Cascading Style Sheets

Giuseppe Della Penna
Università degli Studi di L'Aquila

giuseppe.dellapenna@univaq.it
http://www.di.univaq.it/gdellape

Notes to the English Version

These slides contain an English translation of the didactic material used in the Web Engineering course at University of L'Aquila, Italy.

The slides were initially written in Italian, and the current translation is the first result of a long and complex adaptation work.

Therefore, the slides may still contain some errors, typos and poorly readable statements.

I'll do my best to refine the language, but it takes time.

Suggestions are always appreciated!

Cross-Browser Compatibility

Standards and Quirks mode

- Browsers support two rendering modes: *Quirks mode* and *Standards mode*. The Standards mode follows the W3C specifications, thus is (hopefully) browser independent.
- The Quirks mode follows the formatting rules of the specific browser, with its limitations and extensions.
- Quirks mode exists to make the browser compatible with old sites, which were developed with very *browser-dependent* code. Today, it is *necessary to* develop new sites in Standards mode. By default, browsers use Quirks mode. To enter Standards mode all you need is to write the correct doctype declaration the beginning of the document, as follows:
  - To use **XHTML transitional**:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

  - To use **XHTML strict**:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

The Style Sheets

- A CSS is a text document, consisting of a set of *style rules*.
- A document can have more than one associated style sheet.
- A style sheet, in general, can be *embedded* in a document or *attached* to it. HTML:
  - embedded CSS style sheets are written in the document <head>, inside a <style> tag with type = "text/css".
  - it is also possible to write a style rule directly into a specific element using its style attribute.
  - finally, CSS style sheets can be imported from an external resource by writing, in the <head> section, a <link> tag with type = "text/css", rel = "stylesheet" and href = “style_sheet_uri”.

Multiple Style Sheets

- If you specify multiple style sheets for the same document, CSS generally combines them in order of inclusion.
- However, it is possible to *conditionally* include a style sheet (or a group of sheets) in a document based on various criteria:
  - Output media
  - User preferences

Multiple Style Sheets

Media Types

- When you embed or link a style sheet in an HTML document, you can specify in the <style> or <link> element the *media* attribute.
- The media value is a comma-separated list of names of media descriptors, which identify the type of output for which the style sheet is suitable for:
  - screen (default)
  - tty, tv, projection, handheld: indicate different types of visual terminals
  - print: the style sheet used for printing (useful, e.g., to eliminate background colors and optional elements used in the on-screen rendering)
  - aural, braille: rendering to speech and braille synthesizers
  - all: for all types of media

Multiple Style Sheets

Alternative Style Sheets

- Moreover, you can give a page three different styles for the same media type :
  - **Persistent style:** it is always loaded by the browser. The styles embedded in the document are always persistent.
  - **Preferred style:** this is the default style that will be combined with the persistent, if present. It is indicated by putting a title = "style_name" attribute in the <link> tags.
  - **Alternative styles**: these are styles that can be loaded alternately to the preferred one, depending on user preferences. They are indicated by inserting the attribute title = "style_name" in the <link> tag and changing the rel attribute to "alternate stylesheet".

Rules

- A CSS rule defines a *formatting style* and a *class of elements* it must be applied to.
- *A formatting style* is in turn defined by a list of properties, with the syntax **property: value**, surrounded by curly braces and separated by a semicolon.
- The *classes element* are defined through special patterns called *selectors*.
- An example of an abstract rule is
  SEL {P1: V1 [!important] P2: V2 P3: V3}
- The optional **!important modifier**, written after the value (but before the separator) of any property, is used to increase the *priority of the rule* during the *cascading* process, as we will see later.

Rules

Simple selectors

- A simple selector is a base selector followed by zero or more *attribute selectors*, *class selectors*, *ID selectors*, *pseudo classes* and *pseudo elements*.
- There are two *basic selectors*:
  - The *universal selector* (*) that matches any element.
  - The *type selectors* (strings representing names of elements), which match any element with the given name.

