Cascading Style Sheets

- CSS: Cascading Style Sheet
- CSS 1: W3C recommendation (17 Dec 1996)
- CSS 2.1: W3C Candidate Recommendation (19 July 2007)
- CSS 3: Working Draft
- Resources:
  - CSS2.1 standard, [http://www.w3.org/TR/CSS21/](http://www.w3.org/TR/CSS21/)
  - W3C CSS Tutorial, [http://www.w3.org/Style/Examples/011/firstcss](http://www.w3.org/Style/Examples/011/firstcss)
CSS Syntax

- CSS is based on rules
- A rule is a statement about one stylistic aspect of one or more XHTML element
- A style sheet is a set of one or more rules that apply to an XHTML document

```
Selector    Declaration
h1           {color:blue; font-size:12px;}
```

Cascading Style Sheets

- The term “cascading” means that a document can include more than one style sheet
- In this case, visualization follows priority rules
  - User Style
  - Inline Style (inside HTML tag)
  - Internal Style (usually on the HTML head section)
  - External Style
  - Browser Default Style
External style

- Link to an external style sheet using the `<link>` element

```html
<head>
  <link rel="stylesheet" type="text/css" href="stile.css">
</head>
<body>
<h1>Questo testo è di colore verde, e utilizza il font "verdana" a 17 pixel</h1>
<h2>Questo testo è di colore rosso, e utilizza il font "arial" a 18 pixel</h2>
</body>
```

---

External style

- Alternative method
- `@import` directive in the `<style>` element

```html
<head>
  <style>
    @import url(stile.css);
  </style>
</head>
<body>
...  
</body>
```
**Internal style**

- `<style>` element inside the document header

```html
<head>
<style type="text/css">
  h1 { font-size:17px; font-family:verdana; color:green; }
  h2 { font-size:18px; font-family:arial; color:red; }
</style>
</head>
```

**Inline style**

- `<style>` attribute within an XHTML element

```html
<h1 style="font-size:17px; font-family:verdana; color:green;">Questo testo e' di colore verde, e utilizza il font verdana a 17 pixel</h1>
```
Priority rules

- Rules can be marked as “important”

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

Tree structure and inheritance

- XHTML documents are trees
- Styles are inherited along trees

- When two rules are in conflict the most specific wins
- Example
  ```
  body {color: green}
  hl {color: red}
  ```
**Pseudo class selector**

- Used to style an element based on something other than the structure of the document
  - E.g., the state of a form element or link

```css
/* makes all unvisited links blue */
a:link {color:blue;}
/* makes all visited links green */
a:visited {color:green;}
/* makes links red when hovered or activated */
a:hover, a:active {color:red;}
/* makes table rows red when hovered over */
tr:hover {background-color: red;}
/* makes input elements yellow when focus is applied */
input:focus {background-color: yellow;}
```
Meaningful XHTML

- Meaningful elements
  - h1, h2, ...
  - ul, ol, and dl
  - strong and em
  - blockquote and cite
  - abbr, acronym, and code
  - fieldset, legend, and label
  - caption, thead, tbody, and tfoot
- Id and class names
  - Allow to give extra meaning
- Div and span
  - Add structure to document

Div element

- Stands for “division”
- Used to group block-level elements
  - Provides a way of dividing a document into meaningful areas
- Use only if necessary and not redundant

```html
<div id="mainNav">
  <ul>
    <li>Home</li>
    <li>About Us</li>
    <li>Contact</li>
  </ul>
</div>
```

```html
<ul id="mainNav">
  <li>Home</li>
  <li>About Us</li>
  <li>Contact</li>
</ul>
```
Span element

- Used to group or identify inline elements

```html
<h2>Where’s Durstan?</h2>
<p>Published on <span class="date">March 22nd, 2005</span> by <span class="author">Andy Budd</span></p>
```

