

Cascading Style Sheets

Lecture 11

Chapter 7-9 in HTML Text

Applying Styles

- The “style” of an element can be defined in many ways.
- Each element may have its own style attribute

```
<span style="font-style:italic">Hello Italics</span>
```

The “value” of the style attribute is expressed in a **style sheet language** - most often, CSS

Style Sheets

- **CSS: Cascading Style Sheets**
 - Allows style to be specified independently of HTML
 - Advantages:
 - Consistent look and feel across entire page/site
 - Less cluttered HTML*
 - Much easier to change format of pages

Styles

- CSS can be found in many places in HTML
 - Inline (style attribute)
 - Embedded style sheet (style element)
 - External (link to .css page)
- Regardless of where it is found, styles are always expressed as **attribute / value** pairs
 - `attribute : value`
 - The CSS attribute/value pair syntax is **not** what makes CSS so useful however (more to come)

Styles

- Each element in HTML has **many** supported CSS style attributes
- Most attributes are valid across multiple elements, ex.
- There are too many attributes to go over here.
 - There are many online resource to help
 - The textbook is **excellent** - it covers the vast majority of important CSS properties

Non-inline styles

- Inline styling offers no advantage over the old HTML styling tags
 - HTML is cluttered with styling information
 - Each instance of an element needs to be maintained separately
- Solution: Style Sheets
 - Embedded
 - External

Example: inline styles

```
<html>
  <head>...</head>
  <body style="background-color:red">
    <p style="background-color:yellow"> Paragraph 1</p>
    <p style="background-color:yellow"> Paragraph 2</p>
    <p style="background-color:yellow"> Paragraph 3</p>
    <p style="background-color:yellow"> Paragraph 4</p>
  </body>
</html>
```

Redundant - if we want to change the style of all paragraphs we need to search for and change each instance!

Also - the structure and content of the document is obscured

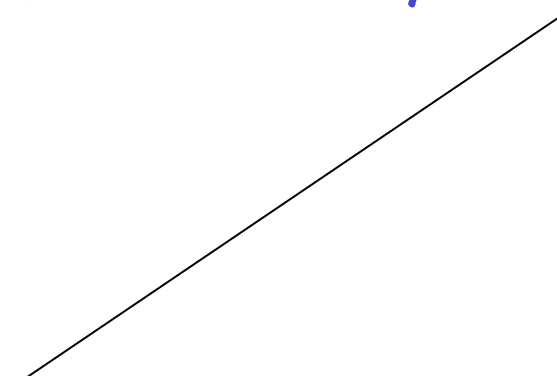
Embedded Style Sheets

```
<html>
  <head>...
    <style type="text/css">
      p {
        background-color:yellow
      }
      body {
        background-color:red
      }
    </style>
  </head>
  <body>
    <p> Paragraph 1</p>
    <p> Paragraph 2</p>
    <p> Paragraph 3</p>
    <p> Paragraph 4</p>
  </body>
</html>
```

Changing paragraph style
now only requires edits to
style tag in head

External style sheets

```
<html>
<head>...
  <link rel="stylesheet" type="text/css" href="style1.css"/>
</head>
<body>
  <p> Paragraph 1</p>
  <p> Paragraph 2</p>
  <p> Paragraph 3</p>
  <p> Paragraph 4</p>
</body>
</html>
```



Contents of style1.css

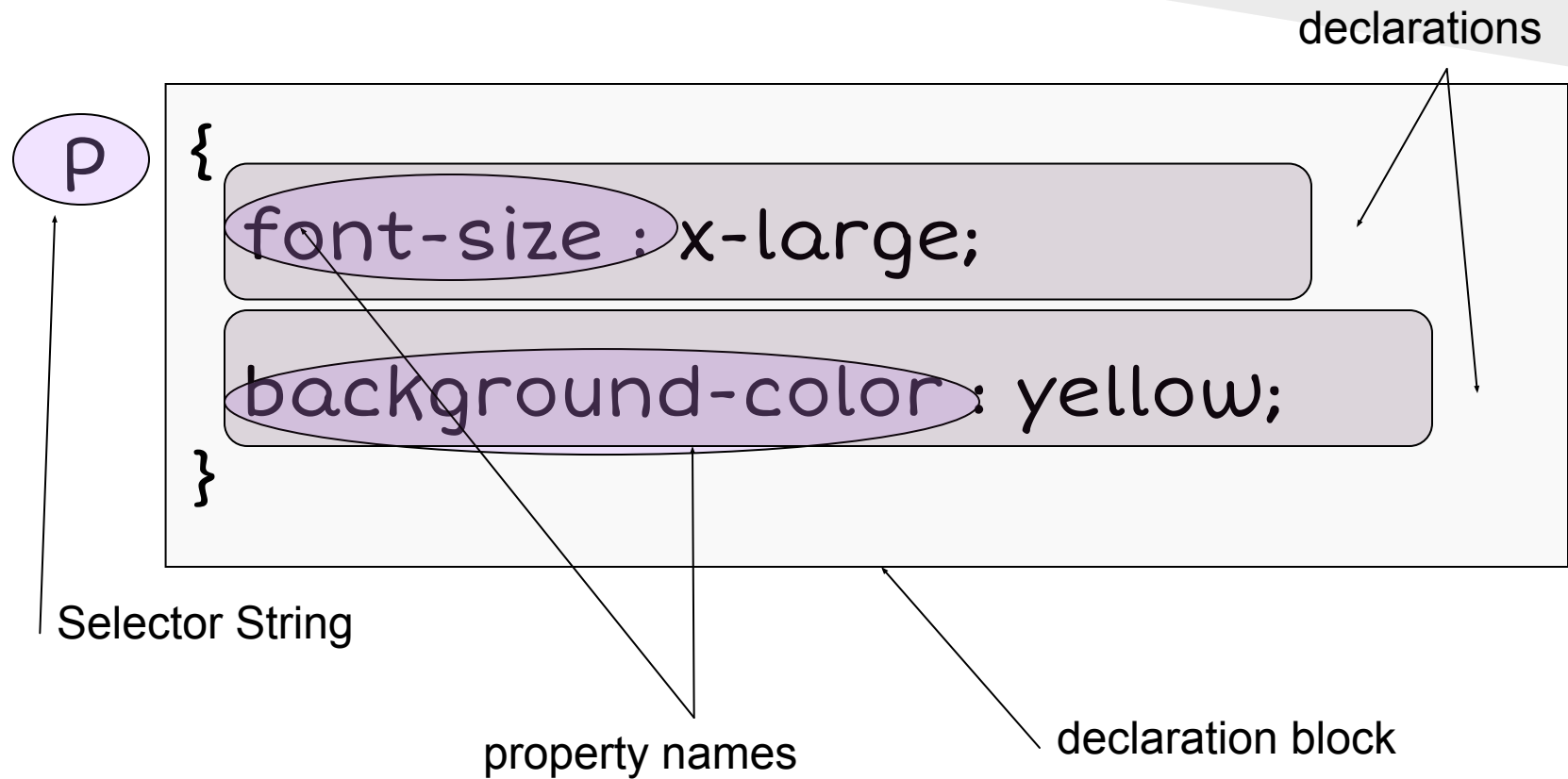
```
p {background-color:yellow};
```

```
body {background-color:red};
```