Rules

Attribute Selectors

- A base selector can be followed by one or more attribute selectors. The selectors of this type can be written as follows:
  - **[A]** (the element must have an attribute A)
  - **[A = V]** (the element must have an attribute A with value V)
  - **[A ~ = V]** (the element must have an attribute A whose value is a space separated list containing V)
  - **[A | = V]** (the element must have an attribute A whose value is a list separated by '-' containing V)

Rules

Class Selectors

- When working with HTML, there is a special abbreviated syntax, called *class selector*, to work with the class attribute of the elements
  - The syntax "S.C", applicable to any simple selector S is equivalent to S [class ~= C].
  - As a special case, you can write a class selector alone (while in general attribute selectors should follow another valid selector), which implies an universal selector: .C is equivalent to *.C, i.e., *[class ~= C]

Rules

ID Selectors

- In XML (and HTML), you can assign each element a unique ID.
- This ID can be used in CSS to apply formatting to a specific item.
- The ID selector can be placed after each base selector and has the syntax #ID.
  - The switch **S#ID** matches the element which corresponds to the selector S and has the specified ID
  - It is possible (and common) to use ID selectors alone, just like class selectors, implying a universal selector: **#ID** is equivalent to *#ID, i.e., the element (of any type) with the specified ID.

Rules

Pseudo-classes

- The *pseudo-classes* identify items based on soma special properties.
  - **:first-child** (the element is the *first child* of its parent)
  - **:link** *(unvisited links)*
  - **:visited** *(visited links)*
  - **:hover** (the element currently *indicated by the user*, for example by moving the mouse over it)
  - **:focus** (the currently *focused* element, namely, the one that accepts keyboard input)
  - **:active** (the currently *activated* element, for example by a mouse click)

Rules

Pseudo-elements

- It is also possible to apply formatting to fictitious elements, not really part of the document and/or delimited by tags. These elements are called *pseudo-elements*.
  - **:first-line** (the first line of the text block contained in an element)
  - **:first-letter** (the first character of the text block contained in an element)
  - **:before,:after** (indicate the text position preceding and following an element, used in conjunction with the CSS *content* property)

Rules

Combination of selectors

- Two selectors S and T can be combined in a third selector in various ways:
- S T (space between)
  This selector matches the elements designated by T only if they are descendants of an element that matches S
- S > T
  This selector matchis the elements designated by T only if they are children of an element that matches S
- S + T
  This selector matches the elements designated by T only if they immediately follow an element that matches S and has the same parent.
- S, T
  This selector matches the elements designated by S and T (logical OR, *selector grouping)*

Style Properties Value Deduction

- During the rendering process, CSS must determine the style to be assigned *to each element of the document*.
  - This involves calculating the value *of each style property* that the element can have.
- To calculate the value of a property P of an element E, CSS proceeds as follows:
  1. **Cascading**
     The property has a style specified through CSS rules?
  2. **Inheritance**
     The property can be inherited from the parent?
  3. **Default**
     The property has the value given by the default stylesheet.

Cascading

- In a style sheet you can write several rules that match the same element (and it is often very useful).
- Moreover, CSS implicitly provides two other style sheets:
  - The user style sheet. The user may provide style rules, such as the base font size.
  - The style sheet. Every browser should have its own default style sheet.
- When CSS calculates the value of *each individual style property*, it takes in consideration all the rules that match the element to be formatted in any style sheet, and selects one through a **waterfall** process.

Cascading

Selection Rules

- When multiple rules match the same element, the final value of each property is selected using the following procedure.
- **Priority of Origin**
  - !important value from the user style sheet
  - !important value! from the author style sheet
  - value from the author style sheet
  - value from tha user style sheet
  - Default value from the browser style sheet
- **Specificity** *(for properties with the same priority of origin)*
  - A selector that is more specific to a particular element takes precedence over a more generic one.
  - In HTML, the rules included in the *style* attributes have the greatest specificity.
- **Order** *(for properties with same priority of origin and specificity)*
  - A rule takes precedence over those that *precede* it.

