INTERNET SYSTEMS

CASCADING STYLE SHEETS

TOMASZ PAWLAK, PHD
INSTITUTE OF COMPUTING SCIENCE, POZNAN UNIVERSITY OF TECHNOLOGY

PRESENTATION OUTLINE

- Motivation and history
- CSS basics
- CSS details

MOTIVATION AND HISTORY

MOTIVATION

- In the original HTML (1.0/2.0) the content was mixed in a single document with:
  - Structural (but non-semantic) information
  - Definition of presentation (style)
    - E.g., colors, fonts, text-sizes

MOTIVATION

- In the original HTML (1.0/2.0) the content was mixed in a single document with:
  - Structural (but non-semantic) information
  - Definition of presentation (style)
    - E.g., colors, fonts, text-sizes
- Similarly formatted parts of the webpage have similar style definitions
  - Low maintainability:
    - A simple change in the style must be manually propagated to all usage places
  - Redundancy:
    - The same code fragments are copied multiple times across HTML document
    - Redundant code significantly increases size of webpages → longer page load times
  - Browser performance:
    - Style calculation and drawing must be done for each HTML element separately
  - Interoperability
    - It is almost impossible to prepare a single page that adapts to different devices
    - No support for dynamic style changes (e.g., high contrast for accessibility, skins…)

CASCADING STYLE SHEETS (CSS)

- A language for styling HTML documents
  - Format independent from HTML
    - Applicable to XML, SVG, XUL
- CSS style definitions can be shared
  - By multiple HTML elements
  - By multiple HTML documents
    - E.g., in separate *.css files
- Support for multiple output devices
  - Screen
    - Including adjustments for different screen sizes
  - Printer
  - Speech (screen readers)

CASCADING STYLE SHEETS FEATURES

- Separation of concerns
  - Presentation and content
- Rule-based declarative language
- Inheritance and overriding
  - Hence: “Cascading” in name
- Styling support
  - Layout
  - Colors
  - Fonts
  - Gradients
  - Graphical transformations
  - Animations
  - Generated content
  - Web fonts

HISTORY

- 1994: Håkon Wium Lie (co-worker of Tim Berners-Lee) proposed CSS
- 1996: W3C published first CSS Recommendation (CSS level 1)
  - Font properties, colors, alignment, margins, borders, paddings, unique ids,…
- 1996: Many poorly standardized alternatives in the market
  - E.g., JavaScript Style Sheets (uses JS code to set styling information)
- 1996/97: Internet Explorer 3 and Netscape 4 have limited support for CSS
- 03.2000: The first browser with full CSS1 support:

HISTORY

- 1994: Håkon Wium Lie (co-worker of Tim Berners-Lee) proposed CSS
- 1996: W3C published first CSS Recommendation (CSS level 1)
  - Font properties, colors, alignment, margins, borders, paddings, unique ids,…
- 1996: Many poorly standardized alternatives in the market
  - E.g., JavaScript Style Sheets (uses JS code to set styling information)
- 1996/97: Internet Explorer 3 and Netscape 4 have limited support for CSS
- 03.2000: The first browser with full CSS1 support:
  - Internet Explorer 5 for Macintosh*

* source: https://www.w3.org/Style/CSS/software.en.html#w26

HISTORY – TOWARDS MODERN CSS

- CSS1 has many issues and was imprecise – hence difficult to consistently implement in browsers
- 1997: W3C Recommendation for CSS level 2
- 1998: W3C began work on CSS level 3
- 1999: W3C published CSS3 Working Draft
- 2004: W3C Candidate Recommendation for CSS level 2 revision 1
  - Original CSS2 Recommendation was reverted to Candidate Recommendation

HISTORY – TOWARDS MODERN CSS

- CSS1 has many issues and was imprecise – hence difficult to consistently implement in browsers
- 1997: W3C Recommendation for CSS level 2
- 1998: W3C began work on CSS level 3
- 1999: W3C published CSS3 Working Draft
- 2004: W3C Candidate Recommendation for CSS level 2 revision 1
  - Original CSS2 Recommendation was reverted to Candidate Recommendation
- 2005: CSS2.1 reverted to Working Draft

HISTORY – TOWARDS MODERN CSS

- CSS1 has many issues and was imprecise – hence difficult to consistently implement in browsers
- 1997: W3C Recommendation for CSS level 2
- 1998: W3C began work on CSS level 3
- 1999: W3C published CSS3 Working Draft
- 2004: W3C Candidate Recommendation for CSS level 2 revision 1
  - Original CSS2 Recommendation was reverted to Candidate Recommendation
- 2005: CSS2.1 reverted to Working Draft
- 2007: CSS2.1 published again as Candidate Recommendation

HISTORY – TOWARDS MODERN CSS

- CSS1 has many issues and was imprecise – hence difficult to consistently implement in browsers
- 1997: W3C Recommendation for CSS level 2
- 1998: W3C began work on CSS level 3
- 1999: W3C published CSS3 Working Draft
- 2004: W3C Candidate Recommendation for CSS level 2 revision 1
  - Original CSS2 Recommendation was reverted to Candidate Recommendation
- 2005: CSS2.1 reverted to Working Draft
- 2007: CSS2.1 published again as Candidate Recommendation
- 2009: CSS2.1 updated, still in Candidate Recommendation

HISTORY – TOWARDS MODERN CSS

- CSS1 has many issues and was imprecise – hence difficult to consistently implement in browsers
- 1997: W3C Recommendation for CSS level 2
- 1998: W3C began work on CSS level 3
- 1999: W3C published CSS3 Working Draft
- 2004: W3C Candidate Recommendation for CSS level 2 revision 1
  - Original CSS2 Recommendation was reverted to Candidate Recommendation
- 2005: CSS2.1 reverted to Working Draft
- 2007: CSS2.1 published again as Candidate Recommendation
- 2009: CSS2.1 updated, still in Candidate Recommendation
- 2010: CSS2.1 reverted to Working Draft

HISTORY – TOWARDS MODERN CSS

- CSS1 has many issues and was imprecise – hence difficult to consistently implement in browsers
- 1997: W3C Recommendation for CSS level 2
- 1998: W3C began work on CSS level 3
- 1999: W3C published CSS3 Working Draft
- 2004: W3C Candidate Recommendation for CSS level 2 revision 1
  - Original CSS2 Recommendation was reverted to Candidate Recommendation
- 2005: CSS2.1 reverted to Working Draft
- 2007: CSS2.1 published again as Candidate Recommendation
- 2009: CSS2.1 updated, still in Candidate Recommendation
- 2010: CSS2.1 reverted to Working Draft
- 2011: CSS2.1 published as Proposed Recommendation, then as Recommendation

HISTORY – TOWARDS MODERN CSS

- CSS1 has many issues and was imprecise – hence difficult to consistently implement in browsers
- 1997: W3C Recommendation for CSS level 2
- 1998: W3C began work on CSS level 3
- 1999: W3C published CSS3 Working Draft
- 2004: W3C Candidate Recommendation for CSS level 2 revision 1
  - Original CSS2 Recommendation was reverted to Candidate Recommendation
- 2005: CSS2.1 reverted to Working Draft
- 2007: CSS2.1 published again as Candidate Recommendation
- 2009: CSS2.1 updated, still in Candidate Recommendation
- 2010: CSS2.1 reverted to Working Draft
- 2011: CSS2.1 published as Proposed Recommendation, then as Recommendation
- 2011-2015: Several erratum published

