Cascading Style Sheets (CSS)

Sivan Toledo
Tel-Aviv University

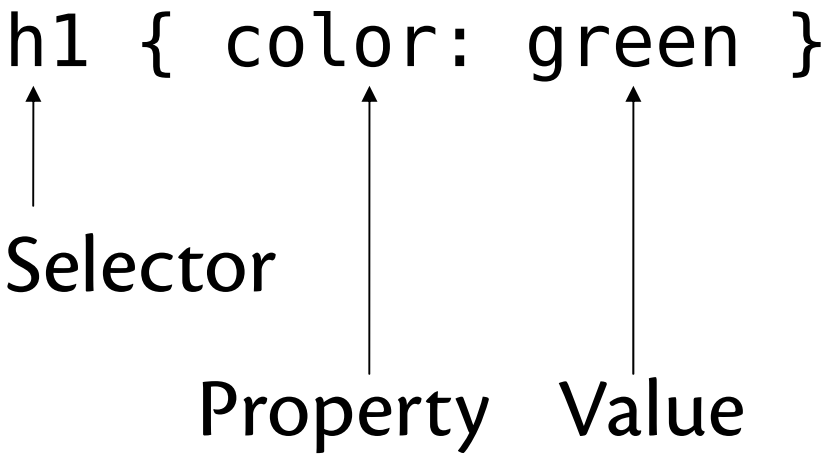
A Simple Example (HTML Integratation)

```
<html>
  <head>
    <title>A CSS Example</title>
    <style type="text/css">

      h1 { color: green }

    </style>
  </head>
  <body>
  ...
</html>
```

The Structure of a Rule

CSS is a declarative language: you declare style rules that the browser applies to document elements

The structure of a simple rule:



Multiple Selectors

We can apply a single rule to multiple selectors:

```
h1, h2, h3 { color: green }
```

Don't forget the commas (a sequence of selectors with no commas has a different meaning; more on that later)

Multiple Rules & Grouping Declarations

```
h1 { color:       green }
h1 { font-weight: bold  }
```

or

```
h1 { color:       green;
     font-weight: bold
   }
```

The semicolon is a separator (can put one after the last declaration)

Inheritance

```
body { color: green }
p    { color: red   }
```

The body **and all its children** in the document tree will be printed in green, except for paragraphs and their subtrees, which will be printed in red

Most of the properties are inherited by children elements in the document, unless they are overridden by other rules

Non Inheritance

Some properties (not many) are not inherited; for example, the background is not inherited in order to preserve correct tiling of background images

Shorthand Properties

Some properties are shorthand notation for groups of more specific properties

```
h1 { font: 36pt italic }

h1 { font-size:  36pt;
     font-style: italic }

blockquote { margin: 1em 0em 1em 0em }

blockquote { margin-top  : 1em;
             margin-right: 0em; ...
```

Units: em

```
h1 { font-size:   4em   }      of the parent
p  { text-indent: 1.8em }   of this element
```

em=font size of this element or its parent
Origin of the term: the width of the letter "M"

Using "em" rather than absolute units (pt, mm, cm) ensures that font-related sizes and lengths are chosen relative to the user's base size; this causes web pages to respect the user's default size and allows users to scale up/down the text

More on em

font-size defines em relative to the parent's font size; other properties define em relative to this element's font size

Use em for almost everything and absolute units for almost nothing

Use abstolute units only when the absolute size of the medium is known (e.g., printing on A4 paper) or when an element has a rigid pixel size (images)

Units: Percentages

```
body { margin-left:  10%;
       margin-right: 10% }
```

Percentages of the width/height of the rectangle that is used to lay out the element (here the width of the browser window or the frame)

Selectors: By the Class Attribute

```
h1 { color: green }
```

a **type** selector

```
.question { color: green }
.answer   { color: red   }
```

class selectors

```
<p class="question">
  How much is a cent?
</p>

<p class="answer">
  A cent is 0.01 of one Dollar.
</p>
```

Selectors: By the ID Attribute

```
#para821 {
  text-decoration: line-through
}

<p class="question">
  Who is the US president?
</p>

<p class="answer"
   id="para821">
  The US president is Bill Clinton.
</p>
```

