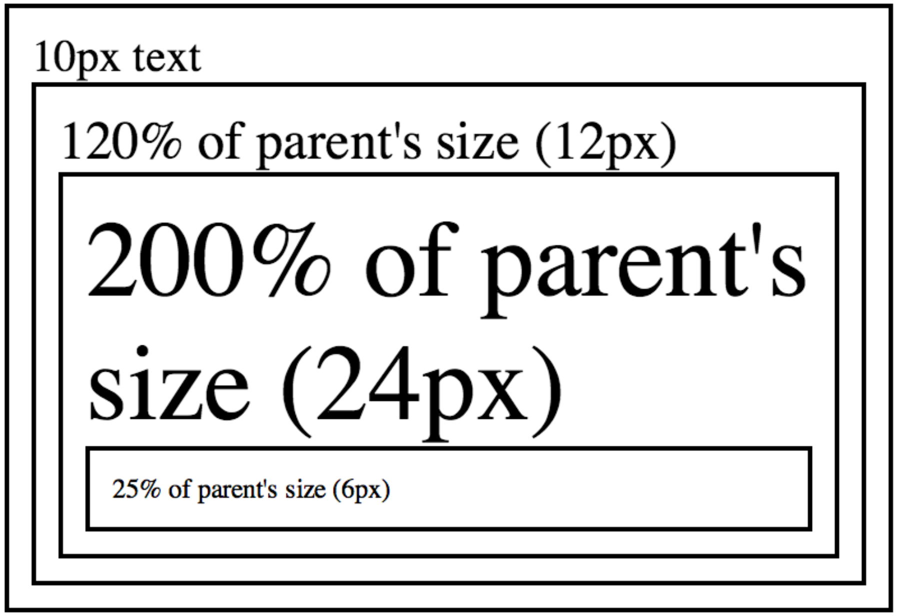
Note that margins are proportional to font-size.

fontsize2.html

`font-size`, continued

`font-size` can be specified as a percentage of the parent's `font-size`. Note the nesting of the `div`s.

<div style="font-size: 10px">
10px text
<div style="font-size: **<u>120%</u>**">
120% of parent's size (12px)
<div style="font-size: **<u>200%</u>**">
200% of parent's size (24px)
<div style="font-size: **<u>25%</u>**">
25% of parent's size (6px)

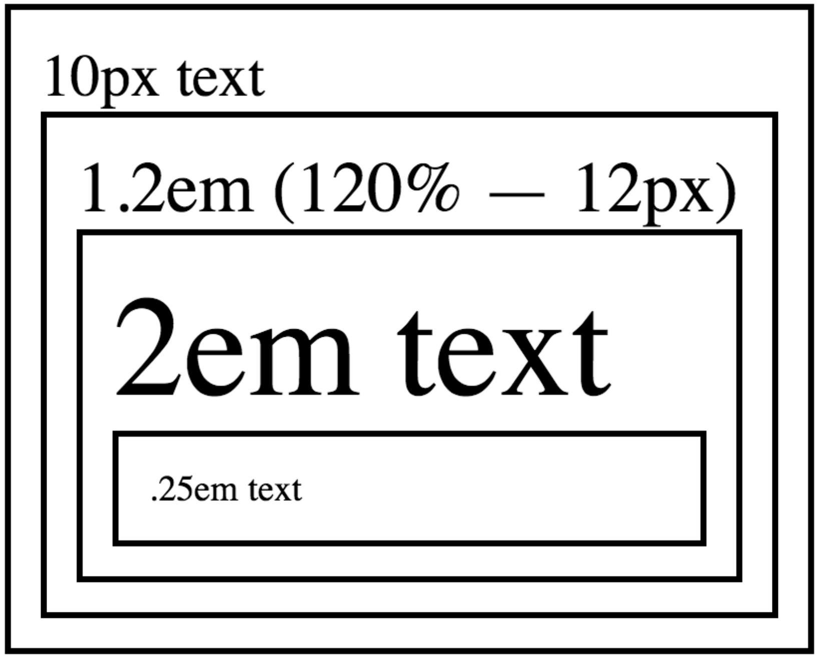

10px text
120% of parent's size (12px)
200% of parent's size (24px)
25% of parent's size (6px)

fontsize3.html

`font-size`, continued

The unit `em` provides another way to specify sizing relative to the parent's font-size.

<div style="font-size: 10px">
10px text
<div style="font-size: **<u>1.2em</u>**">
1.2em (120% &mdash; 12px)
<div style="font-size: **<u>2em</u>**">
2em text
<div style="font-size: **<u>.25em</u>**">
.25em text



fontsize4.html

A `font-size` scheme

HFHC recommends specifying `font-size` for `body` and then making everything relative to it:

```
body { font-size: 12pt }
h1    { font-size: 150% }
h2    { font-size: 120% }
…
```

Poll: What `font-size` practices do web developers in our class use?

`font-weight`

The initial value of `font-weight` is normal. Setting it to `bold` produces bold text.

`lighter` and `bolder` produce relative lightening and darkening IF a font is available in additional weights.

Numeric weights 100, 200, ..., 900 can also be specified. `normal` is defined as 400, `bold` is 700.

How many weights does each provide?

`font-weight`, cont.

default	Helvetica Neue	Myriad Pro
this is 100	this is 100	this is 100
this is 200	this is 200	this is 200
this is 300	this is 300	this is 300
this is 400	this is 400	this is 400
this is 500	**this is 500**	this is 500
this is 600	**this is 600**	**this is 600**
this is 700	**this is 700**	**this is 700**
this is 800	**this is 800**	**this is 800**
this is 900	**this is 900**	**this is 900**

fontweight1.html

`font-style`

Any of three values can be specified for `font-style`:

normal
italic (not italic<u>S</u>)
oblique

An italic style of a font is typically slanted, possibly lighter, and usually has accentuated serifs, if serifs are present as they are in this text.

An oblique style is simply slanted.

`font-family`

We've seen `font-family` declarations like these:

```
font-family: sans-serif
font-family: cursive
```

These use CSS <u>keywords</u> to specify a <u>font family</u> that should be used for the selected elements.

`font-family` is an inherited property.

`font-family`, continued

The CSS 2.1 REC defines five *generic font families*, with these associated keywords:

```
serif
sans-serif
monospace
cursive
fantasy
```

Remember: Don't put quotes around CSS keywords!

`font-family`, continued

`serif`: "Glyphs of serif fonts tend to have finishing strokes, flared or tapering ends, or have actual serifed endings." Proportionately spaced, typically.

Times: ABCIefghEFGH

Palatino: ABCIefghEFGH

Charlemagne: ABCIEFGH

What are some differences between them? What's in common?

`font-family`, continued

`sans-serif`: "...tend to have stroke endings that are plain—with little or no flaring, cross stroke, or other ornamentation." Typically proportionately spaced.

Calibri:	ABCIefgh
Helvetica:	ABCIefgh
Geneva:	ABCIefgh

`font-family`, continued

`monospace`: "The sole criterion of a monospace font is that all glyphs have the same fixed width. "

Courier New:	ABCIefghW
Lucida Console:	ABCIefghW
Prestige Elite Std:	ABCIefghW

Part of the art is making thin characters wide and wide characters thin. Note the 'i's and 'W's.

`font-family`, continued

`cursive`: "Glyphs in cursive fonts ... generally have either joining strokes or other cursive characteristics beyond those of italic typefaces. ... the result looks more like handwritten pen or brush writing than printed letterwork."

Brush Script Std: ABCIefgh

Casual: ABCIefgh

Edwardian Script ITC: ABCIefgh

Sidebar: Phototypesetters

Phototypesetters preceded laser and inkjet printers.

One design used black celluloid discs with transparent characters arranged along a ring. The disc rotated to position each character then light was projected through it onto photosensitive paper. Optical zoom produced varying sizes. There was one face per disc, with four mounts.

UA CS had one through the 70s and 80s. Professors could submit *camera-ready copy* of articles to publishers. Alternative: Fixed-width line printer output that was manuallly retyped. Could you correctly retype a full page of Java code?

Many of the books on the wall in GS 701 were phototypeset.

`font-family`, continued

`fantasy`: "Fantasy fonts are primarily decorative while still containing representations of characters."

Cracked:	ABCIefgh
Casual:	ABCIefgh
Curlz MT:	ABCIefgh

`font-family`, continued

Here are the five families:
serif, sans-serif, monospace, cursive, fantasy

Classify these:

American Typewriter: ABCIefgh

OCR A Std: ABCIefgh

Gabriola: ABCIefgh

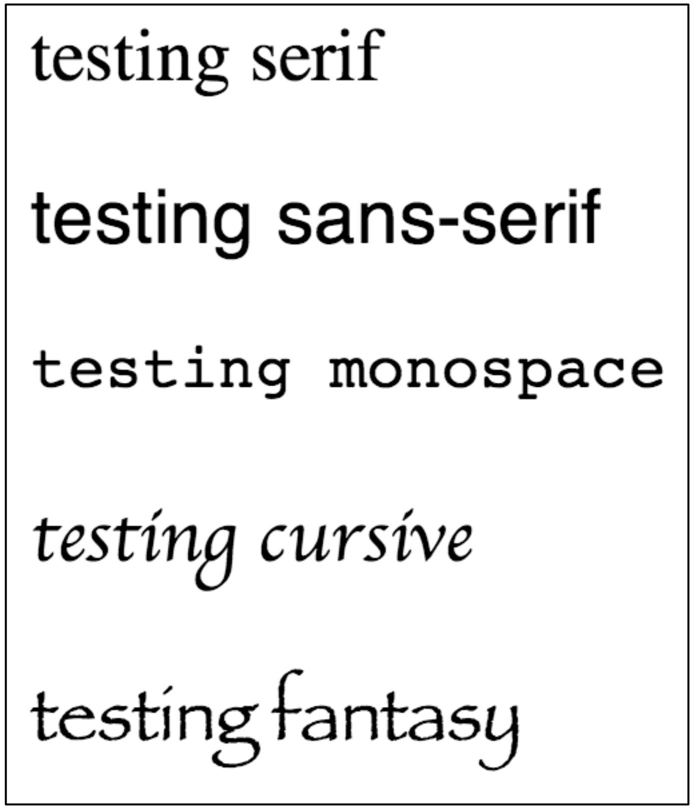
Simbraille: ⠁⠃⠉⠊⠑⠋⠛⠓

The five-family taxonomy is far from perfect!

`font-family`, continued

A font that meets the family characteristics is guaranteed to be present for all generic families but it is <u>not</u> guaranteed it will be the same in all browsers.