HISTORY – TOWARDS MODERN CSS

- CSS1 has many issues and was imprecise – hence difficult to consistently implement in browsers
- 1997: W3C Recommendation for CSS level 2
- 1998: W3C began work on CSS level 3
- 1999: W3C published CSS3 Working Draft
- 2004: W3C Candidate Recommendation for CSS level 2 revision 1
  - Original CSS2 Recommendation was reverted to Candidate Recommendation
- 2005: CSS2.1 reverted to Working Draft
- 2007: CSS2.1 published again as Candidate Recommendation
- 2009: CSS2.1 updated, still in Candidate Recommendation
- 2010: CSS2.1 reverted to Working Draft
- 2011: CSS2.1 published as Proposed Recommendation, then as Recommendation
- 2011-2015: Several erratum published
- 12.04.2016: CSS2 revision 2 published as Working Draft

HISTORY – TOWARDS MODERN CSS

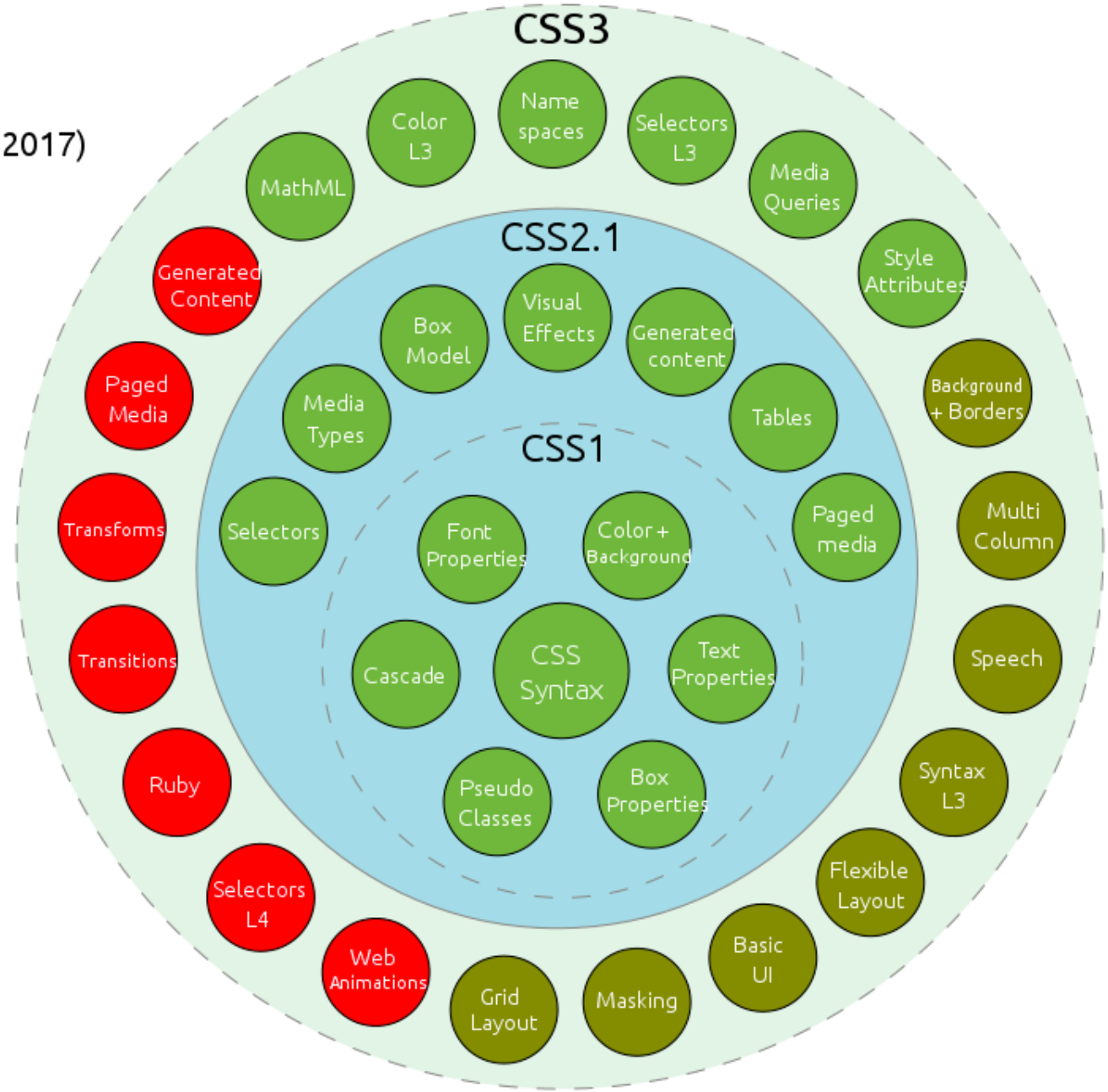
- The main difficulties in standardizing CSS2 come from organization of standard: this one big document, but rarely changes were made for more than one section
- CSS level 3
  - A large collection of documents called "modules"
  - Each module extends features defined in CSS2
    - Over 130 modules are defined*
- CSS level 4 follows the modular model
  - Each module extends features defined in the level 3 modules
  - New modules that do not extend base ones are assigned with level 1
    - E.g., flexbox

* Source: https://www.w3.org/Style/CSS/specs

CSS3

Taxonomy & Status (September 2017)

- W3C Recommendation
- Candidate Recommendation
- Last Call
- Working Draft
- Obsolete or inactive



Source: https://commons.wikimedia.org/wiki/File:CSS3_taxonomy_and_status_by_Sergey_Mavrody.svg

BROWSER SUPPORT

- All major browsers support currently:
  - CSS2.1
  - Some CSS3 and CSS4 specifications
- Details:
  - https://caniuse.com/#compare=edge+81,firefox+75,chrome+81,safari+13.1,opera+68,ios_saf+13.4,android+81,and_chr+81&compare_cats=CSS

CSS BASICS

A SIMPLE CSS DOCUMENT

- The CSS document consists of the CSS rules
- Each rule consists of
  - Selector(s) – the conditional part
    - Multiple selectors are treated as alternatives
  - List of pairs of property and its value

```
selector1 {
    /* comment */
    property1: value;
    property2: value;
}

selector2, selector3 {
    property3: value;
    property4: value;
}
```

CSS SELECTORS

- A Selector points at an HTML element to which the rule applies the style
  - Technically, the selector points at a particular DOM element
  - Selectors may be recurrent
- CSS Selectors Level 3 is the latest W3C Candidate Recommendation
  - Released as W3C Recommendation in 2011
  - Reverted to W3C Candidate Recommendation in 2018
  - Published again as W3C Recommendation in 2018
- CSS Selectors Level 4 is the latest W3C Working Draft (as of April 2020)

BASIC CSS SELECTORS

Selector	Matches	CSS Level
`E`	an element of type E	1
`E:link`	an E element being the source anchor of a hyperlink of which the target is not yet visited (:link) or already visited (:visited)	1
`E:active`	an E element during certain user actions	1
`E::first-line`	the first formatted line of an E element	1
`E::first-letter`	the first formatted letter of an E element	1
`.myclass`	all elements with class="myclass"	1
`#myid`	the element with id="myid"	1
`E.warning`	an E element whose class is "warning" (the document language specifies how class is determined)	1
`E#myid`	an E element with ID equal to "myid"	1
`E F`	an F element descendant of an E element	1
`*`	any element	2
`E[foo]`	an E element with a "foo" attribute	2
`E[foo="bar"]`	an E element whose "foo" attribute value is exactly equal to "bar"	2
`E[foo~="bar"]`	an E element whose "foo" attribute value is a list of whitespace-separated values, one of which is exactly equal to "bar"	2
`E[foo|="en"]`	an E element whose "foo" attribute has a hyphen-separated list of values beginning (from the left) with "en"	2
`E:first-child`	an E element, first child of its parent	2
`E:lang(fr)`	an element of type E in language "fr" (the document language specifies how language is determined)	2
`E::before`	generated content before an E element's content	2
`E::after`	generated content after an E element's content	2
`E > F`	an F element child of an E element	2
`E + F`	an F element immediately preceded by an E element	2