ID Selector as an Inline Style Attribute

```
<p class="answer"
   style="text-decoration:
              line-through ">
  The US president is Bill Clinton.
</p>
```

Exactly the same meaning as a rule with an ID selector

Usually not a good idea

Combining Selectors

```
p.obsolete { text-decoration:
                line-trough }
```

Applies of elements of type `p` and class `obsolete`

Additional examples later

Contextual Selectors

```
h3 strong { color: red }
```

Applies to elements of type `strong` that are inside elements of type h3

```
ul        { list-style: decimal     }
ul ul     { list-style: lower-roman }
ul ul ul  { list-style: lower-alpha }

p.question em { font-style: italic }
```

Selectors: Pseudo-Classes for Anchors, Input

```
a:link    { color: blue }
a:hover    { text-decoration: underline }
a:active  { color: red }
a:visited { color: black; text-... }

input:focus { outline: solid black }
```

Selectors: Pseudo-Elements

```
p:first-line {
  font-variant: small-caps }

p.largeinitial:first-letter {
  font-size: 3em }

p.answer:before {
  content:     "Question: ";
  font-weight: bold }

body:after {
  content: "© 2006 by Sivan Toledo" }
```

Advanced Attribute Selectors

```
[value]       { font-weight: italic }
input[value] { font-weight: italic }
```

Applies to elements (`input` elements) that have a `value` attribute (no matter what its value is)

```
input[value="Type your name here"] {...}
```

```
input[value~="yes"] {...}
```

Applies if the word `yes` appears (**space separation**)

The Language Pseudo-Attribute

```
body {
  font-family: Georgia, serif;
  direction:   ltr
}

:lang(iw) :lang(he) {
  font-family: Arial sans;
  direction:   rtl
}

[lang|="he"] { direction: rtl }
```

CSS2 only!

Not as comprehensive, but CSS1

Advanced Contextual Selectors

```
body > p.footnote { font-size: 0.8em }
```

Applies to p elements with class `footnote` whose parent is the body element

```
p + p { text-indent: 1.8em }
```

Applies to p elements that directly follow another p element (but not to a p that follow headings, etc)

```
p:first-child { font-weight: bold }
```

Font: Font Families

```
body { font-family:
          "Bistream Vera Sans Mono",
          "Courier New",
          "Computer Modern Typewriter",
          monospace }
```

Chooses the first family that the browser can find; monospace is a generic font type, not a real family name; **always finish lists with a generic name:**

serif sans-serif monospace *cursive* fantasy

Font Sizes

```
h1 { font-size: 2em     }      relative size
h1 { font-size: 200%    }  exactly the same
h1 { font-size: larger  }     relative, 120%
h6 { font-size: smaller }     relative, 1/1.2

12pt, 2mm, 8px     absolute, not recommended

xx-small, x-small, small, medium,
large, x-large, xx-large

      absolute, browser chooses, approx x1.2 factors
```

Font Variations

`font-style:` normal | *italic* | *oblique*

`font-variant:` normal | SMALL-CAPS

Font Weights

`font-weight:`			
`100`			
`200`			
`200`		like this	(light)
`300`			
`400`	`normal`	like this	(regular)
`500`		like this	(medium)
`600`		**like this**	(demi)
`700`	`bold`	**like this**	(bold)
`800`		**like this**	(xtra-bold)
`900`			

But most families have only 2 weights

Also relative specs: `lighter`, `bolder` (than parent)

Font Properties Shorthand & System Fonts

```
p { font: italic bold 1.2em monospace }
```

The shorthand can set the style, variant, weight, size, line-height (more later), and family

Specifying various system fonts:

```
font:   caption | icon | menu
      | message-box
      | small-caption
      | status bar
```

These set all the properties, but we can override

Wide and Narrow Fonts

```
font-stretch: normal | wider | narrower
              condensed | expanded | ...
```

Most fonts don't have width variations, but some do:

Some families have several widths

Some families have several widths

Some families have several widths

Some families have several widths

(The browser may be able to stretch/condense; this is usually not very aesthetic)

Adjusting the Size of Alternate Fonts

```
body {
  font:                10pt Verdana, serif
  font-size-adjust: 0.58
}
```

