

Tutorial 3:

Working with

Cascading Style Sheets



Computer Science Department
University of Central Florida

COP 3175 – Internet Applications



Objectives

- Review the history and concepts of CSS
- Explore inline styles, embedded styles, and external style sheets
- Understand style precedence and style inheritance
- Understand the CSS use of color
- Explore CSS styles for fonts and text
- Review and compare different image formats



Objectives

- Display an animated graphic
- Apply a background image to an element
- Float elements on a Web page
- Explore the properties of the box model
- Apply border styles to an element



Introducing Cascading Style Sheets

- What are Style sheets?
 - **Style sheets** are declarations that describe the layout and appearance of a document
 - As seen in Tutorial 1, HTML specifies a document's content and structure
 - But HTML does not necessarily describe the appearance of documents
 - To describe the design (appearance) of a web page, you should use style sheets



Introducing Cascading Style Sheets

- **Cascading Style Sheets (CSS)**
 - a style sheet language used on the Web
 - There are many style sheet languages
 - But CSS is by far the most common on the web
 - Guess what?
 - We've actually been using CSS since Tutorial 1
 - ...each time we used the style attribute
 - The style attribute is part of the specification for HTML
 - but the text (value) of the attribute is written in CSS



Introducing Cascading Style Sheets

- **Cascading Style Sheets (CSS)**
 - CSS specifications are maintained by the World Wide Web Consortium (W3C)
 - Several versions of CSS exist with varying levels of browser support:
 - CSS1, CSS2, CSS 2.1, and CSS3



Cascading Style Sheets

- **CSS1** introduced in 1996, and made styles for the following document features:
 - Fonts
 - set font size, type, and other properties
 - Text
 - alignment, underlining, italics, etc.
 - Color
 - specifying background and foreground colors
 - Backgrounds: set background image for elements
 - Block-level Elements
 - setting margins and borders of block level elements



Cascading Style Sheets

- **CSS2** introduced styles for the following document features:
 - Positioning
 - placing elements at specific locations on the page
 - Visual Formatting
 - clipping and hiding element content
 - Media Types
 - Interfaces
 - control appearance of scroll bars and mouse cursors
- **CSS 2.1** did not add any new features to the language



Cascading Style Sheets

- **CSS3** (which is still in development) will introduce styles for the following document features:
 - User Interfaces
 - Accessibility
 - support users with disabilities
 - Columnar layout
 - give more layout options
 - International Features
 - support for wide variety of languages and typefaces
 - Mobile Devices
 - Scalable Vector Graphics



Applying (Using) a Style Sheet

- There are three ways to apply a style to an HTML or XHTML document:
 - 1) **Inline Styles**
 - 2) **Embedded Style Sheet**
 - 3) **External Style Sheet**

- Each approach has advantages and disadvantages, which we will explore



Using Inline Styles

■ Inline styles

- These are the styles that we've already been using
- They are easy to use and interpret because they are applied directly to the elements they affect

■ Syntax:

```
<element style="style1: value1; style2: value2; style3: value3; ...">
```

- where style1, style2, style3, and so forth, are the names of the style properties, and value1, value2, value3, and so on, are the values associated with each property



Using Inline Styles

- Inline styles

- Example:

- center an h1 tag and display it in a red font:

- ```
<h1 style="text-align: center; color: red">Sunny Acres</h1>
```

- Advantages of Inline Styles:

- Easy to understand and interpret

- Since they are applied directly to elements they affect



# Using Inline Styles

---

- Disadvantages of Inline Styles:
  - The very benefit of inline styles is actually their downfall!
  - They are cumbersome!
  - Example:
    - Imagine if you have a large website (100's of pages) with a consistent theme (color, look, etc) across those pages
      - Now, if you used inline styles to make all of your headings, what must you do if you want to change the theme of the site from blue to green?
      - You would have to open up EACH and EVERY page (100s of pages) and modify the inline style color attribute, changing it to green...**on EACH AND EVERY page**

FAIL



# Using Inline Styles

---

- Disadvantages of Inline Styles:

- Additionally, many developers argue that inline styles are not consistent with the goal of separating Web content from the styles.

- Example:

- Here is an old, deprecated way of aligning an h1 tag:

```
<h1 align="right">...</h1>
```

- Here's the new way using styles:

```
<h1 style="text-align: right">...</h1>
```

- There is very little difference between these two methods
  - A goal of style sheets is to separate document content from the style of the document. Ideally, the HTML code and CSS styles should be separate



# Using Embedded Styles

## ■ Solution: **Use embedded styles**

- The power of using style sheets becomes evident when you move style definitions away from document content
- One way to do this is to collect all styles together and place them in the head section
  - Known as embedded styles

### ■ Syntax:

```
<style type="text/css">
 style declarations
</style>
```

- Where *style declarations* are the declarations of the different styles to be applied to the document



# Using Embedded Styles

- Embedded Styles:

- Each style declaration has the syntax:

- `selector {style1: value1; style2: value2: ... }`

- where selector identifies an element (or elements) within the document and the 'style: value' is a pairing of a style and its respective value

- Example:

- Make all h1 headings, in a document, as centered red text
- Embed the following in the document header:

```
<style type="text/css">
 h1 {text-align: center; color: red}
</style>
```





# Using Embedded Styles

## ■ Embedded Styles:

- You can also apply the same style to several elements
  - Simply enter the elements, separating them by commas

## ■ Example:

- Make all h1 and h2 headings, in a document, as right-aligned, blue text
- Make all h3 headings as centered, green text
- Embed the following in the document header:

```
<style type="text/css">
 h1, h2 {text-align: right; color: blue}
 h3 {text-align: center; color: green}
</style>
```



# Using Embedded Styles

---

## ■ Embedded Styles:

- So embedded styles fix the limitation of inline styles
- But what is limitation of embedded styles?
  - Think about the limitation of inline styles...
- Limitation of embedded styles:
  - an embedded style sheet is limited to the page elements of the current document
  - So what's the problem with this?
  - The problem is similar to that of inline styles
    - What if your web site has hundreds of pages
    - You would have to modify the embedded style (in the header), for each and every one of those pages



# Using an External Style Sheet

---

- Solution: External Style Sheets
  - **Place all of your styles in an external file**
    - a .css file
    - This file simply contains all style declarations
      - Looks just like a list of embedded styles
      - Except that the style declarations are not enclosed within the opening and closing <style> tags
  - Then, you simply link to this file from any of the pages on your website
    - This allows the same styles to be applied all pages that link to the style sheet
      - You can now change the styles for an entire web site by modifying one style sheet



# Using an External Style Sheet

---

## ■ Style Comments

- You can add comments to your style sheets just as you do for an HTML file
- The syntax is as follows:  

```
/* comment */
```
- CSS ignores the presence of whitespace
  - So as with HTML, you can place style comments on several lines, making it easier to read



# Using an External Style Sheet

---

## ■ Linking to an External Style Sheet

- From a web page, you can link to an external style sheet by using the link element

- Syntax:

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

- where url is the URL of the external style sheet
  - Link elements for style sheets must be placed in the head section of the web page document
- Note on URL:
    - Interpreted the same way as URLs for linked web pages
      - With respect to file locations (in folders)
      - If no folder is listed, the css file is assumed to be in same folder



# Understanding Cascading Order

---

- Which style has precedence?
  - Problem:
    - You can apply styles to a web page in a variety of ways
    - Which style is ultimately used?
    - Example:
      - Consider a web site, whose pages link to an external style sheet that sets all h1 elements in a bold, green font.
      - At the same time, the author uses some inline fonts that set some of the h1 tags to a bold, purple font that is centered.
      - Additionally, browsers, by default, render h1 elements in a regular black font that is not centered.
      - So you effectively have three styles for one h1 element.
      - **Which style is used? Which takes precedence?**



# Understanding Cascading Order

---

## ■ Style Precedence:

- a rule that determines which style is applied when one or more styles conflict.

## ■ The General Rule:

- In the case of conflict, the more specific style has precedence over the more general style

## ■ Example:

- The most general styles are those built into a browser
  - The reasons browsers know to indent blockquote elements or know to display h1 elements in a large fonts is because they apply a default, internal style sheet
- This default browser style sheet is used unless a different style is specified by the Web page author or the user



# Understanding Cascading Order

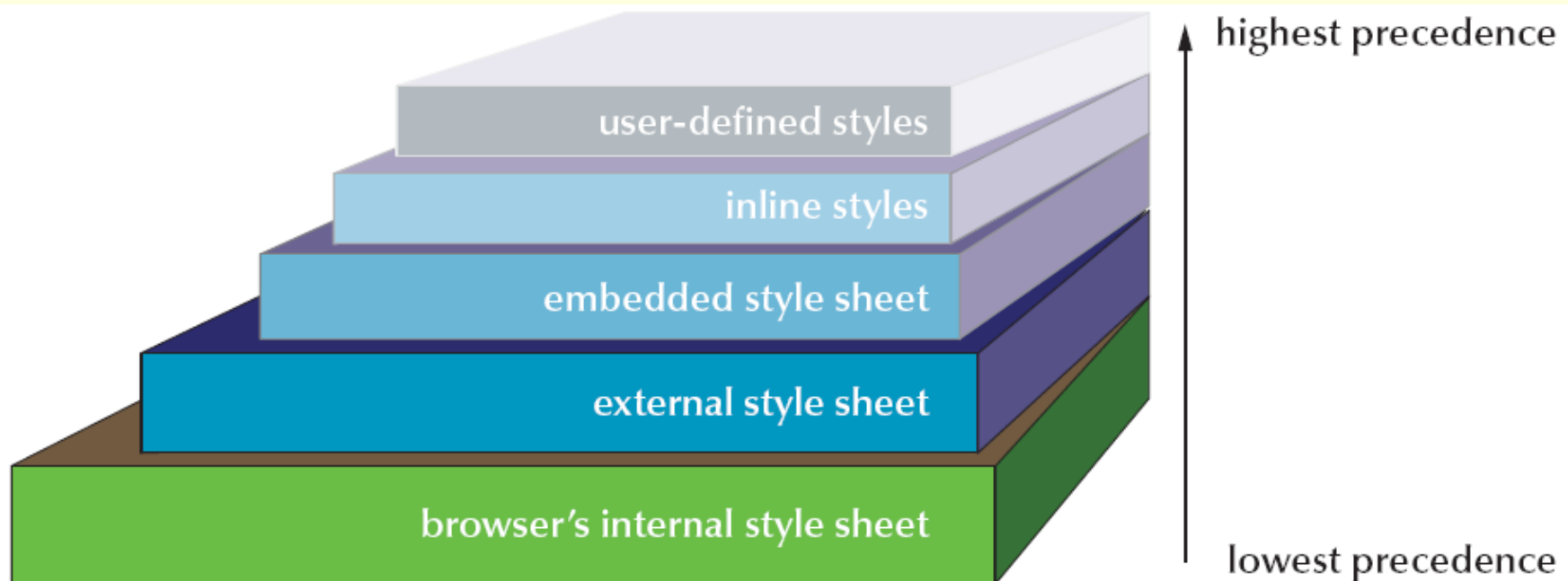
---

- Style Precedence:
  - Next three levels:
    - Next three precedence levels are those defined by the web site author
    - External style sheets
      - These have precedence over browser's built-in styles
    - Embedded style sheets
      - These have precedence over external style sheets
    - Inline styles applied to specific elements
      - These have precedence over embedded style sheets
    - User styles
      - Users can specify their own styles (ex: for accessibility).
      - These styles take precedence over everything else





# Understanding Cascading Order





# Understanding Cascading Order

- Style Precedence:

- Conflicting styles on same level:

- If two styles, on the same level, have a conflict, the one that was declared last has the same precedence.