BASIC CSS SELECTORS

Selector	Matches	CSS Level
`E`	an element of type E	1
`E:link`	an E element being the source anchor of a hyperlink of which the target is not yet visited (:link) or already visited (:visited)	1
`E:active`	an E element during certain user actions	1
`E::first-line`	the first formatted line of an E element	1
`E::first-letter`	the first formatted letter of an E element	1
`.myclass`	all elements with class="myclass"	1
`#myid`	the element with id="myid"	1
`E.warning`	an E element whose class is "warning" (the document language specifies how class is determined)	1
`E#myid`	an E element with ID equal to "myid"	1
`E F`	an F element descendant of an E element	1
`*`	any element	2
`E[foo]`	an E element with a "foo" attribute	2
`E[foo="bar"]`	an E element whose "foo" attribute value is exactly equal to "bar"	2
`E[foo~="bar"]`	an E element whose "foo" attribute value is a list of whitespace-separated values, one of which is exactly equal to "bar"	2
`E[foo|="en"]`	an E element whose "foo" attribute has a hyphen-separated list of values beginning (from the left) with "en"	2
`E:first-child`	an E element, first child of its parent	2
`E:lang(fr)`	an element of type E in language "fr" (the document language specifies how language is determined)	2
`E::before`	generated content before an E element's content	2
`E::after`	generated content after an E element's content	2
`E > F`	an F element child of an E element	2
`E + F`	an F element immediately preceded by an E element	2

Each time an element E is referenced, it is meant as an element selected by selector E (recurrence happens here)

BASIC CSS PROPERTIES: FONTS

```
/* A comma separated list of fonts. A browser will use the first available font from the list. */
font-family: Arial, Helvetica, sans-serif;
/* Generic family names - defined in settings of a browser */
font-family: serif, sans-serif, monospace, cursive, fantasy;

/* A set of absolute size keywords based on the current default font size (which is medium).
    xx-small, x-small, small, medium, large, x-large, xx-large */
font-size: large;
/* Larger or smaller than the font size of the parent element, by roughly the ratio used to separate the
absolute size keywords above. */
font-size: larger;
font-size: smaller;
/* <length> values */
font-size: 12px;
font-size: 0.8em;
/* <percentage> values */
font-size: 80%;

/* Font-styles */
font-style: normal;
font-style: italic;
font-style: oblique;

/* Font-weights */
font-weight: normal;
font-weight: bold;
/* Weight relative to the parent */
font-weight: lighter;
font-weight: bolder;
/* Fine-grained weight 100-900 by 100, normal is 500 */
font-weight: 500;
```

FONTS – GOOD PRACTICES

- Do not use proprietary fonts
  - Your clients may not have them
  - http://www.cssfontstack.com/
- Use the fonts with the symbols in the language of your site
  - Use universal (Unicode) fonts for multilingual sites
- Be consistent – use at most three different fonts per site

FONTS – GOOD PRACTICES

- Do not assume any particular size of the rendered text
  - The size of the rendered text depends on many conditions:
    - Browser text rendering implementation
    - Graphic card and drivers
      - Text may render differently even on two screens attached to the same computer
    - Operating system and version
  - A pixel-based size is not accessible
    - User cannot increase the size of text
    - The text may render very small on high-density screens (e.g., smartphones)
- em and rem are recommended units for setting font size
  - em specifies font size relatively to the font size of the parent element or the default size unless set
  - rem specifies font size relatively to the font size of the root element or the default size unless set
  - Calculate em/rem value by dividing the desired size in pixels by the parent/root font size in pixels
- The font size should reflect the importance of the text in the page

BASIC CSS PROPERTIES: COLORS

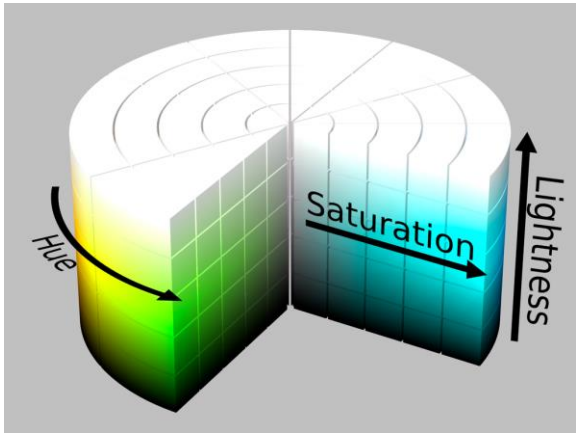
```
/* A CSS Level 1 color
   16 colors taken from the Windows VGA palette without
   defined RGB values in CSS1 specification. */
color: red;

/* The only color added in CSS Level 2 (Revision 1)
   In CSS2.1 each color has assigned RGB values */
color: orange;

/* CSS Level 3 color, sometimes called a SVG or X11 color */
color: antiquewhite;

/* The color 'lime' with the 3-character dash notation;
    each character refers to one channel in order: R, G, B;
    duplicate each symbol to get equivalent 6-character notation */
color: #0f0;

/* The color 'lime' with the 6-character dash notation */
color: #00ff00;

/* A color using the available functional notations */
color: rgb(34, 12, 64);
color: rgba(34, 12, 64, 0.3);
color: hsl(40, 50%, 50%);
color: hsla(40, 50%, 50%, 0.4);

/* Use the color of the direct ancestor of this element */
color: inherit;

/* Use the value of the ‘color’ property in an other property */
-other-color-property: currentcolor;
```

BASIC CSS PROPERTIES: COLORS

```
/* A CSS Level 1 color
   16 colors taken from the Windows VGA palette without
   defined RGB values in CSS1 specification. */
color: red;

/* The only color added in CSS Level 2 (Revision 1)
   In CSS2.1 each color has assigned RGB values */
color: orange;

/* CSS Level 3 color, sometimes called a SVG or X11 color */
color: antiquewhite;

/* The color 'lime' with the 3-character dash notation;
    each character refers to one channel in order: R, G, B;
    duplicate each symbol to get equivalent 6-character notation */
color: #0f0;

/* The color 'lime' with the 6-character dash notation */
color: #00ff00;

/* A color using the available functional notations */
color: rgb(34, 12, 64);
color: rgba(34, 12, 64, 0.3);
color: hsl(40, 50%, 50%);
color: hsla(40, 50%, 50%, 0.4);

/* Use the color of the direct ancestor of this element */
color: inherit;

/* Use the value of the ‘color’ property in an other property */
-other-color-property: currentcolor;
```