If Verdana is available, it will be used at 10 pt

Otherwise the size of the alternate font will be adjusted so that its x-height will be

0.58 x 10pt = 5.8pt

The value none overrides an inherited value

Text Decorations & Transformations

```
text-decoration: none
                 underline
                 overline
                 line-through
                 blink                    yuk!
```

This property is not inherited

```
text-transformation: capitalize
                     uppercase
                     lowercase
                     none
```

Display Modes

`display:`	`none`	*not displayed at all*
	`block`	*in a box (rectangle)*
	`inline`	*flows with the text*
	`list-item`	*self explanatory*
	`run-in`	*fancy heading style*
	`compact`	*fancy heading style*
	`marker`	*marker for a list item*
	`table`	
	`inline-table`	
	`table-...`	

Controls how an element is displayed

Simple Display Modes

none: not displayed at all (useful in scripts)

block: starts and ends on a new line, like `p`, `h1`, `div`

inline: flowed layout, like `em`, `strong`, `span`

list-item: in a box with a label, like `li`

Run-in Displays

```
h3         { display:      run-in;
             font-weight: bold }
h3:after { content: "." }
```

Background. By the 1950s, Roman Catholic theological and biblical studies had begun to move away from the neo-scholasticism and biblical literalism that the reaction to the Modernist heresy had enforced from after the First Vatican Council well into the 20th century.

Compact Displays (See Also Floats)

Second Vatican Council

The Second Ecumenical Council of the Vatican, or Vatican II, (Vatican two) was an Ecumenical Council of the Roman Catholic Church opened under Pope John XXIII in 1962 and closed under Pope Paul VI in 1965.

Lumen Gentium One of the principal documents

Controlling List Items: The Labels

`list-style-type:`
  `none` (suppresses the label, not the counter),
  `disc, circle, square,`
  `decimal, decimal-leading-zero,`
  `lower-roman, upper-roman,`
  `lower-alpha, lower-latin,`
  `upper-alpha, upper-latin,`
  `hebrew,`
  `(various other alphabets)`

```
ul { list-style-image: url("star.gif") }
```

(none is also a possible value)

Controlling List Items: Label Position

`list-style-position: inside | outside`

- The Second Ecumenical Council of the Vatican, or Vatican II, (Vatican two) was an Ecumenical Council of the Roman Catholic Church opened ...

- The Second Ecumenical Council of the Vatican, or Vatican II, (Vatican two) was an Ecumenical Council of the Roman Catholic Church opened ...

Controlling List Items: A Shorthand

```
ul > li {
  list-style: circle inside
}
ul ul > li {
  list-style: url("star.gif") outside
}
```

Counters

```
h2:before {
  content: "Chapter " counter(ch) ": ";
  counter-reset:     sec;
  counter-increment: ch }

h3:before {
  content: counter(ch) "." counter(sec);
  counter-increment: sec }
```

Chapter 2: The Vikings in the Mediterranean

1.1 How Did They Get There?

1.2 Trade and Conflict

Non-Arabic Counters

```
counter(appendix, upper-alpha)
counter(ch,       upper-roman)
```

Counters for List Markers

```
ol { counter-reset: itemnr }
li:before {
  display: marker;
  content: counter(itemnr, ".");
  counter-increment: itemnr;
}
```

1 first item in first list

  1.1 first item in inner list *nested use of itemnr*

  1.2 second item in inner list

2 second item in outer list

Spacing for Markers

```
li:before {
  display: marker;
  content: counter(itemnr, ".");
  counter-increment: itemnr;

  marker-offset: 1em;          distance to item
  width:         3em;        width of marker box
  text-align:    left;
}
```

Controlling the Meaning of White-Space Characters

Normally, white space characters, including tabs and new-lines, are ignored:

- new-lines are ignored; text wraps at end of line
- multiple consecutive white-space characters are collapsed into one
- white space at the beginning and at the end of paragraphs is ignored

```
white-space: normal | pre | nowrap
```

Controlling the Flow of Text: Alignment and Indentation

```
text-align: left
            right
            center
            justify
            (a string; for tables; later)

text-indent: 2em
```

(can indent by a percentage of paragraph width; usually a bad idea)