Inheritance

- Many properties (see the specification) are automatically *inherited* by the children of an element, if there are no specific rules that require a different value.
  - This default behavior is very useful when creating complex style sheets.
  - For example, if you specify a font for the P tag, all the tags contained therein (e.g., B) have the same font, unless a style is assigned to them (collectively or individually) that specifies a different font.
- It is also possible to force the inheritance of a property by specifying (where possible) the *inherit* keyword as the value of the property itself.

Basic Elements of CSS

Measure Units

- The measures are expressed in the CSS language by numbers (also floating point and in some cases negative) immediately followed by the name of the measurement unit.
  - Inserting a space between the number and the unit name usually makes the code unusable!
  - The zero measure can be specified without a unit.
- There are two classes of units: *relative* and *absolute*.
- The *relative units* are **em** (current font-size), **ex** (current x-height) and **px** (device pixels).
  - Relative units are very useful if you want to automatically adjust sizes based on the output device and its settings (e.g., printer or browser with various font sizes)
- The *absolute* units are **in** (inches), **cm** (centimeters), **mm** (millimeters), **pt** (points = 1/72 of an inch), **pc** (picas = 12 points).
  - They are useful only when the output device is unique and precisely defined.
- In many cases, the measures can also be expressed as *percentages*. The reference measurement is usually the same property of the container element.

Basic Elements of CSS

Colors

- The colors can be defined in the CSS via the numerical RGB specification or through their own name.
- The *RGB specification* allows you to define any RGB color through the value of its three components *red*, *green* and *blue*.
- *RGB hexadecimal strings* have the form **#RRGGBB,** where each pair of digits represents the hexadecimal value of the corresponding component.
  - The short form **#RGB** is the number where each component has the value of the corresponding digit repeated twice.
- *RGB decimal strings* are obtained using the construct **rgb(R, G, B),** where R, G and B are numbers between 0 and 255 or percentages (i.e., fractions of the maximum 255).
- Finally, the colors for which a name is defined are **maroon** (#800000), **red** (#ff0000), **orange** (#ffA500), **yellow** (#ffff00), **olive** (#808000), **purple** (#800080), **fuchsia** (#ff00ff), **white** #ffffff, **lime** (#00ff00), **green** (#008000), **navy** (#000080), **blue** (#0000ff), **aqua** (#00ffff), **teal** (#008080), **black** (#000000), **silver** (#c0c0c0) and **gray** (#808080)

Basic Elements of CSS

Shortand Properties

- The CSS language has many properties that are often set in a group, such as the three properties that define a border (color, width, style) or the font properties(family, size, weight, ...).
- For this reason, there are also the so-called *shortand* properties, which allow, with their particular syntax, to set in a single operation the values of several properties.
- In a shorthand property the values of each "component" property are simply separated by a space. The order is not usually significant, because there is no ambiguity in the language.
  - If one or more properties are omitted in the shorthand notation, their value is set to the default one.
- For example, the CSS font property can be used to set all the font-style font-variant, font-weight, font-size, line-height and font-family peoperties.

Borders

- Most items can have a border on the four sides of their boxes. Each border can have different characteristics (color, thickness, style). It is also possible, for table cell borders, to specify how neighboring edges should be combined.
- Note: table borders specified through CSS are independent from those shown by the *border* attribute of <table>.
- The border color can be specified by symbolic names (e.g., white), or through their RGB components in hex (e.g., #FFFFFF) or decimal (e.g., rgb(255,255,255)) form
- The border thickness is a value that can be specified in any of the measurement units understood by CSS (e.g., px, pt, mm, ...)
- The main styles for the borders are *dotted*, *dashed*, *solid*, *double*, *groove*, *ridge*, *inset*, *outset*. However, most browsers only support solid, dotted and dashed styles.
- For all of the border properties there exist shorthands that allow one to set the same values for all sides at once and/or specify the three properties (color, thickness, style) with a single statement.