BASIC CSS PROPERTIES: COLORS

```
/* A CSS Level 1 color
   16 colors taken from the Windows VGA palette without
   defined RGB values in CSS1 specification. */
color: red;

/* The only color added in CSS Level 2 (Revision 1)
   In CSS2.1 each color has assigned RGB values */
color: orange;

/* CSS Level 3 color, sometimes called a SVG or X11 color */
color: antiquewhite;

/* The color 'lime' with the 3-character dash notation;
    each character refers to one channel in order: R, G, B;
    duplicate each symbol to get equivalent 6-character notation */
color: #0f0;

/* The color 'lime' with the 6-character dash notation */
color: #00ff00;

/* A color using the available functional notations */
color: rgb(34, 12, 64);
color: rgba(34, 12, 64, 0.3);
color: hsl(40, 50%, 50%);
color: hsla(40, 50%, 50%, 0.4);

/* Use the color of the direct ancestor of this element */
color: inherit;

/* Use the value of the ‘color’ property in an other property */
-other-color-property: currentcolor;
```



https://commons.wikimedia.org/wiki/File:HSL_color_solid_cylinder_saturation_gray.png

BASIC CSS PROPERTIES: BACKGROUNDS

```
/* Hexadecimal value - the same values as for the "color" property are allowed */
background-color: #bbff00;
/* Background image */
background-image: none;
background-image: url(http://www.example.com/bck1.png);
/* Multiple values, successive backgrounds are stacked from top to bottom */
background-image: url(http://www.example.com/bck1.png), url(http://www.example.com/bck2.png);
/* Keyword values: top, bottom, left, right, center */
background-position: left;
/* <percentage> values */
background-position: 25% 75%;
/* <length> values */
background-position: 1cm 2cm;
/* Multiple values */
background-position: 0px 0px, center;
/* Background position */
/* One-value syntax: repeat-x, repeat-y, repeat, space, round, no-repeat */
background-repeat: repeat;
/* Two-value syntax: horizontal | vertical */
background-repeat: no-repeat repeat;
/* Background size */
background-size: cover;
background-size: contain;
/* One-value syntax: the width of the image (the height is set to 'auto') */
background-size: 50%;
background-size: 3em;
background-size: 12px;
/* Two-value syntax: the first value: width of the image, the second value: height */
background-size: 50% auto;
background-size: 3em 25%;
background-size: auto 6px;
```

COLORS AND BACKGROUNDS – GOOD PRACTICES

- Use contrasting colors for text and background
- Pick different colors of similar intensity for backgrounds
  - Use darker colors for text
- Use color to highlight important stuff
- Use colors that reflect purpose of messages:
  - e.g., red for error, yellow for warning, blue for information
- Use tools that help finding sets of colors according to good practices
  - Adobe Color CC
  - http://paletton.com

SHORTHAND PROPERTIES

- Many CSS properties have shorthand properties that aggregate the simple ones,
  - They enable us to shorten CSS code by setting multiple properties in just one rule
  - Providing the values of the "simple" properties in the shorthand one is optional
    - If we do not set a value for a simple property, **it is implicitly set to its default value by the shorthand property!**
    - It may break the intended inheritance
- Examples
  - `background` is the shorthand for
    - `background-image, background-position, background-size, background-repeat, background-origin, background-clip, background-attachment, background-color`
  - `Font` is the shorthand for
    - `font-style, font-variant, font-weight, font-stretch, font-size, line-height, font-family`

CSS DETAILS

SOURCES OF CSS RULES

- CSS rules may have various sources of different priorities, in order:
  - Author – a CSS attached to document
  - User – a custom user CSS
  - User-agent – a default CSS of a browser

STYLE COMPUTING ALGORITHM

- Cascade rules
  - Filter the set of all rules for the rules applicable to **a particular media type**, **a DOM element**, and **a property**.
  - Take a set of indiscernible rules on the top of the ranking built in order:
    - User important declarations (declared with `!important` annotation)
    - Author important declarations (declared with `!important` annotation)
    - Author normal declarations
    - User normal declarations
    - User-agent declarations
  - Take a set of indiscernible rules on the top of the ranking built on rule **specificity**
  - Take the last specified rule
    - Imported style sheets are before the importing ones
- If cascading results in no rules, then inherit rules:
  - If the element is not the document root, use the computed value for the parent element
- If inheritance results in no rules, then use the initial value of the property from the specification

PRIORITIES OF CSS RULES – EXAMPLE 1

```
/*
    Media type: all
    Source: Website
    Specificity: 0.0.0.1
*/
p {
    color: black !important;
}

/*
    Media type: all
    Source: Website
    Specificity: 0.0.0.1
*/
p {
    color: white !important;
}

/*
    Media type: all
    Source: Website
    Specificity: 0.0.0.1
*/
p {
    color: orange;
}
```

1. Filter the rules applicable to the screen media type, the paragraph p, and the color property

PRIORITIES OF CSS RULES – EXAMPLE 1

```
/*
    Media type: all
    Source: Website
    Specificity: 0.0.0.1
*/
p {
    color: black !important;
}

/*
    Media type: all
    Source: Website
    Specificity: 0.0.0.1
*/
p {
    color: white !important;
}

/*
    Media type: all
    Source: Website
    Specificity: 0.0.0.1
*/
p {
    color: orange;
}
```

1. Filter the rules applicable to the screen media type, the paragraph p, and the color property
2. Filter the rules w.r.t. sources

PRIORITIES OF CSS RULES – EXAMPLE 1

```
/*
    Media type: all
    Source: Website
    Specificity: 0.0.0.1
*/
p {
    color: black !important;
}

/*
    Media type: all
    Source: Website
    Specificity: 0.0.0.1
*/
p {
    color: white !important;
}
```

```
/*
    Media type: all
    Source: Website
    Specificity: 0.0.0.1
*/
p {
    color: orange;
}
```

1. Filter the rules applicable to the screen media type, the paragraph p, and the color property
2. Filter the rules w.r.t. sources
3. Filter the rules w.r.t. specificity

PRIORITIES OF CSS RULES – EXAMPLE 1

```
/*
    Media type: all
    Source: Website
    Specificity: 0.0.0.1
*/
p {
    color: black !important;
}
```

```
/*
    Media type: all
    Source: Website
    Specificity: 0.0.0.1
*/
p {
    color: white !important;
}
```

```
/*
    Media type: all
    Source: Website
    Specificity: 0.0.0.1
*/
p {
    color: orange;
}
```

1. Filter the rules applicable to the screen media type, the paragraph p, and the color property
2. Filter the rules w.r.t. sources
3. Filter the rules w.r.t. specificity
4. Take the last specified rule

PRIORITIES OF CSS RULES – EXAMPLE 2

```
/*
    Media type: print
    Source: Website
    Specificity: 0.0.0.1
*/
@media print {
    p {
        color: black;
    }
}

/*
    Media type: all
    Source: Website
    Specificity: 0.0.0.1
*/
p {
    color: white;
}

/*
    Media type: all
    Source: Website
    Specificity: 0.0.0.2
*/
body p {
    color: blue;
}
```

PRIORITIES OF CSS RULES – EXAMPLE 2

```
/*
    Media type: print
    Source: Website
    Specificity: 0.0.0.1
*/
@media print {
    p {
        color: black;
    }
}
```

```
/*
    Media type: all
    Source: Website
    Specificity: 0.0.0.1
*/
p {
    color: white;
}

/*
    Media type: all
    Source: Website
    Specificity: 0.0.0.2
*/
body p {
    color: blue;
}
```

1. Filter the rules applicable to the screen media type, the paragraph p, and the color property

PRIORITIES OF CSS RULES – EXAMPLE 2

```
/*
    Media type: print
    Source: Website
    Specificity: 0.0.0.1
*/
@media print {
    p {
        color: black;
    }
}
```

```
/*
    Media type: all
    Source: Website
    Specificity: 0.0.0.1
*/
p {
    color: white;
}

/*
    Media type: all
    Source: Website
    Specificity: 0.0.0.2
*/
body p {
    color: blue;
}
```