The box model

- One of the cornerstones of CSS
- Dictates how elements are displayed and, to a certain extent, how they interact with each other
- Every element on the page is considered to be a rectangular box
The box model

- Content
  - The content of the box, where text and images appear
- Padding
  - Clears an area around the content
  - The padding is affected by the background color of the box
- Border
  - A border that goes around the padding and content
  - The border is affected by the background color of the box
- Margin
  - Clears an area around the border
  - The margin does not have a background color, it is completely transparent

Example

- Padding, borders, and margins are optional and default to zero

```css
#myBox {
    margin: 10px;
    padding: 5px;
    width: 70px;
}
```
The box model

- Total element width = width + left padding + right padding + left border + right border + left margin + right margin
- Total element height = height + top padding + bottom padding + top border + bottom border + top margin + bottom margin
- Example
  - W3Schools.com
  - http://www.w3schools.com/Css/css_boxmodel.asp

Positioning schemes

- Three basic positioning schemes in CSS
  - Normal flow
  - Floats
  - Absolute positioning
- Unless specified, all boxes start life being positioned in the normal flow
  - The position of an element’s box in the normal flow will be dictated by that element’s position in the (X)HTML
**Normal flow**

- Block-level boxes will appear vertically one after the other
  - The vertical distance between boxes is calculated by the boxes' vertical margins
- Inline boxes are laid out in a line horizontally
  - Their horizontal spacing can be adjusted using horizontal padding, borders, and margins
  - Vertical padding, borders, and margins will have no effect on the height of an inline box

**Display property**

- Allows to control element visualization (block or inline)
- Changing an inline element to a block element, or vice versa, can be useful for making the page look a specific way
- Example
  - W3Schools.com
  - [http://www.w3schools.com/Css/css_display_visibility.asp](http://www.w3schools.com/Css/css_display_visibility.asp)

```css
li {display:inline;}
span {display:block;}
```
Box Positioning

- A block can be positioned in different ways to which correspond different positioning schemes
  - Static: normal block
  - Relative: the offset values are relative to the block position in the normal flow. If a relative block B follows a relative block A, the offset is respect to the position of A without the offset
  - Absolute: the box position is determined by the top, left, right, bottom properties and is relative to the containing block
  - Fixed: the box is fixed with respect to some reference (the viewport as an example)

Relative positioning

- It is possible to shift one element “relative” to its starting point by setting a vertical or horizontal position

```css
#myBox {
  position: relative;
  left: 20px;
  top: 20px;
}
```
Absolute positioning

- Takes the element out of the flow of the document, thus taking up no space
- Other elements in the normal flow of the document will act as though the absolutely positioned element was never there

Fixed positioning

- A subcategory of absolute positioning
  - A fixed element’s containing block is the viewport
- Allows to create elements that always stay at the same position in the window

- Note: in case of overlaps the z-index property specifies the stack order of an element (which element should be placed in front of, or behind, the others)
Fixed positioning

- Can be used to create complex frame-like presentations

```
#header { position: fixed; width: 100%;
  height: 15%; top: 0; right: 0;
  bottom: auto; left: 0; }
#sidebar { position: fixed; width: 10em;
  height: auto; top: 15%; right: auto;
  bottom: 100px; left: 0; }
#main { position: fixed; width: auto;
  height: auto; top: 15%; right: 0;
  bottom: 100px; left: 10em; }
#footer { position: fixed; width: 100%;
  height: 100px; top: auto; right: 0;
  bottom: 0; left: 0; }
```

Examples

- W3Schools.com
  - [http://www.w3schools.com/Css/css_positioning.asp](http://www.w3schools.com/Css/css_positioning.asp)

  The main problem people have with positioning is remembering which type of positioning is which. Relative positioning is “relative” to the element’s initial position in the flow of the document, whereas absolute positioning is “relative” to nearest positioned ancestor or, if one doesn’t exist, the initial container block.