Chrome on Mac



testing serif

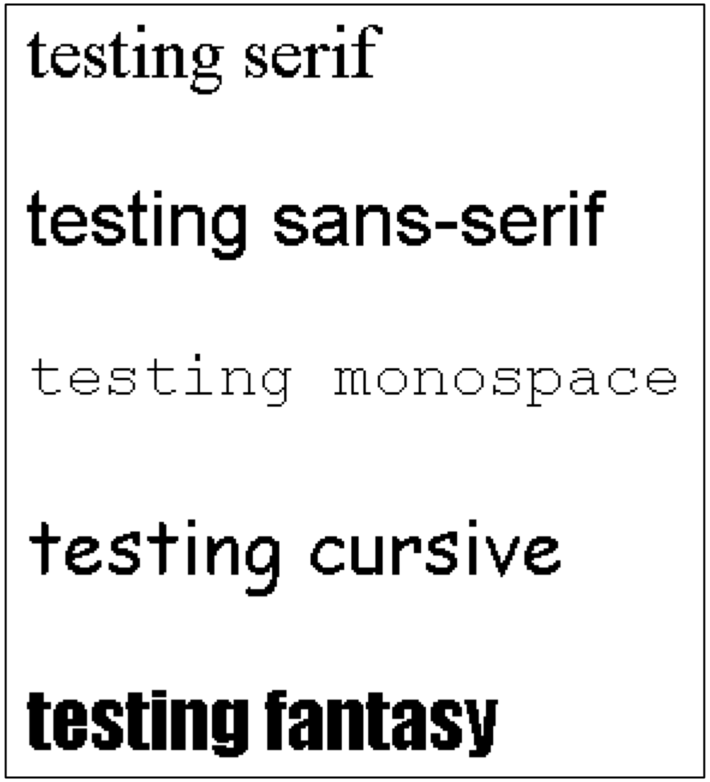
testing sans-serif

testing monospace

testing cursive

testing fantasy

Chrome on Windows



testing serif

testing sans-serif

testing monospace

testing cursive

testing fantasy

allfams.html

`font-family`

A font-family declaration has this general form:

font-family: 1st choice, 2nd choice, ..., last choice

Example from HFHC, p. 317:

font-family: Verdana, Geneva, Arial, sans-serif

Each family is tried in turn until one that's present is found.

<u>The last entry should always be one of the five generic families</u>, which are all guaranteed to be present, to assure basic control over appearance.

font-family

At hand...

font-family: 1st choice, 2nd choice, ..., last choice

Font <u>names</u> can quoted or not, but don't quote the generic font-family keywords!

font-family: "Helvetica Neue", "Courier New", "Gill Sans Ultra Bold", "Gabriola", fantasy

font-family: Helvetica Neue, Courier New, Gill Sans Ultra Bold, Gabriola, fantasy

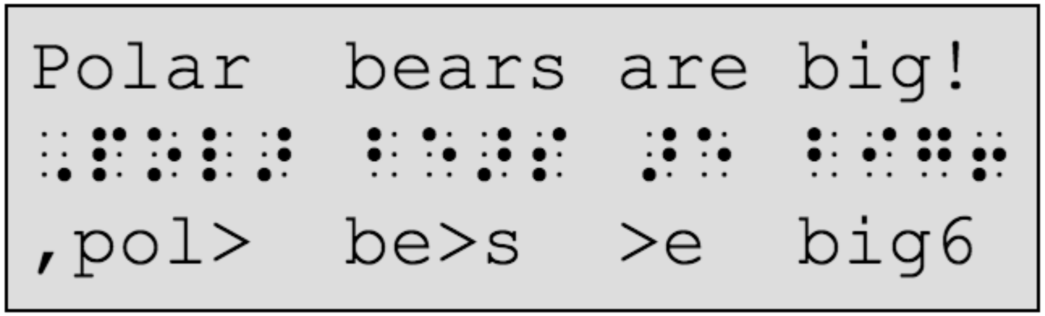
Do the examples above show a practical series?

fam2.html

@font-face

"The `@font-face` rule allows for linking to fonts that are automatically fetched and activated when needed."

```
<style>
  @font-face {
    src: url("fonts/SimBraille.ttf");
    font-family: Braille; }
</style>
<div class="brlx">Polar  bears are big!</i>
<span style="font-family: Braille">,pol> be>s >e big6
</span>
,pol>  be>s  >e  big6
</div>
```


Polar bears are big!

,pol> be>s >e big6

fontface1.html

`@font-face`, continued

`@font-face` is defined in *CSS Fonts Module Level 3*, so it's considered to be part of CSS 3, not CSS 2.1.

It is an "at-rule".

Various browsers support various font file formats, but modern browsers support .woff files (Web Open Font Format, a REC).

Remember that such fonts have to be fetched over the net. simbraille.ttf: 9.5K, ArialUnicode.ttf: 23M!

HFHC pp. 323-326 has more on @font-face and font files.

The `font` shorthand

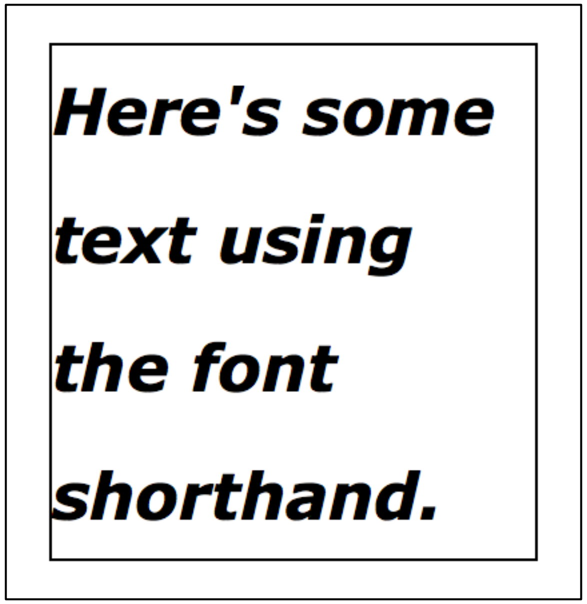
The `font` shorthand property can set many properties:

```
font-style
font-weight
font-family
font-size
font-stretch
font-variant
line-height
```

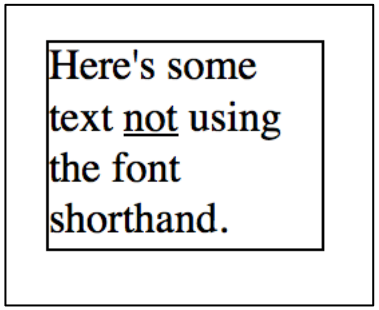
`font`, continued

Example:

```
font: bold italic 1.5em/2em
  Verdana, Geneva, Arial, sans-serif;
```



Here's some text using the font shorthand.



Here's some text not using the font shorthand.

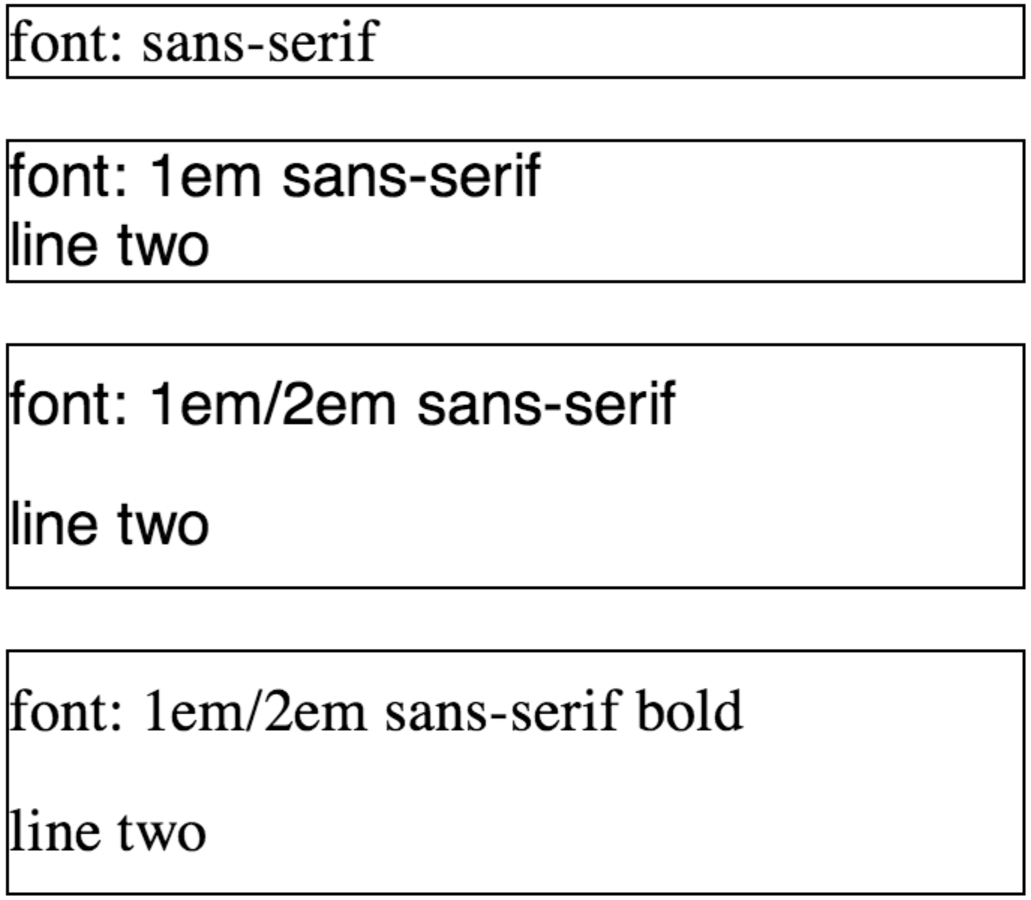
font-family list must be last!

font1.html

`1.5em/2em` sets `font-size` and `line-height`, respectively.