CSS Syntax

- W3C maintains the standard for CSS
 - CSS 1, CSS 2, CSS 2.1, CSS 3
- Syntax largely unchanged - it's the attribute set that changes (in a limited way)
- While there remains some small discrepancies, all modern browsers will render CSS 2.1 the same
 - ... as long as you are in standards mode!!!
- CSS 3 is the newest standard – adds rounded corners, drop shadows, gradients, and a variety of other nice features
 - Browser support is still highly variable

CSS Syntax



CSS Comments

- `/* CSS Support C++ style multiline comments */`
 - No single line comments
- Entity references (> instead of >)
 - Must be used for embedded css
 - Must **not** be used in external

Selector Strings

- When a browser encounters an element, it searches through each declaration block
 - Matches the element against a selector
- What makes CSS complex is the flexibility in which selectors can be defined and applied.

Type selectors

```
p { background-color:violet }
```

```
h1, h2, h3 {  
  font-weight : bold;  
  font-size: x-large }
```

Note, semi-colon is not required on last rule

```
span { font-style : italic }  
* { color:blue }
```

- Multiple element types are separated by commas
- All instances of the given element are formatted according to the declaration block

Type selectors

Heading 1

This is the paragraph within heading 1

Heading 2

This is the paragraph within heading 2. It also has some *italic text* within a `span` element

ID Selectors

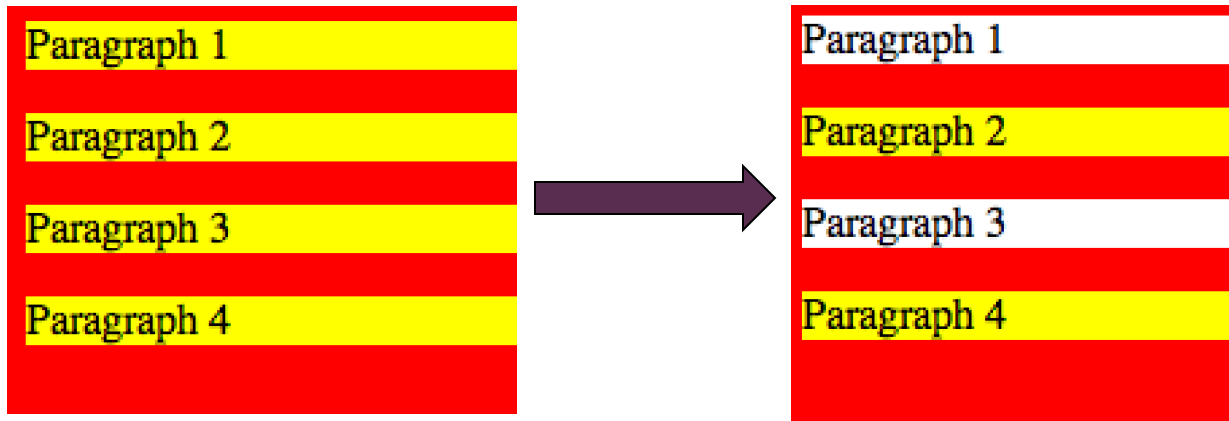
Each element within an HTML document can have an id attribute. They must be unique across the page

```
<html>
<head>...
  <link rel="stylesheet" type="text/css" href="style1.css"/>
</head>
<body>
  <p id="p1"> Paragraph 1</p>
  <p id="p2"> Paragraph 2</p>
  <p id="p3"> Paragraph 3</p>
  <p id="p4"> Paragraph 4</p>
</body>
</html>
```


ID Selectors

In some instances, you may want a **particular** instance of the paragraph element to be formatted differently

```
p { background-color : yellow }  
#p1, #p3 {background-color : white }
```



ID Selectors

- Heavy use of ID selectors tends to degrade / reduce the power of CSS
 - If every element is specified explicitly in separate CSS rules, the CSS can't be maintained either!
- Best used where specific ID's are re-used *between* pages, but should retain some formatting uniqueness

Class selectors

- Often, you do not want **all** instances of a particular element to look the same
 - However - you have a small set of “classes” or “types” of each element
 - “Normal” paragraphs
 - “Quoted” paragraphs
 - “Special” paragraphs
- CSS Allows you define classes **independently** from element type

Class selectors

`.normal {background-color:white}`
`.quoted {font-style:italic}`
`.special {background-color:yellow}`

```
<body>  
  <p class="normal"> Paragraph 1</p>  
  <p class="quoted"> Paragraph 2</p>  
  <p class="normal"> Paragraph 3</p>  
  <p class="special"> Paragraph 4</p>  
</body>
```

Paragraph 1

Paragraph 2

Paragraph 3

Paragraph 4

Class selectors

Class selectors don't relate to an individual element type however

```
<body>  
  <p class="normal"> this is my paragraph  
  with some <span class="special"> special text </span>  
  embedded in it. </p>  
  <p class="special"> Paragraph 4</p>  
</body>
```

this is my paragraph with some **special text**
embedded in it

Paragraph 4

Class selectors

... but they can be combined with element types...

```
.normal {background-color:white}  
.quoted {font-style:italic}  
.special {background-color:yellow}  
p.special {  
    font-size:x-large;  
    font-weight:bold  
}
```

Notice that the yellow background was still applied

this is my paragraph with some **special text** embededd in it.

Paragraph 4

Multiple class assignment

You can also assign multiple classes to a single element instance

Ex. helpful when defining multiple font classes and multiple background formats used independently

Multiple Class Assignment

```
.textSubtle { font-style:italic }  
.textLoud {font-size: x-large}  
.backSoft {background-color:gray}  
.backSpecial{background-color:red}
```

<p class="textLoud"> Paragraph 1</p>

<p> Paragraph 2</p>

<p class="textSubtle backSpecial">
Paragraph 3</p>

<p class="textLoud backSoft">
Paragraph 4</p>

Paragraph 1

Paragraph 2

Paragraph 3

Paragraph 4

Pseudo-Classes

- CSS defines several pseudo-classes of its own relating to anchors (hyperlinks)
 - `a:visited`: applied to all links visited “recently”
 - `a:link`: applied to all links not marked as visited
 - `a:active`: applied when link is clicked, but not releases
 - `a:hover`: applied when mouse is over link
- You may use these selectors to define your own styles

Descendent Selectors

- You can also be more specific when using type selectors
- Specifying multiple element types within the selector string **without** comma establishes a “containment” constraint

```
ul span {font-variant:small-caps}
```

The rule above **only** pertains to spans inside an unsorted list

Child and siblings

div > p selects paragraphs that are **direct** children of div elements (not descendants)

h1 + p selects paragraphs that are immediate (next) siblings (adjacent) of h1 elements

h1 ~ p selects paragraphs that are siblings (in general) of an h1 element.