Borders

CSS Properties

- border (border-top, border-right, border-bottom, border-left)
  - **Values:** {*width, style, color*}
- border-color (border-top-color,border-right-color,border-bottom-color,border-left-color)
  - **Values:** *color* | transparent
- border-style (border-top-style,border-right-style,border-bottom-style,border-left-style)
  - **Values:** **none** | *border style name* | inherit
- border-width (border-top-width,border-right-width,border-bottom-width,border-left-width)
  - **Values:** *measure*
- border-collapse
  - **Values:** collapse | **separate**
    Elementi: tables and internal inline elements

Background

- All the *block* elements can have a background consisting of a solid color or an image
- In the case of image backgrounds, the file to be used must be indicated through the construct url('...') and it is possible to specify:
  - In which position to display the image w.r.t. to the element background.
  - If the image should be repeated to fill the entire surface available to the element.
  - If the image has to "scroll" with the content of the window or remain fixed.
- Thanks to their versatility, backgrounds are often used for improper purposes, such as creating graphics (buttons, structural elements of the page, etc.) that can not be achieved simply by importing images through the HTML <img> tag.

Background

CSS Properties

- background-color
  - **Values:** *color* | transparent
- background-attachment
  - **Values:** scroll | fixed | inherit
- background-image
  - **Values:** *url* | none | inherit
- background-position
  - **Values:** left top | left center | left bottom | right top | right center | right bottom | center top | center center | center bottom | *x% y%* | *measure measure* | inherit
- background-repeat
  - **Values:** repeat | repeat-x | repeat-y | no-repeat | inherit

Formatting

characters

- color (font color)
  - **Values:** *colore*
- font-family (font)
  - **Values:** one or more font names, separated by commas, in order of priority
- font-size
  - **Values:** *measure*
- font-style (italics, etc.)
  - **Values:** normal | italic | oblique
- font-variant (small caps, etc.)
  - **Values:** normal | small-caps
- font-weight (bold, etc.)
  - **Values:** normal | bold | bolder | lighter
- text-decoration (underlined, etc.)
  - **Values:** none | underline | overline | line-through
- text-transform (upper/lower case)
  - **Values:** capitalize | uppercase | lowercase | none

Formatting

paragraphs

- line-height
  - **Values:** *measure* | normal
- text-align (paragraph horizontal alignment)
  - **Values:** left | right | center | justify
- vertical-align (paragraph vertical alignment)
  - **Values:** top | middle | bottom
  - **Elements:** table cells
- text-indent (paragraph left indentation)
  - **Values:** *measure*
- word-spacing
  - **Values:** *measure* | normal
- letter-spacing
  - **Values:** *measure* | normal

Lists

- Using CSS, you can create bulleted and numbered lists of simple types, with standard images or numbers (Arabic, Roman, etc..) as bullets, using the list-style-type attribute.
- The most advanced list features, made available through the list-style-image attribute, allow the use of images as bullets. In this case, the list-style-type attribute is ignored.
- When you create custom lists with image bullets, you should always adjust the indents, margins and padding of the elements in order to achieve the desired visual effect.
  - Depending on your browser, the margin and/or the padding of list-item elements determine the indentation of the element itself and/or space between the bullet and the associated text.
- The list-type attributes can be applied to all items whose *display* property is set to *list-item*.

Lists

CSS Properties

- list-style-type (standard bullets)
  - **Values:** disc | circle | square | decimal | decimal-leading-zero | lower-roman | upper-roman | lower-greek | lower-latin | upper-latin | armenian | georgian | lower-alpha | upper-alpha | none |
- list-style-image (image bullets)
  - **Values:** *uri* | none
- list-style-position (position of the bullet relative to item text)
  - **Values:** inside | **outside**

Box Model

Control the Box Generation

- It is possible to specify how the box associated with an element should be generated.
- display
  - **Values:** inline | block | list-item | none
  - **Block** generates a block-like box, which occupies a horizontal and vertical space disjoint from the other boxes, the div or p
  - **Inline** generates an inline box, which becomes part of the text flow without interrupting it (such as b or span)
  - **List-item** displays the box as a list element (like li)
  - **None** disables the generation of the box, removing the associated element from the document.