`font`, continued

The <u>family must follow a font size</u> *and* <u>that pair must be last</u>! Good and bad examples:



font: sans-serif

font: 1em sans-serif
line two

font: 1em/2em sans-serif

line two

font: 1em/2em sans-serif bold

line two

font2.html

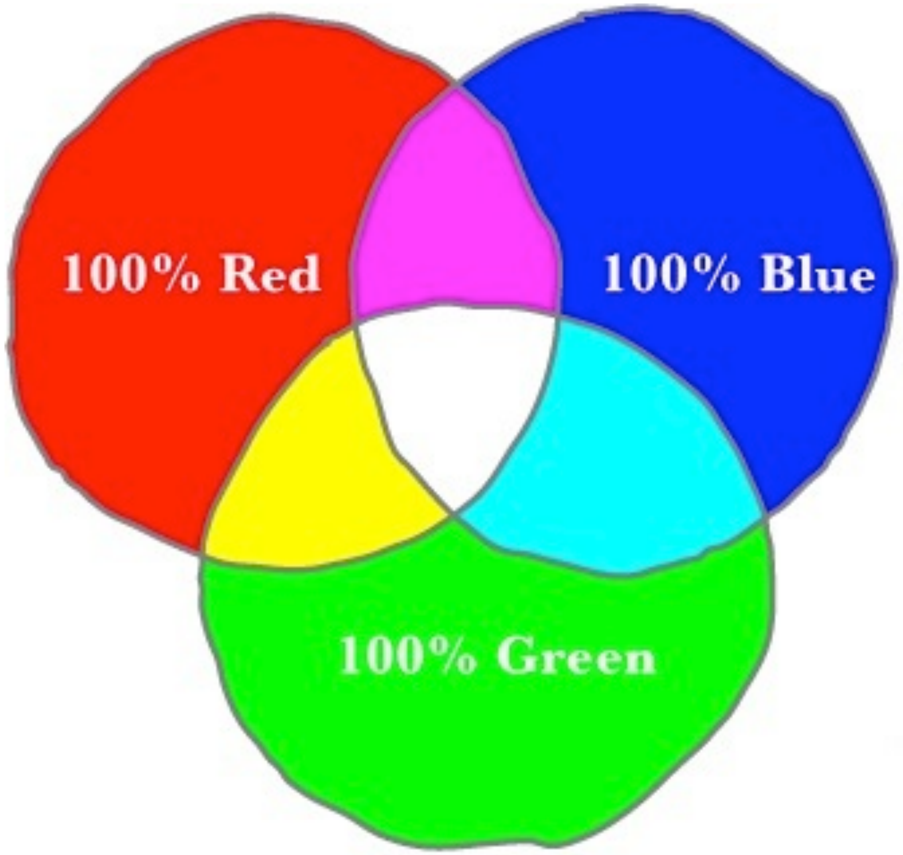
Color

Color basics

Conventional technology for displays produces color by using varying intensities of red, green, and blue light.

100% (full brightness) red, green and blue produce a pure white.

What are the colors produced by the 100% pairings?

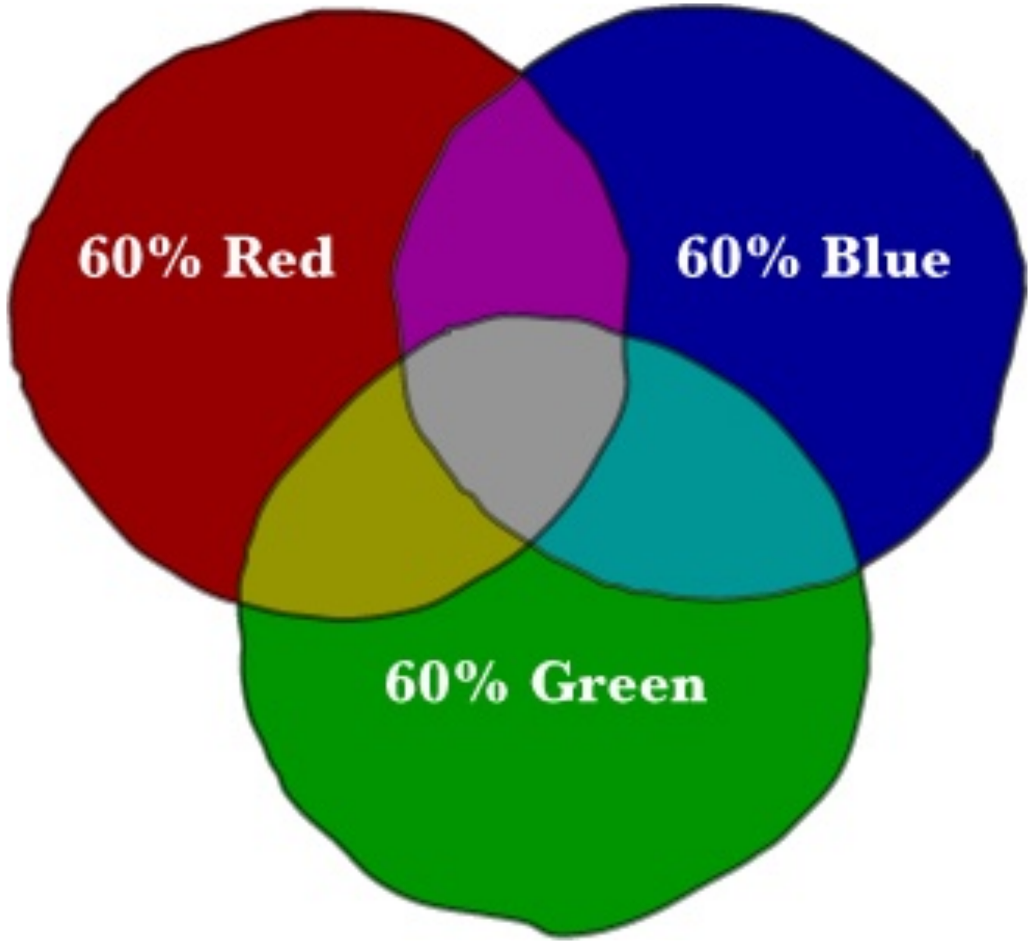


Diagrams from HFHC pp. 340-341

Color basics, continued

Lower intensities produce muted colors.

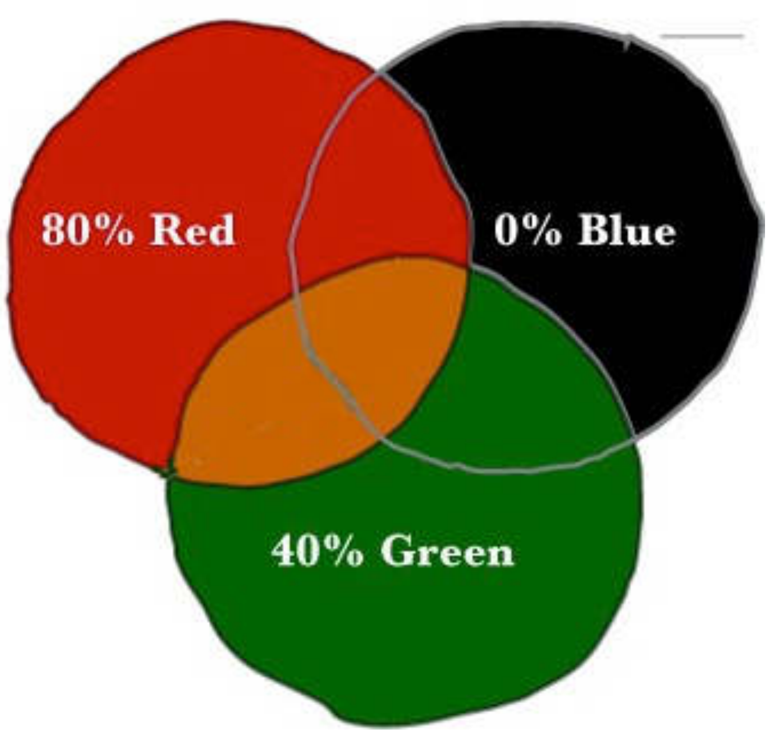
What's
60% red plus
60% green plus
60% blue?



Color basics, continued

Combining lots of red with half as much green gives us an orange.

The HTML `input` element provides an easy way to experiment with RGB combinations:



```
<input type=color style="width: 200px; height: 200px">
```

<u>With Chrome</u> we'll see a platform-specific color picker, typically allowing red, green and blue values from 0-255.

color1.html

Colors in CSS

Here are some of the CSS properties that specify coloration:

```
color
background-color
border-color
text-decoration-color
```

`border-color` (a shorthand)

CSS 2.1 has 17 color keywords: aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, orange, purple, red, silver, teal, white, and yellow

Colors in CSS, continued

Here are RGB values for some of the colors:

red	255	0	0	
maroon	128	0	0	(dark red)
green	0	128	0	
navy	0	0	128	(dark blue)
purple	128	0	128	
fuchsia	255	0	255	(bright purple)
silver	192	192	192	(light gray)
black	0	0	0	

Use the "gear" to see RGB values in color2.html.

Colors in CSS, continued

CSS Color Module Level 3 defines about 150 color keywords:

aliceblue, antiquewhite, aqua, aquamarine, azure, beige, bisque, black, blanchedalmond, blue, blueviolet, brown, burlywood, cadetblue, chartreuse, chocolate, coral, cornflowerblue, cornsilk, crimson, cyan, darkblue, darkcyan, darkgoldenrod, darkgray, darkgreen, darkgrey, darkkhaki, darkmagenta, darkolivegreen, darkorange, darkorchid, darkred, darksalmon, darkseagreen, darkslateblue, darkslategray, darkslategrey, darkturquoise, darkviolet, deeppink, deepskyblue, dimgray, dimgrey, dodgerblue, firebrick, floralwhite, forestgreen, fuchsia, gainsboro, ghostwhite, gold, goldenrod, gray, green, greenyellow, grey, honeydew, hotpink, indianred, indigo, ivory, khaki, lavender, lavenderblush, lawngreen, lemonchiffon, lightblue, lightcoral, lightcyan, lightgoldenrodyellow, lightgray, lightgreen, lightgrey, lightpink, lightsalmon, lightseagreen, lightskyblue, lightslategray, lightslategrey, lightsteelblue, lightyellow, lime, limegreen, linen, magenta, maroon, mediumaquamarine, mediumblue, mediumorchid, mediumpurple, mediumseagreen, mediumslateblue, mediumspringgreen, mediumturquoise, mediumvioletred, midnightblue, mintcream, mistyrose, moccasin, navajowhite, navy, oldlace, olive, olivedrab, orange, orangered, orchid, palegoldenrod, palegreen, paleturquoise, palevioletred, papayawhip, peachpuff, peru, pink, plum, powderblue, purple, red, rosybrown, royalblue, saddlebrown, salmon, sandybrown, seagreen, seashell, sienna, silver, skyblue, slateblue, slategray, slategrey, snow, springgreen, steelblue, tan, teal, thistle, tomato, turquoise, violet, wheat, white, whitesmoke, yellow, yellowgreen

See them all in colors3.html!

The `rgb(...)` specification

An alternative to color keywords is a numerical RGB specification. Either percentage values or integers in the range 0-255 can be specified.