Floating

- A floated box can either be shifted to the left or the right until its outer edge touches the edge of its containing box, or another floated box
- Often used for images and when working with layouts
  - Example
  - http://www.w3schools.com/Css/css_float.asp

```css
img {
    float: right;
}
```

Floating

- Floated boxes aren’t in the normal flow of the document, so block boxes in the regular flow of the document behave as if the floated box wasn’t there
Floating

- If all three boxes are floated left
  - Box 1 is shifted left until it touches its containing box
  - Other two boxes are shifted left until they touch the preceding floated box

Examples/chapter03/floating-boxes.htm

Floating

- If the containing block is too narrow for all of the floated elements to fit horizontally
  - The remaining floats will drop down until there is sufficient space
  - If the floated elements have different heights, it is possible for floats to get “stuck” on other

Not enough horizontal space

Box 1
Box 2
Box 3 drops

Different height boxes

Box 1
Box 2
Box 3 gets “stuck” on Box 1
Line boxes and clearing

- Line boxes next to a floated box are shortened to make room for the floated box, and flow around the float
  - Floats were created to allow text to flow around images

```
No boxes floated
```

```
Image floated left
```

Line boxes shorten to make room for the floated image

---

Line boxes and clearing

- To stop line boxes flowing around the outside of a floated box, you need to apply a clear to that box
  - The clear property can be left, right, both, or none, and indicates which side of the box should not be next to a floated box

```
p { clear: left; }
```

```
Image floated left
```

Line boxes shorten to make room for the floated image

```
Second paragraph cleared
```

Margin added to clear float
Page layout

- Possibility to control page layout without needing to use presentation markup
- CSS layout has a rather undeserved reputation of being difficult
  - Mostly due to a proliferation of different layout techniques available on the Web
- Tasks
  - Horizontally centering a design on a page
  - Creating two- and three-column float-based layouts
  - Creating fixed-width, liquid, and elastic layouts
  - Making columns stretch to the full height of the available space

Centering a design

- Long lines of text can be difficult and unpleasant to read
- Rather than spanning the full width of the screen, centered designs span only a portion of the screen, creating shorter and easier-to-read line lengths
- Two basic methods
  - Use auto margins
  - Use positioning and negative margins
Auto margins

- Define the width of the wrapper div
- Set the horizontal margin to auto

```html
<body>
  <div id="wrapper">
  </div>
</body>
```

- The most common approach
- IE 5.x and IE 6 do not support auto margins

Positioning and negative margins

- Define the width of the wrapper div
- Set the position property of the wrapper to relative
- Set the left property to 50%
- Apply a negative margin to the left side of the wrapper, equal to half the width of the wrapper

```html
#wrapper {
  width: 720px;
  position: relative;
  left: 50%;
}

#wrapper {
  width: 720px;
  position: relative;
  left: 50%;
  margin-left: -360px;
}
```
Float-based layouts

- Simply set the width of the elements you want to position, and then float them left or right
- Two-column floated layout
- Three-column floated layout

Two-column floated layout

- (X)HTML framework
  - Main navigation on the left side of the page
  - Content on the right
- For accessibility reasons the content area is above the navigation in the source
  - The main content is the most important thing on the page and so should come first in the document
  - There is no point forcing screenreader users to trawl through a potentially long list of links before they get to the content
Two-column floated layout

• Create a virtual gutter by floating one element left and one element right

```html
#content {
  width: 520px;
  float: right;
}
#mainNav {
  width: 180px;
  float: left;
}
#footer {
  clear: both;
}
```

Better: add horizontal padding

```html
#mainNav {
  padding-top: 20px;
  padding-bottom: 20px;
}
#mainNav li {
  padding-left: 20px;
  padding-right: 20px;
}
#content h1, #content h2,
#content p {
  padding-right: 20px;
}
```
Two-column floated layout

Three-column floated layout

- (X)HTML framework
  - similar to the two column layout, but two new divs inside the content div

```html
<div id="content">
  <div id="mainContent">
    ...
  </div>
  <div id="secondaryContent">
    ...
  </div>
</div>
```
Three-column floated layout