1. Filter the rules applicable to the screen media type, the paragraph p, and the color property
2. Filter the rules w.r.t. sources

PRIORITIES OF CSS RULES – EXAMPLE 2

```
/*
    Media type: print
    Source: Website
    Specificity: 0.0.0.1
*/
@media print {
    p {
        color: black;
    }
}

/*
    Media type: all
    Source: Website
    Specificity: 0.0.0.1
*/
p {
    color: white;
}
```

```
/*
    Media type: all
    Source: Website
    Specificity: 0.0.0.2
*/
body p {
    color: blue;
}
```

1. Filter the rules applicable to the screen media type, the paragraph p, and the color property
2. Filter the rules w.r.t. sources
3. Filter the rules w.r.t. specificity

PRIORITIES OF CSS RULES – EXAMPLE 3

```
/*
    Media type: all
    Source: Website
    Specificity: 0.0.1.1
*/
a:link {
    color: black;
}

/*
    Media type: all
    Source: Website
    Specificity: 0.0.1.1
*/
a:hover {
    color: gray;
}

/*
    Media type: all
    Source: Website
    Specificity: 0.0.0.1
*/
body {
    color: orange;
}
```

PRIORITIES OF CSS RULES – EXAMPLE 3

```
/*
    Media type: all
    Source: Website
    Specificity: 0.0.1.1
*/
a:link {
    color: black;
}

/*
    Media type: all
    Source: Website
    Specificity: 0.0.1.1
*/
a:hover {
    color: gray;
}

/*
    Media type: all
    Source: Website
    Specificity: 0.0.0.1
*/
body {
    color: orange;
}
```

1. Filter the rules applicable to the screen media type, the paragraph p, and the color property

PRIORITIES OF CSS RULES – EXAMPLE 3

```
/*
    Media type: all
    Source: Website
    Specificity: 0.0.1.1
*/
a:link {
    color: black;
}

/*
    Media type: all
    Source: Website
    Specificity: 0.0.1.1
*/
a:hover {
    color: gray;
}
```

```
/*
    Media type: all
    Source: Website
    Specificity: 0.0.0.1
*/
body {
    color: orange;
}
```

1. Filter the rules applicable to the screen media type, the paragraph p, and the color property
2. Inherit value from the parent element (body)

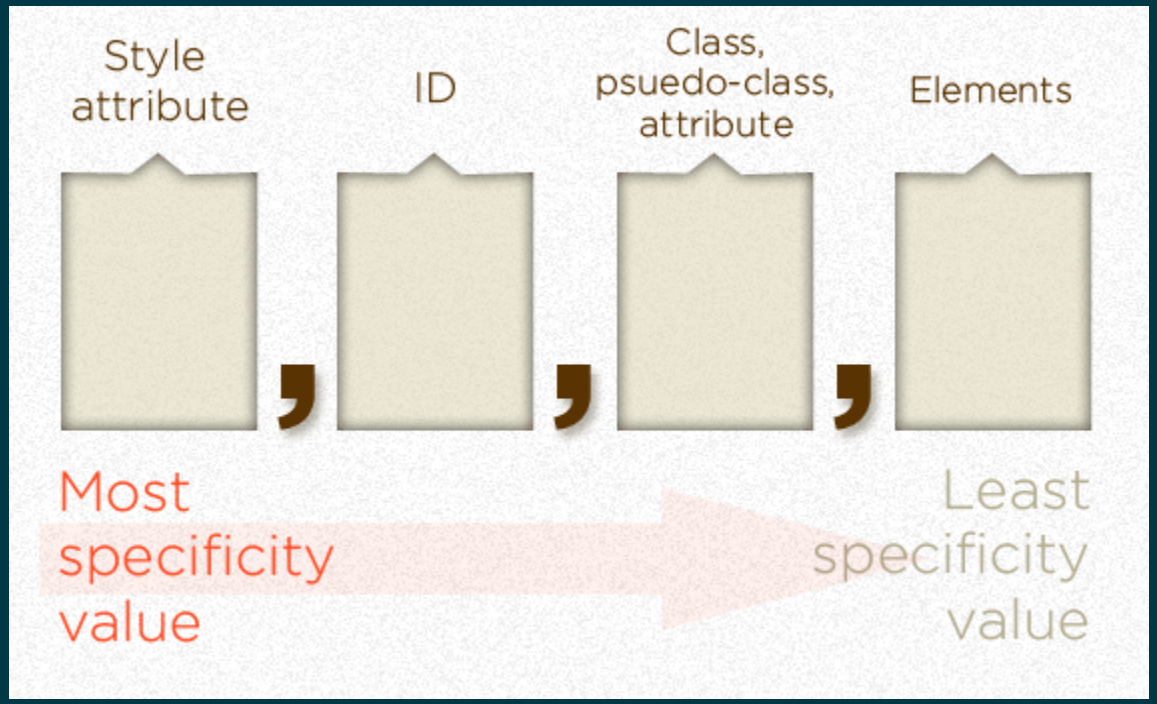
HOW TO ATTACH CSS TO HTML DOCUMENT

- CSS documents are typically attached in the **<head/>** section but this is rather a good practice than a rule

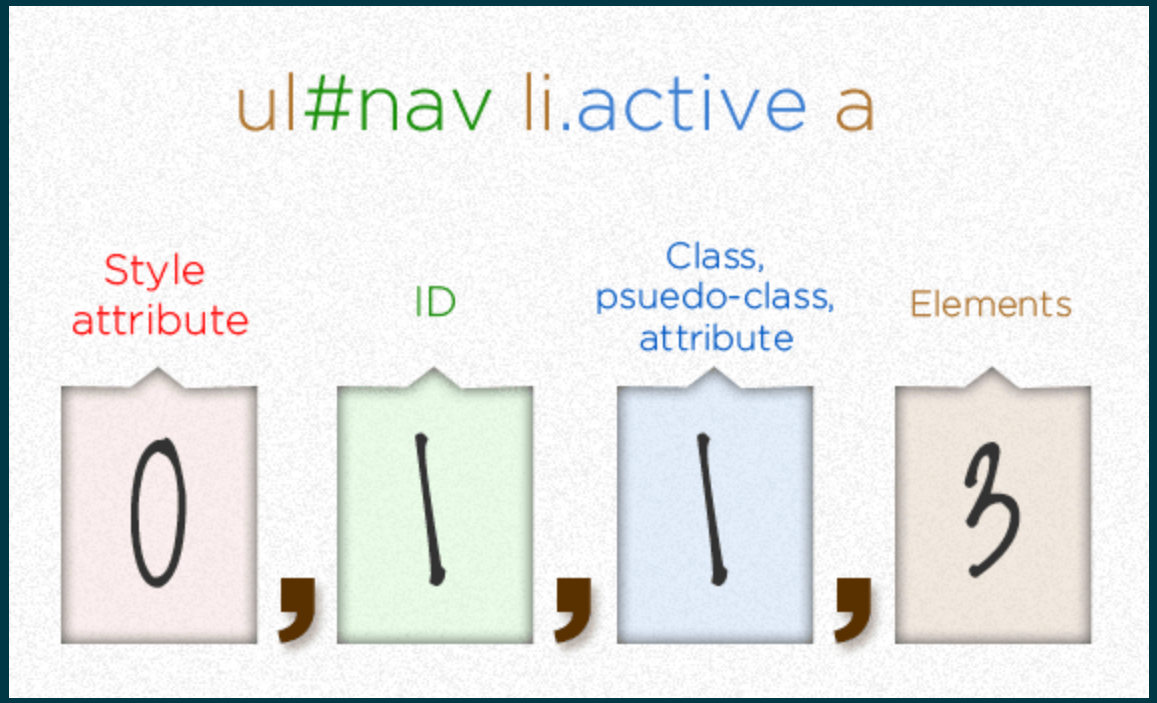
```html
<head>
    <!-- Attach CSS file to HTML document -->
    <link href="style.css" rel="stylesheet" type="text/css" />

    <!-- Attach CSS file to HTML document only when printing -->
    <link href="print.css" rel="stylesheet" type="text/css" media="print" />

    <!-- Embed CSS inside HTML document -->
    <style>
        body {
            color: black;
        }
    </style>
</head>
```