Box Model

Showing and Hiding Elements

- After generating the box related to an element, you can specify whether the contents of the box should be rendered or not.
- visibility
  - **Values:** **visible** | hidden
  - **Visible** (default) shows the element.
  - **Hidden** hides the element.
- By setting the property to hidden, the element box is not removed from the document flow, so its footprint is still considered in the layout calculation.

Box Model

Content Management

- Generally, the content of a block is limited to the size of the block itself. The content, however, may be larger than its container.
- In these cases, you can specify how to handle the part that overflows the container.
- overflow
  - **Values:** **visible** | hidden | scroll | auto
  - **Visible** (default) allows the extra content to be rendered outside the container.
  - **Hidden** hides the piece of content overflowing the container.
  - **Scroll** brings up the scroll bars inside the container, so that the content can be scrolled. The bars appear in any case, even if the content does not overflow.
  - **Auto** makes scroll bars appear inside the container, so that its content can be scrolled, only if it overflows the container.

Box Model

Margins and Spaces

- margin (margin-right, margin-left, margin-top, margin-bottom)
  - The margin is the space between the border of an element box and that of surrounding objects.
  - **Values:** *measure*
- padding (padding-top, padding-right, padding-bottom, padding-left)
  - The padding is the empty space between the border of an element box and the contents of the box itself.
  - **Values:** *measure*
- In many cases, margin and padding have the same visual effect, but you must always use the attribute that is logically suitable for your purpose.

Box Model

Sizing

- The size of each element can be set in various ways, completely overwriting or constaining the natural size calculated by the browser.
- In the first case, you can specify absolute units or percentages that are applied to the corresponding dimensions of the container.
- In the second case, you can specify minimum or maximum sizes, expressed as absolute units or percentages.

Box Model

Sizing Properties

- width, height
  - **Values:** *measure* | **auto** | inherit
  - Set the element width and height, respectively. The auto value corresponds to the natural size, calculated through the other element properties.
- max-height, max-width, min-height, min-width
  - **Values:** *measure* | inherit
  - Set minimum or maximum size of the element.

Box Model

Positioning

- Items can be placed in four different ways, specified with the CSS attribute *position*:
  - **Static:** the object is placed in its natural position, given by the text flow (default).
  - **Relative:** the object is located at the coordinates set by the left, right, top and bottom attributes *relative to* its natural position.
  - **Absolute:** the object is located at the coordinates set by the left, right, top and bottom attributes, relative to the upper left corner of its nearest container with *non static positioning*.
  - **Fixed:** The object is located at the coordinates set by the left, right, top and bottom attributes , relative to the upper left corner of the viewport.
  - Elements with absolute or fixed positioning are *removed from the text flow*. Their position does not depend on the elements that surround them, though, if absolute, they continue to flow along with the rest of the page.

Box Model

Positioning Properties

- position
  - **Values:** static | relative | absolute | fixed
  - Determines the type of positioning for the element
- top, left, right, bottom
  - **Values:** *measure*
  - Determines the element position, according to the rules defined by the value of the position property
- z-index
  - **Values:** *number* | auto | inherit
  - Determines the element position on the Z axis. Higher values move the element towards the user.

Box Model

Floats

- The floating technique allows to remove elements from the text flow and place them in a dynamic way on the left or right edge of the container.
- Floating elements are always distributed in the best possible way according to the available space.
- The text outside floating elements flows around their margin.
- This type of effect is often used to create menus, column layouts, etc.

Box Model

Floats Properties

- float
  - **Values:** left | right | none
  - Set the object as floating on the left or right side of the container. The value none disables floating.
- clear
  - **Values:** left | right | both
  - The clear property set on an element requires all the floats of the type specified (left, right or both) to be placed on the page before the element itself.