- Float the main content left and the secondary content right, inside the already floated content div
  - Divides the second content column in two, creating a three-column effect

```css
#mainNav
#content

#mainContent
  float: left

#secondaryContent
  float: right

#footer

#mainContent {  
  width: 320px;  
  float: left;  
}
#secondaryContent {  
  width: 180px;  
  float: right;  
}

#secondaryContent h1, #secondaryContent h2,  
#secondaryContent p {  
  padding-left: 20px;  
  padding-right: 20px;  
}
```

Three-column floated layout

- Better: remove the padding from the content element and apply it to the content of the secondary content
Three-column floated layout

Examples/chapter08/3-col-fixed.htm

Fixed-width, liquid, and elastic layout

- **Fixed-width layout**
  - Column widths defined in pixels
  - Very common as they give the developer more control over layout and positioning

- **Downsides**
  - Do not make good use of the available space: columns are always the same size no matter the window size
  - Usually work well with the browser default text size, but if you increase the text size a couple of steps, sidebars start running out of space and the line lengths get too short to comfortably read
Liquid layouts

- Dimensions are set using percentages instead of pixels
  - Very efficient use of space
- If the design spans the entire width of the browser window, line lengths can become long and difficult to read
  - Solution: make the wrapper span just a percentage, e.g. 85 percent

```
#wrapper {
  width: 85%;
}
```

Liquid layouts

- Set the width of the navigation and content areas as a percentage of the wrapper width
  - 2-percent virtual gutter between the navigation and the wrapper to deal with any rounding errors and width irregularities that may occur

```
#wrapper {
  width: 85%;
}
#mainNav {
  width: 23%;
  float: left;
}
#content {
  width: 75%;
  float: right;
}
```
Liquid layouts

- The widths of the content divs are based on the width of the content element and not the overall wrapper
  - Width of secondary content area = width of the main navigation area?

```
#wrapper {
  width: 85%;
}
#mainNav {
  width: 23%;
  float: left;
}
#content {
  width: 75%;
  float: right;
}
```

- 3 columns liquid layout

```
#mainContent {
  width: 66%;
  float: left;
}
#secondaryContent {
  width: 31%;
  float: right;
}
```
Elastic layouts

- With liquid layouts
  - Line lengths can get uncomfortably long on large resolution monitors
  - Lines can become very short and fragmented in narrow windows or when the text size is increased a couple of steps
- In elastic layouts the width of elements is relative to the font size (ems) instead of the browser width
  - When the font size is increased the whole layout scales
- Allows to keep line lengths to a readable size
  - Particularly useful for people with reduced vision
Elastic layouts

- Trick to simplify design: set the base font size so that 1em roughly equals 10 pixels
  - The default font size on most browsers is 16 pixels
  - Ten pixels are 62.5 percent of 16 pixels
- Set the font size on the body to 62.5%

```
body {
  font-size: 62.5%;
}
```

Elastic layouts

- 1em now equals 10 pixels at the default font size
- Convert the fixed-width layout into an elastic layout by converting all the pixel widths to em widths

```
#wrapper {
  width: 72em;
  margin: 0 auto;
  text-align: left;
}
#mainNav {
  width: 18em;
  float: left;
}
#content {
  width: 52em;
  float: right;
}
#mainContent {
  width: 32em;
  float: left;
}
#secondaryContent {
  width: 18em;
  float: right;
}
```
Three columns elastic layout

Default text size

Three columns elastic layout

Increased text size

Examples/chapter08/3-col-elastic.htm
Elastic-liquid hybrid

- Combines both elastic and liquid techniques
- Works by setting the widths in ems, then setting the maximum widths as percentages

```css
#wrapper {
  width: 72em;
  max-width: 100%;
  margin: 0 auto;
  text-align: left;
}
#mainNav {
  width: 18em;
  max-width: 23%;
  float: left;
}
#content {
  width: 52em;
  max-width: 75%;
  float: right;
}
#mainContent {
  width: 32em;
  max-width: 66%;
  float: left;
}
#secondaryContent {
  width: 18em;
  max-width: 31%;
  float: right;
}
```