Interline Spacing

```
p { line-height: 1.5 }
```

Lines are spaced at a minimum of 150% of the font size; e.g., if em=10pt then lines are at least 15pt apart (could be more if lines contain tall things)

Inherited elements will have line height of 150% of **their** font size

```
line-heigt: 1.5em | 150%
```

inherited as a fixed length

Interword Spacing and Letter Spacing

```
word-spacing: normal | 0.2em
```

Usually best to leave this alone

```
letter-spacing: normal | 0.1em
```

A value of 0 is useful (instructs the browser not to compress/expand spacing); also useful with all caps:

```
h2 { text-transform: uppercase;
     letter-spacing: 0.33em }
```

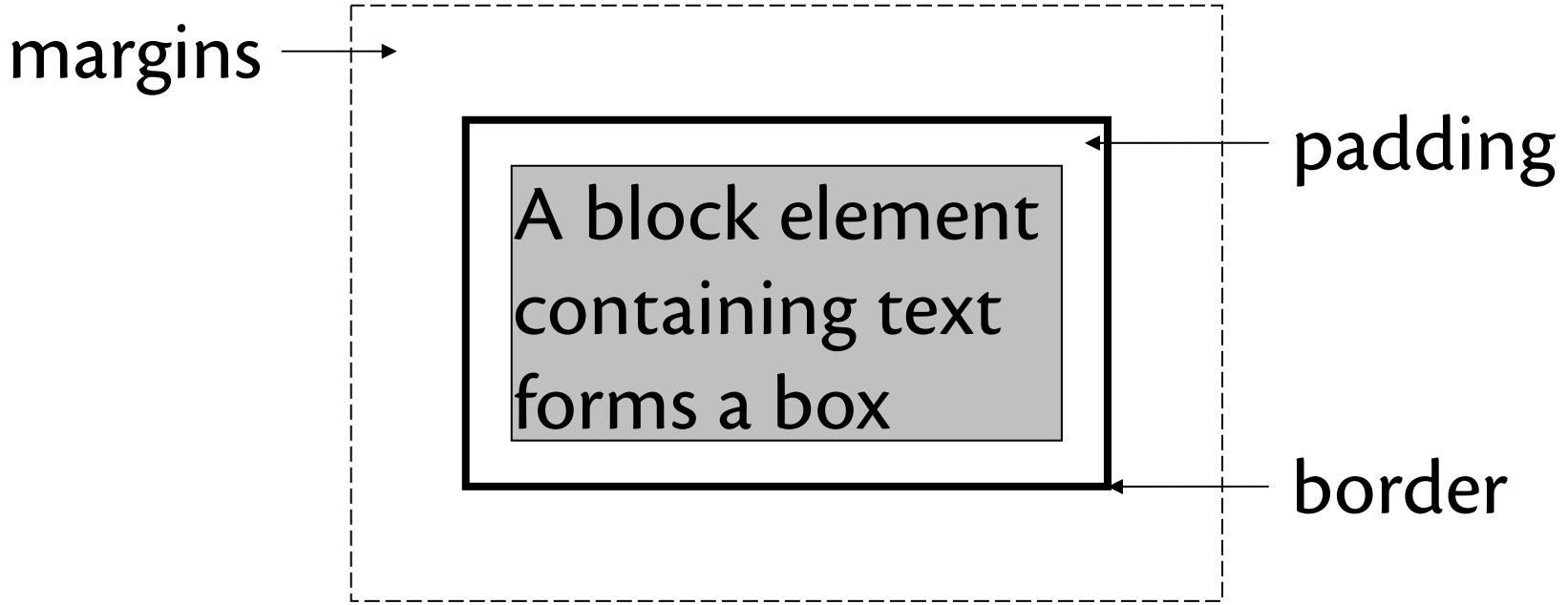
Vertical Alignment

```
vertical-align: baseline                    default
                sub | super
                middle                vertical centering
                text-top | text-bottom
                top | bottom                confusing
                50% | -75% | ...
                1ex | ...
```

A length raises or lowers the baseline by that amount; a percentage by a fraction of the line height

The Space Around Boxes

The browser lays out a document by putting boxes inside boxes (and by flowing text inside boxes)