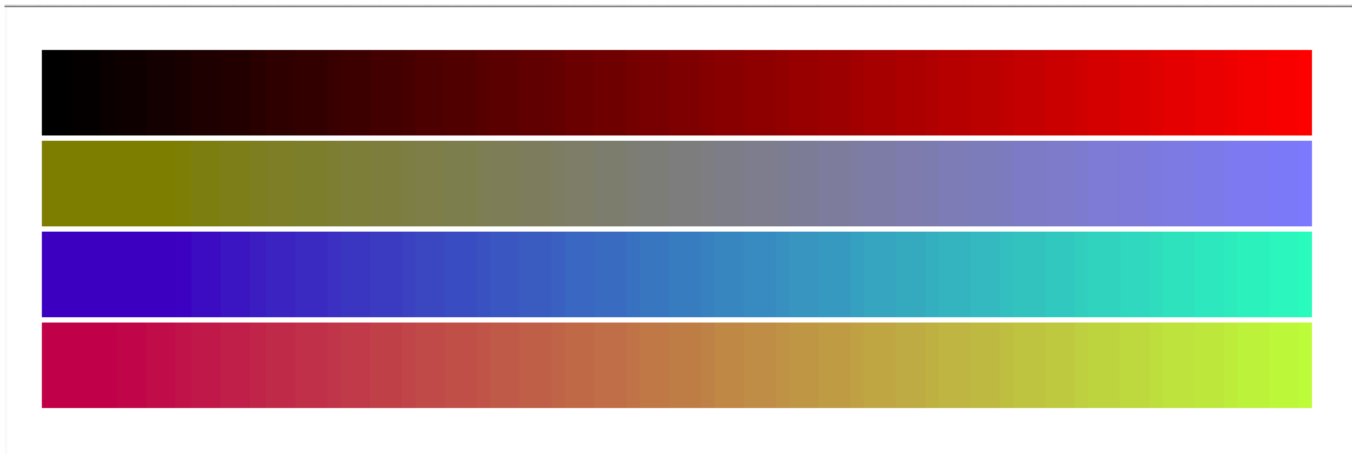
Three ways to say "navy":

```
color: navy
color: rgb(0%, 0%, 50%)
color: rgb(0, 0, 128)
```

You <u>cannot</u> mix percentages and integers, like `rgb(0, 0, 50%)`.

`rgb(...)`, continued

Problem: Using only what we've seen, produce this: (rgb2.html)



Transparency with `rgba(...)`

CSS Color Module Level 3 has an `rgba(...)` specification that allows opacity to be specified.

rgba(255, 0, 0, 1.0) is the same as rgb(255, 0, 0), a fully opaque red.

rgba(255, 0, 0, 0.2) is a 20% opaque red.

Experiment with different background colors using rgba1.html.

Hexadecimal color specification

Color can also be specified using three two-digit hexadecimal (base 16) values.

Here are three ways to make text purple:

```
color: purple
color: rgb(128, 0, 128)
color: #800080
```

A specification like `#ABC` means `#AABBCC`.

If you know hexadecimal, feel free to hex colors but we won't be using hex in this class.

HFHC covers hexadecimal on pp. 345-347.

The `display` property

The `display` property

An element's `display` property determines the type of element box used to display it.

By default, for example, the `p` and `div` elements have `block` display, `em` and `span` have `inline` display, and `style`'s `display` is none.

But, any element can be given any of the 19 different `display` types.

`display`, continued

Let's revisit blockinline.html and ...

- Give all elements `inline` display
- Give all elements `block` display
- Mix them up!

Experimentation

Experiments with `ul` in dispex.html:

- `display: inline`
- `display: inline-block`
  - Try `vertical-align: middle` and `top`
- `display: table-cell`
  - Add `vertical-align: middle`

Also, try `inline` display for `li`.

`display: none`

No element box is generated if an element's `display` is `none`.

The default `display` for `head` is `none`. No element boxes are generated for `head` or its descendants.

`display` is <u>not</u> inherited but `none` applies to all descendants.

`visibility`

The default value of `visibility` is `visible`. It is inherited.

If it's `hidden`, full-size boxes for the element and all descendants are generated but nothing is drawn for them.

Try it with dispex.html

Layout and positioning

Elements flow onto the page

Browsers "flow" elements onto a page in the order they appear in the document.

As we've seen, elements with block display generate line breaks whose sizes depend on the involved margins.

Elements with inline display flow onto a line until they reach the edge of their container, which causes a new line to be started.

Elements with inline display have horizontal margins but <u>no</u> vertical margins. The tallest element on a line determines the height of the line.

In various cases, elements flow into other elements.

flow1.html

Specifying `width`

The `width` property can be used to specify the width of the <u>content area</u> of an element.

`width` can be absolute or a percentage of the <u>containing block's content area width</u>.

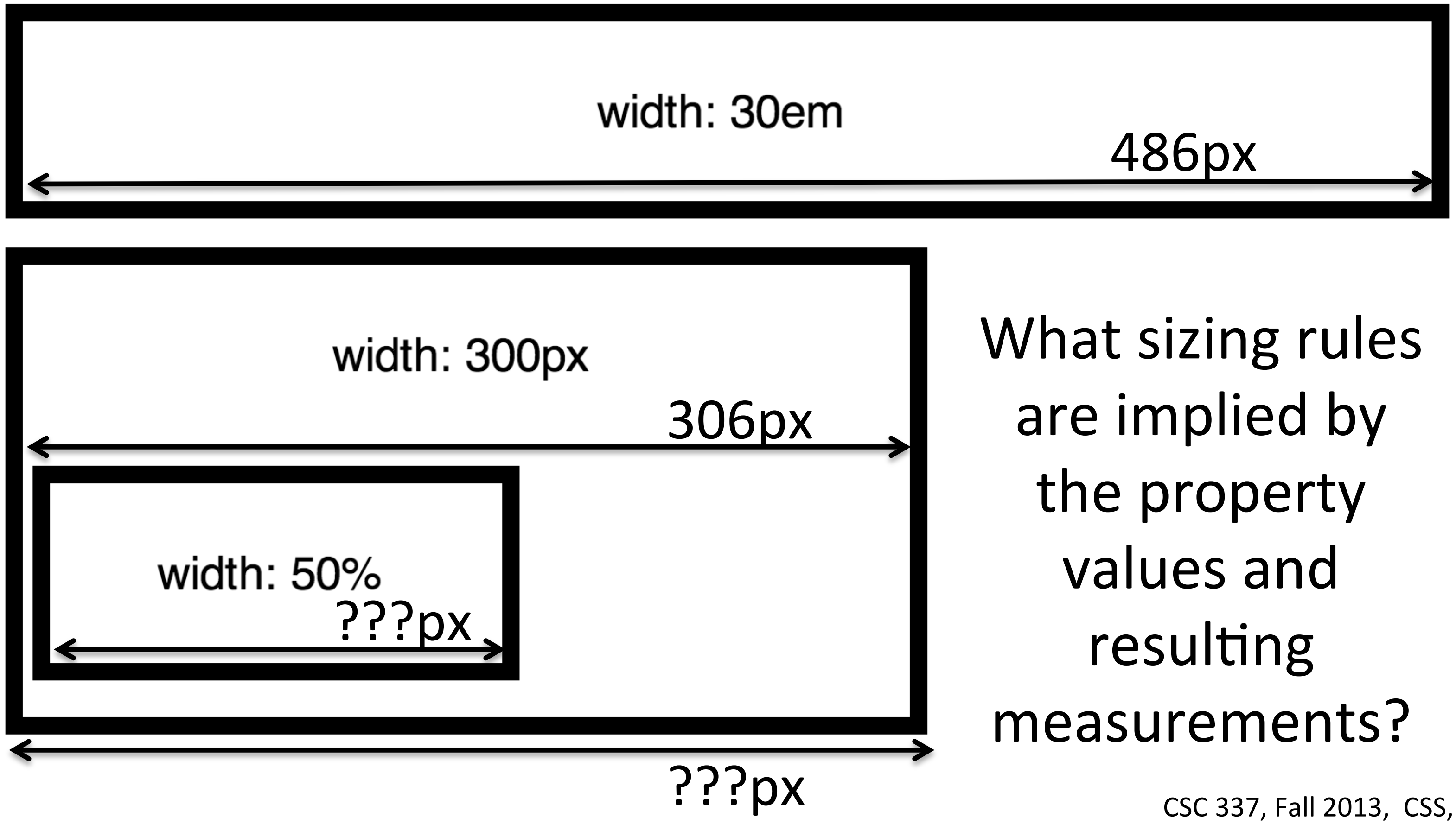
```
<div style="width: 30em">
    Lorem ipsum dolor sit amet...
</div>
<div style="width: 300px">
    Duis aute irure dolor...
    <div style="width: 50%">
        Duis aute irure dolor...
</div></div></div>
```

width1.html

`width`, continued

```
div { padding: 3px;
      border: 6px solid;
      margin: 10px 0px;
```

width1a.html



What sizing rules are implied by the property values and resulting measurements?

Floats

"The `float` CSS property specifies that an element should be taken from the normal flow and placed along the left or right side of its container, where text and inline elements will wrap around it."—MDN

Let's try float1.html

Floats, continued

Things to observe or try in float1.html:

- The paragraph elements, with silver backgrounds, extend the full width of the page, under the float.
- The text and images in the paragraphs do <u>not</u> go under the float.
- What happens if the float is moved lower in the HTML file?
- What happens with a second float to the right? How about a float to the left?
- With a bigger margin will text go to the right of a float:right?

Next: Match float2.html

~~Field Trip!~~

(Postponed due to government shutdown.)

The `clear` property

The `clear` property is used to specify that an element is to be positioned so that it is <u>not beside a float</u>.

The declaration `clear: left` effectively says, "Place me far enough down the page so that no float is on my left." Or, "Get me clear of floats on my left."

`clear` can be `left`, `right`, `both`, `none`, or `inherit`.