Elastic-liquid hybrid

- On browsers that support max-width, this layout will scale relative to the font size but will never get any larger than the width of the window
A few more interesting tasks

- Vertical navigation bars
- Horizontal navigation bars
- Image maps
- Form styles

Lists and navigation bars

- Simple list
  ```html
  <ul>
  <li>Read emails</li>
  <li>Write book</li>
  <li>Go shopping</li>
  <li>Cook dinner</li>
  <li>Watch Scrubs</li>
  </ul>
  ```

- To add custom bullets
  - Use the list-style-image property, but little control over the position of the bullet image
  - Turn list bullets off and add custom bullet as a background image on the list element
Custom bullets

- Remove indentation by zeroing down the margin and padding on the list
- Remove the default bullet setting the list style type to none
- Adding padding to the left side of the list item creates the necessary space for the bullet

\[
\begin{align*}
ul & \{ \\
& \text{margin: 0;} \\
& \text{padding: 0;} \\
& \text{list-style-type: none;} \\
\}
\end{align*}
\]

\[
\begin{align*}
li & \{ \\
& \text{background: url(bullet.gif) no-repeat 0 50%;} \\
& \text{padding-left: 30px;} \\
\}
\end{align*}
\]

Vertical navigation bar

- HTML framework

\[
<ul> \\
<li><a href="home.htm">Home</a></li> \\
<li><a href="about.htm">About</a></li> \\
<li><a href="services.htm">Our Services</a></li> \\
<li><a href="work.htm">Our Work</a></li> \\
<li><a href="news.htm">News</a></li> \\
<li><a href="contact.htm">Contact</a></li> \\
</ul>
\]
### Vertical navigation bar

- Remove the default bullets and zero down the margin and padding

```css
ul {
    margin: 0;
    padding: 0;
    list-style-type: none;
}
```

- HOME
- ABOUT
- OUR SERVICES
- OUR WORK
- NEWS
- CONTACT

### Vertical navigation bar

- Rather than style the list items, style the enclosed anchors
  
  - To create a button-like hit area, set the display property of the anchors to block, and then specify the anchor’s dimensions (e.g. 200 pixels wide and 40 pixels high); the line height is set to 40 pixels in order to center the link text vertically
  
  ```css
  ul a {
      display: block;
      width: 200px;
      height: 40px;
      line-height: 40px;
      color: #000;
      text-decoration: none;
  }
  ```

- The last couple of rules are just stylistic, setting the color of the link text and turning off the underlines
Vertical navigation bar

- Pixy rollover technique: the rollover graphic is applied as a background image to the anchor link

```html
ul a {
  display: block;
  width: 200px;
  height: 40px;
  line-height: 40px;
  color: #000;
  text-decoration: none;
  background: #94B8E9 url(images/pixy-rollover.gif) no-repeat left middle;
  text-indent: 50px;
}
```

A single image composed of both the up and hover state images

---

Vertical navigation bar

- Apply the hover and selected states
  - Simply shift the background image on the anchor links to the right, uncovering the hover state graphic

```html
a:hover, .selected a {
  background-position: right bottom;
  color: #fff;
}
```
**Horizontal navigation bar**

- **HTML framework**

```html
<ul>
  <li><a href="#">Home</a></li>
  <li><a href="#">About</a></li>
  <li><a href="#">News</a></li>
  <li><a href="#">Products</a></li>
  <li><a href="#">Services</a></li>
  <li><a href="#">Clients</a></li>
  <li><a href="#">Case Studies</a></li>
</ul>
```