Margins are always transpararent

Margins

```
margin-top:    1em; margin-right:  2em;
margin-bottom: 1em; margin-left:   2em;
```

or

```
margin: 1em 2em 1em 2em;
```

(starts at top, clockwise)
(auto-completion if fewer than 4 values)

`auto` (later), `10%` (% width of enclosing element)

Collapsing Margins

Touching top and bottom margins of boxes that are on top of one another collapse: the border separation is the maximal margins, not the sum

Left/right margins do not collapse

Margins do not collapse if something (nonzero padding, border) separates them; Consider a list-item inside a list, followed by a paragraph; if the list has padding/border, the margins will not collapse

Padding

```
padding-top, padding-right, ...
padding: 1em
```

Same rules as the margins properties

The padding is part of the box (unlike the margins which are around the box); the background of the box is painted on the padding area, but not on the margins
But the background of the root **is** extended into its margins (because it has no parent)

Borders and Border Properties

Borders have 3 properties and 4 sides:

```
border-left-color, border-top-color, ...
border-left-style, ...
border-left-width, ...
```

many shorthand properties:

```
border
border-left, border-top, ...
border-color, border-style,
              border-width
```

Border Styles and Widths

`none`	*default*
`hidden`	*space reservation but nothing is displayed*
`solid, dotted, dashed`	*line styles*
`double`	*parallel lines that fill the given width*
`groove, ridge`	*3D effect around*
`inset, outset`	*3D on entire area*
`thin, medium, thick`	*predetermined widths (document consistency)*
`0.1em, 2px, 1mm`	

Outlines Instead of Borders

An outline is a border for which no space is reserved

Useful for flashing as part of the user interface (when the mouse hovers, for example, or to indicate focus)

Three properties and one shorthand:
`outline-color, outline-style, outline-width, outline`

Box-Size Control

Normally, the width of boxes is determined by the width of the containing box minus padding, border, and margins; height by the amount of material

We can set the width and/or height explicitly using the `width` and `height` properties (length, percentage, or `auto` to return to normal); usually not a good idea

But placing some constraints can be a good idea

Constraining Box Sizes

`min-width`
`max-width`
`min-height`
`max-height` *(height constrainsts are rarely useful)*

What to do when a box overflows (extends beyond its containing box)?

`overflow:`	`visible`	*default, may stick out*
	`hidden`	*hide the part that sticks out*
	`scroll`	*always use a scrollbar*
	`auto`	*a scrollbar, only if necessary*

Floating Elements

```
float: left | right | none
```

Causes the element to be displayed in block mode; it will float to the left or the right until it hits the margin/border/padding of another block element

Margins of floating elements don't collapse

```
h3 { float: left; width: 5em }
p  { margin-left: 6em }
```

Clear: Don't Float Near Me

```
clear: none | left | right | both
```

Prevents a floating element from attaching to the cleared side; instead, the floating elements will move down until they have enough space

```
img {
      float: left;
      clear: left
}

h2  { clear: both }
```

Positioning Boxes

`position:`	`static`	*default*
	`absolute`	*implies display: block*
	`fixed`	*implies display: block*
	`relative`	*default*