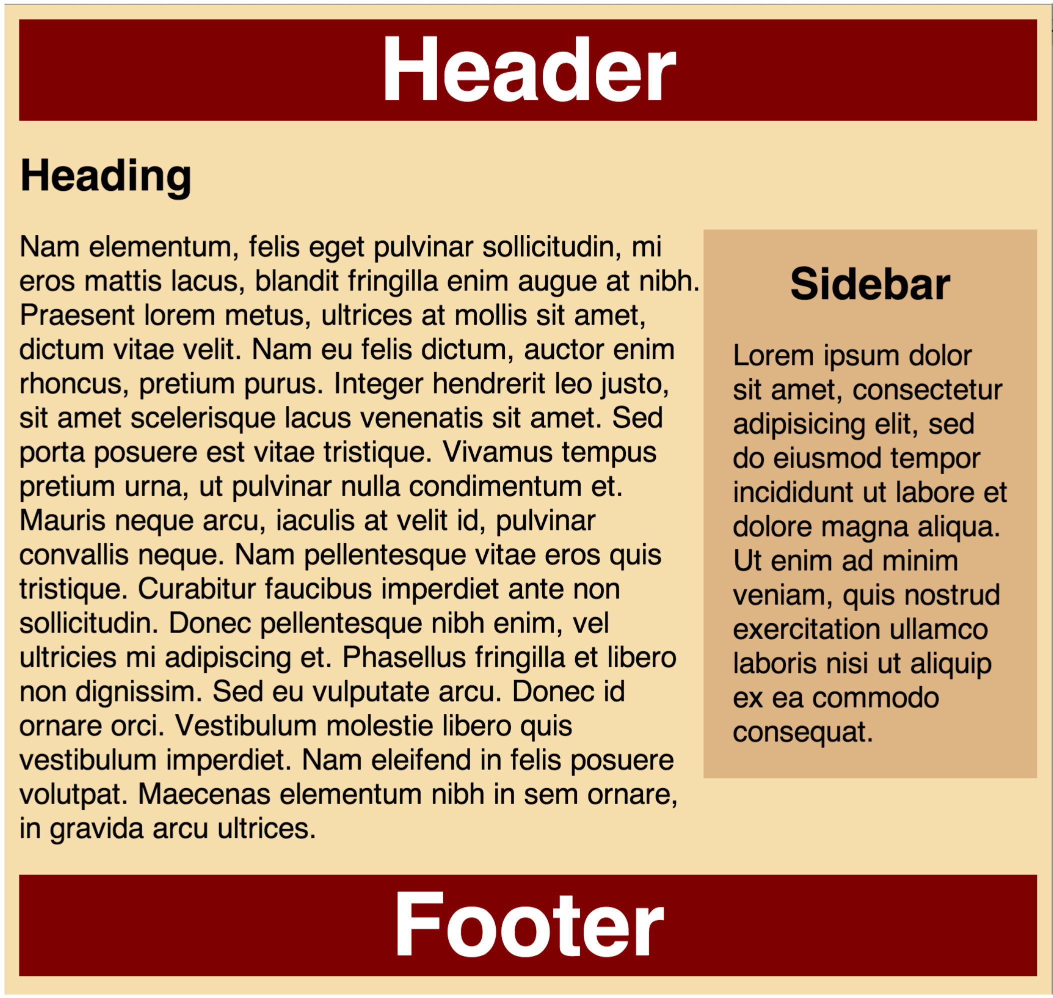
Problem: Run a series of specials down the right side of float1.html.

`clear`, continued

This is float5.html.

Try resizing and notice the interaction between the sidebar (a float) and the footer.

How can we fix it with a `clear`?


Header

Heading

Nam elementum, felis eget pulvinar sollicitudin, mi eros mattis lacus, blandit fringilla enim augue at nibh. Praesent lorem metus, ultrices at mollis sit amet, dictum vitae velit. Nam eu felis dictum, auctor enim rhoncus, pretium purus. Integer hendrerit leo justo, sit amet scelerisque lacus venenatis sit amet. Sed porta posuere est vitae tristique. Vivamus tempus pretium urna, ut pulvinar nulla condimentum et. Mauris neque arcu, iaculis at velit id, pulvinar convallis neque. Nam pellentesque vitae eros quis tristique. Curabitur faucibus imperdiet ante non sollicitudin. Donec pellentesque nibh enim, vel ultricies mi adipiscing et. Phasellus fringilla et libero non dignissim. Sed eu vulputate arcu. Donec id ornare orci. Vestibulum molestie libero quis vestibulum imperdiet. Nam eleifend in felis posuere volutpat. Maecenas elementum nibh in sem ornare, in gravida arcu ultrices.

Sidebar

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Footer

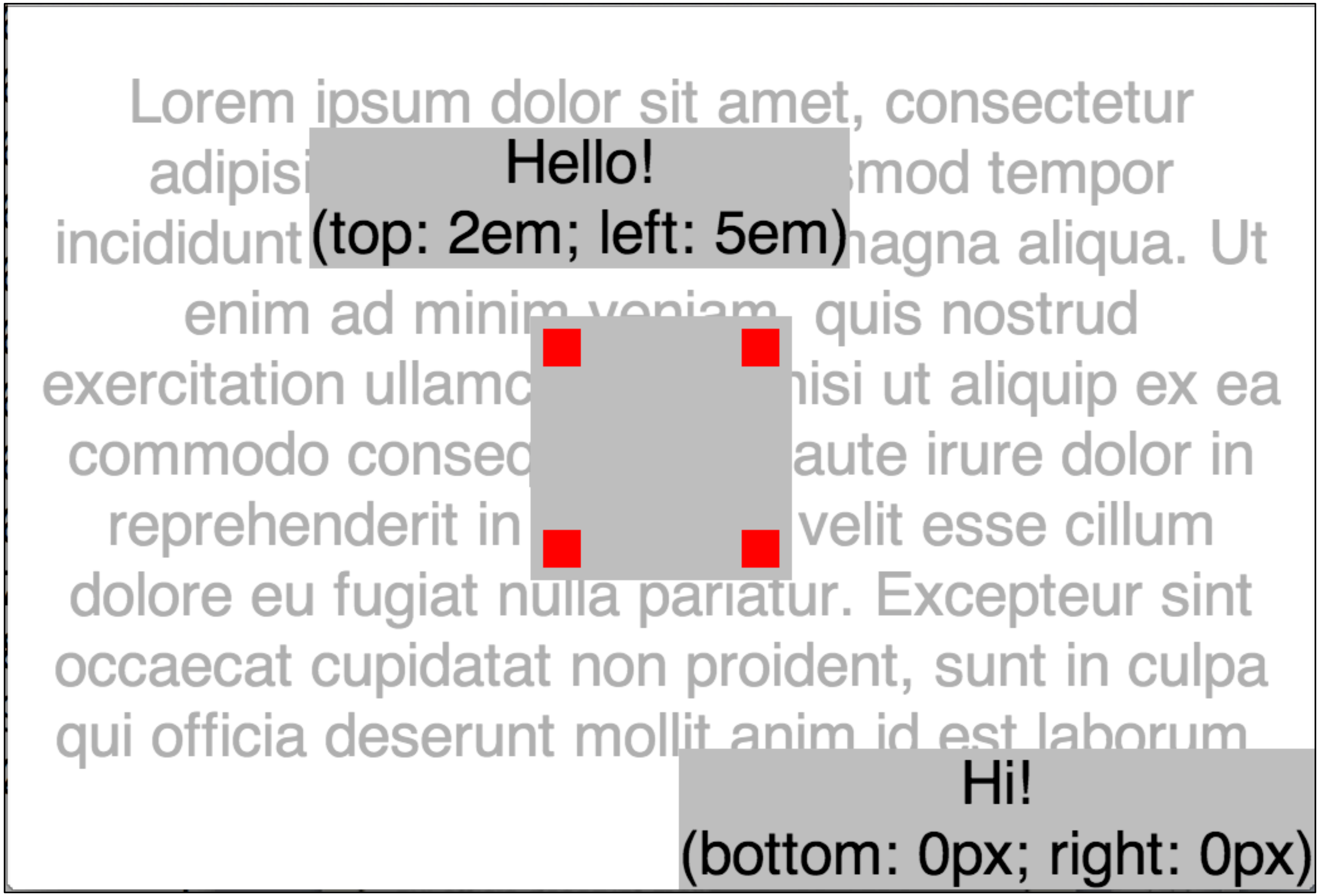
Absolute positioning

Elements have `position: static` by default, causing them to be laid out as part of the flow as we've seen thus far.

`position: absolute` causes an element to be taken out of the flow and positioned at a particular location.

Unlike a float, an absolutely positioned element has no effect on the layout of other elements.

Absolute positioning, continued



Lorem ipsum dolor sit amet, consectetur adipisi mod tempor incididunt agna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamc nisi ut aliquip ex ea commodo conseq aute irure dolor in reprehenderit in velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum

Hello!
(top: 2em; left: 5em)

Hi!
(bottom: 0px; right: 0px)

abs1.html

Absolute positioning, continued

```
.absbox { background-color: silver; text-align: center; margin: 0px }
.redsq { position: absolute; width: 10px; height: 10px;
         background-color: red}

<p style="color: rgb(70%, 70%, 70%); text-align: center">Lorem ...

<p class="absbox" style="position: absolute; top: 2em; left: 5em">
Hello!<br>(top: 2em; left: 5em)

<p class="absbox" style="position: absolute; bottom: 0px; right: 0px">
Hi!<br>(bottom: 0px; right: 0px)

<div class=absbox
style="position: absolute; top: 35%; left: 40%; width: 20%; height: 30%">

  <div class=redsq style="top: 5%; left: 5%"></div>
  <div class=redsq style="top: 5%; right: 5%"></div>
  <div class=redsq style="bottom: 5%; left: 5%"></div>
  <div class=redsq style="bottom: 5%; right: 5%"></div>
</div>
```

Absolute positioning, continued

Points to note:

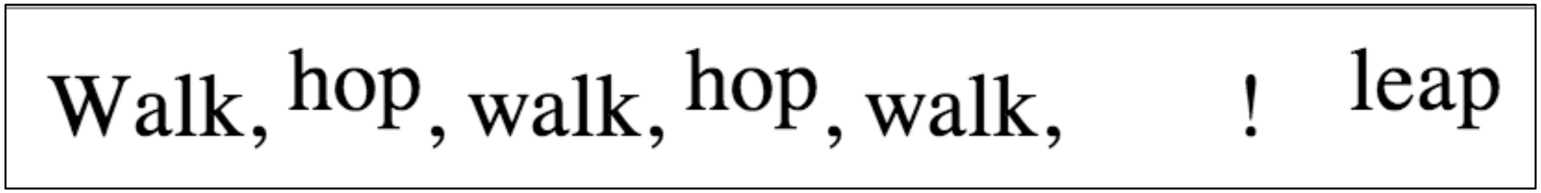
- `position: absolute`
- `top/right/bottom/left` specify distance from container's respective edge. They are known as *offset properties*.
- Percentages are used to center box.
- Red squares in centered box are absolutely positioned wrt. to their container.
- Negative offsets can be used!

Relative positioning

`position: relative` causes space to be reserved as normal for an element box but then the element box is drawn at an offset.

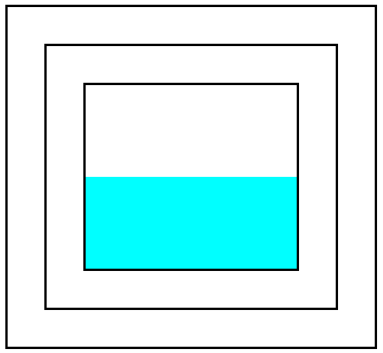
```
.up { position:relative; bottom: 0.3em }
.upover { position: relative;
            bottom: 0.3em; left: 3em; }
```

```
Walk, <span class=up>hop</span>,walk,
<span class=up>hop</span>, walk,
<span class=upover>leap</span>!
```



Positioning problem

We want

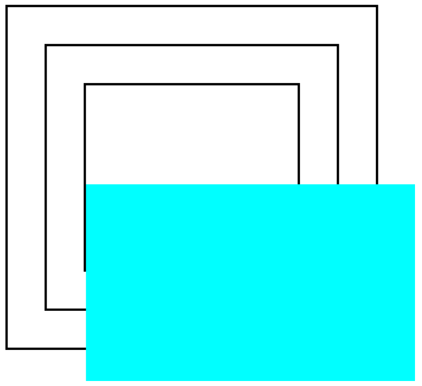


rel2a.html

We try

```
<style>
  div { border: 1px solid; margin: 1em;  }
  .half {position: absolute; top: 50%; bottom: 0px;
          width: 100%; margin: 0; border: none }
</style>
<div style="width: 10em">
  <div>
    <div style="height:5em">
      <div class="half" style="background-color: aqua"></div>
    </div>
  </div>
</div>
```

We get



rel2.html

Positioning problem, solved

The problem is that absolute positioning is relative to the nearest ancestor with non-`static` positioning.