SELECTOR SPECIFICITY

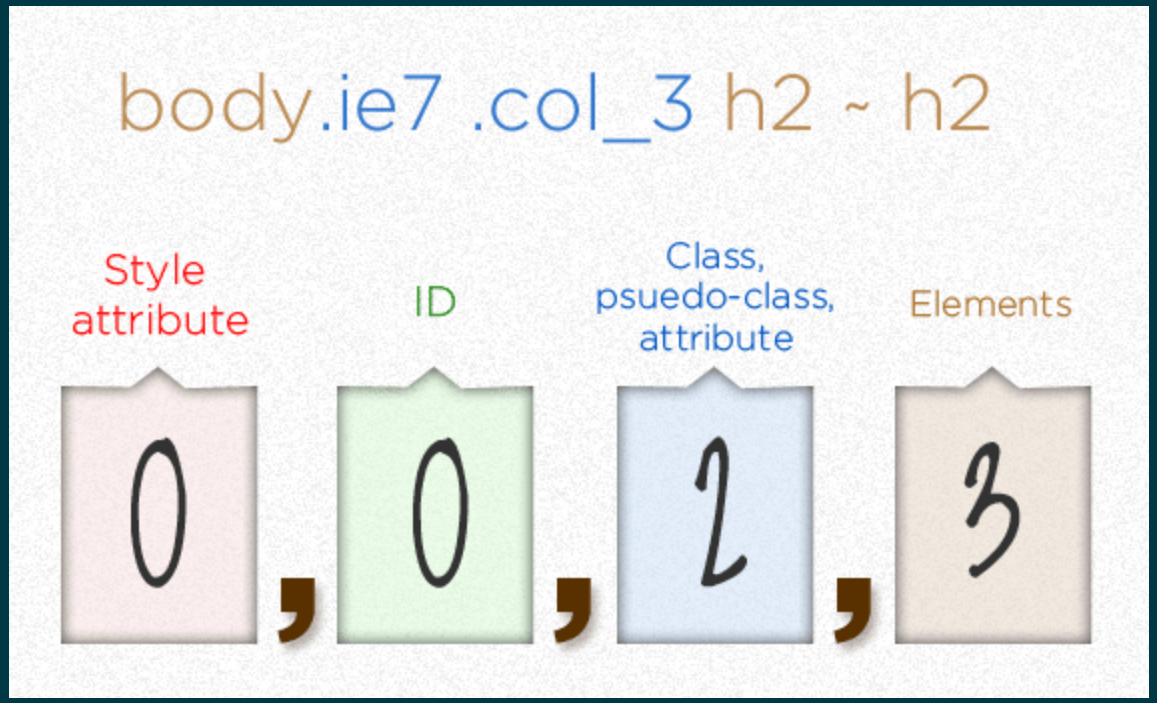
- Specificity is an ordered tuple of integers
  - Each integer is a number of specific terms in a selector, in order:
    - Style attribute
    - IDs
    - Classes, pseudo-classes (:), attributes
    - Elements, pseudo-elements (::)