When the positioning rule is not static, the actual position is determined by

`top: <length> | <percentage> | auto`

and by `right, left, bottom`

Fixed Positioning

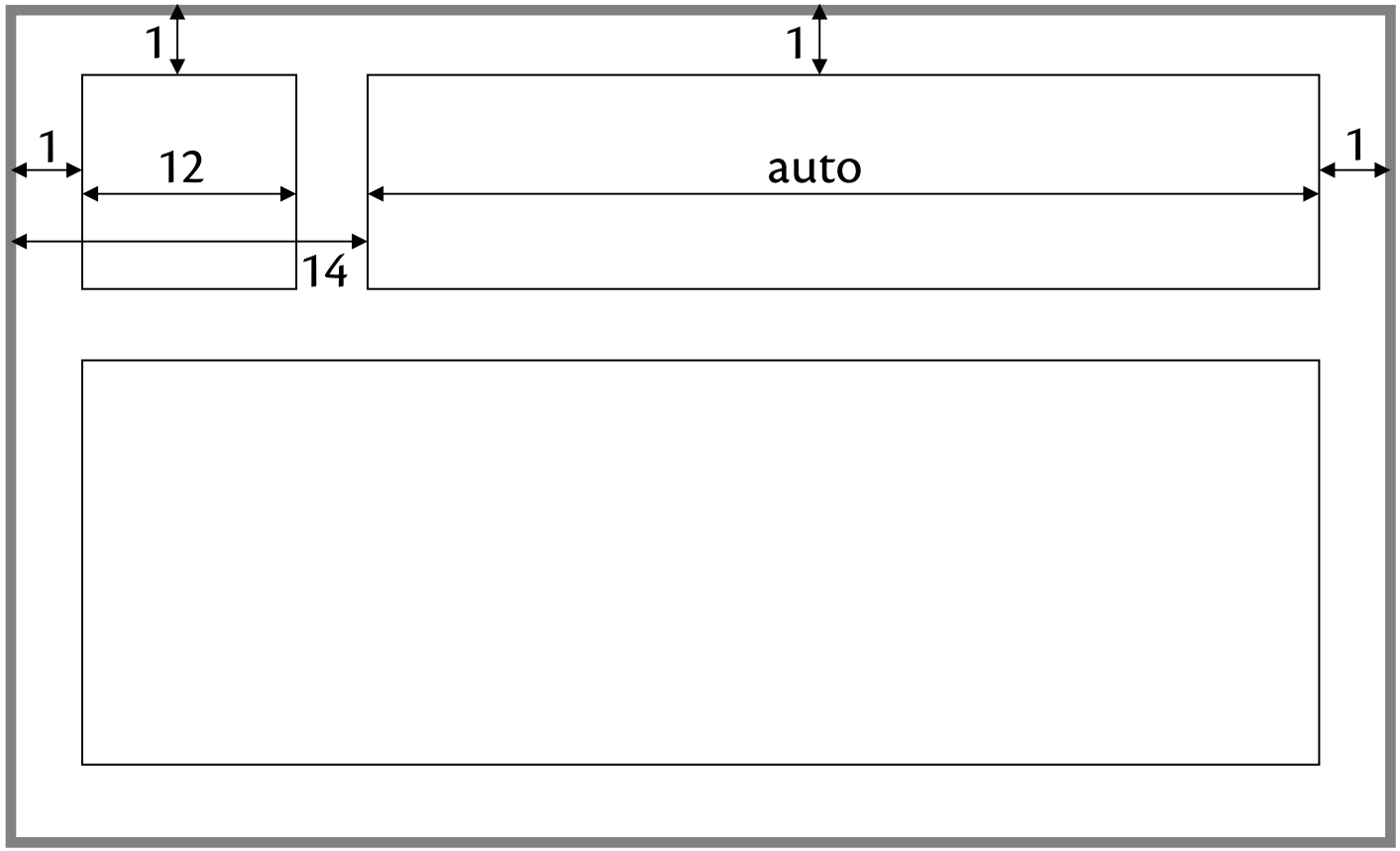
A block that is displayed at a fixed position relative to the viewport (the browser's window or the printed page)

```
div.lefttopbox {
  position: fixed;
  top: 1em;      down from the top of the window
  left: 1em;     from the left end of the window
  width: 12em; height: 12em;
  ...
}
```

Fixed Positioning (Continued)

```
div.righttopbox {
  position: fixed;
  top: 1em;
  right: 1em; left: 14em;
  width: auto; height: 12em;
  ... }

div.mainbox {
  position: fixed;
  top: 14em: height: auto;
  left: 1em; right: 1em;
  ... }
```

Fixed Positioing: the End Result



(This is why we do not need frames with CSS)

Absolute Positioning

Similar to fixed, but with two differences

- The positioning specification is relative to the containing block: the nearest ancestor in the document tree that uses a non-static positioning rule
- Special meaning for the value `auto` for the properties `top`, `bottom`, `left`, `right`: means "approximately where the block would have been statically positioned"

Example of Absolute Positioning

```
When we think of Beethoven<span
class="marginnote>1770-1827</span>, we
think of ...

span.mntext {
  position: absolute;
  top: auto;
  right: 1em;
  width: 12em;
}
```

We assume that the marginal note will fit into the right margin

Relative Positioning

The positioning of the element is shifted up/down and left/right relative to where it would have been positioned; applied all kinds of displays, not just block

```
span.superscript { top: -0.5em;
                   font-size: 0.75em }
```

Moves a span of text up and shrinks it a bit[1].