We can use relative positioning with no offsets to cause the containing `div` to be that nearest ancestor:

```
<div style="width: 10em">
   <div>
      <div style="height: 5em; position: relative">
         <div class="half" style="background-color: aqua">
         </div>
```

Fixed positioning

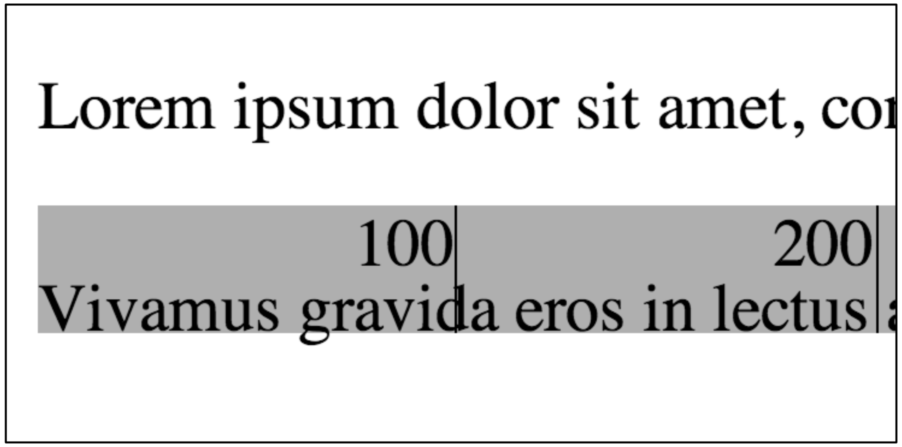
`position: fixed` causes an element box to be shown at a fixed position in the browser's window.

```
    p { line-height: 3em }
    .ruler { position: fixed; width: 100%;
                height: 30px; top: 50%; overflow: hidden;
                background-color: rgba(0,0,0,.3) }
    .px100 {  width: 99px; border-right: 1px solid;
                text-align: right; height: 30px; float: left }
```

(In the slide, "overflow: hidden;", "rgba(0,0,0,.3)" and "float: left" are underlined.)

```
<div class=ruler>
<div class=px100>100</div>
<div class=px100>200</div>
...
```

fixed1.html



Lorem ipsum dolor sit amet, co

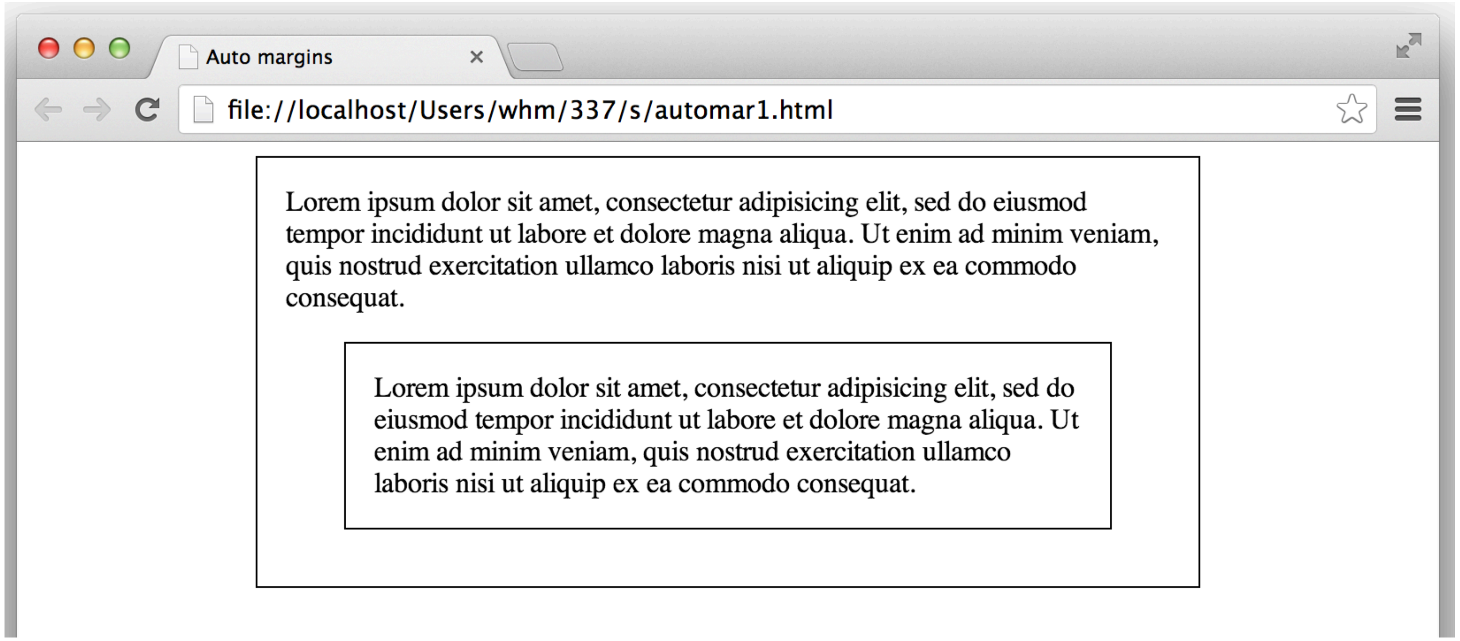
100 200

Vivamus gravida eros in lectus

Centering boxes

Element boxes can be horizontally centered in their container by using `auto` for horizontal margins:

```
div { border: thin solid; padding: 1em }
#content { width: 500px; margin: 0px auto }
#inner { width: 80%; margin: 1em auto }
```



Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

automar1.html

More on layout and positioning

Chapter 11 in HFHC works through a great series of examples in detail. Go through them!

Chapter 10 in *CSS: The Definitive Guide*, 3rd ed. by Meyer has great technical detail on floating and positioning, with lots of good stuff on offset properties.

Negative values for margins create lots of possibilities but the CSS 2.1 REC says "...there may be implementation-specific limits".

CSS tables provide a table-based layout mechanism. We'll not talk about that now, in favor of first looking at tables of data in HTML.

Field Trip!
(Emergency funding
approved!)

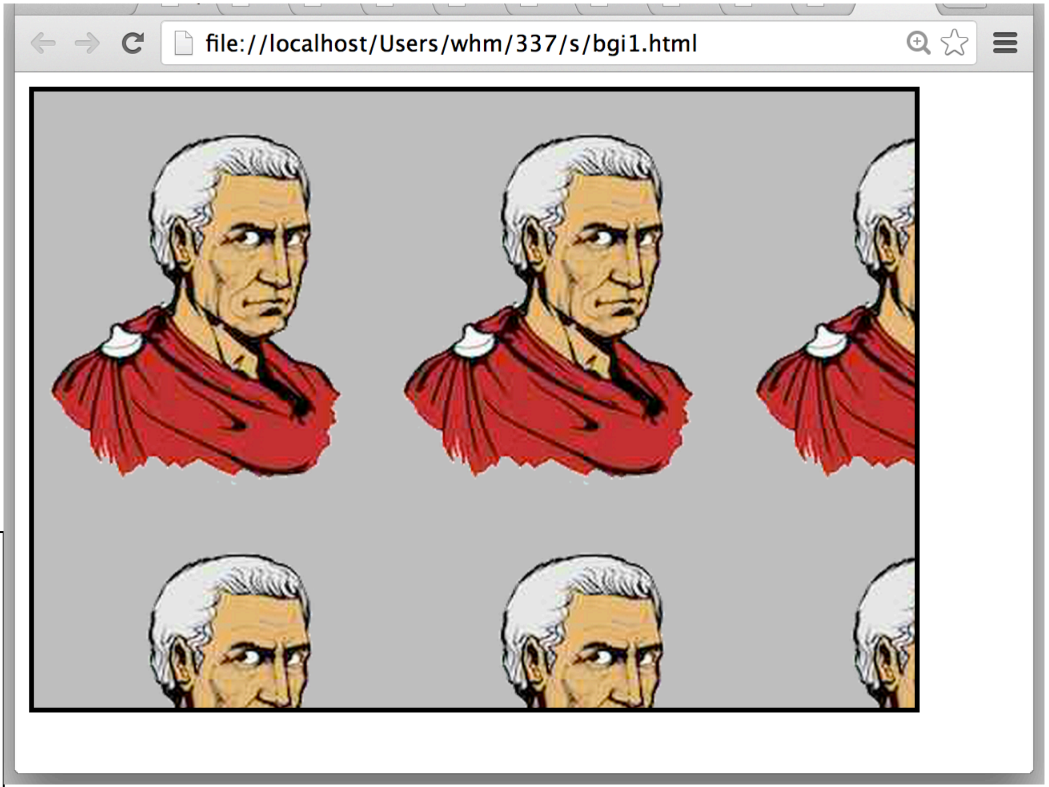
Background Images

background-image

A variety of effects can be created with background images. Here's a simple example:

```
#div1 {
  background-image: url(caesar200.png) /* 200x239px, w/xp */
  width: 500px;
  height: 350px;
  border: solid;
  background-color:
    silver;
}
<div id=div1>
</div>
```

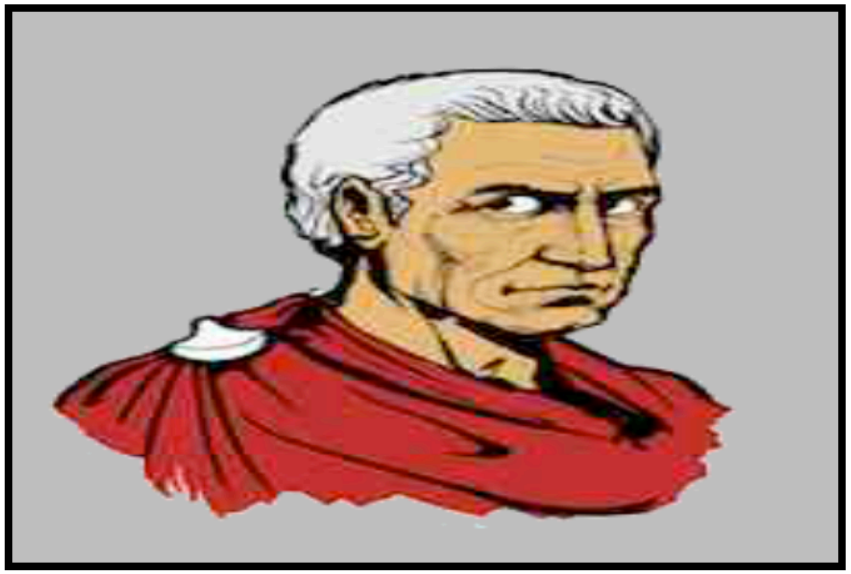
Note the horizontal and vertical repetition ("tiling"), and that the image is on top of the background color.