CSS Media Queries

- Media queries are one of the most important features introduced by CSS3 and supported by all modern browsers.
- Media queries can be used:
  - Within the *media* attribute of the link tag, to import a stylesheet only if the query is satisfied, or
  - Directly in the stylesheet, enclosing a set of rules between braces preceded by the **@media** *at-rule*
- A media query consists of a *media type* (*screen*, *print*, etc.) combined through an AND with e sequence of *media features*, whose support may vary in browsers.
  - orientation [portrait | landscape]
  - width: <measure>, min-width: < measure >, max-width: < measure >
- It is possible to merge (OR) several media queries using a comma, for example:

```
@ media screen and (min-width: 300px),
screen and (orientation: landscape) {...}
```

- An interesting support library (try it!) uses javascript to make media queries available even in older browsers: http://code.google.com/p/css3-mediaqueries-js/

Responsive Design

- **Responsive design** is a layout design technique introduced by Ethan Marcotte in his 2010 article on A List Apart: http://alistapart.com/article/responsive-web-design
- With this technique the website layout *dynamically adapts to the size and the characteristics of the output device*.
- The three components of a responsive design are:
  - fluid layout (typically grid-based)
  - CSS3 media queries
  - flexible images
- When CSS media queries were not available, responsive design was achieved with the help of fluid design supported by scripts, but now you can get much more advanced effects using only stylesheets.
- By applying specific changes to the stylesheets through media queries, it is possible, for example, to hide items, relocate others, decrease borders and spacings, etc..

Responsive Design

Example

- For "normal" browsers
  - Fluid design, with percentage widths (possibly also for images). A liquid design makes the layout robust also for devices that do not support media queries.
- For tablets:
    - For example: *@media only screen and (min-width: 768px) and (max-width: 959px)*
  - Remove "creative" paddings and spacings, slightly decrease the font size, etc..
- For mobiles:
    - For example: *@media only screen and (max-width: 767px)*
  - Fixed design, for example 320 pixels wide, or use a min-width on the main elements so that they don't become too narrow.
  - Linearize columns, hide secondary elements (parts of the header and footer, etc.), show more compact menus.

A Grid Layout with Floats

- Now we will see how to create a **liquid grid layout**, useful when webpage elements should be aligned in rows and columns, exploiting only *floats*.
- This kind of structure is the basis for **responsive** layouts, because it adjusts to the size of the browser. It is also possible to **nest** these grids, to obtain very complex effects.
- Sources:
  - The base of this layout is the 960 grid system (http://960.gs/) which, however, is fixed (width 960 pixels) and does not use media queries.
  - The media queries part is inspired by Skeleton (http://www.getskeleton.com/), which, however, remains fixed for widths greater than 960 pixels.
  - Finally, the inspiration for the liquid version of the grid is taken from http://www.designinfluences.com/fluid960gs/.
  - See also the introductory article on fluid grids by Ethan Marcotte: http://alistapart.com/article/fluidgrids

A Grid Layout with Floats

The rows

- This design allow to place up **to 16 columns on each row**.
- Row cells have a 1% **spacing** on both sides.
- The *container* and *row* classes represent the grid and the row containers, respectively::
  - .container { position: relative; width: 98%; padding: 0;}
    - The grid container has a small extra margin . It is not strictly needed, and it is safe to write rows outside a container.
  - .row { margin-bottom: 10px; }
    - Rows have a little spacing below, that can be removed.

A Grid Layout with Floats

The columns

- The *column* class, define tghe common column properties:
  - .container .column, .container .columns { float: left; display: inline; margin-left: 1%; margin-right: 1%; }
- *Alpha* and *omega* (i.e., first and last) columns have no lateral margin:
  - .column.alpha, .columns.alpha { margin-left: 0; }
    .column.omega, .columns.omega { margin-right: 0; }
- Classes *one*, *two*, etc. define the horizontal span of the cell, in columns (1-16):
  - .container .one.column { width: 4.25%; }
    .container .two.columns { width: 10.5%;}
  - Horizontal widths for columns from three to sitexteen are, respectively, 16.75%, 23%, 29.25%, 35.5%, 41.75%, 48%, 54.25%, 60.5%, 66.75%, 73%, 79.25%, 85.5%, 91.75%, 98%