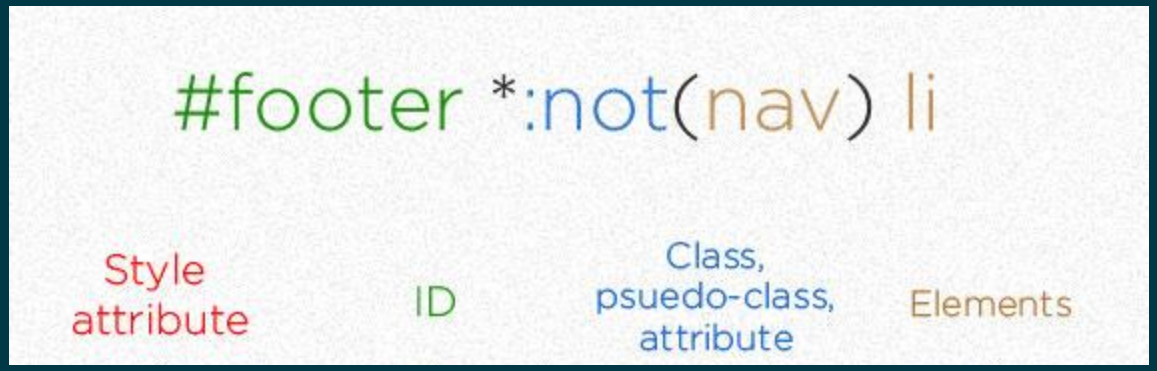
SELECTOR SPECIFICITY



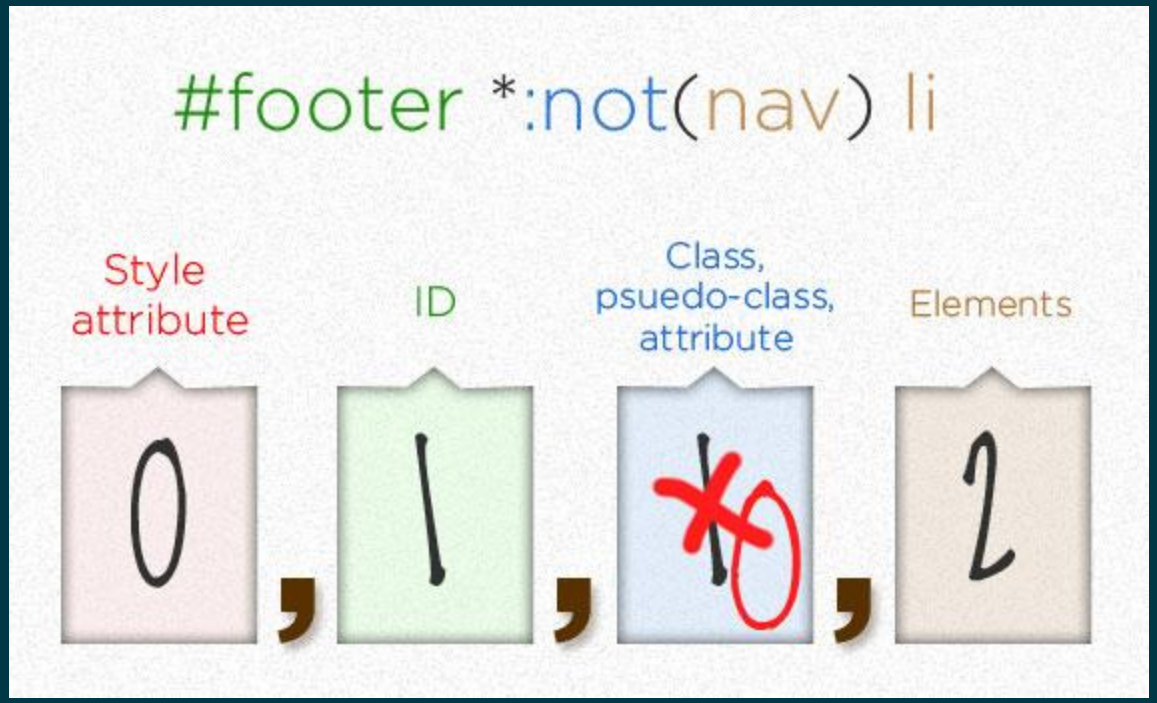
SELECTOR SPECIFICITY



SELECTOR SPECIFICITY

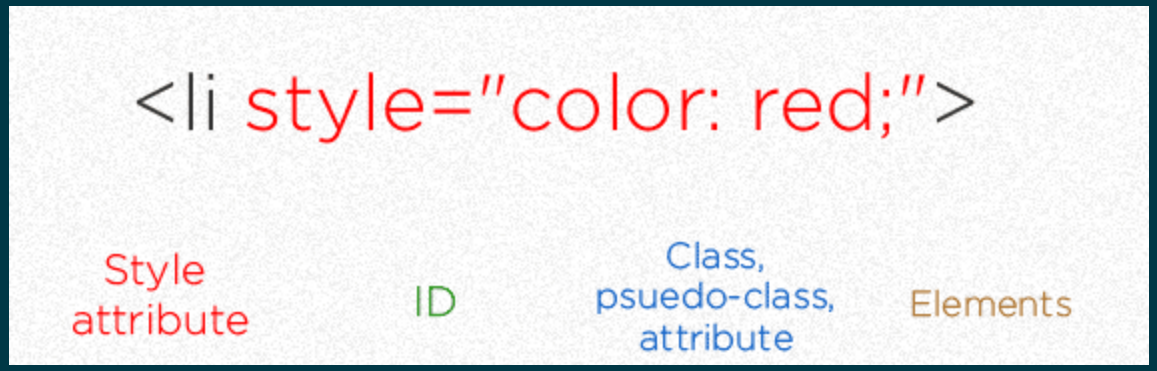


SELECTOR SPECIFICITY

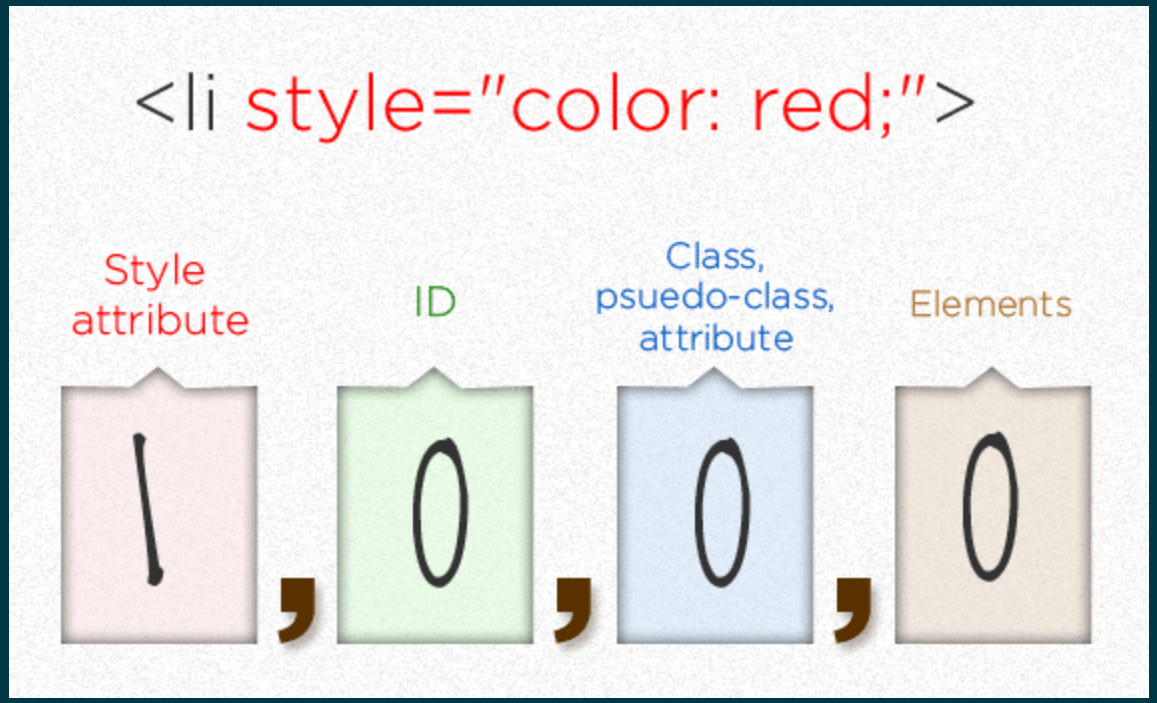


`:not()` is not counted as pseudo-class
universal selector (*) is not counted

SELECTOR SPECIFICITY



SELECTOR SPECIFICITY



SELECTOR SPECIFICITY

ul > li ul li ol li::first-letter

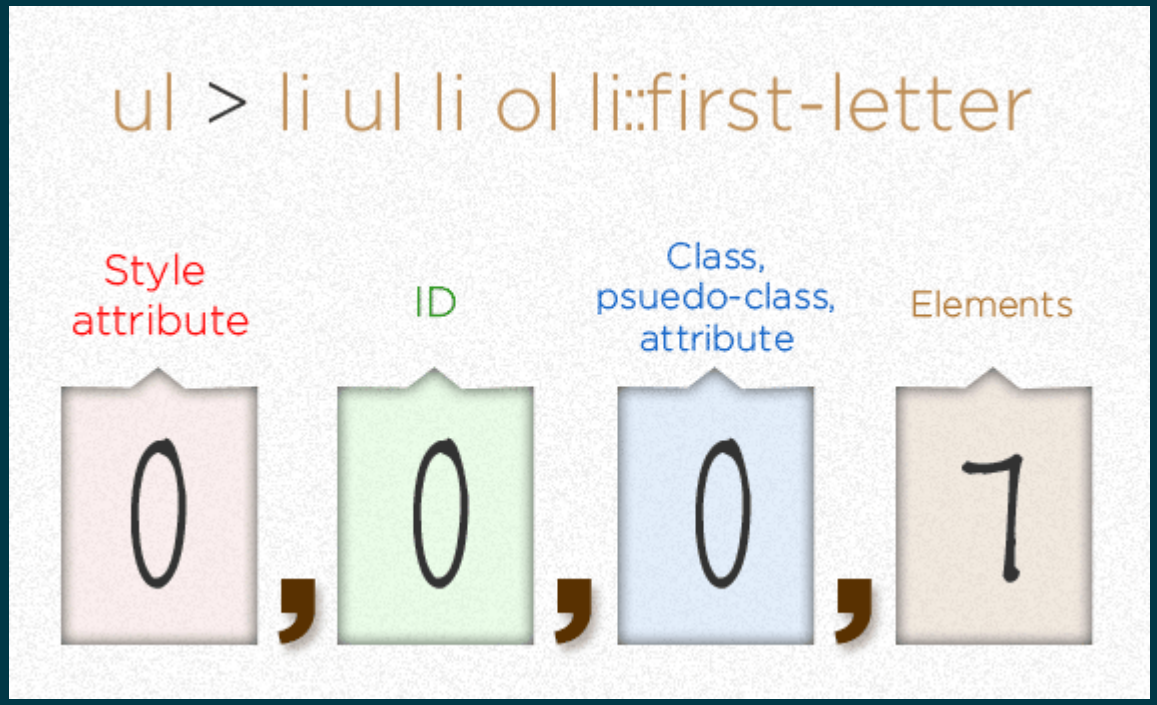
Style attribute

ID

Class, psuedo-class, attribute

Elements

SELECTOR SPECIFICITY



`::first-letter` and `::first-line` are as pseudo-elements

THE `!important` EXCEPTION

- `!important` can be considered as one additional integer to the left of specificity tuple
- From the rule computing algorithm, if two rules with `!important` are applicable, the one with higher specificity is applied