Examples/chapter06/horizontal-nav.htm

**Horizontal navigation bar**

- Zero down the padding and margins, and remove the default bullets
- Horizontal navigation bar 720 pixels wide, with a repeating orange gradient as a background

```css
ul {
  margin: 0;
  padding: 0;
  list-style: none;
  width: 720px;
  background: #FAA819;
  url(images/mainNavBg.gif) repeat-x;
}
```
Horizontal navigation bar

- Lists are currently displayed vertically
- Two methods to make it display horizontally
  - Set the list items to display inline
  - Float all list items left

Floating method
  - Warning: when an element is floated, it no longer takes up any space in the flow of the document
  - The parent list has no content and collapses down, hiding the list background

Solution: float the parent element as well

The rollover state is simply a change in link color

Examples

http://www.w3schools.com/Css/css_navbar.asp
CSS image maps

- Allow to specify regions of an image to act as hotspots
- Image maps are still valid in HTML
  - But they mix presentation with content
- It is possible to create simple image maps with a combination of lists, anchors, and some CSS code

The goal

Examples/chapter06/imagemap.htm
CSS image maps

- Add the image to the page inside a named div

```html
<div id="pic">
  <img src="images/group-photo.jpg" width="640" height="425" alt="Richard, Andy and Jeremy" />
</div>
```

- Add a list of links to each person’s website
  - Each list item needs to be given a class to identify the person in that list item
  - Give each link a title attribute containing the name of the person: tooltip showing who the person is will be displayed on most browsers when the link is hovered over

```html
<ul>
  <li class="rich">
    <a href="http://www.clagnut.com/" title="Richard Rutter">
      Richard Rutter</a>
  </li>
  <li class="andy">
    <a href="http://www.andybudd.com/" title="Andy Budd">
      Andy Budd</a>
  </li>
  <li class="jeremy">
    <a href="http://www.adactio.com/" title="Jeremy Keith">
      Jeremy Keith</a>
  </li>
</ul>
```
CSS image maps

- Set the width and height of the div so that it matches the dimensions of the image
- Set the position property of the div to relative
  - Important: it allows the enclosed links to be positioned absolutely, in relation to the edges of the div, and hence the image

```css
#pic { width: 640px; height: 425px; position: relative; /* The key to this technique */ }
```

- Remove the list bullets and zero down the list's margin and padding

```css
#pic ul { margin: 0; padding: 0; list-style: none; }
```

CSS image maps

- Style the links
  - Set their widths and heights to create the desired hit area
  - Links can then be positioned absolutely over the correct people, forming the hotspots
- The link text is still displayed
  - It is necessary to hide it off the screen by using a large, negative text indent

```css
#pic a { position: absolute; width: 100px; height: 120px; text-indent: -1000em; }
```
CSS image maps

• The individual links can be positioned over the relevant people

```css
#pic .rich a {
  top: 15px;
  left: 95px;
}
#pic .andy a {
  top: 115px;
  left: 280px;
}
#pic .jeremy a {
  top: 250px;
  left: 425px;
}
```

• Finally, to create the rollover effect, a solid white border is applied to the links when they are hovered over

```css
#pic a:hover {
  border: 1px solid #fff;
}
```
Simple form layout

- Short and relatively simple forms are easiest to fill in when the form labels appear vertically above their associated form elements.

Useful HTML form elements

- Fieldset element: to group related blocks of information

```html
fieldset {
  border: solid 0 transparent;
}
```
Useful HTML form elements

- Label element: to add a meaningful and descriptive label to each form element
  - Real benefit: to increase form usability for people using assistive devices
- Two ways to associate a label with a form
  - Implicitly, by nesting the form element inside the label element
  - Explicitly, by setting the for attribute of the label equal to the id name of the associated form element

```html
<label>email <input name="email" type="text"/></label>

<label for="email">email</label>
<input name="email" id="email" type="text"/>
```

The goal

![Diagram of form elements]
Simple form layout

- HTML code

```html
<fieldset>
  <legend>Your Contact Details</legend>
  <p>
    <label for="author">Name:</label>
    <input name="author" id="author" type="text" />
  </p>
  <p>
    <label for="email">Email Address:</label>
    <input name="email" id="email" type="text" />
  </p>
  <p>
    <label for="url">Web Address:</label>
    <input name="url" id="url" type="text" />
  </p>
</fieldset>
```

Simple form layout

- General styles for the fieldset and legend elements
  - The fieldsets must be vertically separated using margins
  - The contents can be given breathing space using padding
  - Light background, with a slightly darker, 1-pixel border

```css
fieldset {
  margin: 1em 0;
  padding: 1em;
  border: 1px solid #ccc;
  background: #f8f8f8;
}
legend {
  font-weight: bold;
}
```
Simple form layout

- Position the labels so they appear vertically above the form elements
  - Labels are inline elements by default
  - Setting their display property to block will cause them to generate their own block box, forcing the input elements onto the line below
  - The width of text input boxes varies from browser to browser: set the width of text input boxes

```html
label {
  display: block;
}
input {
  width: 200px;
}
```

Simple form layout

- Unlike text areas and text inputs, radio buttons and check boxes need to be handled differently
  - Rather than having their labels above them, these elements usually have their labels to the right of them
  - When stacked vertically all the elements are left aligned, making them easier to select
Simple form layout

- The width of the text boxes was defined by applying a width to the input element
  - The input element covers other form widgets such as check boxes, radio buttons, and submit buttons
- The best way to distinguish between input elements is to give them a class

```html
<fieldset>
  <legend>Remember Me</legend>
  <p>
    <input id="remember-yes" class="radio" name="remember" type="radio" value="yes" />
    <label for="remember-yes">Yes</label>
  </p>
  <p>
    <input id="remember-no" class="radio" name="remember" type="radio" value="no" checked="checked" />
    <label for="remember-no">No</label>
  </p>
</fieldset>
```

Simple form layout

- Override the previously set input width by setting the width of radio buttons to auto
  - The same can be done for check boxes and submit buttons

```css
input.radio, input.checkbox, input.submit {
  width: auto;
}
```

- Floating the radio buttons left will bring them back on the same level as their labels
  - A small amount of right margin will provide the desired spacing between the two elements

```css
input.radio {
  float: left;
  margin-right: 1em;
}
```
Longer form layout

• For longer and more complex forms, vertical space becomes an issue
  ◦ To improve scanning and reduce the amount of vertical space used, it makes sense to position the labels and form elements horizontally

Instead of setting the label to be a block-level element, float the labels left
Give the label a width so that all of the form elements line up

This width causes a large gap between the radio buttons
  ◦ Set the width on these labels explicitly

```css
label {
  float: left;
  width: 10em;
}

#remember-me label {
  width: 4em;
}
```
Complex form layout

- Example

```html
<p>
  <label for="dateOfBirth">Date of Birth:</label>
  <input name="dateOfBirth" id="dateOfBirth" type="text" />
  <label id="monthOfBirthLabel" for="monthOfBirth">
    Month of Birth:
  </label>
  <select name="monthOfBirth" id="monthOfBirth">
    <option value="1">January</option>
    <option value="2">February</option>
    <option value="3">March</option>
  </select>
  <label id="yearOfBirthLabel" for="yearOfBirth">
    Year of Birth:
  </label>
  <input name="yearOfBirth" id="yearOfBirth" type="text" />
</p>
```

Form labels are important for the accessibility.