Example

div > p { color:red;}

h1 + p {text-decoration: underline}

h1 ~ p {color: blue}

<section>

<p> paragraph above h1</p>

<h1> header</h1>

<p> paragraph right next to h1 </p>

<p> paragraph that is sibling to h1</p>

<div>

<p>Nested paragraph in div</p>

</div>

</section>

paragraph above h1

header

paragraph right next to h1

paragraph that is sibling to h1

Nested paragraph in div

More Combinations

```
.special span {color:blue}  
ul ol li { letter-spacing:1em}
```

Rule:

- No Commas means descendent relationship
- Comma means “or”
#p4, .special {font-style:italic}

At-Rules

- Just like in any other language, CSS allows you to import rules
 - `@import url("general-rules.css");`
- Restrictions:
 - all import statements must come before any rules
 - url can be relative or absolute
 - relative url **is relative to the importing css file**
 - duplicate rules in document override imported rules*

Cascading Style Sheets

The power of CSS also rests in its ability to resolve “collisions” between rules.

```
p {color:red}  
#p1 {color:green}  
.special {color:gray}
```

Paragraph 1

Paragraph 2

Paragraph 3

```
<p>Paragraph 1 </p>  
<p id="p1" class="special">Paragraph 2</p>  
<p class="special">Paragraph 3</p>
```

How does CSS decide the rule to use for <p> elements?

Which rule to apply?

- For every property, for every element, the browser must decide which CSS rule to use
- To do this, the browser performs a multi-stage sort on all rules for each property/element
- This sort, and its application, is called ***rule-cascading***
- First Stage: Sort according to *origin of rule and importance*

Rule origin

Origin refers to “who” is supplying the rule

- **Author**: The author of the web page, either via an embedded or external style sheet
- **User-Agent**: Browser typically has a set of default style, especially for text
- **User**: Most modern browsers allow users to either directly or indirectly create their own CSS rules (preferences)

Rule Importance

- Each “origin” may also specify rule as being “important”.
- This designation provides a hint as to how the sorting should proceed

```
p {text-indent:3em; font-size:larger !important}
```

Stage 1 Sorting

- 1) Important declarations - user
- 2) Important declaration - author
- 3) Normal declaration - author
- 4) Normal declaration - user
- 5) Any declaration - user agent

Colliding rules are placed into one of five “bins”. Each bin is examined, in order. If the first bin encountered has only one rule, that rule is chosen

Stage 2 Sort: Specificity

If multiple rules exist within the same “origin bin”, then rule that is most “specific” is used.

Specificity Bins:

- 1) ID selectors
- 2) Class and pseudo-class selectors
- 3) Descendent and Type Selectors
 - The more types in descendents, the more specific
- 4) Universal selectors (*)

Rules with combination selectors are placed in both bins

Multi-bin Tie Breaking

- Specificity Bins:
 - ID selectors
 - Class and pseudo-class selectors
 - Descendent and Type selectors
 - Universal selectors (*)
- Example
 - p.special: Bin 2 and 3
 - *.special: Bin 2 and 4
 - Multiple rules found in bin 2. Search for next repetition of either rule in lower bins.

Further tie breaking

- If two rules of identical specificity are found:
 - If there is a style attribute for the element, use that
 - Otherwise, list rules in order in which they appear, top to bottom (Note, imports are always before inline rules)
 - Use the one that was defined **last**
- Note: CSS properties are always used instead of HTML deprecated attributes
 - Ex: image tag - height attribute is overridden by the height style

Rule Inheritance

When no rule is present for an element “x”, CSS derives properties from element x’s parent

```
body {color:red}  
#p1 {color:green}
```

```
<p>Paragraph 1 </p>  
<p id="p1">Paragraph 2</p>  
<p id="p2">Paragraph 3</p>
```

Paragraph 1

Paragraph 2

Paragraph 3

Selecting for media

- An author can apply different stylesheets according to the “media” the page is displayed on
- Common Media Types:

all	print
aural: speech synthesizer	projections
braille: tactal displays	screen
handheld	tv

Media types

- The most common use for this is specifying different style when printing
 - Good example: don't use background colors when printing!

```
link rel="stylesheet" type="text/css" href="style1.css"
  media="screen, tv, projection" />
link rel="stylesheet" type="text/css" href="style2.css"
  media="print, handheld" />
```

Most browser will automatically apply correct sheet

Font Families

- Font Family: Collection of related fonts
 - All share the same basic appearance
 - Differences: font weight, italics, decorations (small-caps, strikethrough, etc)

Times New Roman

Times New Roman

TIMES NEW ROMAN

Verdana

Verdana

VERDANA

American Typewriter

American Typewriter

AMERICAN TYPEWRITER

Comic Sans MS

Comic Sans MS

COMIC SANS MS

BANK GOTHIC

BANK GOTHIC

BANK GOTHIC

Font Families

- Caution: many fonts are system-dependent
 - Font names present on a Linux system may not be available when viewed on Windows
 - The fonts available depend on the client's machine, NOT the server, web page, or browser!
- Use comma-separated values
`font-family:"Edwardian Script", "French Script", cursive`
- The last entry should be a CSS generic family