[1] This does not affect the interline spacing, unlike the vertical-align property

Overlaps and Z-Axis Ordering

When we specify positioning, the browser allows elements to overlap (this makes `float` better for marginal notes and images, for example)

We can control which element is "closer to the viewer" by giving elements a `z-order`; higher values are closer to the viewer

Example: Relative, Absolute, Z-Order

```
When we think of
<span class="imgtext">
<img src="beethoven.jpg" />
Beethoven</span>, we think of his ...

span.imgtext {
  position: relative;
  z-order: 2 }
span.imgtext > img {
  position: absolute;
  top: -1em; left: -1em
  z-order: 1 }
```

When we think of  Beethoven, we think of his symphonies, but also ...

Visibility

```
visibility: hidden
            visible   default
```

Setting visibility to hidden hides the element, but space is still reserved for it

May when we want to toggle visibility but without changing the layout of the document (e.g., when revealing the answer of a question in an on-line exam)

Positioning Summary

- May cause overlaps; therefore, not very useful
- Mainly useful for partitioning the document into several distinct areas without using frames
- Can also be used for various visual effects and for scripted animations (e.g., elements that gradually move into their final position)

Colors

Predefined: black, white, red, green, yellow, plue, purple, silver, gray, maroon, fuchsia, lime, olive, navy, teal, aqua; **don't use other named colors**

RGB colors:

`rgb(100%, 50%, 50%)`	*good notation*
`rgb(255, 128, 128)`	*0-255, not as good*
`#FF8080`	*same value, hexadecimal*
`#A83`	*yuk! same as #AA8833*

System colors: Background, Menu, MenuText, ...

Background Images

```
body {
  background-image: url(mylogo.gif);
  background-color:   rgb(80%, 80%, 80%);
}
```

The image is displayed on top of the colored background; if the image is partially transparent, the color will show underneath; the color will be shown alone if the image cannot be loaded/shown

Background-Image Repetition

`background-repeat:`

`repeat`	*default*
`repeat-x`	
`repeat-y`	
`no-repeat`	

Background-Image Positioning

```
background-attachment:
                scroll      default
                fixed       may be useful for body

background-position:
                10% 20%     x-y values
                10%         same for x and y
                1em 1em
                2em
                top left
                center center
                ...
```

Tables!

Before we begin: HTML tables can be used to achieve many layout effects; don't do it; use CSS positioned element and floating elements instead

Two layout modes for the table element itself:

`table`	*default, a block element*
`inline-table`	*a box that flows with the text*

Table Layout Algorithms

`table-layout:`	`auto`	*default; clever but slow*
	`fixed`	*a faster algorithm*

`fixed` is used only if `width` is not `auto` (even though the table's `width` is often ignored):

- If a column element has a width or the cell in the first row has a width, that width is used
- If all columns have width (under the definition above), that determines the table's width
- Otherwise, undetermined columns have equal widths, the rest of the table's width

Collapsing a Column

Setting the `visibility` of a column to `collapse` includes the column in column-width calculations but does not show it

This allows scripts to quickly hide and show columns: the browser does not need to recomputed the column widths, only to shift columns to the left or right

The Border Model of a Table

`border-collapse: collapse` *default*
`separate`

In the collapsing model, at most one border separates adjacent cells

Each cell, row, and column, row group, column group, and the entire table may have a border

The border that is actually shown next to each cell is the strongest of those that apply to that location

Border Strengths

thick lines > think lines
double lines > single lines
solid lines > dashed lines > dotted lines
3D style > 2D style

If borders differ only in color, then
cell > row > row group > col > col group > table

Separated Borders

Mainly useful for 3D borders (`inset` and `outset`)

```
table {
  border-collapse: separate ;
  border-spacing:  2px      ;
  empty-cells:     show;  default, or "hide"
  ... }
```

`hide` in `empty-cells` will not draw their border

The Z-Ordering of Table Elements

What is the background of a cell inside a row and a column, inside a row group and a column group, if all of these (and the table) have background?