Source: https://developer.mozilla.org/en-US/docs/Web/CSS/Specificity

THE `!important` EXCEPTION

- `!important` can be considered as one additional integer to the left of specificity tuple
- From the rule computing algorithm, if two rules with `!important` are applicable, the one with higher specificity is applied
- Usage of `!important` is a **bad practice** because it makes debugging difficult by breaking the natural cascading of styles

Source: https://developer.mozilla.org/en-US/docs/Web/CSS/Specificity

THE `!important` EXCEPTION

- `!important` can be considered as one additional integer to the left of specificity tuple
- From the rule computing algorithm, if two rules with `!important` are applicable, the one with higher specificity is applied
- Usage of `!important` is a **bad practice** because it makes debugging difficult by breaking the natural cascading of styles
- Some rules of thumb from Mozilla:
  - Always look for a way to use specificity before even considering `!important`
  - Only use `!important` on page-specific CSS that overrides site-wide or foreign CSS (from external libraries).
  - Never use `!important` when you're writing a plugin/library.
  - Never use `!important` on site-wide CSS.

Source: https://developer.mozilla.org/en-US/docs/Web/CSS/Specificity

INHERITANCE AND OVERRIDING

- Some **properties are inheritable** from ancestor to descendant node in DOM
  - Most of the text-related properties are inheritable (`color, font-*,…`)
  - Most of the box-related properties are not inheritable (`border, display, width, height,…`)
  - See the reference for the details whether a certain property is inheritable
    - https://developer.mozilla.org/en-US/docs/Web/CSS/Reference
- Advantages
  - Inheritance prevents web developers from repeating code for nested tags
  - Less-rules = quicker page load (transfer) and more responsive UI
- To **override** inherited rules just define a separate rule specific to a nested element

WHITESPACE AND VALID CHARACTERS

- Whitespace
  - Between properties and rules is ignored
  - In property values or selectors is significant
    - However successive white characters are treated as one
- Valid names
  - Case-insensitive in ASCII range (a-z)
    - Exception: ids, classes, font name and URIs
  - Valid characters
    - 0-9, a-z, _, -
    - ISO-10646 characters U+00A0 and higher
    - Characters U+0100 and higher must be escaped using \, their hex code and a space, e.g., `\26 B` for `&B`
  - Name must not start with
    - Number
    - Two hyphens
    - Hyphen followed by digit

PAGE RENDERING ALGORITHM

- Page rendering algorithm has five stages
  - **JavaScript execution** – execute any pending scripts, because they can change CSS
  - **Style calculation** – inherit CSS rules and calculate rules that apply for particular DOM elements
  - **Layout** – calculate positions and sizes of DOM elements w.r.t. CSS rules (e.g., margins, sizes,…)
  - **Painting** – draw graphics w.r.t. CSS rules (e.g., colors, backgrounds,…)
  - **Compositing** – merge layers of page into one, removing possible overlapping

PAGE RENDERING ALGORITHM

- Each time a CSS property is changed, a part of the page rendering algorithm must be executed
- **Reflow** – process of recalculating layout for a part of DOM that is affected by change of a layout property
  - Reflow may trigger repaint:
- **Repaint** – process of redrawing a part of DOM that is affected by change of a paint-only property
  - Repaint may trigger composite:
- **Composite** – process of merging layers of page into one

TIPS FOR FAST PAGE RENDERING

TIPS FOR FAST PAGE RENDERING

- Remove unnecessary rules
  - Especially remove vendor-specific rules for widely supported properties
  - Use CSS features like inheritance

TIPS FOR FAST PAGE RENDERING

- Remove unnecessary rules
  - Especially remove vendor-specific rules for widely supported properties
  - Use CSS features like inheritance
- Use highly selective selectors
  - If many DOM elements match a rule, consider splitting the rule into two
  - This is especially important for the last term of the selector: Browsers keep a hash table of selectors, where the key is the last term of the selector

TIPS FOR FAST PAGE RENDERING

- Remove unnecessary rules
  - Especially remove vendor-specific rules for widely supported properties
  - Use CSS features like inheritance
- Use highly selective selectors
  - If many DOM elements match a rule, consider splitting the rule into two
  - This is especially important for the last term of the selector: Browsers keep a hash table of selectors, where the key is the last term of the selector
- Do not use the universal selector *
  - Use inheritance instead

TIPS FOR FAST PAGE RENDERING

- Remove unnecessary rules
  - Especially remove vendor-specific rules for widely supported properties
  - Use CSS features like inheritance
- Use highly selective selectors
  - If many DOM elements match a rule, consider splitting the rule into two
  - This is especially important for the last term of the selector: Browsers keep a hash table of selectors, where the key is the last term of the selector
- Do not use the universal selector *
  - Use inheritance instead
- Do not use the attribute selectors unless really needed
  - They are slow to evaluate

TIPS FOR FAST PAGE RENDERING

- Remove unnecessary rules
  - Especially remove vendor-specific rules for widely supported properties
  - Use CSS features like inheritance
- Use highly selective selectors
  - If many DOM elements match a rule, consider splitting the rule into two
  - This is especially important for the last term of the selector: Browsers keep a hash table of selectors, where the key is the last term of the selector
- Do not use the universal selector *
  - Use inheritance instead
- Do not use the attribute selectors unless really needed
  - They are slow to evaluate
- Use classes and ids to narrow range of matching DOM elements

TIPS FOR FAST PAGE RENDERING

TIPS FOR FAST PAGE RENDERING

- Change the class for a DOM element instead of changing associated CSS rules
  - Browsers keep precomputed data structures associated with CSS, changes in CSS require rebuilding these structures

TIPS FOR FAST PAGE RENDERING

- Change the class for a DOM element instead of changing associated CSS rules
  - Browsers keep precomputed data structures associated with CSS, changes in CSS require rebuilding these structures
- Do not change page-wide rules unless really needed
  - Browsers are optimized to redraw only parts of pages that are needed

TIPS FOR FAST PAGE RENDERING

- Change the class for a DOM element instead of changing associated CSS rules
  - Browsers keep precomputed data structures associated with CSS, changes in CSS require rebuilding these structures
- Do not change page-wide rules unless really needed
  - Browsers are optimized to redraw only parts of pages that are needed
- Change a set of rules at once, instead of changing rules one-by-one
  - Browsers aggregate successive changes before redrawing page

TIPS FOR FAST PAGE RENDERING

- Change the class for a DOM element instead of changing associated CSS rules
  - Browsers keep precomputed data structures associated with CSS, changes in CSS require rebuilding these structures
- Do not change page-wide rules unless really needed
  - