Lecture 3 – CSS

Cascading style sheets also called simply *style sheets* have the role of holding presentation information separately from the html document which can focus more on the content. The benefits of this separation are manifold. It is easier to change the “look and feel” of a large collection of documents if one needs only to change the style sheet without touching the documents themselves. It is also possible to use the same web pages for different target media (PC screen, small screen of handheld device, printer) and concentrate the required presentation differences into the style sheets. Also for large projects it is possible to have different people with different skills and training work on the documents (probably writers with a deep understanding of the subject the web pages are about) and the style sheets (probably designers with a refined taste).

Adding style to web pages

A style sheet is a text file with the extension .css that consists of a sequence of rules. Each rule consists of two parts: selectors and declarations. The selectors define which HTML elements the rule applies to. The declarations list some CSS attributes and their values. The syntax for CSS is very different from the HTML syntax. For example the rule `p {color:#0000ff;font-size:14pt}` specifies that text within paragraphs should be blue and use a 14-point font. The rule `a,h3 {color:green}` specifies that the text for links and level three headings should be green. CSS supports very powerful selectors that allow rules to precisely target the right elements of the HTML document. We will discuss them at the end of the lecture.

There are multiple ways of adding style information to a web page. The simplest is to add a `<link />` tag inside the head of the document with its href attribute having the URL of the style sheet. For example this tag `<link href="classpage.css" rel="stylesheet" type="text/css" />` instructs the browser to use style sheet `classpage.css` residing in the same directory as the HTML document. Multiple external style sheets can be associated with a document. This makes sense when there is a general style sheet that applies to all web pages on a site and there are style sheets with further details that apply just to specific groups of pages (e.g. one for the pages with user documentation, one for the pages with company announcements, etc.). The `<style></style>` tag that also goes inside the head of the document can be used to specify style rules directly inside the document. This is not a good solution for style information that applies to more than one web page. CSS also supports special rules of the form `@import url(”classpage.css”);` which signal that rules are to be read from a separate file. There is a third form of including information: by using the HTML attribute `style` which works with nearly all HTML tags that can occur inside the body of the document. The value of this attribute must be a list of valid CSS rules separated by semicolons. This is a handy way to test the effect of various CSS declarations, but it should not be used in final web pages as it provides no separation between content and presentation.

CSS attributes controlling text

A very common use of CSS is to control the appearance of text. For example the color of text can be controlled using the `color` or `font-color` attributes (`background-color` controlling the background color for the entire element). The values of these attributes can be either six-digit hexadecimal RGB color codes such as `#00ff00` or textual color names. There are a few hundred of names you can use and entering “css color chart” in a search engine will give many pages listing them all. The `font-size` can specify the font size either numerically using various units, using symbolic names such as `small`, `medium` or `xx-large` or relative sizes such as `larger`, `smaller` or `150%`. `font-family` defines the font family to be used, `font-style` can be used to make the font italic, and `font-weight` to make it bold. It is common to use a coma separated list of fonts as the value o f the `font-family` attribute, and the browser will use the first font in the list it has. Typically one puts to the end of the list generic family names such as `serif`, `sans-serif`, or `monospace` so that if the browser doesn’t have any of the actual fonts we want, at least it can use a font from the same generic family. The nine fonts listed to the right are available to all major browsers on the most popular operating system, so they are considered “web-safe”. The `font` attribute can be used to set the values of font style, weight size and family such as in the declaration `font:italic x-large Arial, sans-serif`. It is common for CSS to have more specific versions of attributes and an attribute that sets many more specific attributes.

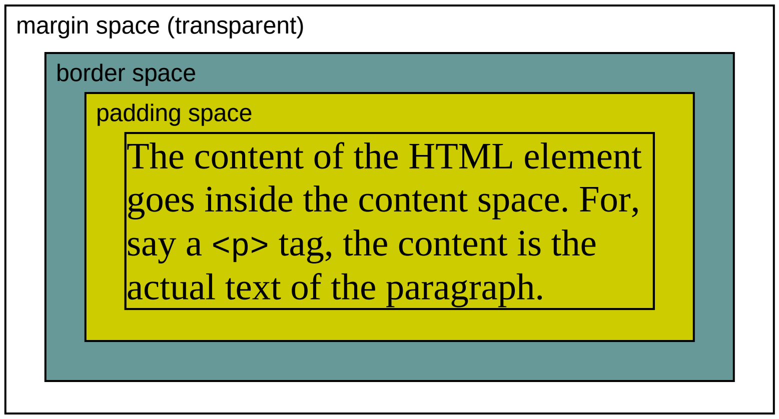
Arial
Arial Black
Comic Sans MS
Courier New
Georgia
Impact
Times New Roman
Trebuchet MS
Verdana

Other attributes control how the text is placed within the element it is contained in. `text-align` can take the values `center`, `justify`, `left` and `right` and `text-indent` sets the size of the indentation of the first line of text (say in a paragraph). `vertical-align` can be used to align text with respect to the element that contains it with values `top` or `bottom`, or to align it with respect to the surrounding text with values such as `baseline`, `sub` and `super`. Thus the declaration list `{vertical-align:super;text-size:smaller}` will make the text it applies to render like smaller superscript text.

Often a chunk of text one wants to control is not an HTML element of its own. For example one may want to italicize a sentence from a paragraph, not the entire paragraph. In such cases the HTML tag `<span></span>` together with the HTML attributes `class` or `id` can be used to turn the text snippet one wants to control into an HTML element that can be the target of CSS rules. Without CSS, `<span></span>` does not affect in any way how the text it encloses is rendered.

CSS attributes controlling spacing

There are CSS attributes controlling the size and spacing of various HTML elements. Before going into details about these attributes, it is useful to understand the general structure of the layout of HTML block elements such as paragraphs for example. At the middle there is a rectangular content space where the text goes. Around it there is some padding space that uses the same background as the content space. Around the padding there is a border that can have a different color and around the border there is the margin that is transparent. One or more of these spaces around the content space can be omitted by setting their width to 0. The CSS attributes `width` and `height` of the size of the content area and `border-width`, `padding`, and `margin` control the width of the other spaces around the content area. With attributes such as `border-top-width`, `margin-left`, and `padding-bottom` one can control the size of these spaces along the four lines framing the content space. You can also control the border style using the `border-style` and other similar attributes. The styles `solid`, `none` and perhaps `double` are the most useful as other border styles such as `groove`, `inset`, `outset`, `ridge`, `dashed` and `dotted` are not rendered consistently by various browsers. There is also a `border-color` attribute. The attribute `border` can be used to set width, style and color. Thus the declaration `border-right:4px solid black` instructs the browser to draw a 4 pixel solid border at the right of the element it applies to.


margin space (transparent)
border space
padding space
The content of the HTML element goes inside the content space. For, say a <p> tag, the content is the actual text of the paragraph.

Many CSS attributes express lengths and sizes. Often they can be expressed as a percentage (it depends on the actual attribute and on the element it applies to what it is a percentage of). When they express absolute measures, various units listed in this table can be used.

Unit	Description
em	Element's font height
Ex	The height of a lower case x
Px	Pixel (exact size depends on display)
In	Inch
Cm	Centimeter
Mm	Millimeter
Pt	Point = inch/72
Pc	Pica = 12 points

Table 1: Length (size) units used by CSS

CSS-controlled positioning

CSS has strong features for controlling the position of various elements on the screen. These features are often put to use in conjunction with the `<div></div>` tag which defines rectangular blocks that include other elements (some of which may be divs). Divs can be used instead of tables to control the overall layout of a web page (as done for example by `http://www.yahoo.com`) and their popularity has been increasing steadily. There are two general strategies to lay out your web page: fixed-width and fluid-width. The first strategy typically relies extensively on the use of CSS positioning and divs and it results in web pages that retain their width irrespective of the size of the browser window: if the window is narrower than the width of the actual content the browser will use a horizontal scroll bar, if the window is wider, the browser will display some extra empty space on one or both sides. It is undesirable to force the user to use the horizontal scroll bar to see the entire page, but if the width of the actual content is within the expected window width for common screen resolutions this does not occur. Fluid-width designs use tables and they resize the width of various elements to fill the actual width of the window. The main disadvantage of fluid-width layout is that it is harder to achieve the desired alignment and spacing of elements; whereas in general text is reformatted by browsers the way the web page authors would want it to, images do not stretch and complex layouts using them are harder to do as fluid-width.