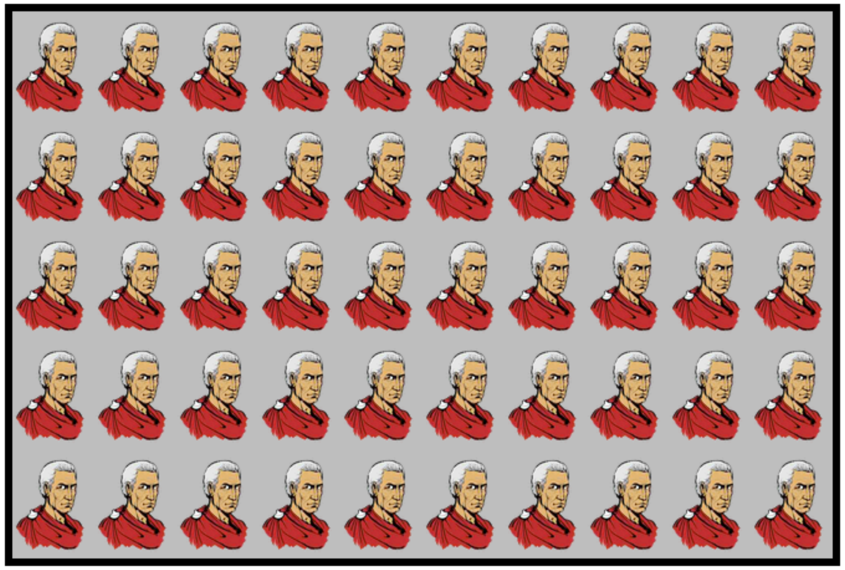

file://localhost/Users/whm/337/s/bgi1.html

bgi1.html

`background-size`

`background-size` can be specified, too:


background-size: 100% 100%


background-size: 10% 20%

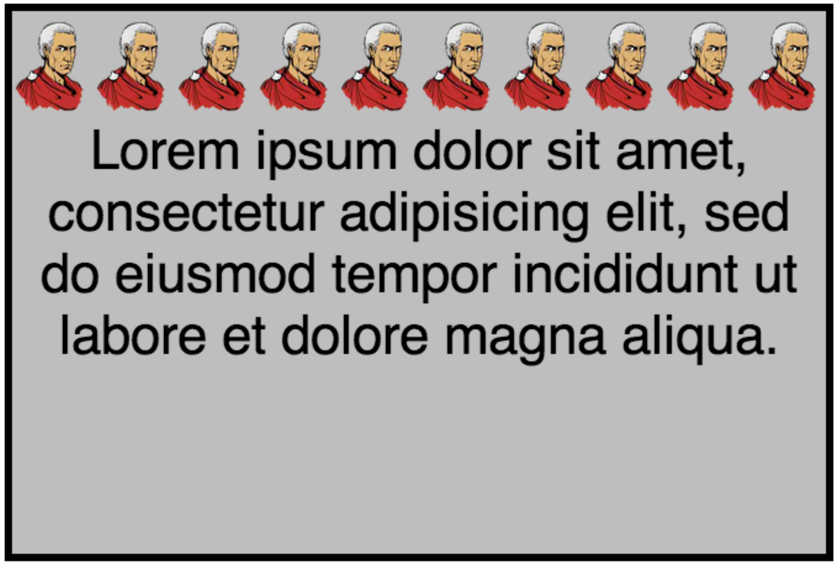
Note:

- Stretch-to-fit on left image.
- Ten across (100/10) and five high (100/20) on right.

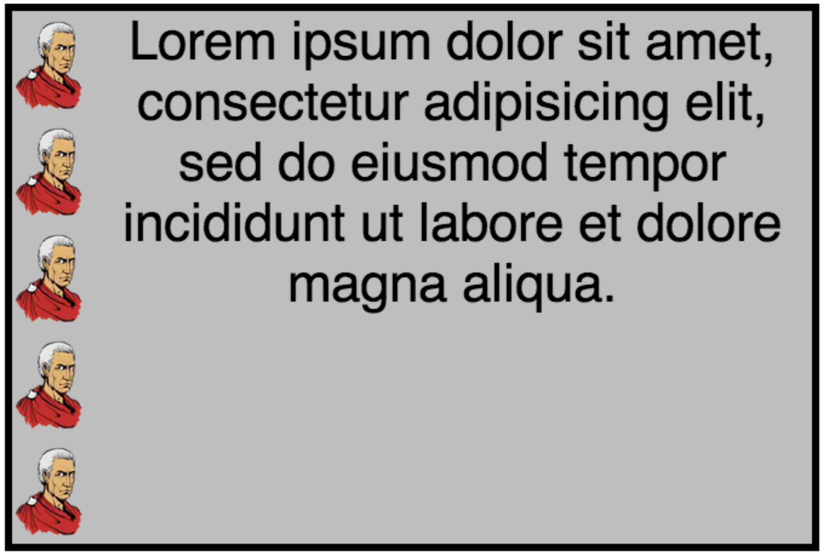
bgi2.html

`background-repeat`

`background-repeat` **can be** `repeat-x`, `repeat-y`, **or** `repeat-none`, **and more.**


Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

background-repeat: repeat-x
font-size: 20px
padding-top: 40px
height: 160px


Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

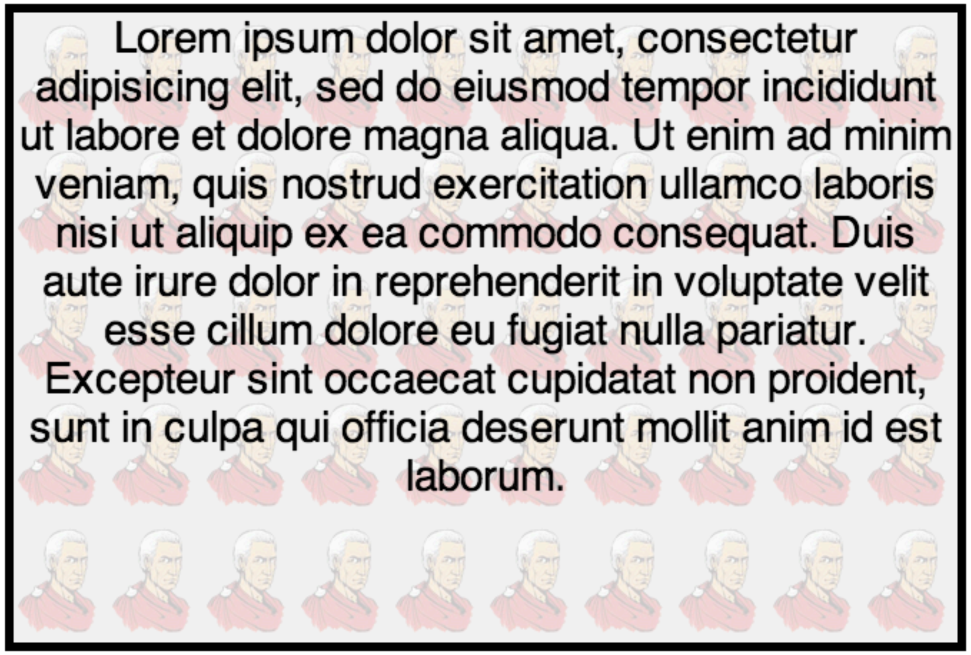
background-repeat: repeat-y
font-size: 20px
padding-left: 30px
width: 270px

bgi3.html

Background layering

If more than one background image is specified, they are layered. Consider this:

```
background-image:  url(white80pixel.png),
                   url(caesar200.png);
```



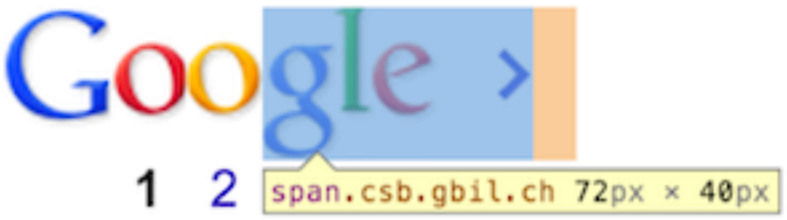
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

What do you think white80pixel.png is?

bgi4.html

`background-position`

Look at this element box:



Here's the markup, broken across lines:

<span class="csb gbil ch"
style="**background**: url(nav_logo161.png)
no-repeat; **background-position: -96px 0;**
width:71px;"></span>

The csb class adds these properties:

height: 40px; display: block

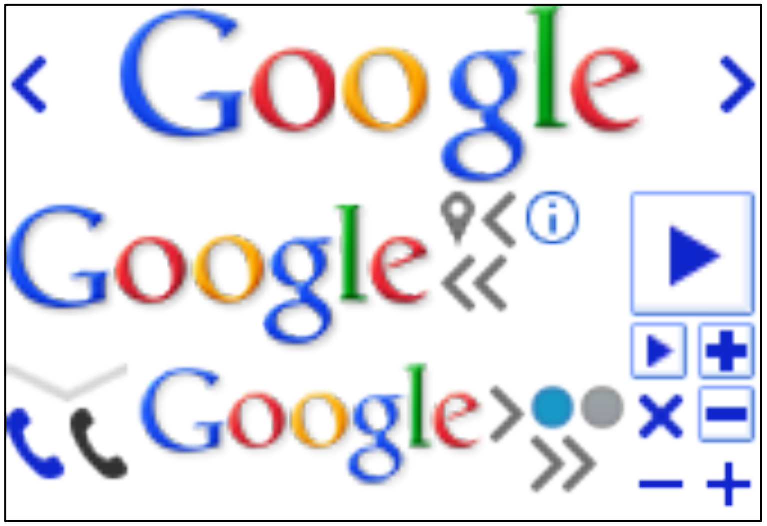
Note use of `background` shorthand!