serif

sans serif

`monospace`

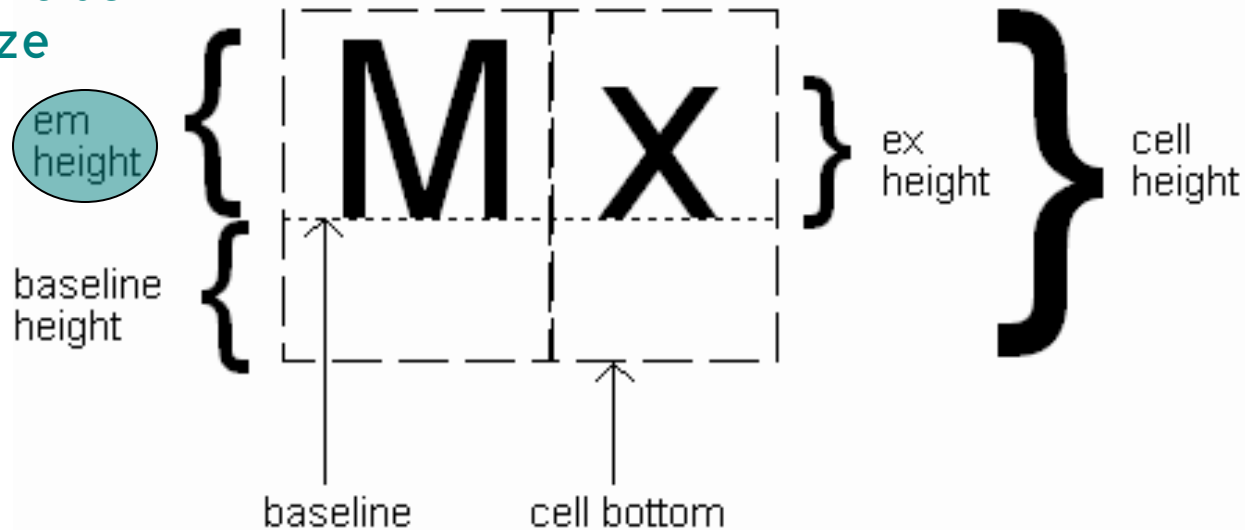
cursive

Font Size

- Font size is specified by the font-size attribute
 - The value is a CSS length, which can be in many different units
 - in, cm, mm
 - pt: 1/72 in
 - pc: 12 points
 - px: 1/96 inch (estimate)
 - em: reference font-size
 - ex: height of lowercase letters (x) expressed in em

CSS Font Properties

Computed value
of **font-size**
property



Typically, the font family dictates all but the font-size

Specify Font Size

- CSS length: in, cm, px, etc.
- Percentage (of parent font-size)
Absolute size keyword:
 - xx-small, x-small, small,
 - medium (initial value),
 - large, x-large, xx-large
 - User agent specific; should differ by ~ 20%
- Relative size keyword: smaller, larger
 - Relative to parent element's font

Additional font properties

font-style: normal, *italic*, *oblique*

font-weight: normal or **bold**

font-variant: normal or SMALLCAPS

Pseudo-Class Selectors

There are some helpful pseudo-classes for text that help you apply rules to specific text

:first-line

:first-letter

Lorem ipsum dolor sit amet, consectetur adipiscing 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

Line Boxes

Text is rendered using line boxes



- Height of line box given by **line-height**
 - Initial value: normal (*i.e.*, cell height; relationship with em height is font-specific)
 - Other values (following are equivalent):

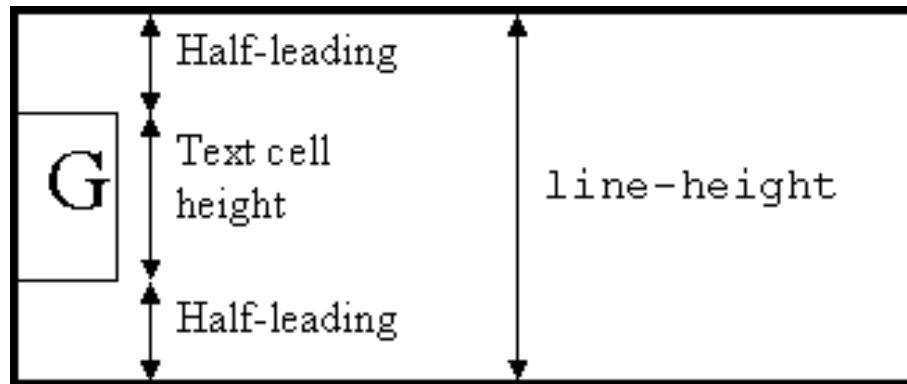
```
line-height:1.5em
```

```
line-height:150%
```

```
line-height:1.5
```

Line Boxes

When line-height is greater than cell height:



Inheritance of line-height:

Specified value if normal or unit-less number

Computed value otherwise

Font Shortcut Declaration

font **shortcut** property:

```
{ font: italic bold 12pt "Helvetica",sans-serif }
```

```
{ font-style: italic;  
  font-variant: normal;  
  font-weight: bold;  
  font-size: 12pt;  
  line-height: normal;  
  font-family: "Helvetica",sans-serif }
```

Font Shortcut Declaration

font shortcut property:

specifying line-height

```
{ font: bold oblique small-caps 12pt/2 "Times New Roman", serif }
```

any order

size and family required,
order-dependent

Note - this has **nothing** to do with division!

Additional Text Formatting

- **text-decoration**: underline, overline, line-through
- **letter-spacing**: normal or CSS length
 - can use negative values
- **word-spacing**: none or CSS length
- **text-transform**: none, capitalize, uppercase, lowercase
- **text-indent**: CSS length (defaults to 0)
 - can also be % of box width
- **text-align**: left, right, center, justified
- **white-space**: normal, pre

Text Color

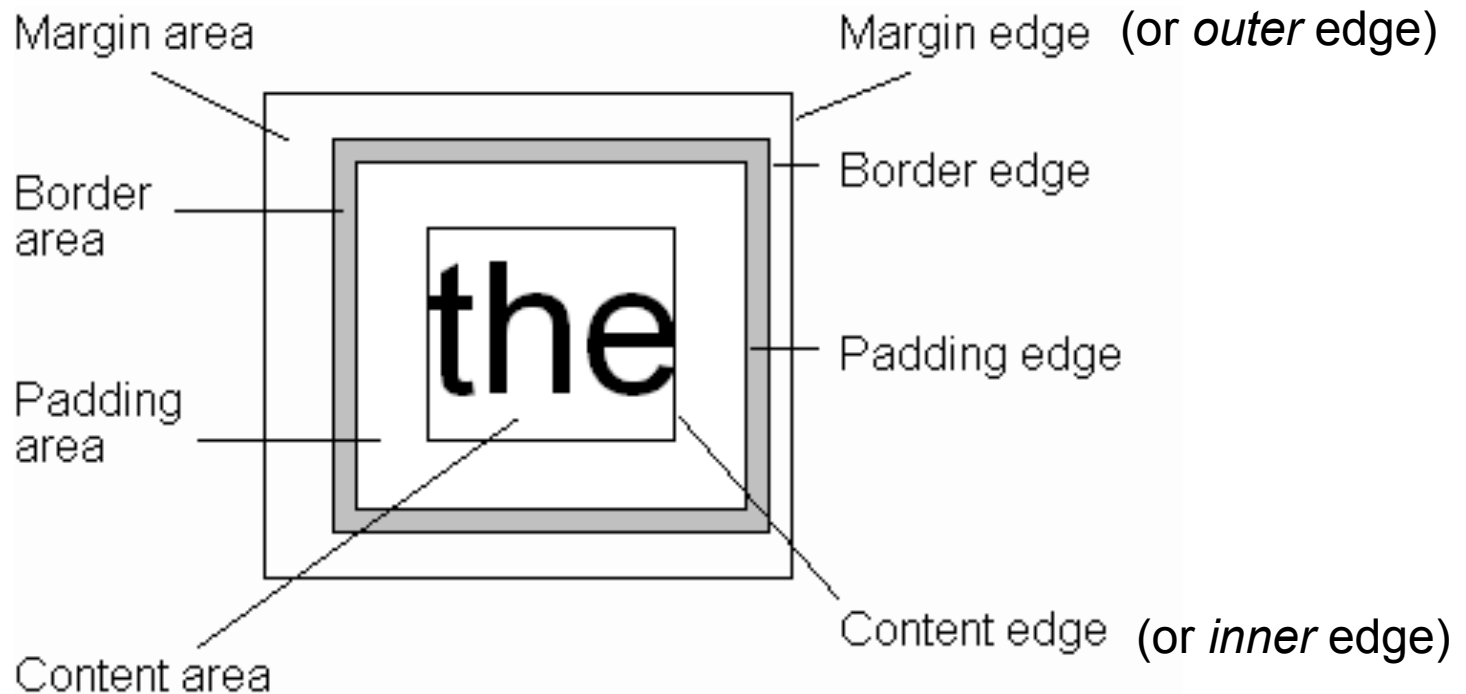
- Font color specified by color property
 - `color:silver`
 - Color name: black, gray, silver, white, red, lime, blue, yellow, aqua, fuchsia, maroon, green, navy, olive, teal, purple, <http://www.w3.org/TR/SVG11/types.html#ColorKeywords>
- More commonly:
 - RGB
 - Hexadecimal values <http://html-color-codes.info/>