The CSS attributes `top`, `left`, `right` and `bottom` can be used to indicate the offset between the element being positioned (including its margin) and its positioning context (e.g. the entire browser window, or the area covered by the HTML element's parent). The CSS attribute `position` can be set to `absolute` to indicate that the offsets are relative to the positioning context of the element or to `relative` to indicate that the offsets are relative to the position where HTML's normal rules would have put the element. The value `fixed` indicates that the position of the element is with respect to the current view in the document (it ignores scrolling). For example the list of declarations `{position:absolute;top:80px;left:40px}` positions the upper left corner of an element 80 pixels lower and 40 pixels to the left of the upper left corner of the positioning context. With relative positioning the parent element leaves blank the space the positioned element would have occupied normally, but with absolute positioning no blank space is kept. The `float` attribute can be used to control how a given element is aligned with respect to the containing box and how other content wraps around it. If it is not set or if it is explicitly set to `none`, the element's position is determined by where it appears in the HTML document and at most one line of surrounding text appears in the same horizontal bar. If it is set to `left` or `right` the elements aligns with that edge of the surrounding box and multiple lines of text can flow around it. The `display` attribute is also often used and it controls various details mostly linked to how the element it applies is positioned with respect to surrounding text. The value `inline` asks the browser to treat the element as text (it can be on multiple lines and standard text wrapping applies) and the value `block` asks it to be enclosed into an invisible rectangular box (like a paragraph or a table cell). The value `none` causes the browser to completely hide the element and render the text around it as if the element didn't exist. Interactive web pages often achieve various effects by manipulating the value of this attribute.

CSS positioning overrides the default positioning rules of the browser and can lead to such typically undesired consequences as moving the element outside the window or causing elements to overlap. But it in combination with client-side programming it helps make pages interactive through techniques such as drag and drop or various animations.

In such contexts page elements often overlap. To control the overlap the `z-index` attribute is used. It has an integer value and the elements with larger z-indices are rendered "on top of" elements with lower indices (they appear closer to the user and cover the other elements).

CSS selectors

So far we focused on the second part of CSS rules: the declarations which set values for various attributes. The first part of a CSS rule is a coma separated list of CSS selectors that define to what elements the declarations should apply. We already saw the basic selector which is simply the name of an HTML tag and causes the rule to apply to all instances of that tag. CSS supports a variety of much more specific selectors some of which are shown in Table 2. For example the ID selector allows CSS to target the element with a given identifier. The identifier of an HTML element is defined by the id attribute whose value has to be unique within the HTML document. The web page author can also define classes of HTML elements using the class attribute and CSS rules can apply to classes also. The descendant selector builds on the containment relation between HTML elements: it allows a rule to target certain elements specifically when they are inside another element. Pseudo-class selectors apply not to elements that the page author explicitly puts into user-defined classes, but to elements that the browser can assign into a pre-defined category. For example a:link applies to links that have not been visited, while a:visited applies to links that have been and a:active to a link that is being clicked on. Another example: `div#main table tr.odd p:hover`. This selector targets paragraphs that the mouse hovers over but only if they are inside a table row of the class "odd" (note that they need not be directly inside the row, they can be inside a cell that is inside a row) with the further constraints that the row be inside a table that is inside a div with the identifier main.

Selector	HTML	CSS.
ID selector	`<p id="last">`	`#last{border:solid red}` or `p#last{…}`
Class selector	`<p class="hot">` `<h2 class="hot">`	`p.hot{font-size:large}` `.hot{color:red}`
Descendant selector	`<p><em>This text</em></p>`	`p em{background-color:blue}`
Pseudo-classes		`a:link,a:active, a:visited,:hover`

Table 2: Some selectors used by CSS

Elements inherit attributes from their ancestors (elements defined by tags that enclose the given element). For example one can set the font for a div, and all elements inside it (paragraphs, tables, cells within tables, etc.) inherit the font from the div the font. Of course individual elements inside the div may override the font. Some CSS attributes are inherited (e.g. the font) others are not (e.g. the position attributes). Yet other attributes are inherited by some elements but not others. Most of these rules for what attributes inherited are intuitive.

Even if we look just at the CSS rules for a given element, multiple CSS selectors can apply. It is common that multiple rules setting the value of the same attribute apply to the element (hence the name “cascading” style sheets). CSS has clearly defined rules for deciding which rule should take precedence. As a rule of thumb, more specific selectors take precedence over less specific selectors. Ids are more specific than classes which are more specific than HTML tag names. Descendant selectors are more specific than single tag selectors, but classes and ids take precedence.

Powerful tools exist that allow you to investigate CSS attribute of elements in actual web pages, track which CSS rules derive from and in general understand better how CSS work. We strongly recommend that you use the firebug add-on to Firefox. It also helps visualize how tags in the HTML source map to elements rendered on-screen, and later in the class will help with client-side programming.