`background-position`, continued

At hand:

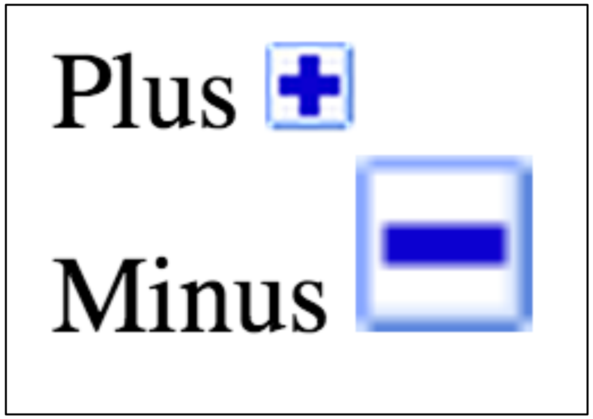
```
background: url(nav_logo161.png) no-repeat;
background-position: -96px 0;
width: 71px; height: 40px;
```

Here's the top of nav_logo161.png, with border added:



`background-position`, continued

```
.pic {   background: url(nav_logo161.png) no-repeat;
     display: inline-block; }
.pic1x {
     width: 13px; height: 13px;
     }
.pic2x { /* everything doubled */
     width: 26px; height: 26px;
     background-size: 334px;
     }
```



bgi5.html

```
Plus <span class="pic pic1x"
   style="background-position: -153px -70px;"></span>
<br>
Minus <span class="pic pic2x"
  style="background-position: -306px -168px;"></span>
```

The Cascade

What is "the cascade"?

Given a collection of CSS rules for a page it's possible that more than one declaration will apply to a given property of a given element.

"The *cascade* takes a unordered list of declared values for a given property on a given element, sorts them by their declaration's precedence [...] and outputs a single cascaded value."

—CSS Cascading and Inheritance Level 3

Specificity

Each selector has a *specificity* that is expressed as a 4-tuple, like this: 0,0,0,0

We compute the specificity of a selector like this:

- Add 0,1,0,0 for every id attribute in the selector.
- Add 0,0,1,0 for every class, attribute, and pseudo-class in the selector.
- Add 0,0,0,1 for every element and pseudo-element in the selector.

The specificity of an inline style is 1,0,0,0.

Specificity, continued

Cheat sheet:

    +0,1,0,0 for each #id
    +0,0,1,0 for each .class and :p-class
    +0,0,0,1 for each element and ::p-element

What is the specificity of each of these?

```
#main > div { ... }
.x .y .z a { ... }
div li a { ... }
div.nav li a { ... }
p, #sidebar, .square { ... }
<span style="...">...
```

Specificity, continued

The simple way to compare specificities: If no values are greater than 9, just remove the commas.

> 0,1,0,0 > 0,0,9,4 (0100 > 0094)
> 0,0,1,1 > 0,0,0,3
> 0,1,0,1 > 0,0,2,0
> 1,0,0,0 > 0,N,N,N
> 0,0,1,0 > 0,0,0,20

Analogy:

> Paint all walls blue.
> Paint all south walls green.
> Paint all walls with no windows red.
> How about a south wall with no window?

Origins of rules

There are three "core" origins for rules:

Author (of the page)

These are rules from external stylesheets (<link>), embedded stylesheets (<style>) and inline styles (style=...)

User (of the browser)

Some browsers allow the user to specify a stylesheet. We haven't seen an example of this yet.

User agent stylesheet (the browser itself)

The rules built into the browser, like shown in Appendix D of the CSS 2.1 REC.

The cascade, by example

An example of the cascade to compute `color` for this paragraph:

```
<div><p class=x> ...
```

(1) Find all property declarations for `color` in all rules having selectors that match the element.

```
p { color: maroon }        /* <link rel=stylesheet> */
div p { color: green }     /* <style> line 10 */
p.x { color: violet }      /* <style> line 14 */
.x { color: orange }       /* <style> line 17*/
p { color: black }         /* browser */
p { color: blue }          /* user stylesheet */
```

Example, continued

(2) Order by origin, with this ascending order of importance: user agent, user, author:

```
p { color: black }          /* browser */
p { color: blue }           /* user stylesheet */
p { color: maroon }         /* <link rel=stylesheet> */
div p { color: green }      /* <style> line 10 */
p.x { color: violet }       /* <style> line 14 */
.x { color: orange }        /* <style> line 17*/
```

If we considered only origin and "last one wins", what would be the `color` of <div><p class=x>...?

Example, continued

(2) continued...

These rules, all by the author of the page, have the same and highest importance. The value of color will come from one of these:

```
p { color: maroon }          /* <link rel=stylesheet> */
div p { color: green }       /* <style> line 10 */
p.x { color: violet }        /* <style> line 14 */
.x { color: orange }         /* <style> line 17*/
```

Example, continued

(3) Compute specificity. Recall:

```
   1,0,0,0 if style=...
  +0,1,0,0 for each #id
  +0,0,1,0 for each .class and :p-class
  +0,0,0,1 for each element and ::p-element
```

```
p { color: maroon }          /* 0,0,0,1 */
div p { color: green }       /* 0,0,0,2 */
p.x { color: violet }        /* 0,0,1,1 */
.x { color: orange }         /* 0,0,1,0 */
```

Example, continued

(4) Order by specificity. If ties in specificity, last one wins.

```
p { color: maroon }          /* 0,0,0,1 */
div p { color: green }       /* 0,0,0,2 */
.x { color: orange }         /* 0,0,1,0 */
p.x { color: violet }        /* 0,0,1,1 */
```

What will be the `color` of <div><p class=x>...?

Experiment with cas1.html

p.s. Repeat for all properties for all elements!

Flashback to slide 29!

From *Selectors, Specificity, and the Cascade* by Meyer:

> "Inheritance is the mechanism by which some property values are passed on from an element to its descendants. When determining which values should apply to an element, a user agent must consider not only inheritance but also the *specificity* of the declarations, as well as the origin of the declarations themselves. This process of consideration is what's known as the *cascade*. We will explore the interrelation between these three mechanisms —specificity, inheritance, and the cascade—in this chapter, but the difference between the latter two can be summed up this way: choosing the result of h1 {color: red; color: blue;} is the cascade; making a span inside the h1 blue is inheritance."

`!important`

If `!important` is added to a declaration, things change.

It looks like this:

```
p { color: red !important }
```

Try it in cas1.html

`!important` should be used sparingly. Lots of `!important`s can be a warning sign.

`!important`, continued

With `!important` in the picture, here's the ordering by origin in the cascade:

user agent declarations
user normal declarations
author normal declarations
<u>author important</u> declarations
<u>user important</u> declarations

The user's stylesheet has the last word! (Why?)

`!important`, continued

To create a user style sheet for Chrome, hit about:version and go to the Profile Path directory.

From there, edit "User StyleSheets/Custom.css"

Try this:

```
p {
    color: yellow !important;
    background-color: black !important;
    }
```

Pseudo-class selectors

Pseudo-class selectors

Pseudo-class selectors allow styling based on characteristics other than element type and attributes.

Here's a rule using the pseudo-class `:hover`

```
div:hover {
    background-color: black;
    }
```

The `:hover` pseudo-class has the element(s) which the cursor is over at any moment.

Try pclass1.html!

What would the selector `div :hover` select?

`:link` and `:visited`

The `:link` and `:visited` pseudo-classes apply only to hyperlinks, which are anchor elements with an `href` attribute.

`:link` is the set of unvisited hyperlinks

`:visited` is the set of visited hyperlinks

These two sets are disjoint—no hyperlink is in both.

Try pclass2.html with (1) Chrome; (2) Safari, with history clearing; and (3) Chrome Incognito.

Lots of pseudo-classes

:active	:indeterminate	:only-of-type
:checked	:in-range	:optional
:default	:invalid	:out-of-range
:dir()	:lang()	:read-only
:disabled	:last-child	:read-write
:empty	:last-of-type	:required
:enabled	:left	:right
:first	:not()	:root
:first-child	:nth-child()	:scope
:first-of-type	:nth-last-child()	:target
:focus	:nth-of-type()	:valid
:fullscreen	:only-child	

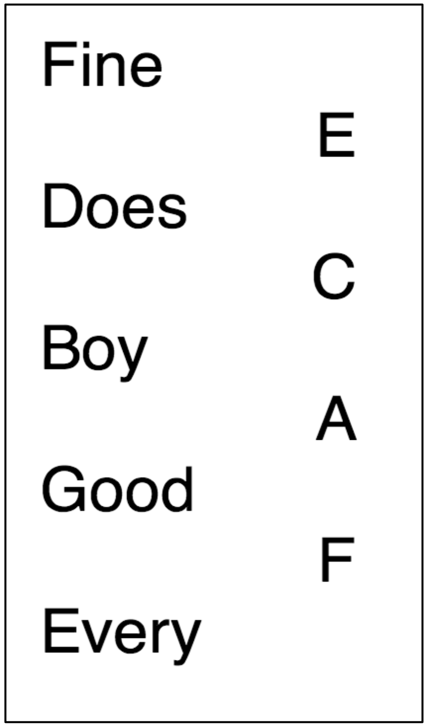
:nth-child

One of the more interesting pseudo-classes is `:nth-child`. Example:

```
li { list-style-type: none }
```

<u>**li:nth-child(even)**</u> { text-align: right }

```
<ul style="width: 5em">
<li>Fine
<li>E
<li>Does
<li>C
<li>Boy
<li>A
<li>Good
<li>F
<li>Every
</ul>
```



```
Fine
            E
Does
            C
Boy
            A
Good
            F
Every
```

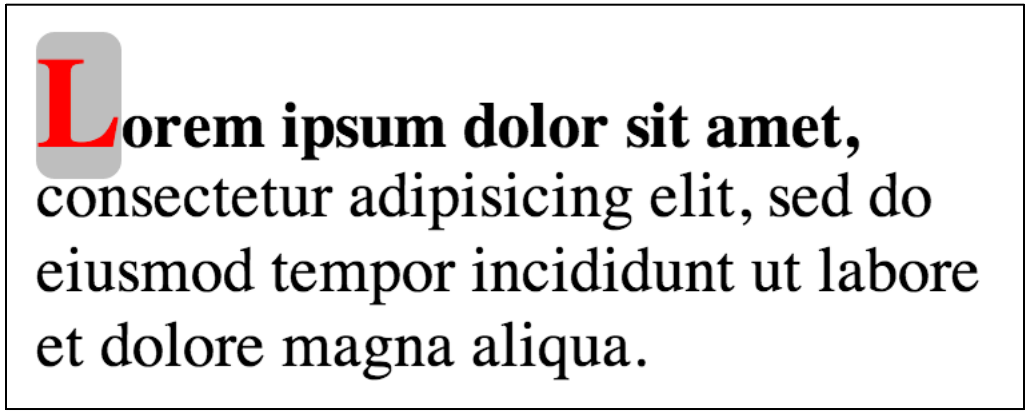
pclass3.html

Pseudo-element selectors

There are also pseudo-element selectors, which allow styling at a level below that of an element.

Example:

```
p::first-line {
    font-weight: bold;
    line-height: 1em
    }

p::first-letter {
    color: red;
    font-size: 2em;
    border-radius: 5px;
    background-color: silver;
    }
```



Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

pelem1.html

Could a `span` be used instead?