Lists and Tables

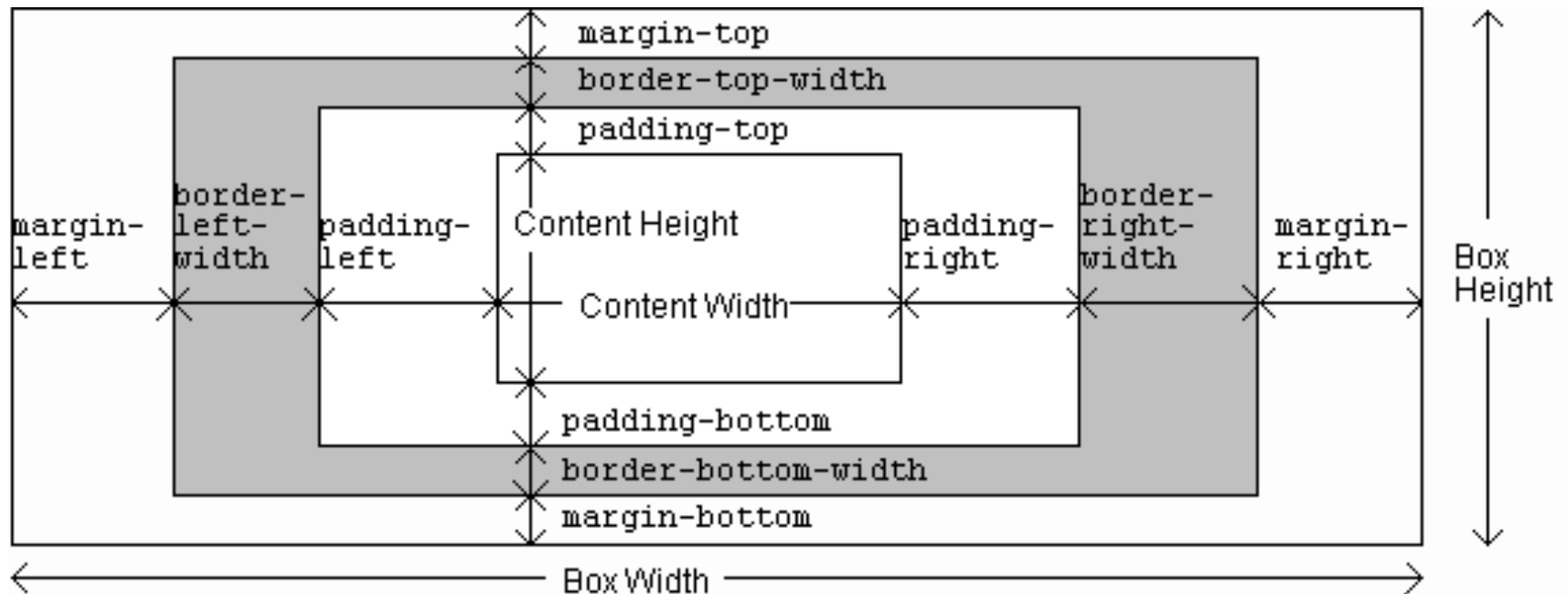
- There are about a dozen CSS properties that help you control the look of lists (ordered and unordered) and tables.
- We won't go over them all here - but check out chapter 8 in the HTML Text
 - You'll use some of these on the next homework.

Box Model

Every rendered **element** occupies a box:



Box Model - Property Names



Box Model Properties

- padding: CSS Length
- border-width: CSS Length
- border-color: Color or “transparent”
- border-style: hidden, dotted, dashed, solid, double groove, ridge, inset, outset
- margin: CSS Length

Duplicate attribute specification

If multiple declarations apply to a property, the last declaration overrides earlier specifications

```
{ border: 15px solid;  
  border-left: 30px inset red;  
  color: blue }
```

