
# ID1354
# Internet Applications

## CSS

**Leif Lindbäck, Nima Dokoohaki**

**leifl@kth.se, nimad@kth.se**

SCS/ICT/KTH

# What is CSS?

- Cascading Style Sheets, CSS provide the means to control and change presentation of HTML documents.

  - A style sheet (CSS code) is a syntactic mechanism for specifying style information.
  - Style sheets allow you to impose a standard style on a whole document, or even a whole collection of documents.
    - Style is specified for tags by the values of its properties.

# Use CSS for Layout!

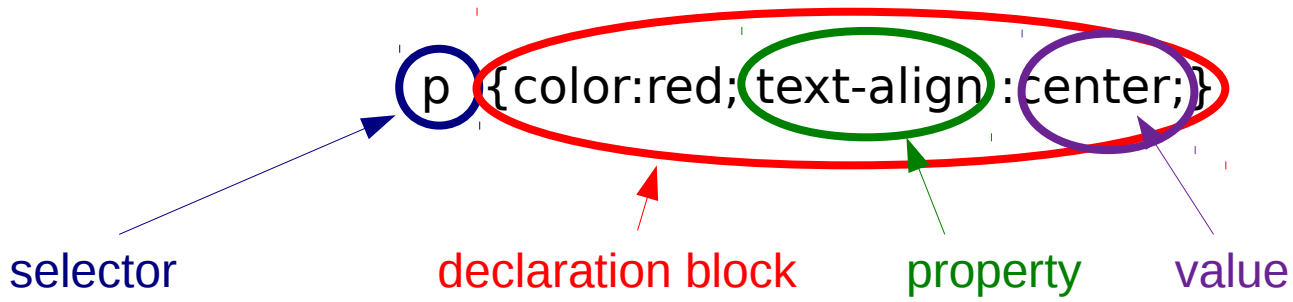
- Remember, HTML is for defining content, not for defining format or layout.
- CSS is the means to define page layout and how HTML elements are displayed.
- Style sheets enables modifying the appearance of an entire web site, just by editing a single file!

# Current CSS Versions

- CSS2.1 is supported by all major browsers.
- CSS3 adds lots of new features and is also fully backwards compatible.
- Do not rely on CSS3 to be fully implemented in all browsers. Check for example at **`http://caniuse.com`**.

# A First Look

- This example will be discussed in detail later in the presentation.
- A sample CSS **rule** is displayed below.
  - The **selector** defines which element(s) is affected by the rule.
  - The **declaration block** contains **property**/**value** pairs separated by semicolon. Each such pair defines some appearance of the selected element(s).



# Levels of Style Sheets

There are **three levels** of style sheets

1. **Inline** - specified as attributes for a tag, apply only to that tag.
   - Avoid this! Using inline tags means duplicated code since rules must be repeated for each tag.

2. **Document-level style sheets** or **internal style sheets** – specified in a HTML document and apply only to that document.
   - Avoid this! Using internal tags means duplicated code since rules must be repeated for each document.

3. **External style sheets** - applied to any number of documents.
   - Use only this level! The same CSS rules are applied to all relevant HTML files.

# External Style Sheets

- External style sheets are separate files, potentially on any server on the Internet. Such files shall have the extension **`.css`**.

- A **`<link>`** tag in a HTML document specifies that the browser is to fetch and use an external style sheet.

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

# CSS Syntax

- Comments are enclosed between **/*** and ***/**

```
/* this is a comment. */
```

- Comments may span multiple lines.

- A style sheet (CSS file) contains a list of style rules, with the syntax specified in the initial example above.

```
hr {color:sienna;}
p {margin-left:20px;}
body {background-image:url("images/back40.gif");}
```

# CSS Errors

- CSS rules with syntax errors or invalid property or value names are silently ignored.
- This might cause much extra work.
- Always validate your CSS files! Use for example the W3C CSS validator,

  `http://jigsaw.w3.org/css-validator/`

# Linking to a Style Sheet

- In a HTML file:

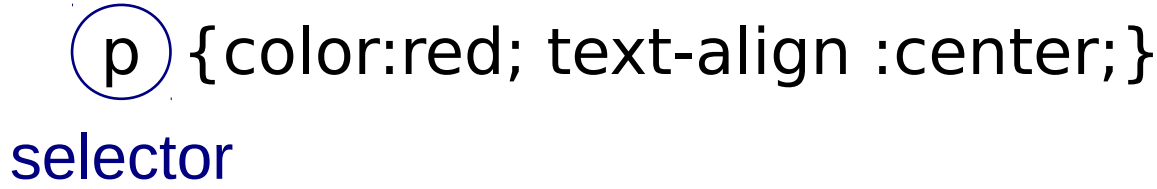
```
<head>
   <link rel="stylesheet" type="text/css"
         href="styles/mystyle.css">
</head>
```

- In the style sheet `mystyle.css`

```
hr {color:sienna;}
p {margin-left:20px;}
body {background-image:url("images/back40.gif");}
```

# Selectors

- The first part of a CSS rule is a selector:




- Determines to which elements the rule aplies.
- There are many different types of selectors.

# Element Selector

- The selector is an element name or a list of element names, separated by commas.
- The rule applies to all elements with the specified name(s).
- Examples:
  - `h1, h3`
  - `p`

# Id Selector

- An id is a logical name for a specific element.
- To specify that a HTML element has a specific id, use the **id** attribute of that element.

```
<p id="wide"> some text </p>
```

The following selector matches all elements with id **wide**, independent of element type.

```
#wide
```

- The following selector matches all elements with id **wide** and type **<p>**.

```
p#wide
```

# Class Selector

- A class is a logical group of elements.
- To specify that a HTML element belongs to a specific class, use the **`class`** attribute of that element.

```
<p class="wide"> some text </p>
```

- The following selector matches all elements of class **`wide`**, independent of element type.

```
.wide
```

- The following selector matches all elements of class **`wide`** and type **`<p>`**.

```
p.wide
```

# Pseudo-Class Selector

- A pseudo-class is a condition, all elements that meet the condition are members of the pseudo-class.
- It is not possible to define new pseudo-classes, but there are many predefined. Some examples are:

| | |
|---|---|
| `:link` | unvisited links |
| `:visited` | visited links |
| `:hover` | applies when the mouse is placed over the element. |
| `:active` | applies while the user is clicking the element. |
| `:focus` | applies when the element has keyboard or mouse focus. |
| `:enabled` | applies to enabled UI components (e.g. buttons). |
| `:disabled` | applies to disabled UI components (e.g. buttons). |

- The following selector matches all elements of pseudo-class `:link`, independent of element type.

```
:link
```

# Combining Selectors

- Selectors are combined by writing them in sequence, separated by a combinator.
- The descendant combinator, whitespace, describes an element that is nested inside another element. It is independent of nesting level. The CSS selector

```
div input
```

(note the whitespace between the tags) applies to the **input** element in the HTML code below.

```
<div ... >
   <form ... >
      <input ... />
      ...
   </form>
</div>
```

# Combining Selectors (Cont'd)

- The child combinator, **>**, describes an element that is nested directly inside another element. The CSS selector

```
div>input
```

does not apply to the **input** element in the HTML below, but the following selector does:

```
form>input
```

```
<div ... >
   <form ... >
      <input ... />
      ...
   </form>
</div>
```

# Combining Selectors (Cont'd)

- Multiple selectors can be combined.
- For example, **`div form>input#username`** means an **`input`** element with id **`username`** that is a child of a **`form`** which is a descendant of a **`div`**.

# Selector Wildcard

- The element name in all types of selectors can be replaced by the wildcard character, **`*`**, which matches all elements.
- **`div>*>input`** means an **`input`** element that is a child of any element which is a child of a **`div`**.

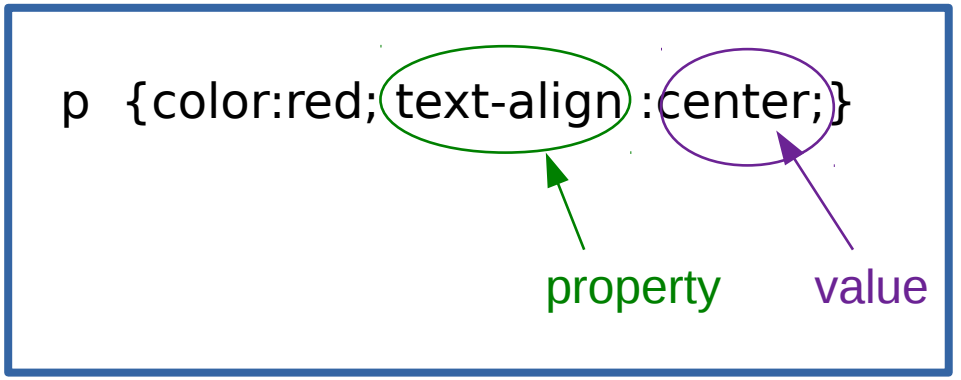
# More Selectors

- There are many more selctors.
- A listing can be found at `http://www.w3.org/TR/selectors/#selectors`

# Question 1

# Properties

- CSS2.1 *includes **60** different properties in **7** categories*:
  - Fonts
  - Lists
  - Alignment of text
  - Margins
  - Colors
  - Backgrounds
  - Borders




- Full list here:
  **`http://www.w3.org/TR/CSS21/propidx.html`**
- CSS3 adds many more, for example animations, columns and graphic transforms.
- Property names are not case sensitive.

# Property Values

- A property value can be a keyword, length, URL or color.
  - Keywords, for example **`left`**, **`small`**, **`green`**, are not case sensitive.
  - A length is a number, maybe with decimal points, plus a unit.
    - Units:
      **`%`** - percentage
      **`px`** - pixels
      **`in`** - inches
      **`cm`** - centimeters
      **`mm`** - millimeters
      **`pt`** – points (1pt = 1/72 inch)
      **`pc`** - picas (12 points)
      **`em`** - 1em is equal to the current font size. E.g., if an element is displayed with a font of 12 pt, then 2em is 24 pt. Very useful since it adapts to font size.
      **`ex`** - is the x-height of a font (x-height is usually about half the font-size)

# Property Values (Cont'd)

- A URL has the form:
  **`url(protocol://server/pathname)`**

- There are many ways to specify a color, for example a color name or a rgb value.
- Standard colors, guaranteed to be displayable by all graphical browsers on all color monitors, are aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, orange, purple, red, silver, teal, white, and yellow.

- A rgb value is a hex, **`#XXXXXX`**, or numeric, **`rgb(n1, n2, n3)`**, where n1, n2, n3 are decimal or percentages.

# Font Properties, Font Family

- The property name is **`font-family`**
- Value is a list of font names. Start with the preferred font, add fallback fonts and end with a generic font name.
  - Browser uses the first in the list it has.
  - Example:

    ```
    font-family: "Times New Roman", Times, serif;
    ```
- Commonly used font combinations can be found at **`http://www.w3schools.com/cssref/css_websafe_fonts.asp`**
- Generic fonts: **`serif`**, **`sans-serif`**, **`cursive`**, **`fantasy`**, and **`monospace`**
- If a font name has more than one word, it should be quoted, e.g., **`"Times New Roman"`**.

# Font Properties, Size

- The property name is **`font-size`**
  - Possible values: a length or a name, such as **`smaller`**, **`xx-large`**, etc.
  - Percentages and **`em`** are the best length units since they are relative.
  - Default text size in browsers is 16px, therefore, default size of 1em is 16px.

# Font Properties, Style

- The property name is **`font-style`**
- Some possible values are **`italic`**, **`normal`**

# Font Properties, Weight

- The property name is **`font-weight`**, specifies degree of boldness.
- Some possible values are **`bolder`**, **`lighter`**, **`bold`**, **`normal`**

# Font Properties, Shorthand

- The property **`font`** can be used as a shorthand for specifying a list of font properties:

```
font: bolder 14pt Arial Helvetica
```

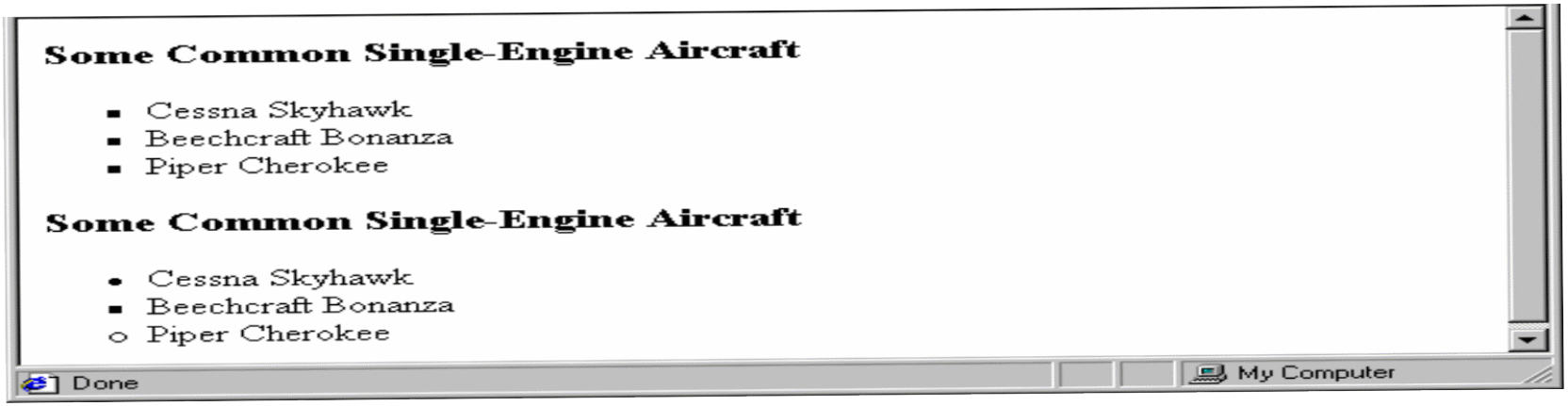
- Order must be: style, weight, size, name(s)

# List properties, Bullet Type

- Property name is **`list-style-type`**

- *Unordered lists*
  - Bullet can be a disc (default), a square, or a circle

```
list-style-type: disc
list-style-type: square
list-style-type: circle
```

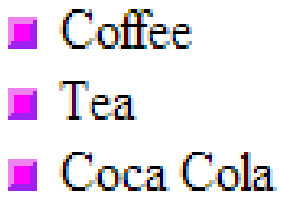


# List properties, Bullet Type (Cont'd)

- Could use an image for the bullets in an unordered list

- Example:

```
list-style-image:url('sqpurple.gif');
```



# List Properties, Numbering

- *On ordered lists,* **`list-style-type`** can be used to change the sequence values.

| Property value | Sequence type | First four |
|---|---|---|
| `decimal` | Arabic numerals | 1, 2, 3, 4 |
| `Upper-alpha` | Uc letters | A, B, C, D |
| `lower-alpha` | Lc letters | a, b, c, d |
| `upper-roman` | Uc Roman | I, II, III, IV |
| `lower-roman` | Lc Roman | i, ii, iii, iv |

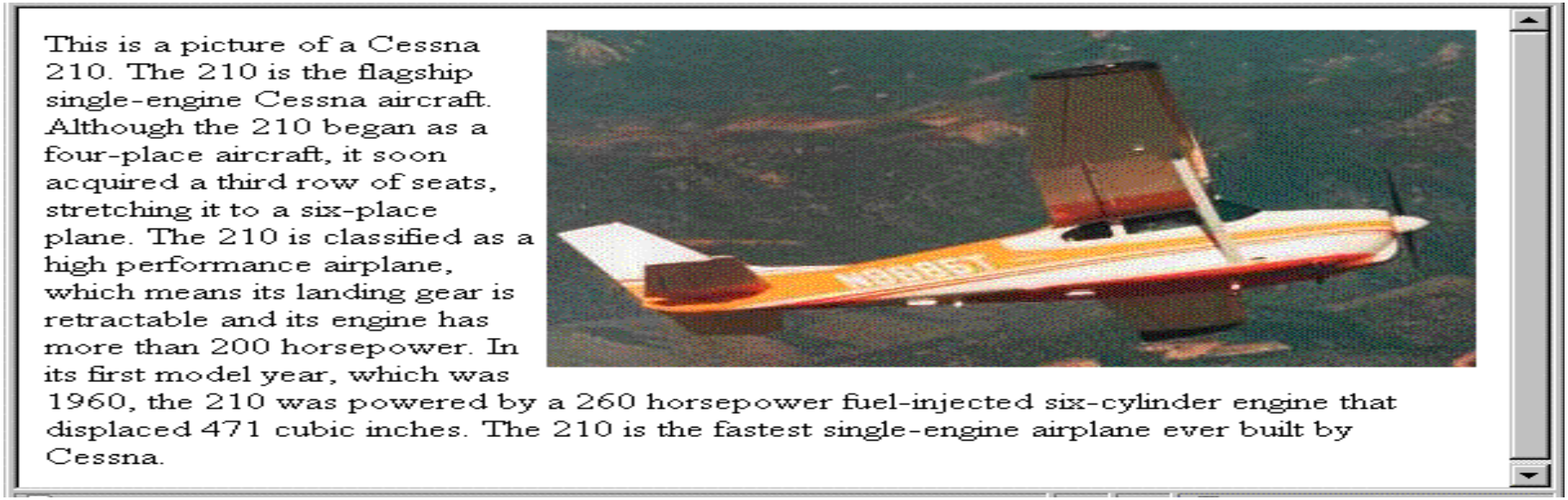
# Text Alignment

- The **`text-indent`** property controls indentation
  - Takes either a length or a % value
  - The **`text-align`** property controls alignment and has the possible values, **`left`** (the default), **`center`**, **`right`**, or **`justify`**

- Sometimes we want text to flow around another element, use the **`float`** property for that.
- The **`float`** property has the possible values, **`left`**, **`right`**, and **`none`** (the default)
- If we have an element we want on the right, with text flowing on its left, we use the default **`text-align`** value (**`left`**) for the text and the **`right`** value for **`float`** on the element we want on the right

# Text Alignment (Cont'd)

- The image has the property **`float: right`**
- The text hast the property **`text-align: left`**



# Color Properties

- The **`color`** property specifies the foreground color.

```
th.red {color: red}
th.orange {color: orange}
```

```
<table>
  <tr>
    <th class = "red"> Apple </th>
    <th class = "orange"> Orange </th>
  </tr>
</table>
```

- The **`background-color`** property specifies the background color.

# Background Images

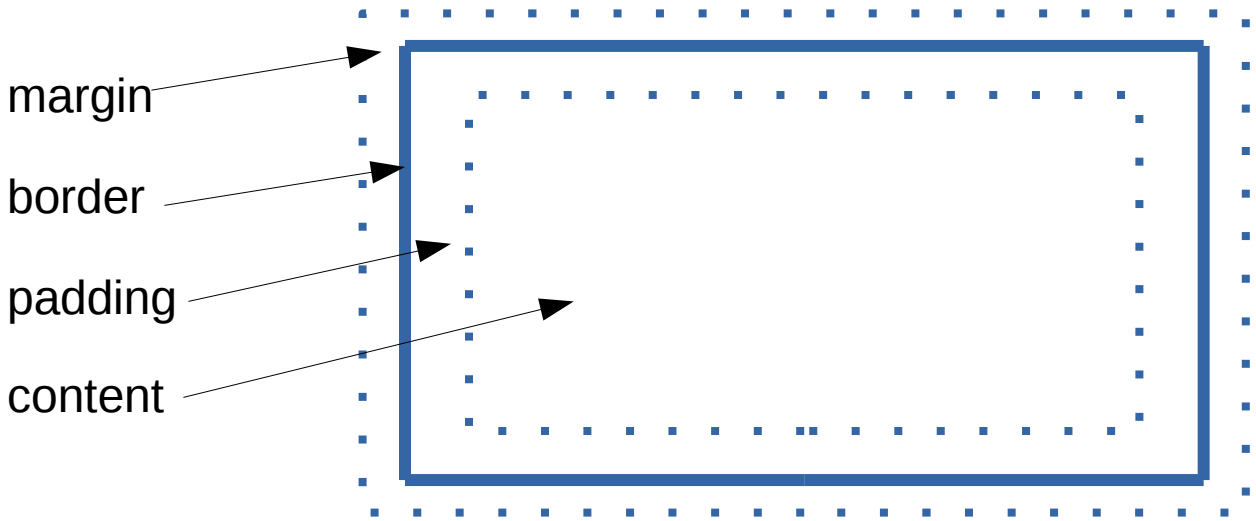
- Specified with the **`background-image`** property.
- Repetition is controlled with the **`background-repeat`** property, which has the values **`repeat`** (default), **`no-repeat`**, **`repeat-x`**, or **`repeat-y`**
- Positioning is controlled with the **`background-position`** property, which hast the values **`top`**, **`center`**, **`bottom`**, **`left`**, or **`right`**

# The Box Model

- All HTML elements are considered as boxes by CSS.
- The box model allows placing a border around elements and space between elements.

# The Box Model (Cont'd)

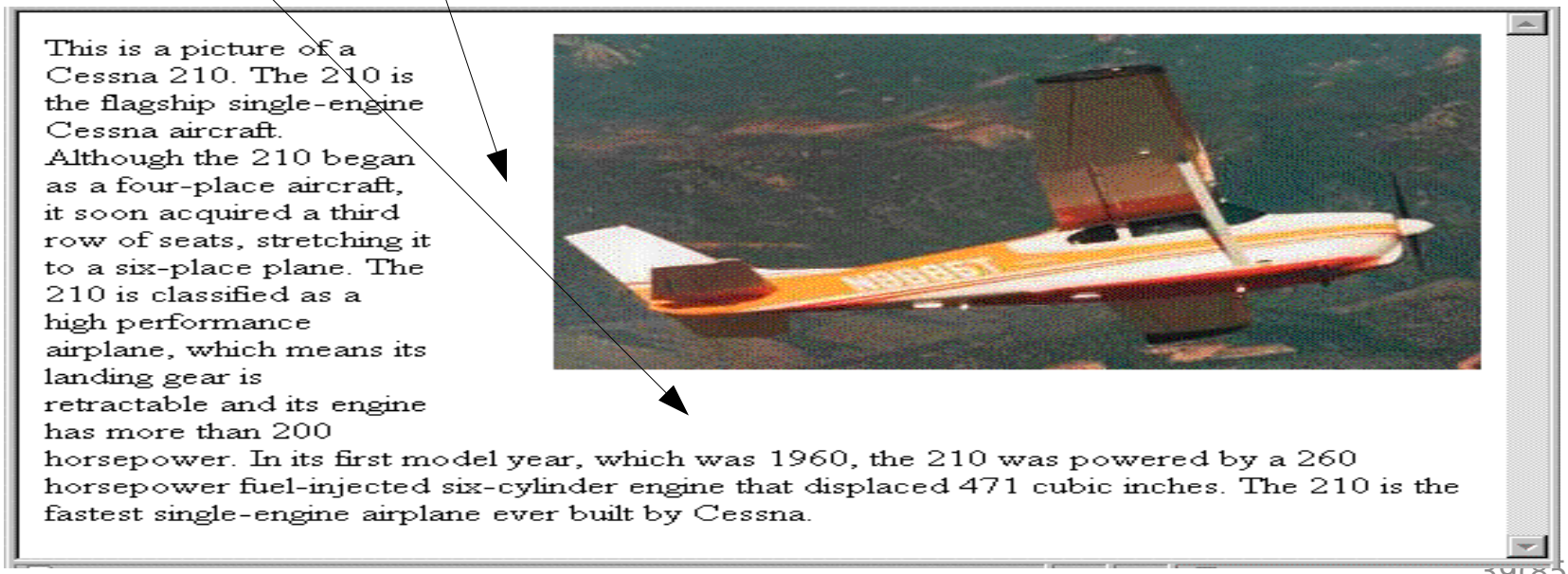
- The margin clears an area around the border. It is completely transparent.
- The border goes around the padding and content. The border is affected by the background color of the box.
- The padding clears an area around the content. It is affected by the background color of the box.
- The content is the content of the box, e.g., text, images, other boxes.



# Box Model, Margin

- The margin is the space between the border of an element and its neighbor element
- The margins around an element can be set with `margin-left`, etc.
- The image below has the following properties.

```
float: right;
margin-left: 0.35in;
margin-bottom: 0.35in"
```



# Box Model, Border

- Border is specified with the **`border-style`** property, which controls whether the element has a border and, if so, the style of the border
- Some possible values are **`none`**, **`solid`**, **`dotted`**, **`dashed`**, and **`double`**
- The border width is specified with the **`border-width`** property, which can have the values **`thin`**, **`medium`** (default), **`thick`**, or a length value in pixels.
  - Can be specified for any of the four borders, e.g., **`border-top-width`**.
- Border color is specified with the **`border-color`** property.
  - Can be specified for any of the four borders, e.g., **`border-top-color`**.
- As for many otherproperties, there is a shorthand:
  **`border: thin solid black`**

# Box Model, Padding

- The padding is the distance between the content of an element and its border
- Controlled by **`padding`**, **`padding-left`**, etc.

# Box Model, Content

- The height and width properties of an element specifies the size of the content.
- Margin, border and padding is outside the specified size.
- Height and width is set with the properties **`height`**, **`max-height`**, **`min-height`** and corresponding for width.

# Question 2

# Display and Visibility

- The **`display`** property specifies if/how an element is displayed. The rule **`display:none`** hides the element, and it does not take up any space.
- **`display`** can also control how and where the element is rendered.
- The **`visibility`** property specifies if an element is visible or hidden. The rule **`visibility:hidden`** hides the element, but it still takes up the same space as when visible.
- **`visibility`** only decides if the element is visible or not.

# HTML Repetition: Block and Inline Elements

- A block element takes up the full width available, and has a line break before and after it.
  Examples: **`<h1>`**, **`<p>`**, **`<ul>`**, **`<table>`**
- Page layout is managed by positioning block elements using CSS.
- A **`<div>`** is a block element that has no other purpose than to define a block. It is often used for layout.
- The 'opposite' of block element is inline element, which does not force line breaks and occupies only the necessary width.
  Examples: **`<td>`**, **`<a>`**, **`<img>`**
- A **`<span>`** is an inline element that has no other purpose than to define the element. It is also often used for layout.

# Block and Inline Elements (Cont'd)

- To get the decired layout, an element can be changed between block and inline.

```
li {
    display: inline;
}

span {
    display: block;
}
```

# Positioning Elements

- Positioning is quite tricky in CSS.
- Elements are positioned using the **`top`**, **`bottom`**, **`left`**, and **`right`** properties.
- How these four properties work depend on the value of the **`position`** property.
- There are four different positioning methods, which are values of the **`position`** property: **`static`**, **`fixed`**, **`relative`**, and **`absolute`**.

# Static Positioning

- Static is the default positioning method.
- **`top`**, **`bottom`**, **`left`**, and **`right`** properties have no effect.
- Elements are positioned according to normal page flow, which has different meaning for block and inline elements.
- Block elements are positioned top to bottom, never side by side. Remember that they are by definition surrounded by line breaks. They always occupy the entire page width.
- Inline elements are positioned left-to-right. All inline boxes on the same line are enclosed in a line box. Anonymous inline boxes are generated for text. Applying margins, paddings, etc to inline boxes is tricky.
- Detailed information on static positioning is found at **`http://www.w3.org/wiki/CSS_static_and_relative_positioning`**

# Fixed Positioning

- An element with fixed position is anchored relative to the browser window.
- The position is decided with **`top`**, **`bottom`**, **`left`**, and **`right`** properties.
- Fixed elements do not move when the window is scrolled.
- Fixed elements may overlap other elements.

# Relative Positioning

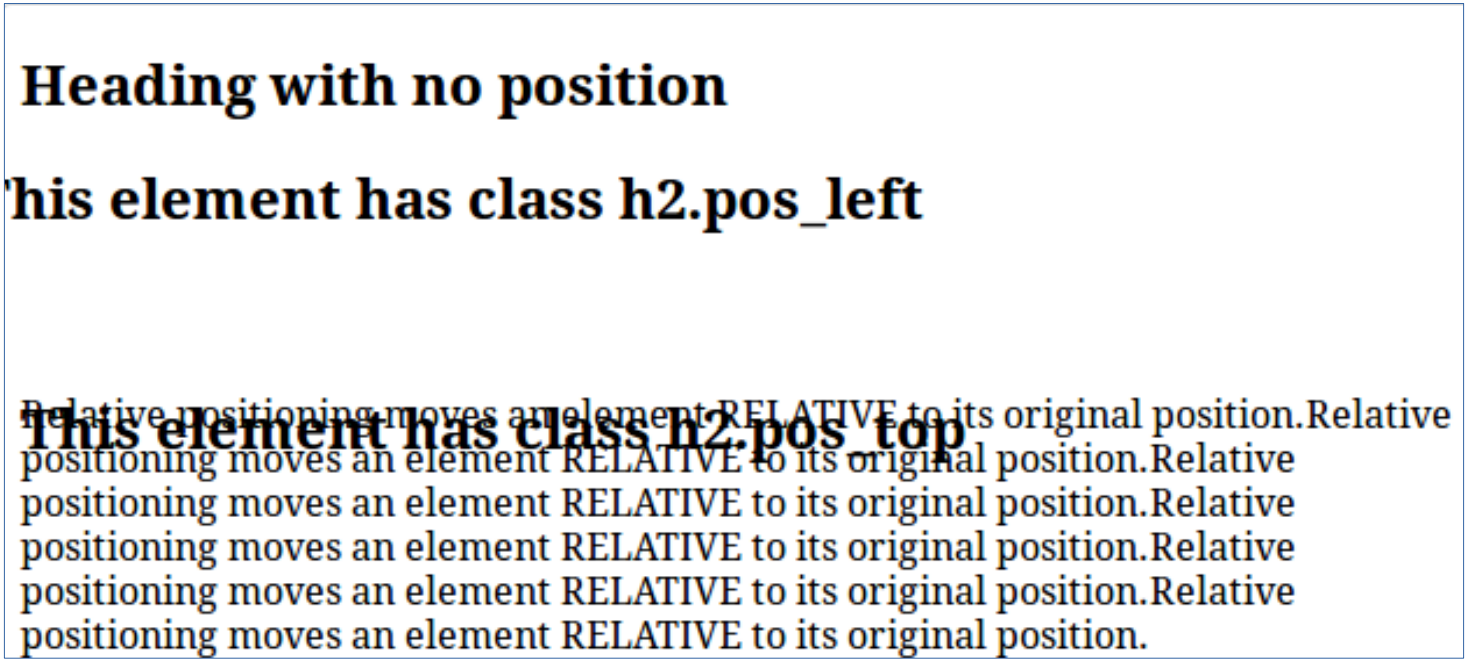
- An element with relative position is first laid out like a static element. Then the generated box is shifted according to the **`top`**, **`bottom`**, **`left`** and **`right`** properties.
- It is only the generated box that is shifted. The element still remains where it was in the static document flow. That's where it takes up space and affects positioning of other elements.
- Relative elements may overlap other elements.
- Detailed information on relative positioning is found at **`http://www.w3.org/wiki/CSS_static_and_relative_positioning`**

# Relative Positioning Example

```
h2.pos_left {
    position: relative;
    left: -20px;
}

h2.pos_top {
    position: relative;
    top: 50px;
}
```

```
<h2>Heading with no position</h2>
<h2 class="pos_left">
    This element has class
    h2.pos_left</h2>
<h2 class="pos_top">
    This element has class
    h2.pos_top</h2>
<p>Relative positioning moves an
element RELATIVE to its original
position...</p>
```



# Absolute Positioning

- An element with **`absolute`** position is positioned relative to the first parent element that has a position other than static. If no such element is found, the containing block is **`<html>`**.
- Absolute elements are removed from the normal flow. The document and other elements behave like the absolute element does not exist.
- Absolute elements may overlap other elements.

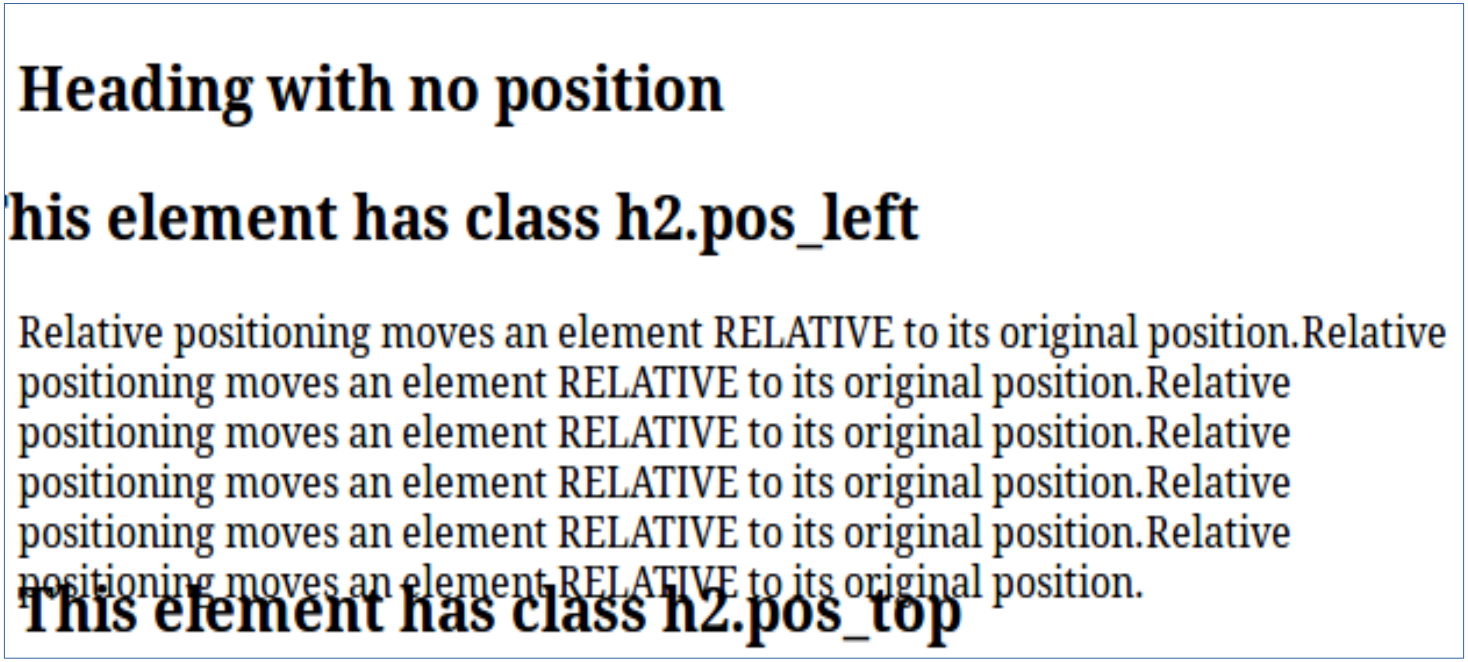
# Absolute Positioning Example

```
.container {
    position: relative;
}

h2.pos_left {
    position: relative;
    left: -20px;
}

h2.pos_top {
    position: absolute;
    top: 80px;
}
```

```
<h2>Heading with no position</h2>
<h2 class="pos_left">
 This element has class
 h2.pos_left</h2>
<div class="container">
 <h2 class="pos_top">
  This element has class
  h2.pos_top</h2>
 <p>Relative positioning moves an
  element RELATIVE to ...</p>
</div>
```



# Overlapping Elements

- The **`z-index`** property specifies the stack order of overlapping elements.
- The value is a positive or negative number:

```
z-index: -1;
```

- An element with greater stack order is always in front of an element with lower stack order.
- If **`z-order`** is not specified, the element positioned last in the HTML code is on top.

# Floating

- A floating element is pushed to the left or right. Elements can not float up or down.
- Elements before the floating element are not affected. Elements after the floating element flows around it.
- Floating elements after each other float next to each other if there is enough space horizontally.
- This means we can have block level elements beside each other!
- Floating is specified with the **`float`** property. Possible values are **`left`**, **`right`** and **`none`** (default).

# Turning off Float

- The **`clear`** property specifies sides of an element where floating elements are not allowed.
- Possible values are **`left`**, **`right`**, **`both`**, **`none`**.

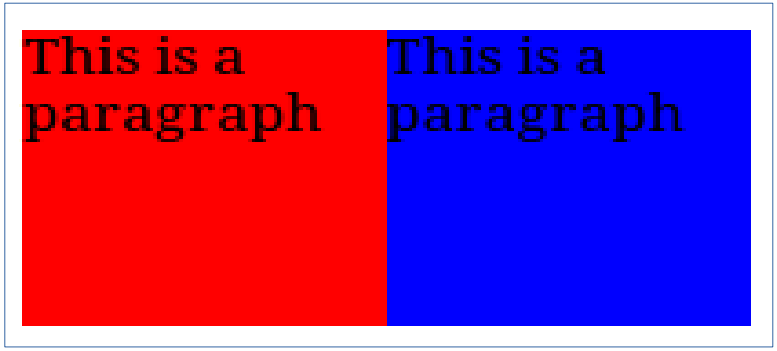
# Float Example

```
p{
  width: 100px;
  height: 100px;
  float: left;
}

p#red{
  background-color: red;
}

p#blue{
  background-color: blue;
}
```

```
<p id="red">
   This is a paragraph</p>
<p id="blue">
   This is a paragraph</p>
```



# Clear Example

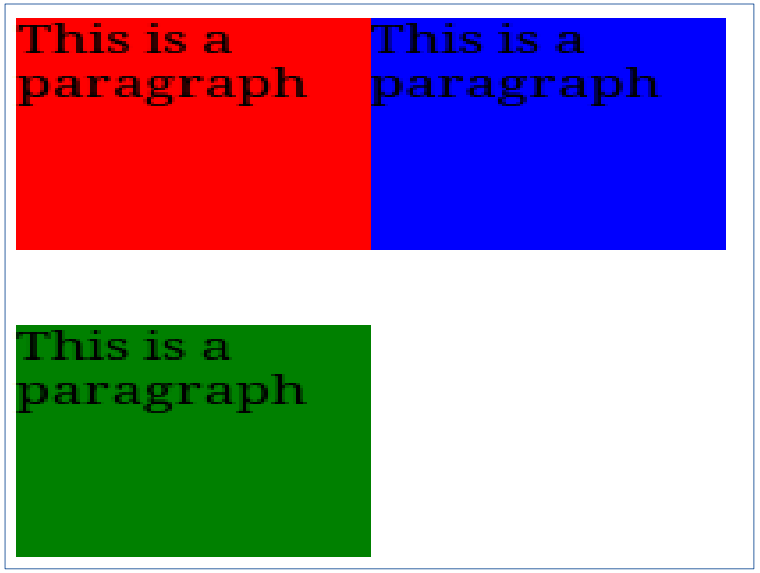
```
p{
  width: 100px;
  height: 100px;
  float: left;
}

p#red{
  background-color: red;
}

p#blue{
  background-color: blue;
}

p#green{
  background-color: green;
  clear: left;
}
```

```
<p id="red">
   This is a paragraph</p>
<p id="blue">
   This is a paragraph</p>
<p id="green">
   This is a paragraph</p>
```



# Aligning Block Elements

- Now we have seen all properties used for layout.
- Since they are quite low level, a lot of experimenting might be needed to achieve an acceptable result.
- An often faced problem is how to align block elements. Some possible solutions to this are found at **`http://www.w3schools.com/css/css_align.asp`**

# Question 3

# Conflicting Property Values

- A conflict occurs when two or more values are specified for the same property on the same element.
- Possible sources of conflict:
  - Between different style sheets.
  - Within the same style sheet.
  - Between inline styles and style sheets.
  - Between document author style sheets, user style sheets, and browser style sheets.

# Assigning Property Values

- This describes the main steps of CSS2.1. They are equal in CSS3, but CSS3 adds more steps.
- The steps to assign a property value are:
  1. Run the cascade process.
  2. If the cascade results in a specified value, use it and transform it into a computed value (e.g., transform `em` and `%` to pixels).
  3. If the cascade does not result in a value, try to inherit the parents computed value.
  4. If no value could be inherited, use the property's initial value.
  5. Transform the computed value into an actual value, e.g., the computed font size is 13pt, but the user agent does not have a 13pt font, so 12pt is used instead.

# Step 1: The Cascade

1. Find all declarations for the specific property for the specific element (e.g., the **`color`** property for the **`p`** element with **`id = msg`**).
2. Sort according to importance and orign.
   - *Importance* means whether a property is marked **`!important`**, e.g, **`p#msg {color:red !important}`**. Note that it is considered bad practice to use **`!important`** declarations.
   - *Orign* is if the property is declared by the author of the web page, the user who browses it, or the browser's default style sheet.

# Step 1: The Cascade, Cont'd

2. (Cont'd) The order of precedence is as follows (lower number wins).
    1. User important declarations
    2. Author important declarations
    3. Author normal declarations
    4. User normal declarations
    5. Browser declarations

# Step 1: The Cascade, Cont'd

3. If there are conflicting rules with the same importance and orign, sort them by specificity. Specificity is a four digit number, the highest number wins.

- Digit 1: Has the value 1 when the property value is specified in a HTML style attribute, and the value 0 when it is specified in a CSS rule.
- Digit 2: The number of id attributes in the selector (e.g., 1 for the selector `p#msg`).
- Digit 3: The number of other attributes in the selector (e.g., 1 for `p.large`)
- Digit 4: The number of element names in the selector (e.g., 2 for `div p#msg`)

# Step 1: The Cascade, Cont'd

4. If there are still conflicts, sort the rules in the order they are specified. The latter specified wins.

# Step 2: Calculate the Computed Value

- If the cascade produces a specified value for the specific property, it is transformed to a computed value.
- This means, for example, that URIs are made absolute and **`em`** and **`ex`** units are computed to pixels or absolute lengths.