The background of table elements is determined by a fixed z-ordering of the elements

cell (on top)

> row > row group

> column > column group

> table

Aligning Text in Cells

`text-align:`	`right`	*as in any block element*
	`left`	*as in any block element*
	`center`	*as in any block element*
	`justify`	*as in any block element*
	`<string>`	*only in tables*

Allows aligning on a decimal point, or on ":" (as in 23:59, for time tables); only used in single-line cells; cells without the special string are aligned to the left of it (think of whole numbers)

Vertical Alignment of Table Cells

`vertical-align:`

- `baseline` — *default: a single baseline for cells in multiple columns*
- `top`
- `bottom`
- `middle`

Table Miscellanea

Cells don't have margins, but they do have padding

```
caption-side: top     default
              bottom
              left
              right
```

Display Modes for XML Tables

In XML, elements that need to form table may have different names:

```
table    table
         inline-table
tr       table-row
col      table-column
td, th   table-cell
caption  table-caption
tbody    table-row-group
thead    table-header-group
tfoot    table-footer-group
col      table-column-group
```

Media-Specific Style Sheets and Printing

```
@media screen {
  body { font-family: Georgia, serif;
         }
}

@media print {
  body { font-family: Times, serif;
           font-size:   11pt; }
}

h1 { font-size: 2em }
...
```

Media Types

(CSS1 did not support media types at all)

`screen`	
`print`	
`aural`	
`braille`	*Braille on an electronic display*
`embossed`	*Braille on paper*
`handheld`	
`projection`	
`tty`	*electronic plain-text displays*
`tv`	*televisions*
`all`	*rules for all devices*

Print-Specific Properties: Page Breaks

`page-break-before:`
- `auto`
- `always`
- `avoid`
- `left` *always, and jump to a left page if necessary*
- `right` *always, and jump to a right page*

`page-break-after:` *same values*

`page-break-inside:`
- `avoid | auto` *(default)* `t`

Print-Specific Properties: Widows & Orphans

`widows: <integer>`

`orphans: <integer>`

Default for both is 2; this means that at least two lines from the end of the paragraph will stay together

and audiences.

Born in Bonn, Germany, he moved to Vienna, Austria, in his early twenties, and settled there, studying with Joseph Haydn and quickly gaining a reputation as a virtuoso pianist.... Unusually among his contemporaries, he worked as a freelance composer, arranging subscription concerts and being supported by a number of wealthy patrons who considered his gifts extraordinary.

Among his most widely-

Print-Specific Properties: Specifying Pages

```
@page { margin: 10%;    percentage of width
        size:   portrait; }
```

Margins in % or absolute units (not em), size in absolute units (usually not useful) or `auto`, `landscape`, or `portrait`

Pseudo-classes:

```
@page :left
@page :right
@page :first
```

Print-Specific Properties: Named Pages

```
@page rotated { size: landscape }

table.verywide { page: rotated }
```

Named pages can have arbitrary names

External Style Sheets: Linking

```
<html>
  <head>
  ...
  <link rel="stylesheet"
        type="text/css"
        href="http://www.w3.org/..." />
  <style type="text/css">
    h1 { color: red } overrides the link's rules
  </style>
```

The `media` attribute of the link can specify the media

External Style Sheets: @import

```
@import "tau-cs.css"
h1 { color: red }
```

overrides rules in the imported style sheets

Importing works similarly to linking

It can be used to created multiple styles that are linked into documents, but are all based on a single master style-sheets with small modifications

Cascading

There can be several rules that apply to a single element: from the document's style sheet, from the browser's, from the user's style sheet; possibly even several rules from the same style sheet

CSS always selects one of the rules and applies it

The selection process is called cascading

Cascading: Assigning Importance

```
body {
  color:              black !important ;
  background-color: white !important ;
}
```

`!imporant` rules override all others

document style sheets > user style sheets

specific rules > general rules

sort the rest by the order given

Inheritance

If no rule applies, the value for the property is usually inherited from the parent in the document

But if there is no rule in the document sheet, for example, the value may not necessarily be inherited from the parent rule in the same sheet: the user may specify a specific rule!

To avoid that, all properties can accept the `inherit` value; this forces inheritance