↑
30px overrides 15px for left border

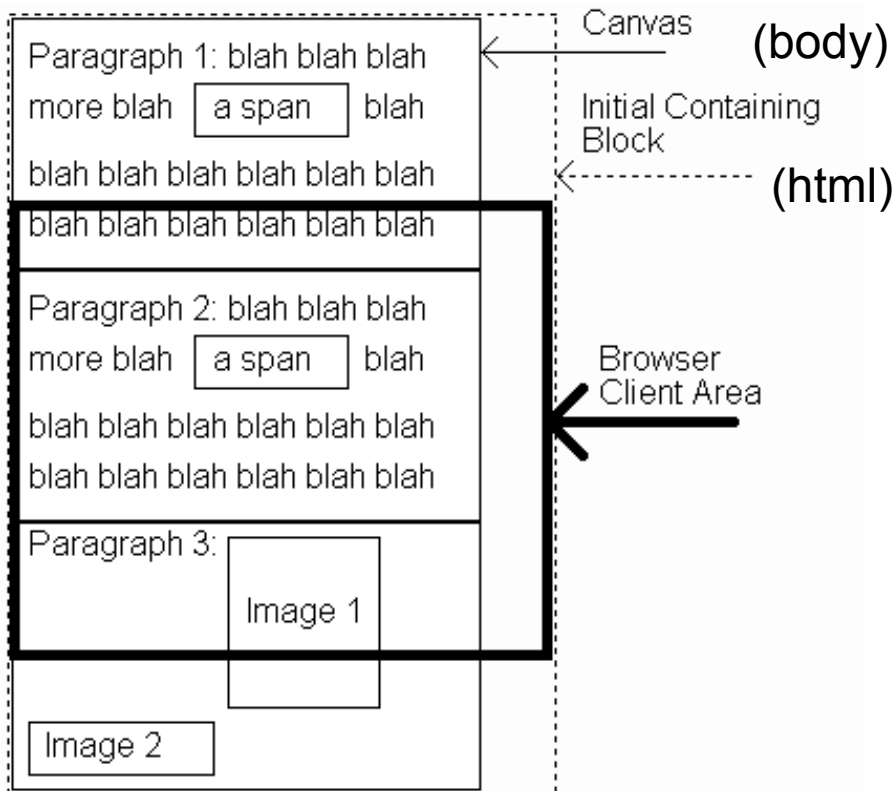
Backgrounds

- **background-color**
 - Specifies background color for content, padding, and border areas
 - Margin area is always transparent
 - Not inherited; initial value transparent
- **background-image**
 - Specifies (using url() function) image that will be **tilled** over an element

Normal Flow Layout

- In normal flow processing, each displayed element has a corresponding box
 - html element box is called initial containing block and corresponds to entire document
 - Boxes of child elements are contained in boxes of parent
 - Sibling block elements are laid out one on top of the other
 - Sibling inline elements are one after the other

Normal Flow Layout



Each element gets its own “box”

Block elements use entire width

Vertically positioned directly below last block

Vertical size is limited to content (or children)

Block Elements

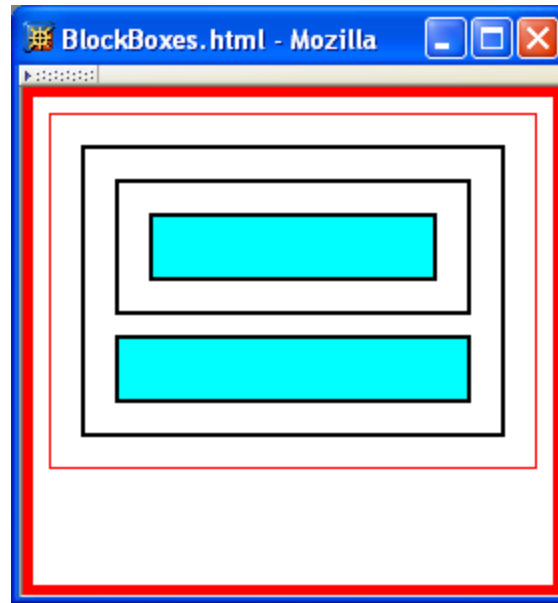
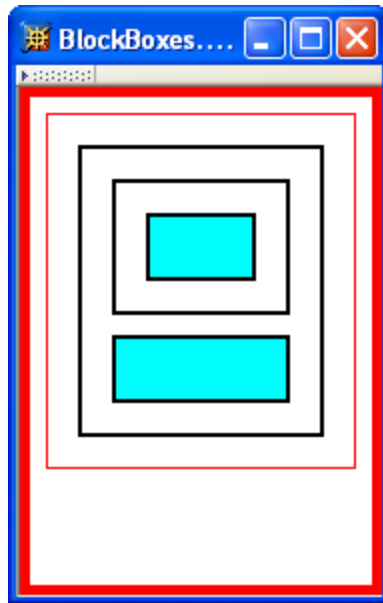
- What is a “block element”?
 - Element with value block specified for its **display** property
 - User agent style sheet (not CSS) specifies default values; typical block elements include `html`, `body`, `p`, `pre`, `div`, `form`, `ol`, `ul`, `dl`, `hr`, `h1` through `h6`
 - Most other elements except `li` and table-related have inline specified for display

Margin Collapsing

- When blocks stack, adjacent margins are **collapsed** to the size of the larger margin
 - Example:
 - Block_{upper} margin = 5
 - Block_{lower} margin = 10
 - Vertical space between two blocks is 10.

Width Attributes

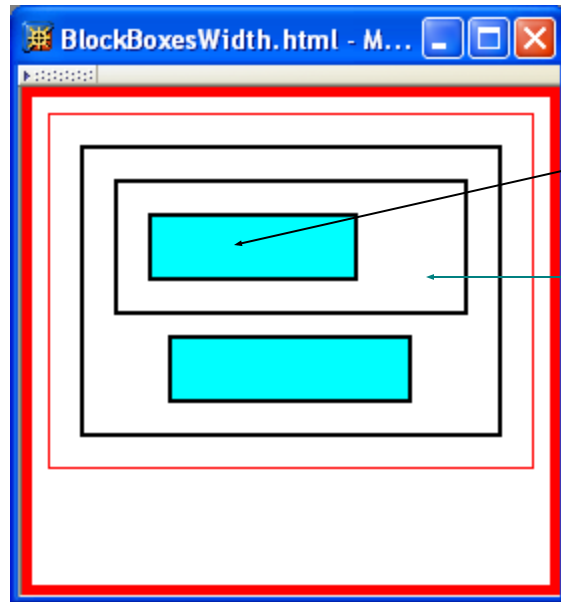
Initial value of width property is auto, which for block boxes means to make the content area **as wide as possible within margin/padding constraints**:



Width of block boxes increases as browser client area is widened

Width Attributes

Can also specify CSS length or percentage (of parent's content width) for width property

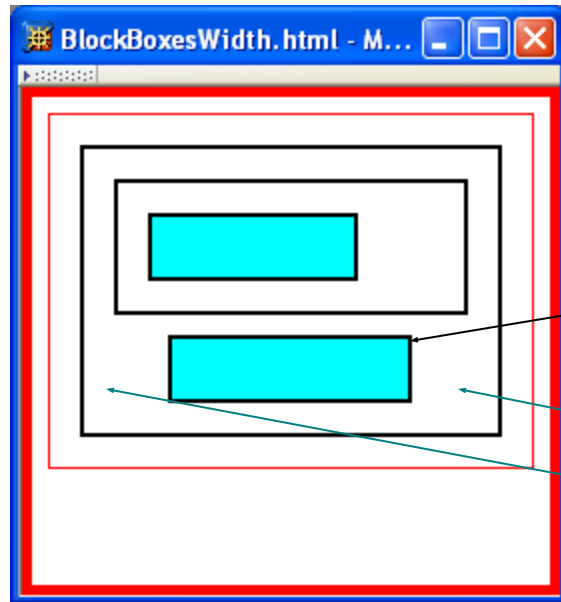


```
#d3 { width:50% }
```

By default, width of right margin is adjusted to accommodate a change to width

Width Attributes

Can also specify CSS length or percentage (of parent's content width) for width property



```
#d4 { width:50%; margin-left:auto;  
      margin-right:auto}
```

Centering can be achieved by setting
both margins to **auto**

min/max properties

If you want your content to be dynamic, within reason, there are four excellent properties to use

- min-width
- max-width
- min-height
- max-height

These are very helpful when defining complex, yet responsive layouts

What if it doesn't fit?

- What if the content within your element doesn't fit in the height or width you've specified?
- The overflow property can be set to the following values:
 - **hidden** (hides the overflowing content)
 - **visible** (content is visible outside the element!)
 - **scroll** (the element **always** has scroll bars)
 - **auto** (the element gets scroll bars when necessary)
 - **inherit** (use the parent settings - default)

More on dimensions

- The CSS box model considers “width” to be the size of the elements **content** only
 - It does not count padding, border-width, or margin
- Unless you are Microsoft in the 90’s and early 2000’s!
 - IE 5 and below counted padding and border-width as part of the overall width!
 - This caused much trouble...

Box models

Microsoft fixed the issue, in “standards mode” for IE 6 and above.

- All was right in the world... however it turns out that Microsoft was probably on to something
- Most of the time, you actually want padding and border to count towards width!

Box models

- So a new standard was created.
 - box-sizing property was added to CSS
- `box-sizing:content-box`
 - (old CSS standard)
- `box-sizing:border-box`
 - (the old IE standard)

Corners and Shadows

- Rounded corners were the rage for years, and probably wasted millions of developer hours!
- Now CSS3 supports them (now no one wants them!)

Lorem ipsum dolor sit amet, consectetur adipiscing 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

`border-radius: 5px`

Supported in IE 9+, and every other browser

Corners and Shadows

- Similarly, drop shadows were forever difficult to implement (google it!)
- Now CSS3 supports drop shadows
 - And Windows 8 and IOS7 decided “flat” was cool

```
box-shadow: 10px 10px 5px #888888;
```

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

Param 1: Horizontal shadow distance

Param 2: Vertical shadow distance

Param 3: Spread - the size of the shadow (optional)

Param 4: Color (optional)

Param 5: Inset (optional) reverses shadow direction

More nice CSS3 stuff

CSS3 adds color gradients, animations, and slew of other features.

Browser support varies though - but its getting close...

Read Chapter 9 for more CSS3 details

Column Layouts

CSS3 adds column-count and column-gap to the mix - allowing text to flow freely within columns

- webkit-column-count
- moz-column-count

Lorem

ipsum dolor sit
amet, consectetur
adipiscing 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

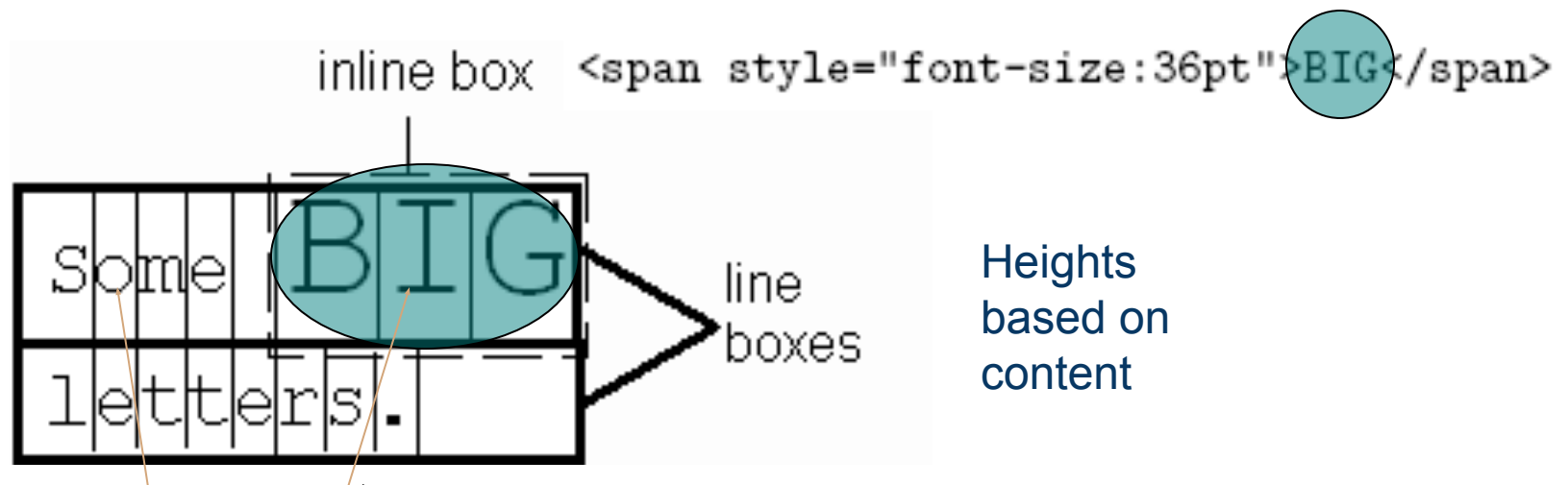
cillum dolore eu
fugiat nulla
pariatur. Excepteur
sint occaecat
cupidatat non
proident, sunt in
culpa qui officia
deserunt mollit
anim id est laborum

IE 10+ supports
columns

Chrome 4+ and Firefox
2+ support it with their
extensions

Inline Element Boxes

Boxes corresponding to **character cells** and **inline elements** are laid out side by side in **line boxes** that are stacked one on top of the other



Character cells aligned by baseline

Vertical Alignment (inline)

```
<p>
  Middle
  
</p>
<p>
  Bottom
  
</p>
<p>
  Top
  
</p>
```

Middle



Bottom



Top



Float Positioning

CSS allows for boxes to be positioned outside the normal flow:

span taken out of normal flow and “floated” to the left of its line box



This text is going to wrap around the big roman numeral embedded within the paragraph because the numeral is floated to the left of the line box.

<p>

This text is going to wrap around the

` I. `

big roman numeral embedded within the paragraph because the numeral is floated to the left of the line box.

</p>

Beyond Normal Flow

- CSS allows for boxes to be positioned outside the normal flow:
- **Absolute** positioning
- **Fixed** positioning

`<p>`

This second paragraph has a note.

``This note is pretty long, so
it could cause trouble...``

`</p>`

style rule that moves `span` relative to
upper left corner of containing
`p` element's box

Absolute positioning

```
p { position:relative;
    margin-left:10em }
.marginNote {
    position:absolute;
    top:0; left:-10em;
    width:8em;
    background-color:yellow }
```

This note is pretty long, so it could cause trouble...

A short note.

This second paragraph has a note.

This third paragraph also has a note.

<p> This second paragraph has a note.

This note is pretty long, so it could cause trouble...</p>

<p>This third paragraph also has a note.

A short note.</p>

Beyond Normal Flow

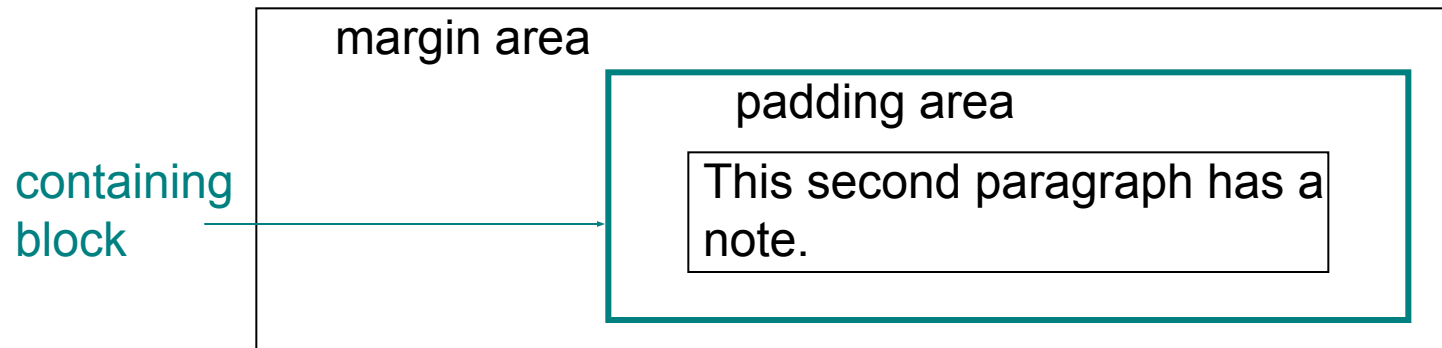
- Properties used to specify positioning:
 - **position**: static (initial value), relative, or absolute
 - Element is **positioned** if this property not static
 - Properties left, right, top, bottom apply to positioned elements
 - Primary values are auto (initial value) or CSS length
 - **float**: none, left, or right
 - Applies to elements with static and relative positioning only

Absolute Positioning

Elements that are *absolute* positioned must have a parent that is *positioned* as well...

```
p { position:relative; margin-left:10em }
```

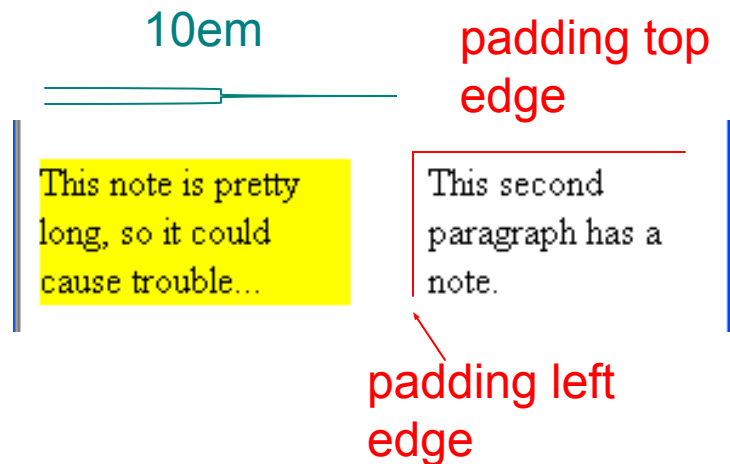
p elements are positioned (but don't move!)



Absolute Positioning

```
p { position:relative; margin-left:10em }  
.marginNote { position:absolute;  
               top:0; left:-10em; width:8em;  
               background-color:yellow }
```

```
<p>  
  This second paragraph has a note.  
  <span class="marginNote">This note is pretty long, so  
  it could cause trouble...</span>  
</p>
```



Absolute Positioning

```
p { position:relative; margin-left:10em }  
.marginNote { position:absolute;  
              top:0; left:-10em; width:8em;  
              background-color:yellow }
```

This note is pretty
long, so it could
cause trouble...

This second
paragraph has a
note.

8em

```
<p>  
  This second paragraph has a note.  
  <span class="marginNote">This note is pretty long, so  
  it could cause trouble...</span>  
</p>
```

Caution w/ absolute positions

Absolutely positioned box does *not* affect positioning of other boxes!

Second absolutely positioned box obscures first

This note is pretty long, so it could
A short note.

This is the first paragraph. No note here.

This second paragraph has a note.

This third paragraph also has a note.

Position-Related Properties

z-index: drawing order for overlaid boxes
(largest number drawn last)

```
#text { position:absolute; top:10px; left:10px;  
        font-family:"Courier",monospace; letter-spacing:0.1ex;  
        background-color:yellow;  
        z-index:1 }  
#overlay { position:absolute; top:10px; left:10px;  
            width:1.1ex; height:4.5em;  
            border:solid red 1px;  
            z-index:2 }
```

In
My
Humble
Opinion

Position-Related Properties

- **display**: value none means that element and its descendants are not rendered and *do not* affect normal flow
- **visibility**: value hidden (initial value is visible) means that element and its descendants are not rendered but still *do* affect normal flow

Browser Compliance

- Only the most recent browsers reliably (and consistently) render complex CSS
 - IE 8 +
 - Firefox 4 +
 - Chrome
 - Opera 9+
 - Safari 3+

You don't stand a chance if you wind up in quirks mode - doctypes are critical.

There are libraries that add CSS support for Internet Explorer - we will discuss later

CSS in Practice

- It tends to be easiest if you first create your “dry” HTML documents first
 - Test your app - no functionality should require CSS.
 - Use good semantic elements
- Next - apply your CSS based on a design (or idea)

Start Page

Please enter your guess:

Guess

This is a simple numeric guessing game. A random number has been computed, and you need to guess what that number is. The random number is between 1 and 10, if you are smart, you should be able to guess it correctly in no more than 4 tries.

Check Page

Please enter your next guess:

Guess

Sorry, thats not the number...

5 was too low

9 was too high

6 was too low

8 was too high

Next topic

- We are going to stay on the browser side of things, and start to add JavaScript to our web pages
- We'll learn how JavaScript can navigate, query, and manipulate the HTML document (DOM)
- We'll take a look at jQuery and work on some AJAX calls to make single-page applications