A Grid Layout with Floats

Floats encapsulation

- To conclude, some cross-browser trick helps to maintain the float columns inside their row:
  - .row:before,.row:after { content: '\0020'; display: block; overflow: hidden; visibility: hidden; width: 0; height: 0; }
  - .row:after { clear: both; }
  - .row{ zoom: 1; }

A Grid Layout with Floats

Example

```
<div clas="row">
        <div class="three columns">
            A
        </div>
        <div class="thirteen columns">
            B
        </div>
</div>
<div clas="row">
        <div class="three columns">
            C
        </div>
        <div class="thirteen columns">
            <div clas="row">
                <div class="eight columns">
                    D
                </div>
                <div class="eight columns">
                    E
                </div>
            </div>
        </div>
</div>
```

A	B	
C	D	E

A Grid Layout with Floats

Media queries

- The fluid layout becomes linear below a certain width: this is achieved by disabling the *floating* and not forcing a column width, so that the columns can "wrap".
- In addition, in this example we fix the size of the entire layout to prevent it from falling below a minimum threshold.

```
@ media only screen and (max-width: 767px) {
  .container {width: 300px;}
  .container .columns, .container .column {margin: 0;}
  .container .one.column, .container .one.columns, ... {float:
  none; width: auto;}
  }
```

Dynamic Content

Prefixes, suffixes, quotes and counters

- quotes
  - **Values:** none | (string string)+
  - Indicates the quotes (open and closed) to use for this element, if necessary. It is possible to specify multiple pairs, one for each level of nesting of the quotation marks.
- counter-reset
  - **Values:** (string [integer])+
  - Clears (or sets to the given number) the value of given counters for the element.
- counter-increment
  - **Values:** (string [integer])+
  - Increments by one (or the given number) the values of the given counter for the element.
- content
  - **Values:** none | (string | counter(C,S) | open-quote | close-quote)+
  - Applies only to pseudo elements :before and :after and specifies the text that should be inserted before or after an element, respectively. The values of open-quote and close-quote are those set by the quotes attribute. C can be any counter visible from the element. S (optional) is one of the values defined for the list-style-type.

Other Properties

- cursor
  - **Values:** [*uri*,]* (auto | crosshair | default | pointer | move | e-resize| ne-resize | nw-resize | n-resize | se-resize | sw-resize |s-resize | w-resize | text | wait | help | progress)
  - Sets the shape of the mouse when it is above the element.
  - The list of URI, optional, indicates one or more external resources to be used as a cursor. The browser uses the first one it can retrieve.
  - In any case, you must also provide one of the standard cursors, as a single value or as the last choice in the list.

Cross-Browser Compatibility

Reset Stylesheet

- Different browsers apply to the properties **different default values,** corresponding to their *default stylesheet*.
- To achieve crossbrowser CSS, you may want **to reset these differences** and create style sheets from a **minimum common basis.** This can be achieved by inserting at the beginning of the style sheet some rules like the following. (http://meyerweb.com/eric/tools/css/reset/)

```
html, body, div, span, applet, object, iframe, h1, h2, h3, h4, h5, h6, p, blockquote, pre, a, abbr, acronym, address, big,
    cite, code, del, dfn, em, img, ins, kbd, q, s, samp, small, strike, strong, sub, sup, tt, var, b, u, i, center, dl, dt, dd, ol,
    ul, li, fieldset, form, label, legend, table, caption, tbody, tfoot, thead, tr, th, td, article, aside, canvas, details,
    embed, figure, figcaption, footer, header, hgroup, menu, nav, output, ruby, section, summary, time, mark, audio,
    video
    { margin: 0; padding: 0; border: 0; font-size: 100%; font: inherit; vertical-align: baseline; }
article, aside, details, figcaption, figure, footer, header, hgroup, menu, nav, section /* HTML5 */
    { display: block; }
body
    { line-height: 1; }
ol, ul
    { list-style: none; }
blockquote, q
    { quotes: none; }
blockquote:before, blockquote:after, q:before, q:after
    { content: ''; content: none; }
table
    { border-collapse: collapse; border-spacing: 0; }
```