- Example:

```
<style type="text/css">
```

```
 h1 {color: orange; text-align: center}
```

```
 h1 {color: blue}
```

```
</style>
```

- How are h1 headings displayed with this style?

- Centered, blue text

- text is blue because blue is the last color declaration, but it is centered because there is only one text-align attribute



# Style Inheritance

---

## ■ Concept of Inheritance

- When there are no conflicts, styles are passed down from the more general levels to the more specific ones
- This is called Style Inheritance
  - So if a style is not specified for an element, that element will take on (inherit) the style of its parent element
- Example:
  - If an external style sheet sets h1 headings to display as blue, this color is assumed for ALL h1 headings unless a different color, in a more specific style level, is specified



# Style Inheritance

---

## ■ Concept of Inheritance

- This concept of inheritance is also true for page elements that are nested within other elements

## ■ Example:

- To set the font color of **every** page element to blue, you can use the following style declaration:

```
body {color: blue}
```

- So now, every element that is nested inside the body tags will display as blue text
  - every h1 heading, every paragraph, every bulleted list, every numbered list...EVERYTHING



# Style Inheritance

---

- Concept of Inheritance

- To **override** style inheritance, you simply specify an alternate style for one of the descendant elements of the parent

- Example:

```
body {color: blue}
```

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

- Here, the text color of all elements will be blue
- However, paragraphs, and any elements inside paragraphs, will display in a red font
- Any changes to a style sheet automatically pass down to all levels...hence “cascading style sheets”



# Applying a Style to a Specific ID

---

- Getting really specific...
  - Thus far, you've seen how to change the appearance of standard elements
    - headings, paragraphs, etc
    - really, any of the default elements
  - What if you want to apply a style to a specific id value
    - meaning, to an "id" that you specified in the code
  - Syntax:
    - `#id {style rules}`
      - where id is the value of the element's id attribute and style rule stands for the styles applied to that specific element



# Applying a Style to a Specific ID

---

- Getting really specific...
  - Example:
    - Given the following h2 element
    - `<h2 id="main">A Fun Family Farm</h2>`
    - Now, you can make the following declaration in the HTML code, as an inline style or embedded css, or within an external style sheet:
      - `#main {color: red}`
        - So all elements, marked as “main”, will display as red text
    - Just “plugging” a concept:
      - Developers can couple this, with the use of `<div>` and `<span>` elements to make fully customized web pages



# Working with Color in HTML and CSS

---

- HTML is a text-based language, requiring you to define your colors in textual terms
- HTML identifies a color in one of two ways:
  - By the color value
  - By the color name
- To have more control and more choices, specify colors using **color values**
- A **color value** is a numerical expression that precisely describes a color





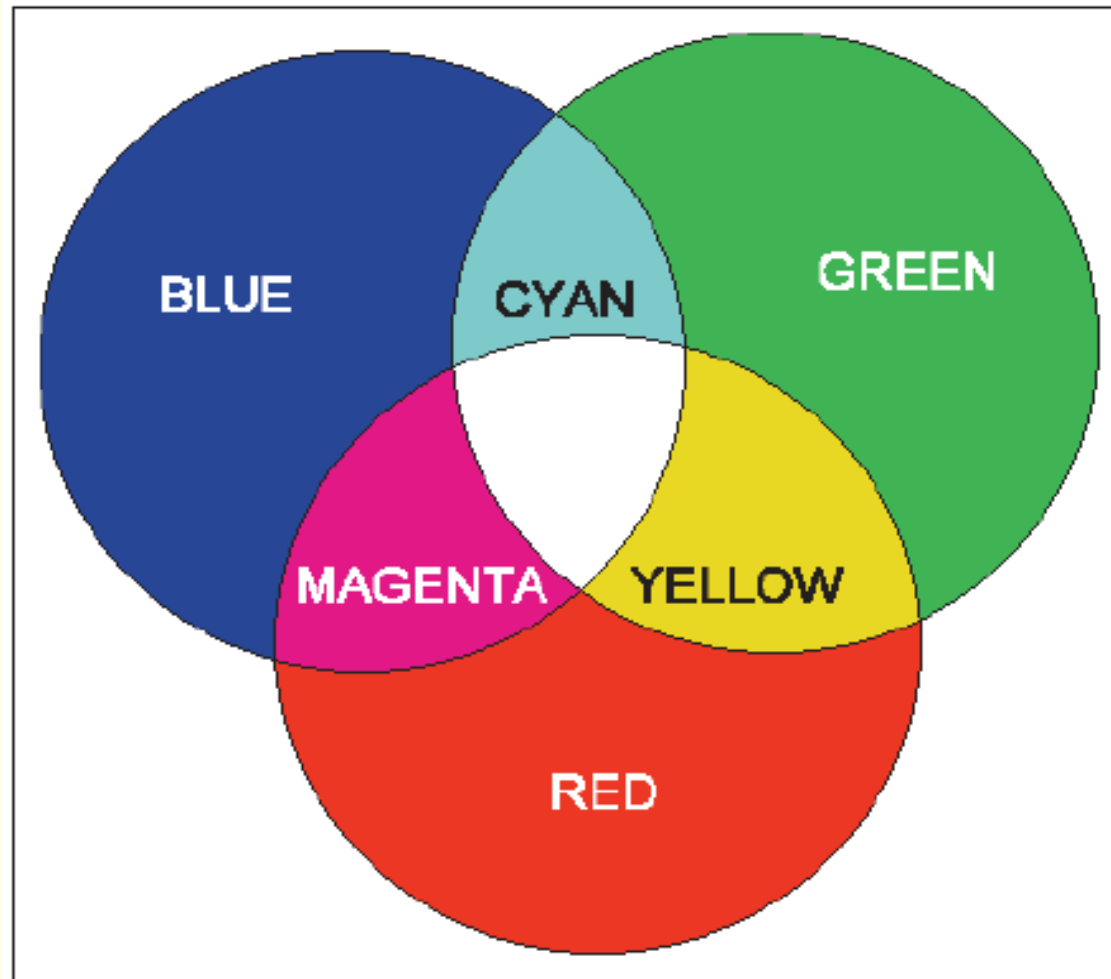
# Working with Color in HTML and CSS

---

- Any color can be thought of as a combination of three primary colors: **red**, **green**, and **blue**
  - Cyan, magenta, yellow, and black (known as CMYK) can be formed by a combination of the aforementioned three colors
- By varying the intensity of each primary color, you can create almost any color and any shade of color
  - This principle allows a computer monitor to combine pixels of red, green, and blue to create the array of colors you see on your screen



# Working with Color in HTML and CSS





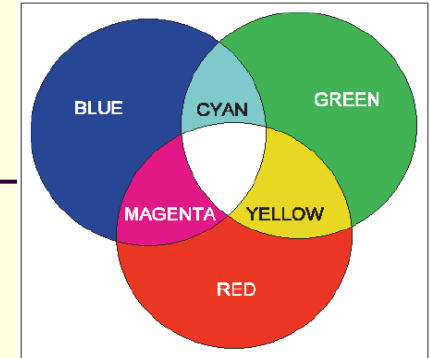
# Working with Color in HTML and CSS

---

- Software programs, such as your Web browser, define color mathematically
  - The **intensity** of each of three colors (RGB) is assigned a number from 0 (absence of color) to 255 (highest intensity)
    - In this way,  $255^3$ , or more than 16.7 million, distinct colors can be defined
  - CSS represents these intensities by a **triplet of numbers**, called an **RGB triplet**, based on the strength of its **R**ed, **G**reen, and **B**lue components



# Working with Color in HTML and CSS



## ■ RGB Triplets:

### ■ Examples:

- `rgb (255,255,0)`
  - Yellow
- `rgb (255,0,255)`
  - Magenta
- `rgb (0,255,255)`
  - Cyan
- `rgb (255,255,255)`
  - white
- `rgb (0,0,0)`
  - black
- `rgb (127,43,241)`
  - Some other color



# Working with Color in HTML and CSS

---

- HTML requires color values be entered as hexadecimals
  - We will not go over this for several reasons:
    - You should be using styles
      - Which means, you should be using the rgb triplets
    - Do you REALLY want a lecture going over hexadecimal and how to convert from decimal to hexadecimal?
      - Yeah. I didn't think so!
  - What you need to know:
    - Some websites, older websites, still use hexadecimal coloring. Be aware of this so you are not surprised.
    - Styles are where it's at! You must use styles.



# Working with Color in HTML and CSS

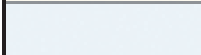
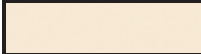

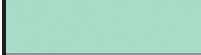

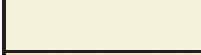


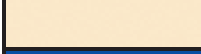




- Using Color Names
  - allows you to accurately display them across different browsers and operating systems
    - You specify a name instead of a rgb triplet
  - The list is short (17 color names), which means it is VERY limiting to Web designers

Color Name	RGB Triplet	Hexadecimal	Color Name	RGB Triplet	Hexadecimal
Aqua	(0, 255, 255)	00FFFF	Olive	(128, 128, 0)	808000
Black	(0, 0, 0)	000000	Orange	(255, 165, 0)	FFA500
Blue	(0, 0, 255)	0000FF	Purple	(128, 0, 128)	800080
Fuchsia	(255, 0, 255)	FF00FF	Red	(255, 0, 0)	FF0000
Gray	(128, 128, 128)	808080	Silver	(192, 192, 192)	C0C0C0
Green	(0, 128, 0)	008000	Teal	(0, 128, 128)	008080
Lime	(0, 255, 0)	00FF00	White	(255, 255, 255)	FFFFFF
Maroon	(128, 0, 0)	800000	Yellow	(255, 255, 0)	FFFF00
Navy	(0, 0, 128)	000080			



# Working with Color in HTML and CSS

## Partial list of extended color names

Sample	Name	RGB	Hexadecimal
	aliceblue	(240,248,255)	#F0F8FF
	antiquewhite	(250,235,215)	#FAEBD7
	aqua	(0,255,255)	#00FFFF
	aquamarine	(127,255,212)	#7FFFD4
	azure	(240,255,255)	#F0FFFF
	beige	(245,245,220)	#F5F5DC
	bisque	(255,228,196)	#FFE4C4
	black	(0,0,0)	#000000
	blanchedalmond	(255,235,205)	#FFEBCD
	blue	(0,0,255)	#0000FF
	blueviolet	(138,43,226)	#8A2BE2
	brown	(165,42,42)	#A52A2A
	burlywood	(222,184,135)	#DEB887



# Defining Text and Background Colors

---

- Now it's time to begin applying these colors to different elements on a web page
  - You've already seen how to change the style of text color
  - Style to define the background color:
    - background-color: color
      - where color is either a color value or a color name
  - Example:
    - h2, h3 headings with white text and a color background

```
<style type="text/css">
 h2, h3 {color: white; background-color: rgb(0, 154, 0)}
</style>
```





# Defining Text and Background Colors

---

- **Deprecated Approaches to Color**
  - There are older, deprecated ways to specify color
  - We will not go over these for obvious reasons
  - But you should be aware of them
    - again, so as not to be surprised
  - See page 139 of textbook for more information



# Brief Interlude: Human Stupidity

---





# Working with Fonts and Text Styles

---

- Choosing a Font:
  - By default, browsers display text in a single font
    - usually Times New Roman
  - You can specify a different font using the style
  - font-family: fonts
    - Where fonts is a comma-separated list of fonts that the browser can use in any element
  - Font names can be of two types:
    - Specific fonts
      - These fonts are actually installed on the user's computer
      - Examples are Times New Roman, Arial, Garamond, etc.



# Working with Fonts and Text Styles

## ■ Choosing a Font:

### ■ Font names can be of two types:

#### ■ Generic fonts:

- a name for a grouping of fonts that share similar appearance
- Browsers support five generic font groups as follows:

	Font Samples		
serif	defg	defg	defg
sans-serif	defg	defg	defg
monospace	de fg	defg	defg
cursive	<b>defg</b>	<i>defg</i>	defg
fantasy	<i>defg</i>	<b>defg</b>	DEFG



# Working with Fonts and Text Styles

---

- Appearance of generic fonts:
  - You can't control this
    - Actual appearance may vary widely across different browsers
  - CSS allows you to specify a list of specific fonts along with a generic font
  - You can list the specific fonts first
  - And if the browser doesn't find those fonts on the system, the generic fonts will be used



# Working with Fonts and Text Styles

---

- Appearance of fonts
  - CSS allows you to specify a list of specific fonts along with a generic font

- Example:

```
font-family: Arial, Helvetica, 'Trebuchet MS', sans-serif
```

- This style tells the browser to first look for Arial; if Arial is not available, try Helvetica and then Trebuchet MS.
      - Not that font names containing one or more spaces must be enclosed in single or double quotation marks
    - Finally, if none of those fonts are found, use a generic sans-serif font

```
body {background-color: white; font-family: Arial, Helvetica, sans-serif}
address {text-align: center}
```



# Setting the Font Size

---

- The style to change the font size of text within an element is:
  - **font-size: *length***
    - where *length* is a length of measure
  - Lengths can be specified in four different ways:
    1. with a unit of measurement
    2. with a keyword description
    3. as a percentage of the size of the containing element
    4. With a keyword, expressing the size relative to the size of the containing element



# Setting the Font Size

- Specifying Lengths using measurement units
  - You can use **Absolute units**
    - These units are FIXED in size, regardless of the device rendering the Web page
    - Absolute units specify a font size using one of five standard units of measurement:
      - Millimeters (mm)
      - Centimeters (cm)
      - Inches (in)
      - Points (pt)
        - What is a point?
        - There are 72 pts in an inch.
      - Picas (pc)
        - There are 6 picas in an inch (and 12 points in a pica)





# Setting the Font Size

- Specifying Lengths using measurement units
  - You can use **Absolute units**
    - Example: say you wanted your text to be ½ inch in size
    - Any of the following styles would work:  
`font-size: 0.5in`  
`font-size: 36pt`  
`font-size: 3pc`
  - When to use absolute units?
    - If you know the exact physical specs of the output device (monitor) and want to fix the text size accordingly.
      - That rarely happens, as Web pages can be displayed on a variety of devices



# Setting the Font Size

---

- Specifying Lengths using measurement units
  - You can also use **Relative units**
    - Relative units are expressed relative to the size of other objects on the Web page
    - Many developers prefer relative units, as it allows the page to be scalable
      - text can be rendered the same way no matter what font size is used by the browser
  - Common Relative units:
    - em unit
    - Percentages
    - Relative keywords



# Setting the Font Size

## ■ Relative Units

### ■ Common Relative units:

#### ■ em unit

- expresses the size relative to the font size of the parent element
- Example: for an h1 heading, the parent element would be the Web page body
- So the style, h1 {**font-size: 2em**}
- sets the font size of h1 headings to twice the font size of the body text
- So if the browser is configured to display body text in 12 pt font, the h1 heading would be displayed in 24 pt font

```
body {background-color: white; font-family: Arial, Helvetica, sans-serif}
h2 {font-size: 2em}
address {text-align: center}
```

h2 headings will be twice  
the size of body text



# Setting the Font Size

---

## ■ Relative Units

### ■ Common Relative units:

- Percentages
- Similar to the em unit
- Percentages are based on the font size of the parent element
- The style:  
`h1 {font-size: 200%}`
- sets the font size of h1 headings to be 200%, or twice, that of body text



# Setting the Font Size

## ■ Relative Units

### ■ Common Relative units:

- Keywords
  - You can use seven keywords to express font sizes
    - xx-small, x-small, small, medium, large, x-large, or xx-large
  - Each browser is configured to display text, at a particular size, for each of these keywords
  - But what is that exact size?
    - That is left up to the browser's style sheet
    - You can't guarantee look of the rendered text
- Relative keywords:
  - You can also use relative keywords to make a font size larger or smaller than the surrounding text
  - Keywords: **smaller** or **larger**



# Setting the Font Size

---

## ■ Relative Units

### ■ Common Relative units:

- Keywords
- Example:
- Make the body text display in a small font, and make h2 text display in a font one size larger (which equates to medium in this case)

```
body {font-size: small}
h2 {font-size: larger}
```



# Spacing and Indentation

---

- CSS & Other Typographic Tasks
  - **Kerning** is the amount of space between characters
    - `letter-spacing: value`
  - **Tracking** is the amount of space between words and phrases
    - `word-spacing: value`
    - value is the size of the space between letters or words
    - size is specified using the same units used for font sizing



# Spacing and Indentation

---

## ■ CSS & Other Typographic Tasks

### ■ **Leading** is the space between lines of text

- `line-height: length`
- where length is a specific length or a percentage of the font size of the text on those lines

#### ■ Example:

```
p {line-height: 2}
```

- makes all paragraphs double spaced

### ■ Indentation:

- `text-indent: value`
- where value is a length expressed in absolute or relative units (or as a percentage of the width of the text block)





# Applying Font Features

---

- To specify font styles, use the following style:
  - **font-style:** *type*
    - where type is one of normal, italic, or oblique
      - italic and oblique are very similar, although different browsers may show a subtle difference
- To control font weight for any page element, use the following style:
  - **font-weight:** *weight*
    - where weight is one of normal, bold, bolder, light, lighter, or a font weight value (see book page 150)



# Applying Font Features

---

- To change the appearance of your text, use the following style:
  - **text-decoration: *type***
    - where type is of none (for no decoration), underline, overline, line-through, or blink
    - You can apply several decorative features together
    - Example:
      - text-decoration: underline overline
  - Note: text-decoration can only be applied to textual elements
    - It can't be applied to nontextual elements
      - Such as inline images



# Applying Font Features

---

- Control the “Case” of text:
  - **text-transform: *type***
    - where *type* is capitalize, uppercase, lowercase, or none (to make no changes to the case)
    - So to make the text fully capitalized:
      - **text-transform: capitalize**
  
- Uppercase letters, small font:
  - **font-variant: *type***
    - where *type* is normal (the default) or small caps (small capital letters)
    - Small caps are all uppercase, but they are in a small font
      - so it doesn't detract from other elements on the page



# Aligning Text Vertically

---

- Aligning text:
  - In Tutorial 1, you learned how to horizontally align text using the text-align style
  - You can also vertically align inline elements, within the context of the surrounding block
  - Style:  
**vertical-align: type**
    - where type is one of the keywords on the next page



# Aligning Text Vertically

- Use the vertical-align attribute  
`vertical-align: type`
  - where type is one of the following keywords

Value	Description
baseline	Aligns the element with the bottom of lowercase letters in surrounding text (the default)
bottom	Aligns the bottom of the element with the bottom of the lowest element in surrounding content
middle	Aligns the middle of the element with the middle of the surrounding content
sub	Subscripts the element
super	Superscripts the element
text-bottom	Aligns the bottom of the element with the bottom of the font of the surrounding content
text-top	Aligns the top of the element with the top of the font of the surrounding content
top	Aligns the top of the element with the top of the tallest object in the surrounding content



# Combining All Text Formatting in a Single Style

- You can combine most of them into a single declaration, using the style
  - `font: font-style font-variant font-weight font-size/line-height font-family`

```
body {background-color: white; font-family: Arial, Helvetica, sans-serif}
h2 {font-size: 2em; letter-spacing: 0.4em; text-indent: 1em }
address {text-align: center; font: normal small-caps 0.8em sans-serif}
```

font style

- Note: properties must be inserted in the ORDER shown



# Working with GIF Images

---

- **GIF (Graphics Interchange Format)** is one of the most commonly used image formats on the Web
  - Compatible with virtually all browsers
  - GIF files are limited to displaying 256 colors
    - So they don't work well for images with lots of colors such as photographs
  - Often used for graphics requiring fewer colors, such as clip art images, line art, logos, and icons
  - Images that require more color depth, such as photographs, can appear grainy when saved as GIF files



# Working with GIF Images

---

- Cool things with GIFs:
  - GIF images support transparency
  - A **transparent color** is a color that is not displayed when the image is viewed in an application
  - Instead, the browser will display whatever is on the background of the page
  - The process of creating a transparent GIF image is dependent on the graphics program





# Working with GIF Images

---

- Cool things with GIFs:
  - GIF images support animation
  - An **animated GIF** is composed of several images that are displayed one after another
    - thereby creating the illusion of motion
    - They were cool back in the day (10+ years ago)
    - Many developers now consider them “cheesy”



# Splash Screen

---

- What is a splash screen?
  - A **splash screen** is a Web page containing interesting animation or graphics that introduces a Web site
  - You then “enter” the website by clicking one or more links within this splash screen
  - This was also very common in the past
  - Not used as often anymore



# JPEG Images

---

- **JPEG** is the other main (commonly used) image format on the Web
  - **JPEG** stands for **Joint Photographic Experts Group**
  - Supports up to 16.7 million colors
  - Most often used for photographs and other images that cover a wide spectrum of color
  - Although jpegs use more colors, they are usually smaller than their GIF counterparts
    - because of the compression algorithm used by jpegs



# PNG Images

---

- A file format called **PNG (Portable Network Graphics)** has been gaining wider acceptance
  - PNG files include most of the same features as GIFs (including transparency and animation)
  - Also, PNG files provide good compression and the full 16.7 million colors available with JPEGs
  - The only problem with PNG:
    - Older browsers don't support them
    - However, this is becoming less of a problem as time goes by



# Comparison of Web Graphic Formats

---

Feature	GIF	JPEG	PNG
Color resolution	256	16.7 million	16.7 million
Useful for line art	Yes	No	Yes
Useful for photographs	No	Yes	Yes
Interlacing/progressive encoding	Yes	Yes	Yes
Compressible	Yes	Yes	Yes
Transparent colors	Yes (1)	No	Yes (multiple)
Supported by older browsers	Yes	Yes	No



# Setting the Image Size

- By default, browsers display an image at its saved size
- You can specify a different size by adding the HTML attributes

`width="value" height="value"`

```
<body>
 <div style="text-align: center">

 </div>
</body>
```



# Setting the Image Size

---

## ■ Note:

- Specifying the image's dimensions, within the browser, does NOT affect/change the file size
  - If you want to decrease the file size of an image, you must do so with an image editing program
- Good idea to specify the image size even if you are not trying to change the dimensions. Why?
  - When a browser encounters an inline image in the HTML file, it calculates the image size and uses that size to properly lay out the page
  - Including the size saves the browser from having to perform the calculation...thus saving rendering time



# Formatting Backgrounds

- You can add a background image to any element
- Syntax:

**background-image: url(*url*)**

- **URL** is the location and filename of the graphic file you want to use for the background of the Web page

```
<body style="background-image: url(background.jpg)">
 <div style="text-align: center">

 </div>
</body>
```





# Background Image Options

- By default, background images are tiled both horizontally and vertically until the entire background of the element is filled up
  - You can specify the direction of the tiling using the style:

**background-repeat: *type***

- where *type* is one of:

Value	Description
repeat	The image is tiled both horizontally and vertically until the entire background of the element is covered
repeat-x	The image is tiled only horizontally across the width of the element
repeat-y	The image is tiled only vertically across the height of the element
no-repeat	The image is not repeated at all



# Background Image Options

The diagram illustrates four different background image options for a rectangular container. On the left, a small box labeled "background image" contains a cluster of colorful circles (yellow, green, blue, and grey) of various sizes. To the right, four larger boxes show how this image is applied:

- background-image: repeat**: The image is tiled both horizontally and vertically, filling the entire container.
- background-image: repeat-x**: The image is tiled only horizontally, creating a repeating pattern across the top of the container.
- background-image: repeat-y**: The image is tiled only vertically, creating a repeating pattern along the left side of the container.
- background-image: no-repeat**: The image is placed only once in the top-left corner of the container.



# Background Image Options

---

- Options for background image placement:
  - Browsers place a background image in an element's upper-left corner
  - You can change this position using the style:  
**background-position: *horizontal vertical***
    - where horizontal is the horizontal position of the image and vertical is its vertical position
      - You can specify a position as the distance from the top-left corner of the element, as a percentage of the element's width or height, or with a keyword
    - Example: **background-position: 30px 42px**
      - Background image placed 30 pixels to the right and 42 pixels down from the upper-left corner of element



# Background Image Options

---

- Options for background image placement:
  - Fixed or not fixed?
    - By default, if a user scrolls through a page, the background element moves with its element
    - You can change this with the style:  
**background-attachment: type**
      - where type is either scroll or fixed
      - scroll means the image scrolls with the element
      - fixed means the image will stay at that exact spot and not move



# The Background Style

---

- You can combine the various background styles into the following single style:

```
background: color url(url) repeat
attachment horizontal vertical
```

- where color, (url), repeat, attachment, horizontal, and vertical are the values just discussed

- Example:

```
background: yellow url(logo.gif) no-repeat
fixed center center
```

- This creates a yellow background, on which the image file, logo.gif, is displayed without being tiled (without repeating). The logo is fixed in the horizontal and vertical center of the element.



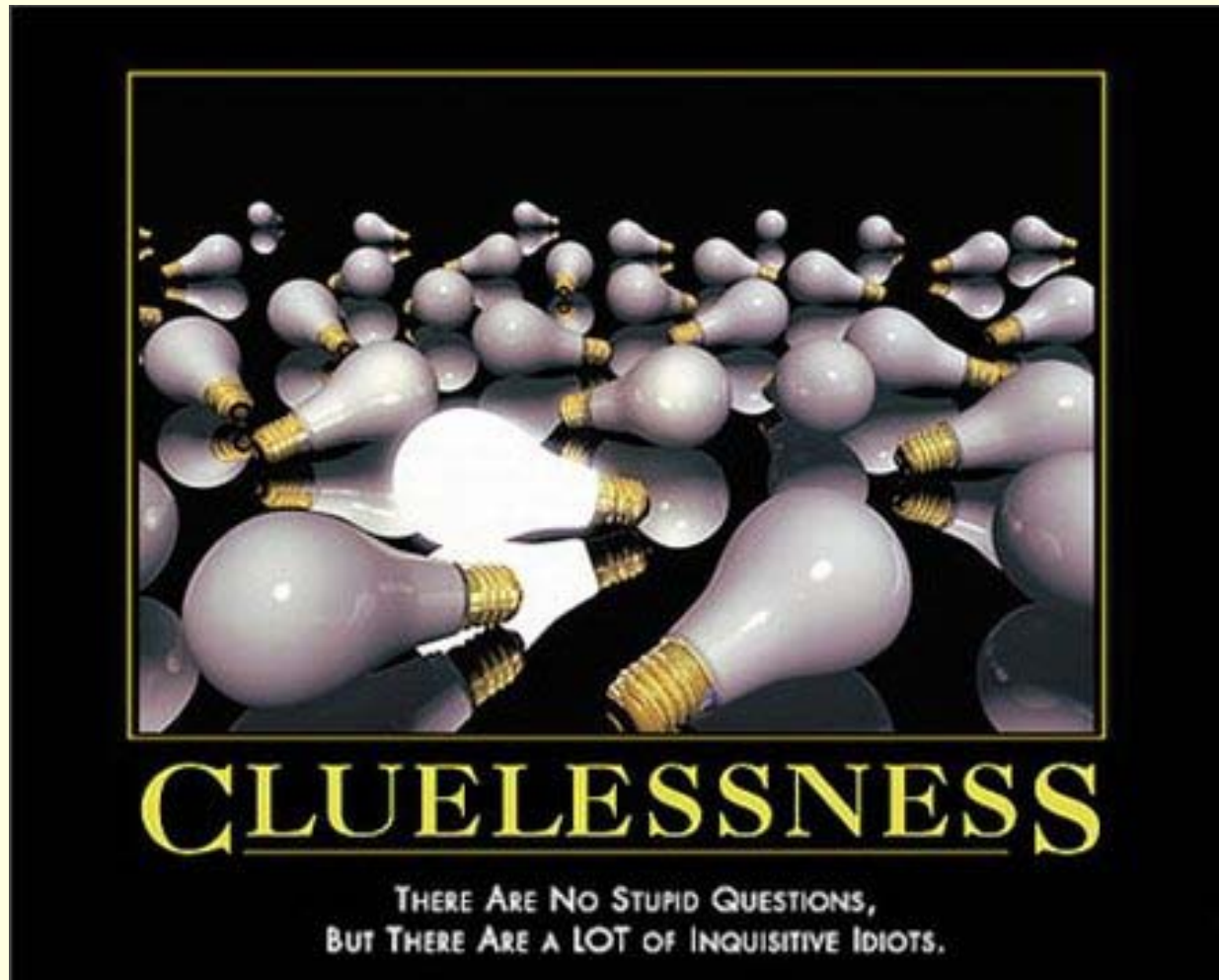
# Tutorial 3: Working with CSS

---

**WASN'T  
THAT  
ASTONISHING!**



# Daily Demotivator



# Tutorial 3: Working with Cascading Style Sheets



Computer Science Department  
University of Central Florida

*COP 3175 – Internet Applications*





# Floating an Element

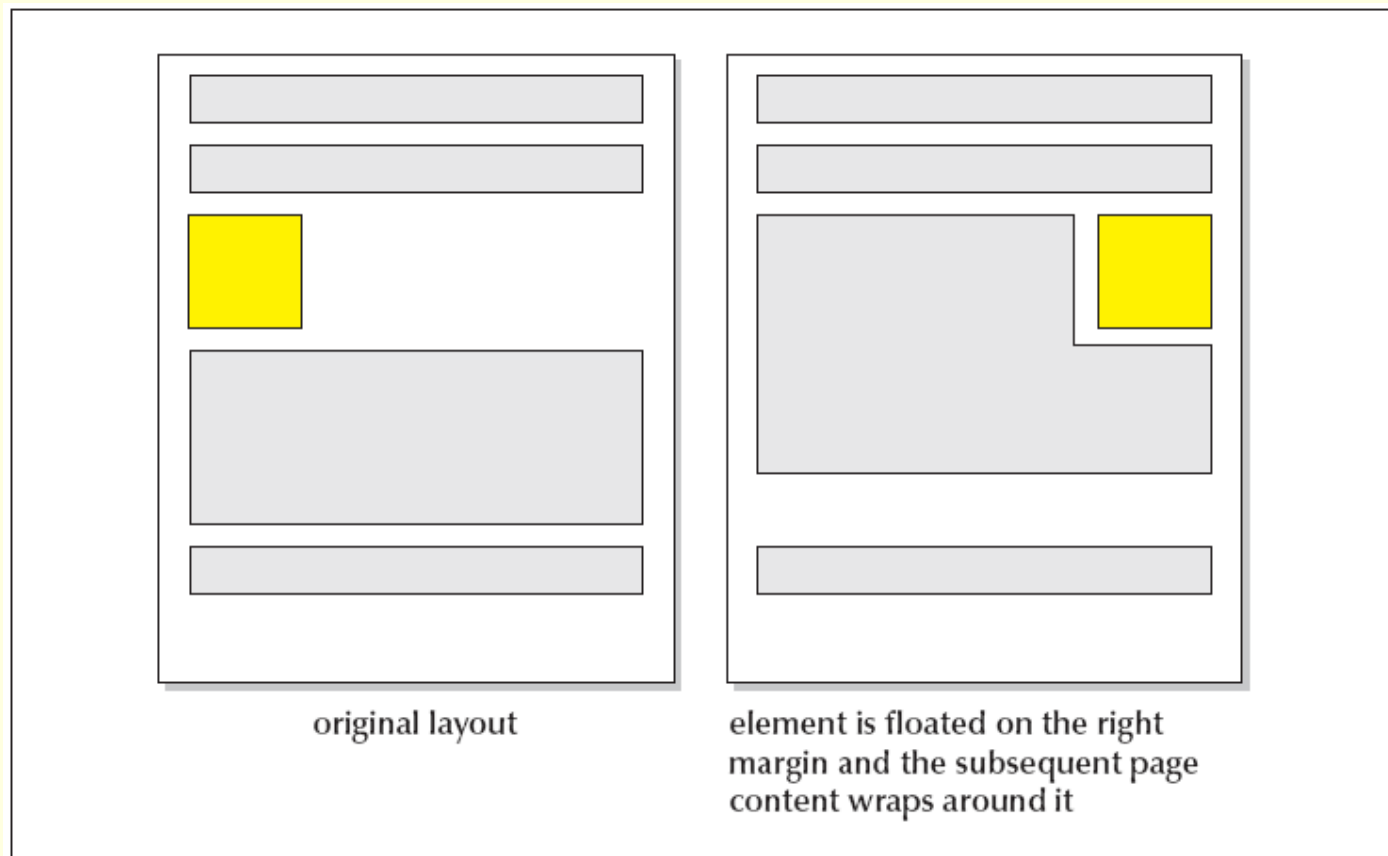
---

- Non-floating elements:
  - Thus far, we've looked at block-level elements that do not float.
  - Meaning, these elements have a space above and below them
    - And nothing else goes on the same line/row as them
  - You can also **float** an element
    - causing it to move out of the normal document flow on the page
    - The other elements on the page that are not floated are then move up to occupy the position previously occupied by the floating element



# Floating an Element

- Example:



original layout

element is floated on the right margin and the subsequent page content wraps around it



# Floating an Element

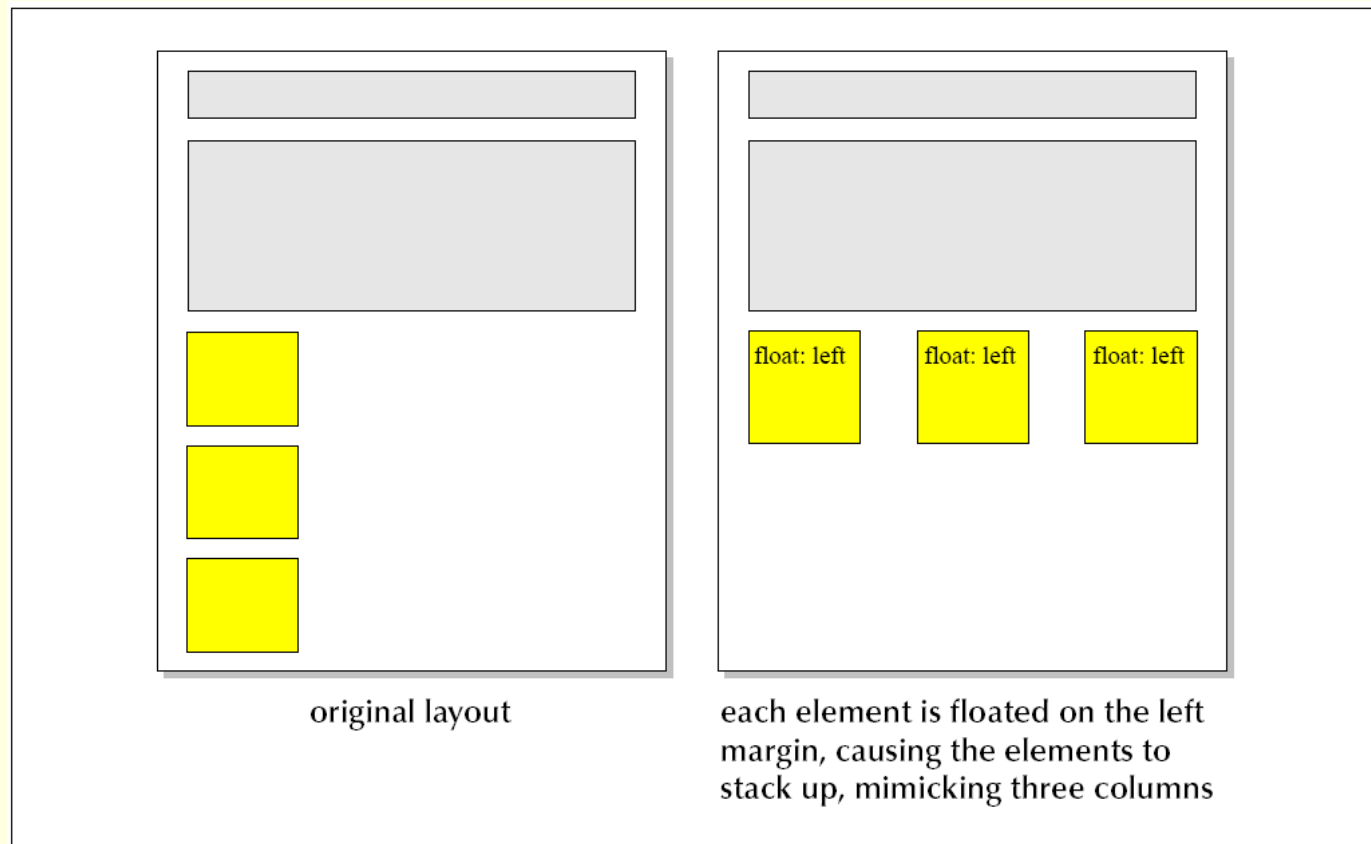
---

- To float an element, use the style **`float: position`**
  - where *position* is none (to turn off floating), left or right
- You can also stack floating elements to create a column effect in your page layout
  - See image on next page...



# Floating an Element

- Floating multiple elements to create columns:





# Floating an Element

---

- Sometimes you do not want an element to wrap around a floating element
  - Example: you may not want headings to wrap around inline images
  - To display an element clear of a floating element, use the style

**clear: position**

- where *position* is none, left, right, or both
- Example:

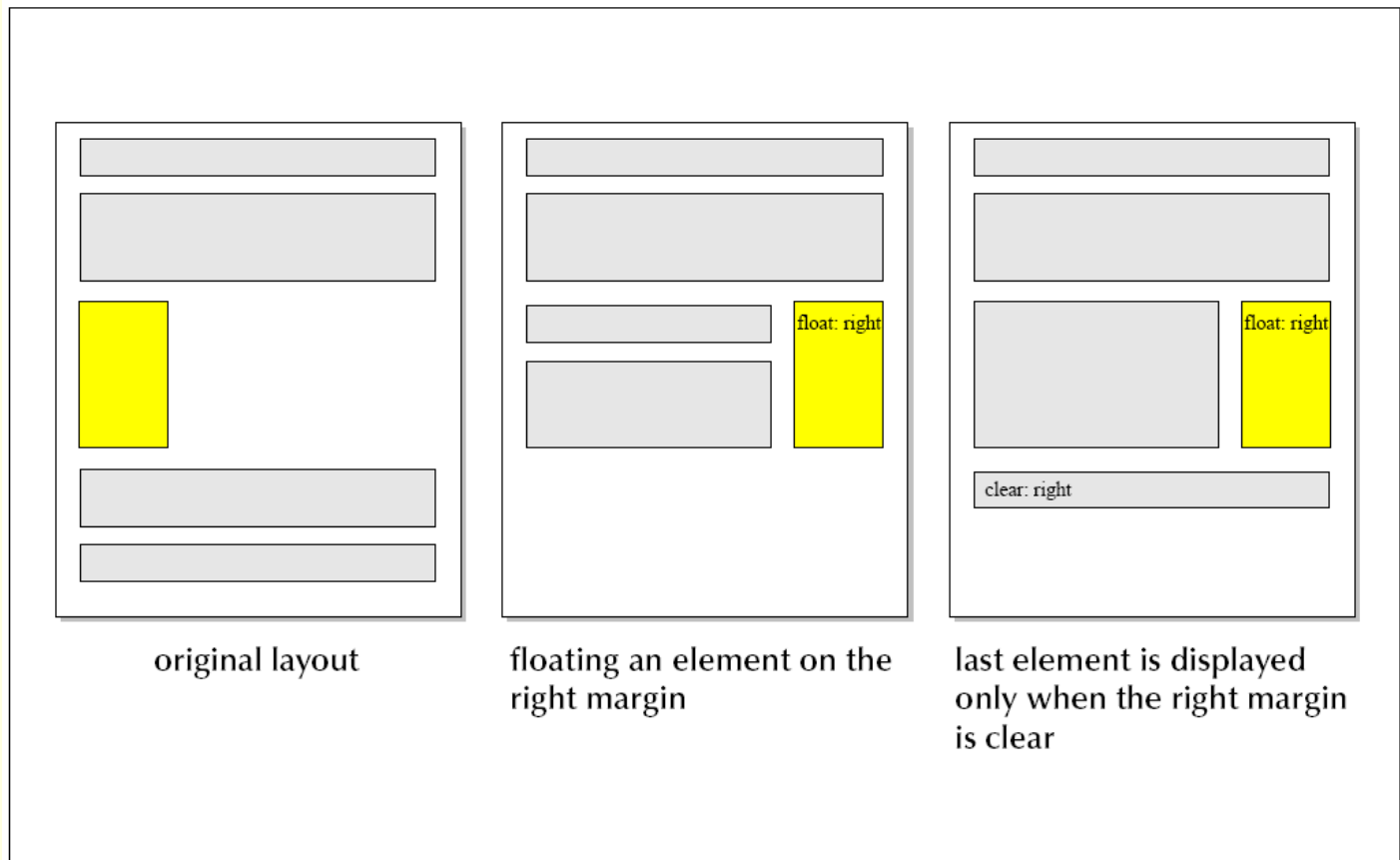
**clear: right**

- Causes the element not to be displayed until the right margin of the parent element is clear of floating objects



# Floating an Element

- Using the clear style:



original layout

floating an element on the right margin

last element is displayed only when the right margin is clear



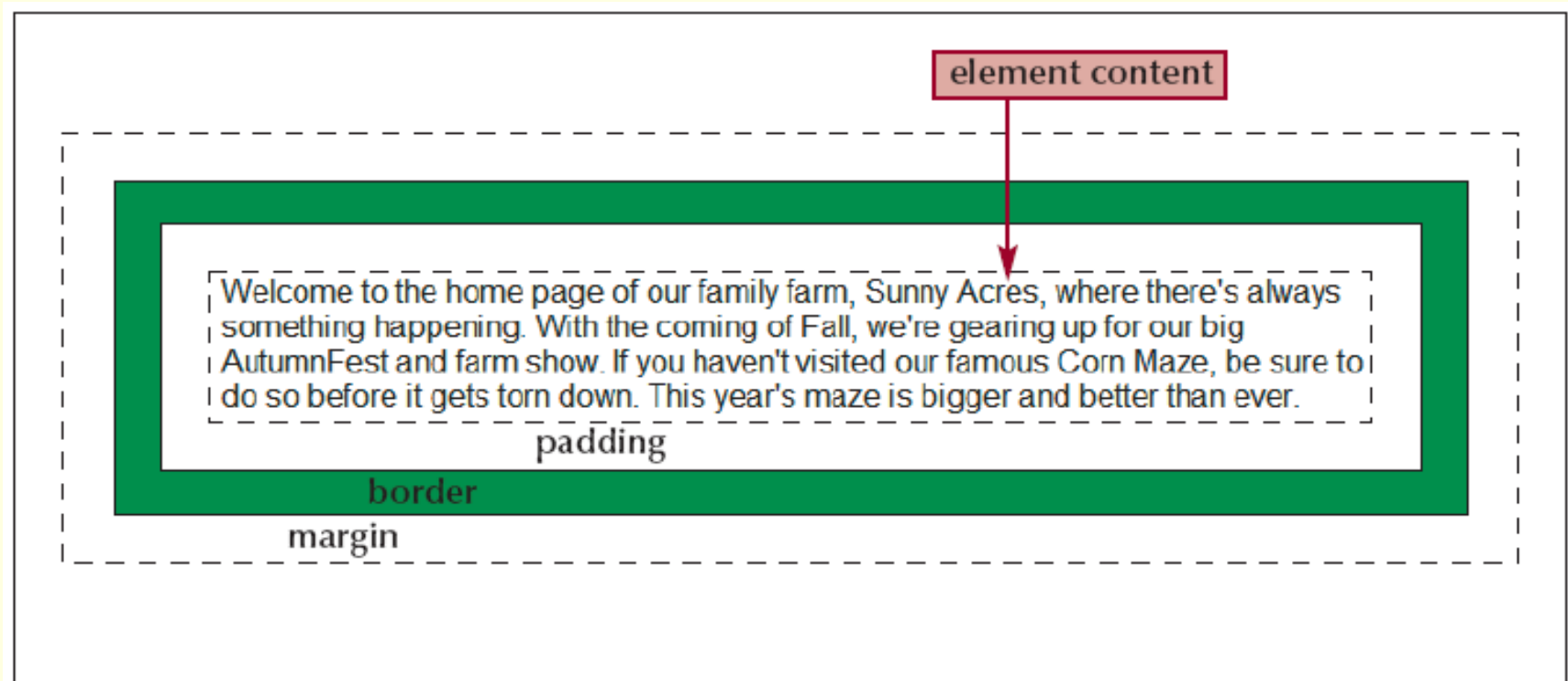
# Working with the Box Model

---

- What is the box model?
  - Think of each element as being housed in a box, and that box has properties that you can modify
  - The **box model** describes the structure of page elements as they are laid out on the Web page
  - The four sections/properties of the box are:
    - 1) The **margin** between the element and other page content
    - 2) The **border** of the box containing the element content
    - 3) The **padding** between the element's content and the box border
    - 4) The **content** of the element itself



# Working with the Box Model







# Margin Styles

---

- Control your margins with the following four styles:
  - `margin-top: length`
  - `margin-right: length`
  - `margin-bottom: length`
  - `margin-left: length`
    - where length is one of the units discussed previously
- Example:

```
h1 {margin-top: 10px; margin-right: 20px;
margin-bottom: 10px; margin-left: 20px}
```

  - creates margins of 10 pixels above and below the h1 heading and margins of 20 pixels to the left and right of the heading



# Margin Styles

---

## ■ Margin Style Options

- The four margin styles can be combined into one single style

**margin: top, right, bottom, left**

- Think of this as moving clockwise around the element, starting with the top margin
- Example:

```
h1 {margin: 10px 20px 10px 20px}
```

  - creates the same set of margins as with the previous example
- Margin values can also be negative.
  - This creates an overlay effect by forcing the browser to render one element on top of another



# Margin Styles

---

## ■ Margin Style Options

- You don't have to supply values for all the margins
  - If you specify three values, they are applied to top, right, and bottom
  - If you specify two values, they are applied to top and right
  - If you specify only one value, it is applied to ALL four margins

## ■ Example:

```
h1 {margin 10px}
```

- Creates a 10-pixel margin around the entire heading



# Padding Styles

---

- Styles to set padding are similar to styles to set margins:
  - `padding-top: value`
  - `padding-right: value`
  - `padding-bottom: value`
  - `padding-left: value`
- You can also combine these into a single style:
  - `padding: top right bottom left`
- If you use a single value, it is applied to all sides:
  - `h1 {padding: 5px}`
    - sets padding space around the h1 heading content to 5 pixels in each direction



# Border Styles

---

- CSS supports three types of styles for the box model border
  - border width, border color, and border style
- As with the margin and padding styles, these styles affect the top, right, bottom, and left borders
  - or all borders at once



# Border Styles

---

- **border-width style:**
  - **border-top-width: length**
  - **border-right-width: length**
  - **border-bottom-width: length**
  - **border-left-width: length**
- Or use the following style to set the width using one line:
  - **border-width: top right bottom left**



# Border Styles

---

- **border-color** style:
  - If a border is present (based on border width), you can set its color:
  - **border-top-color: value**
  - **border-right-color: value**
  - **border-bottom-color: value**
  - **border-left-color: value**
  - **border-color: top right bottom left**
    - where value is a color name, color value, or the keyword “transparent”, which creates an invisible border



# Border Styles

---

- **border-color** style:

- Example:

- Make a style that adds a 4-pixel, red border directly above the address element

```
address {border-top-width: 4px;
border-top-color: red}
```

- Note:

- If you do not use a border, the browser uses the text color of the element within the box.





# Border Styles

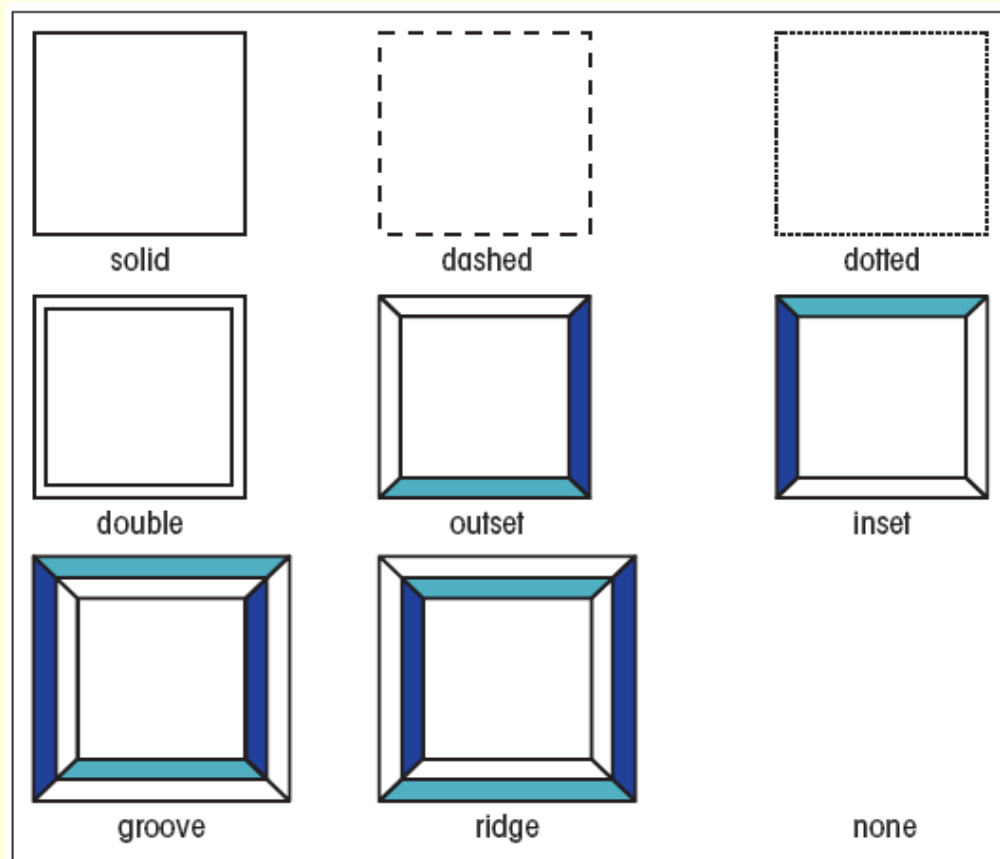
---

- **border-style** style:
  - The final border style defines the actual border style (design)
  - The five border styles are:
    - `border-top-style: type`
    - `border-right-style: type`
    - `border-bottom-style: type`
    - `border-left-style: type`
    - `border-style: top right bottom left`
    - where type is one of nine border styles shown on the following page



# Border Styles

## ■ border-style style:





# Width and Height Styles

---

- Changing the width and height of the whole box
  - Default widths are determined by the browser
    - For inline elements, the width is the width of the element content, and the height is a single line
    - For block-level elements, the width extends across the entire width of the parent element
    - The height of block-level elements expands to meet the content enclosed within that block



# Width and Height Styles

---

- To set the box model width, use
  - width: *length*
    - where *length* is the width of the box content in one of the CSS units of measure.
    - NOTE:
      - Per the CSS specifications, the width value does not take into account the size of the margins, padding space, or borders
        - It only applies to the actual content of the element
      - Most browsers follow the CSS specifications
        - except Internet Explorer
        - Internet Explorer applies the width value to the box model content, padding space, and border



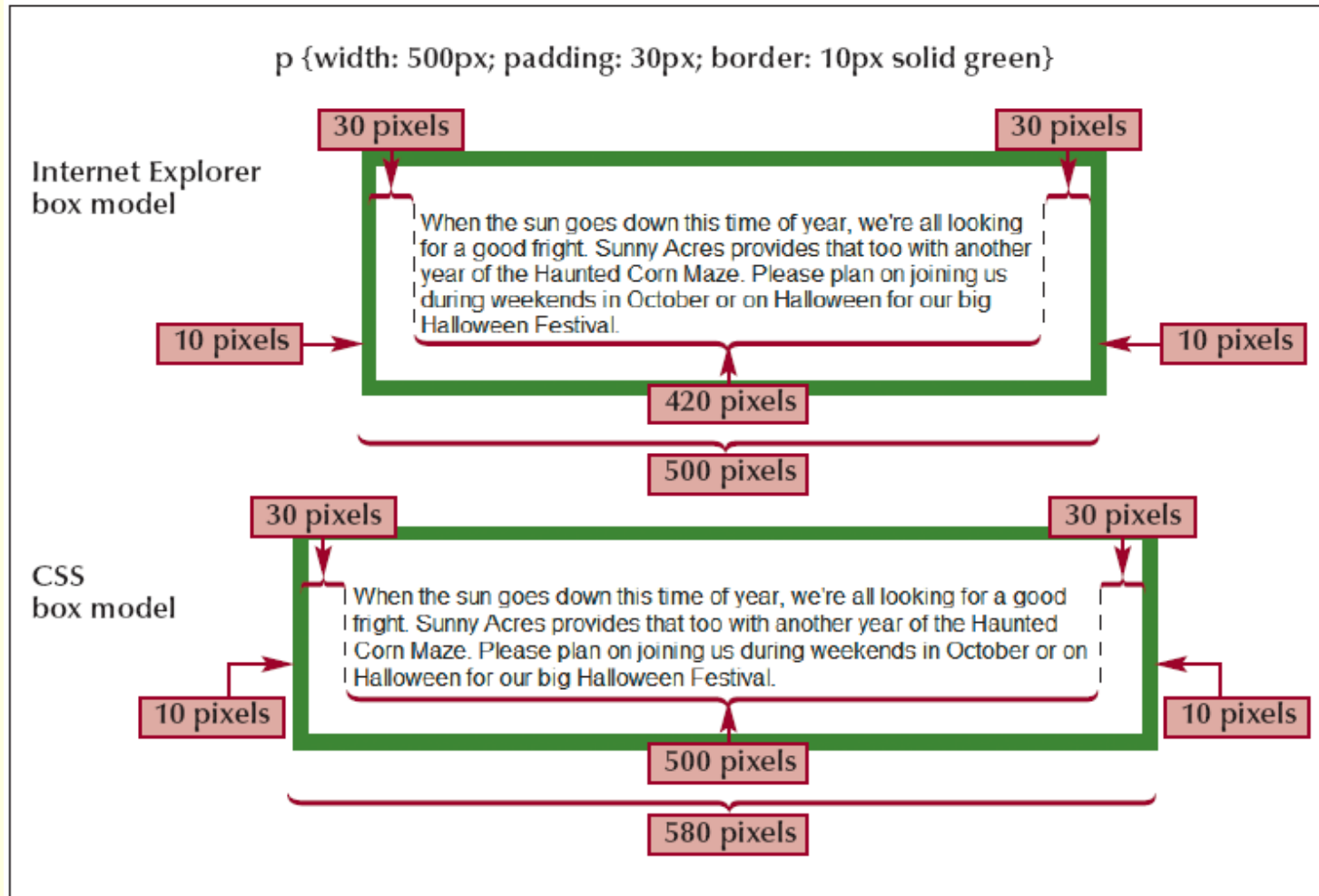
# Width and Height Styles

---

- To set the box model width, use
  - Example:
    - `p {width: 500px; padding: 30px; border: 10px solid green}`
    - What happens in Internet Explorer:
      - results in a paragraph that is 500 pixels wide with 420 of those pixels reserved for the width of the element
      - the other 80 pixels are used for the padding and border
    - What happens in browsers that **follow** CSS specs:
      - results in element content having a width of 500 pixels
      - the width of the entire box, including the padding and borders, is 580 pixels
  - The following picture shows an example of the differences



# Width and Height Styles





# Width and Height Styles

---

- To set the box model height, use
  - height: *length*
    - where *length* is the height of the box content in one of the CSS units of measure
    - If the height value chosen is too small to hold the box's content, the browser will ignore the height value and expand the height of the box as needed
  - Tutorial 4 will show how to override this behavior.



# Brief Interlude: Human Stupidity

---







# Controlling Page Layout with `div` Containers

---

- Using `<div>` elements to control layout
  - Remember from tutorial 1, we “plugged” the idea of `<div>` and `<span>` tags, saying they allow the developer to have full control over page layout
  - Well, you can use `<div>` elements to create “containers” that hold page content
  - These are known as `div` containers
    - You can resize them and float them to create a variety of page layouts



# Controlling Page Layout with div Containers

## ■ Tutorial 3 Example Web site

The screenshot shows the homepage of Sunny Acres. At the top left is the logo with a sun icon. To the right is the address: Tammy and Brent Nelson, 1977 Highway G, Council Bluffs, IA 51503. Below the logo is a navigation menu with links: Home, The Corn Maze, The Haunted Maze, Petting Barn, Produce. A green bar with the word 'Welcome' in white is prominent. The main content area contains several paragraphs of text and a photo of red apples. At the bottom, there are sections for 'Hours' and 'Directions' with bulleted lists of information. A footer at the very bottom repeats the contact information.

**Sunny Acres**  
 Tammy and Brent Nelson  
 1977 Highway G  
 Council Bluffs, IA 51503

[Home](#) [The Corn Maze](#) [The Haunted Maze](#) [Petting Barn](#) [Produce](#)

### Welcome

Welcome to the home page of our family farm, Sunny Acres, where there's always something happening. With the coming of fall, we're gearing up for our big AutumnFest and Farm Show. If you haven't visited our famous Corn Maze, be sure to do so before it gets torn down on November 5. This year's maze is bigger and better than ever.

Farms can be educational and Sunny Acres is no exception. Schools and home-schooling parents, take an afternoon with us at our Petting Barn. We have over 100 friendly farm animals in a clean environment. Kids can bottle feed the baby goats, lambs, and calves while they learn about nature and the farming life. Please call ahead for large school groups.

When the sun goes down this time of year, we're all looking for a good fight. Sunny Acres provides that too with another year of the Haunted Maze. Please plan on joining us during weekends in October or on Halloween for our big Halloween Festival.

Of course, Sunny Acres is above all, a farm. Our Farm Shop is always open with reasonable prices and great produce. Save even more money by picking your own fruits and vegetables from our orchards and gardens.

We all hope to see you soon, down on the farm.

— Tammy & Brent Nelson

#### Hours

- Farm Shop: 9 am - 5 pm Mon - Fri; 9 am - 3 pm Sat
- The Corn Maze: 11 am - 9 pm Sat; 11 am - 5 pm Sun
- The Haunted Maze: 5 pm - 9 pm Fri & Sat
- Petting Barn: 9 am - 4 pm (Mon - Fri); 11 am - 3pm (Sat & Sun)

#### Directions

- From Council Bluffs, proceed east on I-80
- Take Exit 36 North to the Drake Frontage Road
- Turn right on Highway G
- Proceed east for 2.5 miles
- Sunny Acres is on your left, watch for the green sign

SUNNY ACRES • TAMMY & BRENT NELSON • 1977 HIGHWAY G • COUNCIL BLUFFS, IA 51503





# Controlling Page Layout with div Containers

---

- Tutorial 3 Example Web site
  - Site current extends across the screen
    - Bad design
      - Studies show text gets more difficult to read as it extends to beyond 50 or so characters per line
      - Many users may just skip pages with poor design
  - We use div containers to clean up the page and make the design more practical



# Controlling Page Layout with div Containers

- Here's an example from the sample page made in Tutorial 3:
  - Notice the div tag with the id "outer"
  - If we didn't use css to define the behavior of "outer", this tag would do nothing
  - But we will now define, in the css sheet, the behavior of "outer"
  - So everything within the "outer" tags will follow the behavior that we specify

```
<body>
<div id="outer">
 <h1></h1>
 <div id="links">
 Home
 The Corn Maze
 The Haunted Maze
 Petting Barn
 Produce
 </div>
 <h2>welcome</h2>
</div>
<address>
 Sunny Acres ☀
 Tammy & Brent Nielsen ☀
 1977 Highway G ☀
 Council Bluffs, IA 51503
</address>
</div>
</body>
```



# Controlling Page Layout with div Containers

---

- Setting the width of the “outer” container:

```
body {background-color: white; font-family: Arial, Helvetica, sans-serif}
h2 {font-size: 2em; letter-spacing: 0.4em; text-indent: 1em }
h3 {width: 20em; padding-left: 1em}
address {text-align: center; font: normal small-caps 0.8em sans-serif;
 border-top: 0.5em double green; padding-top: 1em}

#promoimage {float: right; margin: 0em 0em 1em 1em}
#outer {width: 50em}
```



# Controlling Page Layout with div Containers

---

- Tutorial 3 Example Web Site:
  - They want a list of links displayed to the left of the page with the follow specs:
    - float it on the left margin of the outer div container
    - Set the width to 10 em
    - Set the background color to white
    - Add an outset border 0.5 em in width

```
#promoimage {float: right; margin: 0em 0em 1em 1em}
#outer {width: 50em}
#links {float: left; width: 10em; background-color: white;
 border-style: outset; border-width: 0.5em}
```



# Controlling Page Layout with div Containers

- Tutorial 3 Example Web Site:
  - Here's the result:





# Controlling Page Layout with div Containers

---

- Tutorial 3 Example Web Site:
  - Here's the result:
    - Floating the links left did save some vertical space
      - But the layout is not attractive
    - We want the main body portion (the Welcome heading and its text) to appear as if it were a separate column to the right of the links
      - So we need to make a container for it!
      - And we need to define the behaviors of this new container using CSS styles





# Controlling Page Layout with div Containers

- Here's the "inner" container
  - Notice the div tag with the id "inner"
  - This tag basically encloses the main portion of the website content
  - It encloses the Welcome heading, the intro paragraph, and other links

```
<div id="inner">
<h2>welcome</h2>
<p>

welcome to the home page of our family farm,
Sunny Acres, where there's always something
happening. with the coming of fall, we're gearing up for our big AutumnFest
and Farm Show. If you haven't visited our famous Corn Maze, be sure to do
so before it gets torn down on November 5. This year's maze is bigger and
better than ever.
</p>

<h3>Directions</h3>

From Council Bluffs, proceed east on I-80
Take Exit 38 North to the Drake Frontage Road
Turn right on Highway G
Proceed east for 2.5 miles
Sunny Acres is on your left; watch for the green sign

</div>
```



# Controlling Page Layout with div Containers

---

- Tutorial 3 Example Web Site:
  - Here's the CSS style defining the behavior of this container:

```
#promoimage {float: right; margin: 0em 0em 1em 1em}
#outer {width: 50em}
#links {float: left; width: 10em; background-color: white;
 border-style: outset; border-width: 0.5em}
#inner {margin-left: 12em; padding-left: 1em;
 border-left: 0.1em solid green}
```



# Controlling Page Layout with div Containers

- Tutorial 3 Example Web Site:

- Here's the resulting layout:

- So we've now used two div containers to separate the content

The screenshot shows a web page for 'Sunny Acres'. The header includes a small image of a farm, the text 'Sunny Acres' with a sun icon, and contact information: 'Jammy and Brent Nielsen, 1973 Hwy G, Council Bluffs, IA 51503'. A navigation menu on the left lists 'Home', 'The Corn Maze', 'The Haunted Maze', 'Petting Barn', and 'Produce'. The main content area has a green 'Welcome' header, followed by a paragraph of text, a photo of red apples, and another paragraph. A red bracket at the bottom indicates the left margin is set to 12 em.



# Setting the Display Style

---

- The page looks much better already
  - But the links look a bit rough
    - They are all bunched together
  - We want each link on a separate line
    - What's one way you already know how to do this?
    - Enclose each link in a `<p>` tag
    - Or insert line breaks
    - However, remember one of the goals of web design:
      - **Separate page content from page layout**
    - So we will use CSS styles to change the display properties of the links



# Setting the Display Style

---

- Setting Display styles

- Syntax:

- **display: type**

- where type is one of the CSS display types on the following page

- Example:

- Links are considered as inline elements
      - Which is why they were all bunched together
    - To display all links as block-level elements, use the style:  
**a {display: block}**
      - Has the effect of having each link be a block-level element



# Setting the Display Style

## Values of the display style

Display	Description
block	Display as a block-level element
inline	Display as an inline element
inline-block	Display as an inline element with some of the properties of a block (much like an inline image or frame)
inherit	Inherit the display property of the element's parent
list-item	Display as a list item
none	Do not display the element
run-in	Display as either an inline or block-level element depending on the context (CSS2)
table	Display as a block-level table
inline-table	Display as an inline table
table-caption	Treat as a table caption
table-cell	Treat as a table cell
table-column	Treat as a table column
table-column-group	Treat as a group of table columns
table-footer-group	Treat as a group of table footer rows
table-header-group	Treat as a group of table header rows
table-row	Treat as a table row
table-row-group	Treat as a group of table rows



# Setting the Display Style

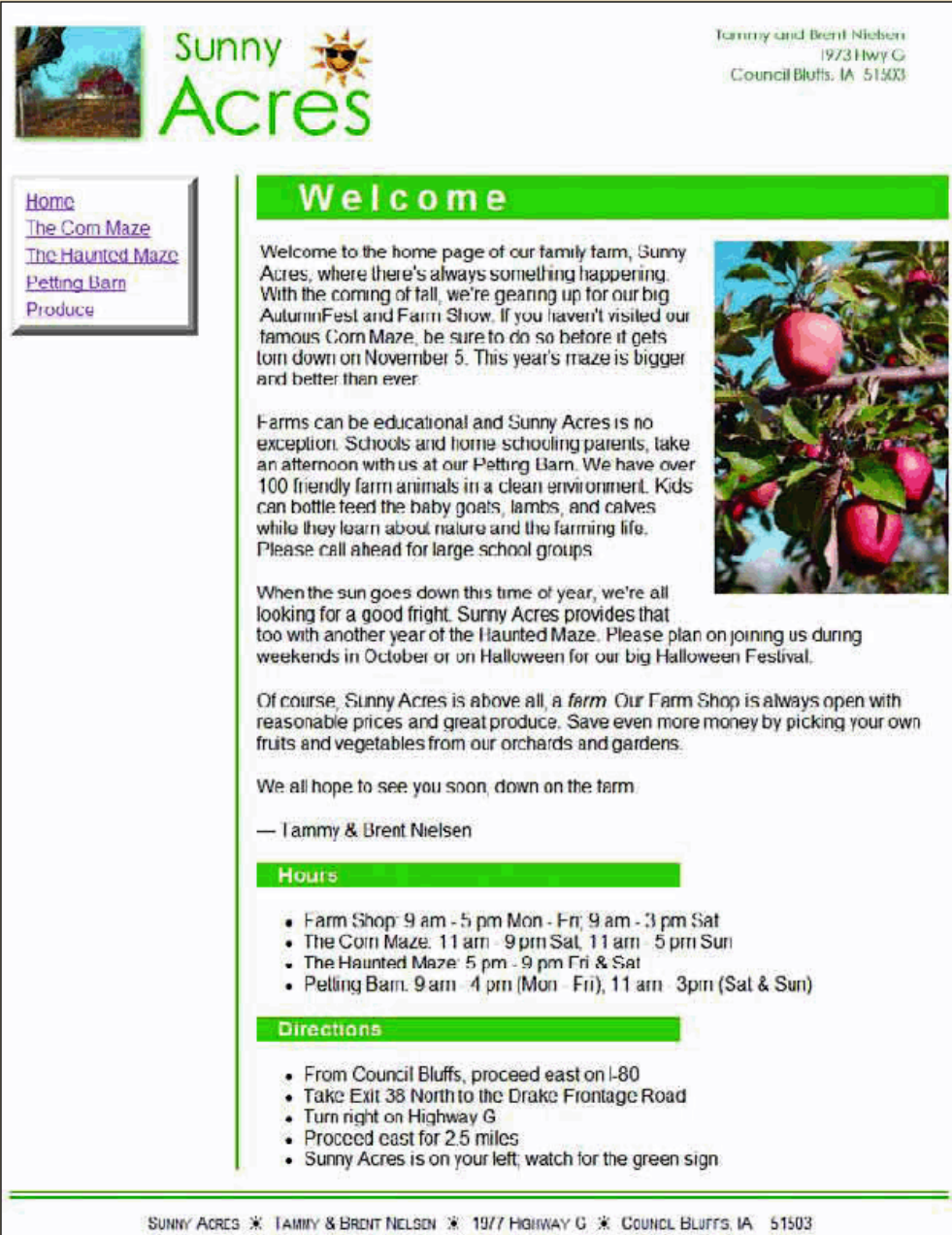
- Tutorial 3 Example Web Site:
  - Making the links as block-level elements

```
body {background-color: white; font-family: Arial, Helvetica, sans-serif}
h2 {font-size: 2em; letter-spacing: 0.4em; text-indent: 1em }
h3 {width: 20em; padding-left: 1em}
address {text-align: center; font: normal small-caps 0.8em sans-serif;
 border-top: 0.5em double green; padding-top: 1em}
a {display: block; margin: 0.3em}

#promoimage {float: right; margin: 0em 0em 1em 1em}
#outer {width: 50em}
#links {float: left; width: 10em; background-color: white;
 border-style: outset; border-width: 0.5em}
#inner {margin-left: 12em; padding-left: 1em;
 border-left: 0.1em solid green}
```

# Final Layout

- Site is now much more presentable and attractive
- Main “outer” container simply controls the overall width of the site
- “links” and “inner” containers represent the two columns that we see



The screenshot shows a website for 'Sunny Acres' with a green and white color scheme. At the top left is a small image of a farm scene. The main header features the text 'Sunny Acres' in green, with a sun icon wearing sunglasses. To the right of the header, contact information for Tammy and Brent Nielsen is provided. A navigation menu on the left lists links for Home, The Corn Maze, The Haunted Maze, Petting Barn, and Produce. The main content area is divided into two columns. The right column has a green header 'Welcome' followed by a paragraph of text, a photo of red apples, another paragraph, and a third paragraph. Below this is a section for 'Hours' and 'Directions', each with a green header and a list of items. At the bottom, a footer contains the website's name, contact info, and address.

**Sunny Acres**

Tammy and Brent Nielsen  
1973 Hwy G  
Council Bluffs, IA 51503

[Home](#)  
[The Corn Maze](#)  
[The Haunted Maze](#)  
[Petting Barn](#)  
[Produce](#)

## Welcome

Welcome to the home page of our family farm, Sunny Acres, where there's always something happening. With the coming of fall, we're gearing up for our big AutumnFest and Farm Show. If you haven't visited our famous Corn Maze, be sure to do so before it gets torn down on November 5. This year's maze is bigger and better than ever.

Farms can be educational and Sunny Acres is no exception. Schools and home schooling parents, take an afternoon with us at our Petting Barn. We have over 100 friendly farm animals in a clean environment. Kids can bottle feed the baby goats, lambs, and calves while they learn about nature and the farming life. Please call ahead for large school groups.

When the sun goes down this time of year, we're all looking for a good fright. Sunny Acres provides that too with another year of the Haunted Maze. Please plan on joining us during weekends in October or on Halloween for our big Halloween Festival.

Of course, Sunny Acres is above all, a *farm*. Our Farm Shop is always open with reasonable prices and great produce. Save even more money by picking your own fruits and vegetables from our orchards and gardens.

We all hope to see you soon, down on the farm.

— Tammy & Brent Nielsen

## Hours

- Farm Shop: 9 am - 5 pm Mon - Fri; 9 am - 3 pm Sat
- The Corn Maze: 11 am - 9 pm Sat, 11 am - 5 pm Sun
- The Haunted Maze: 5 pm - 9 pm Fri & Sat
- Petting Barn: 9 am - 4 pm (Mon - Fri), 11 am - 3pm (Sat & Sun)

## Directions

- From Council Bluffs, proceed east on I-80
- Take Exit 38 North to the Drake Frontage Road
- Turn right on Highway G
- Proceed east for 2.5 miles
- Sunny Acres is on your left, watch for the green sign.

SUNNY ACRES \* TAMMY & BRENT NIELSEN \* 1977 HIGHWAY G \* COUNCIL BLUFFS, IA 51503





# Summary

---

- Learned history and concepts of CSS
- Learned different styles and how they are applied
- Learned CSS use of color and CSS styles for font
- Learned to display an animated graphic
- Learned to float elements and apply style to elements
- Learned the properties of the box model
- Learned how to use div containers



# Tutorial 3: Working with CSS

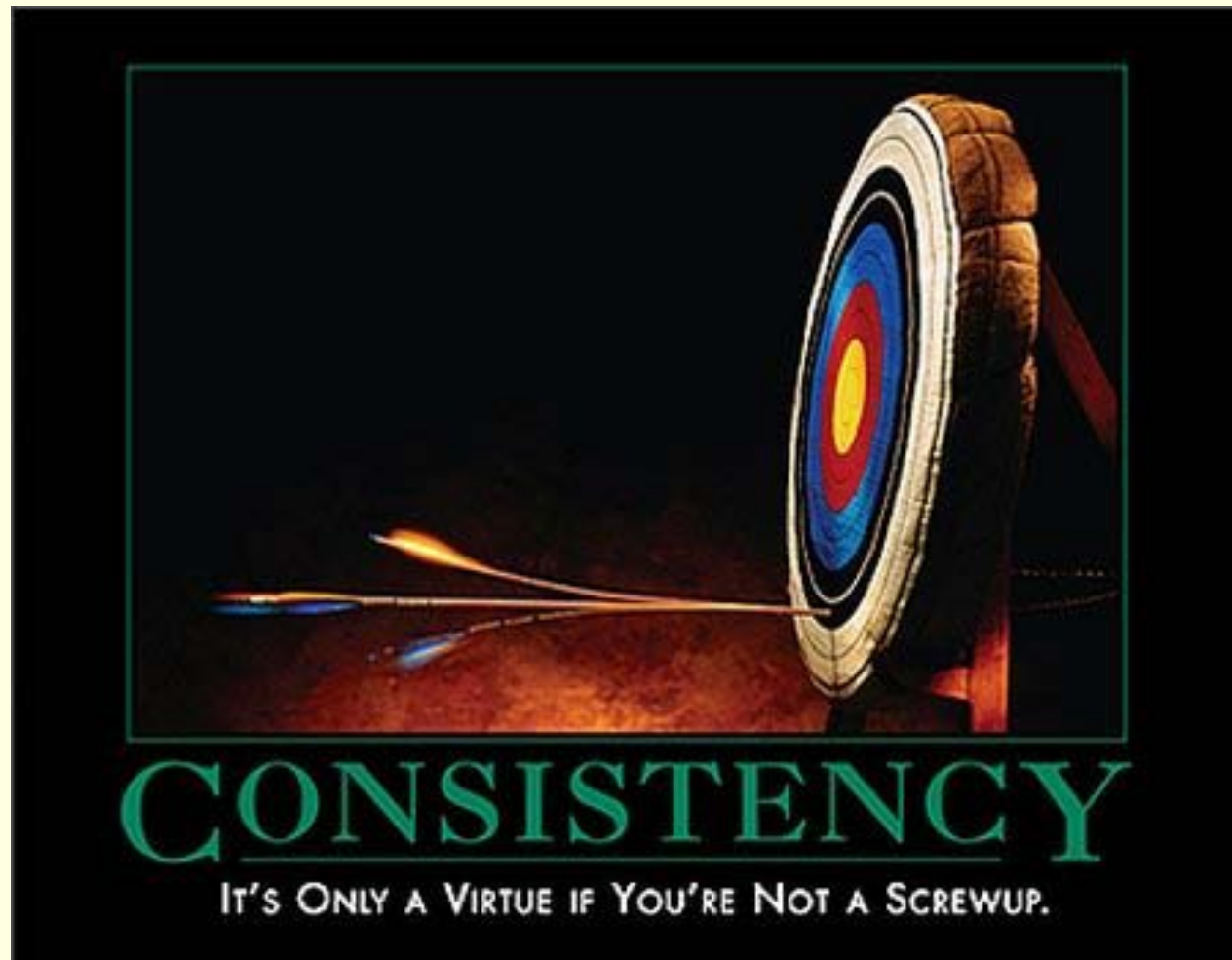
---

**WASN'T  
THAT  
ASTOUNDING!**



# Daily Demotivator

---



# Tutorial 3: Working with Cascading Style Sheets



Computer Science Department  
University of Central Florida

*COP 3175 – Internet Applications*