- However, there are situations when you may not want to display a label for every element:
  - e.g., the date of birth field
Complex form layout

- Hide the “month of birth” and “year of birth” labels
  - Setting the labels’ display property to none would stop the labels from displaying, but it would also prevent many screenreaders from accessing them
  - Solution: position the labels off screen using a large negative text indent
  - To prevent the labels from affecting the layout, the width needs to be zeroed down

```css
#monthOfBirthLabel, #yearOfBirthLabel {
  text-indent: -1000em;
  width: 0;
}
```

Complex form layout

- The various form controls can then be sized individually and given margins to control their horizontal spacing

```css
input#dateOfBirth {
  width: 3em;
  margin-right: 0.5em;
}
select#monthOfBirth {
  width: 10em;
  margin-right: 0.5em;
}
input#yearOfBirth {
  width: 5em;
}
```
Multicolumn check boxes

• Goal: create a two-column layout for large groups of check boxes or radio buttons

• Problem: labels only work for individual elements, not groups of elements

To create the column effect, the check boxes are split into two sets, and each set is wrapped in a div. These elements are then grouped together by wrapping them in a fieldset with a descriptive id.

```html
<fieldset id="favoriteColor">
  <h2>Favorite Color:</h2>
  <div>
    <input class="checkbox" id="red" name="red"
           type="checkbox" value="red" />
    <label>red</label>
  </div>
  <div>
    <input class="checkbox" id="orange" name="orange"
           type="checkbox" value="orange" />
    <label>orange</label>
  </div>
  <br class="clear" />
</fieldset>
```
Multicolumn check boxes

- A generic fieldset style has already been created
  - Override those styles, zeroing down the padding and margin, removing the borders and setting the background color to be transparent

```css
fieldset#favoriteColor {
  margin: 0;
  padding: 0;
  border: none;
  background: transparent;
}
```

- The heading is going to act like a label so it needs to be floated left and given a width of 10ems like the other labels
- The headline also needs to look like a label, so the font weight needs to be set to normal and the font size needs to be reduced
- The two-column layout can then be created by giving the divs a width and floating them left

```css
#favoriteColor h2 {
  width: 10em;
  float: left;
  font-size: 1em;
  font-weight: normal;
}
#favoriteColor div {
  width: 8em;
  float: left;
}
```
**Multicolumn check boxes**

- Because the divs are being floated, they no longer take up any space and appear to spill out of the fieldset

  ![Fieldset with checkboxes](image)

- To force the fieldset to enclose these floats, a clearing element has been inserted after the second div
  - A `<br/>` element is used with a class of `clear`

```css
.clear {
    clear: both;
}
```

**Multicolumn check boxes**

- All the labels in this form have been floated left and set to be 10ems wide
- The labels for the check boxes do not need to be floated and require a much smaller width
- Firefox seems to treat the un floated labels as block-level elements
  - Explicitly set the display property to inline

```css
label {
    width: 3em;
    float: none;
    display: inline;
}
```
Form feedback

- Forms will usually require some type of feedback message to highlight fields that have been missed or incorrectly filled in
  - Usually done by adding an error message next to the appropriate field

```html
<p>
<label for="email">Email Address:</label>
<span class="feedback">Incorrect email address. Please try again.</span>
<input name="email" id="email" type="text" />
</p>
```

Form feedback

- The best approach is to include the error message text inside the form label, and then position it using CSS
Form feedback

- To position the feedback span, set the position of all of the paragraphs in the form to relative, thereby setting up a new positioning context
- Position the feedback span absolutely, so it appears to the right of the text input

```html
<p>
form p {
  position: relative;
}
.feedback {
  position: absolute;
  margin-left: 11em;
  left: 200px;
  right: 0;
}
</p>
```

Form feedback

- Apply the desired styling to the feedback message
  - E.g. text bold red, plus a warning image to the left side of the message

```html
form p {
  position: relative;
}
.feedback {
  position: absolute;
  margin-left: 11em;
  left: 200px;
  font-weight: bold;
  color: #760000;
  padding-left: 18px;
  background: url(images/error.png) no-repeat left top;
}
```
References

- Andy Budd, Cameron Moll, Simon Collison, “CSS Mastery, Advanced Web Standards Solutions”
  - [www.cssmastery.com/](http://www.cssmastery.com/)