# Step 3: Inherit the Computed Value

- If the cascade does not produce a value for the specific property, the value is inherited from the parent element.
- For example, suppose there is a **p** element with an emphasizing element **em** inside:

```
<p>This <em>is</em>the message!</p>
```

If no color has been assigned to the **em** element, the emphasized *is* will inherit the color of the parent element, so if **p** has the color blue, the **em** element will likewise be in blue.

- The computed value is also inherited if the cascade produces a computed value, and that value is **inherit**.

# Step 4: Use the Initial Value

- If neither cascade, nor inheritance produces a computed value, the property's initial value becomes its computed value.
- The initial value of each property is defined in the property's definition.
- In CSS3, the initial value is also used if the cascade produces a computed value, and that value is **`initial`**.

# Step 5: Calculate the Actual Value

- The computed value is ideally used for rendering, but maybe a browser is not able to make use of the computed value.
- For example, a browser may only be able to render borders with integer pixel widths and may therefore have to approximate the computed width.
- The actual value is the computed value after approximations have been applied.

# Crossbrowser Compatibility Issues

- Some browsers, in particular old IE versions, may have unexpected behaviour.
- Always test the web site in all browsers you accept clients to use, and be prepared to spend a lot of time for this.
- It is never acceptable to test a web site in only one browser.
- Remember to add a `!DOCTYPE` declaration to all HTML documents.

# Always Use A Reset Style Sheet

- Different browsers have different default style sheets, therefore they will display the same page differently.
- This makes it difficult to create a design that looks good in all browsers.
- The solution is to always use a reset style sheet, that removes the browser's default values.
- Many reset sheets can be found on the web. There is one example on the course web site.

# Vendor-specific Properties

- Browser makers implement extensions to the CSS specifications.
- This is often done for CSS properties that are not yet released as W3C specifications.
  - Might also be used for experiments.
- A vendor-specific property name starts with **-** or **_**, followed by the vendor identifier.
  - Examples are **-moz** for firefox, **-o** for opera and **-webkit** for safari and chrome.
- Avoid using them unless it's absolutely necessary.
- Will not pass CSS validation.

# Responsive Design

- A responsive web page adapts to the screen resolution on which it is displayed.
- To adapt to the current resolution, the web page might change font size, size of images, number of columns, remove headers, change menu layout, etc.

# Responsive Design (Cont'd)

- Responsive design is a very good practice that we should always strive for.
- It is very annoying for the user if the page has a fixed layout.
  - On big screens it will only occupy a small part of the available space.
  - On small screens the user will be forced to scroll back and forth across the page.

# Media Queries

- The enabling technique for responsive design is CSS3 media queries:

```
@media only screen and (max-width: 480px) {
    /*
        CSS rules that are used for screens
        narrower than 480 pixels.
    */
}
```

- The media query starts with **`@media`**.
- Next is the media type for which the rules shall apply. **`only screen`** means the rules will not apply to printouts.
- Then comes the media features for which the rules shall apply. In this example, the rules will apply to screens narrower than 480 px.

# Use Columns

- There are many approaches to responsive design and many ways to use media queries.
- The chat and cache simulator examples use the method described at
  `http://www.responsivegridsystem.com/`
- The idea is to
  1. Split the page in a number of columns.
  2. Distribute all elements into the columns.
  3. Depending on screen resolution, decide how many columns to use.
     - When more columns are used, more elements will go beside each other.
     - When fewer columns are used, elements will stack on top of each other instead of beside.

# Be Relative

- Other tips are:
  - Avoid fixed positioning.
  - Use relative units, such as **`em`** and percentage.
  - Consider removing elements, for example headers, at low resolution.
  - Replace navigation toolbars with pop-up menus.

# Question 4

# Accessibility

- Many users may be operating in contexts very different from your own:
  - They may not be able to see, hear or move.
  - They may have difficulty reading or comprehending text.
  - They may not be able to use a keyboard or mouse.
  - They may have a text-only screen, a small screen, or a slow Internet connection.
  - They may not speak or understand fluently the language in which the document is written.
  - They may be in a situation where their eyes, ears, or hands are busy or interfered with (e.g., driving to work, working in a loud environment, etc.).

# Accessibility Guidelines

- *Guideline 1, Use text alternatives.*
  Provide text alternatives to auditory and visual content.
  - Some people cannot use images, movies, etc., but may still use pages that include equivalent information.
- Provide a text equivalent for every non-text element, use for example the `alt` attribute of the `img` element to provide a text alternative for an image.
  - A text equivalent for an image of an upward arrow that links to a table of contents could be *Go to table of contents*.

# Accessibility Guidelines, Cont'd

- *Guideline 2, Don't rely on color alone.* Ensure that text and graphics are understandable when viewed without color.
  - If color is used to convey information, people who cannot differentiate between certain colors and users with devices that have non-color or non-visual displays will not receive the information.
- Ensure that all information conveyed with color is also available without color, for example from context or markup.

# Accessibility Guidelines, Cont'd

- *Guideline 3, Use HTML and CSS properly.*
  Mark up documents with the proper structural elements. Control presentation with style sheets rather than with presentation elements and attributes.
  - Misusing markup for a presentation effect (e.g., using a table for layout or a header to change the font size) makes it difficult for users with specialized software to understand the organization of the page or to navigate through it.
  - Using presentation markup rather than structural markup to convey structure (e.g., constructing what looks like a table of data with an HTML `pre` element) makes it difficult to render a page intelligibly to other devices.

# Accessibility Guidelines, Cont'd

- *Guideline 4, Provide clear navigation mechanisms.* Clear and consistent navigation mechanisms: navigation bars, a site map, etc., increases the likelihood that a person will find what they are looking for at a site.
  - Clear and consistent navigation mechanisms are important to people with cognitive disabilities or blindness, and benefit all users.
- Clearly identify the target of each link. Link text should be meaningful enough to make sense when read out of context, either on its own or as part of a sequence of links.
  - For example, write "Information about version 4.3" instead of "click here".

# Accessibility Guidelines, Cont'd

- The complete set of accessibility guidelines can be found in the W3C recommendation *Web Content Accessibility Guidelines 1.0*, available at **`http://www.w3.org/TR/WCAG10/`**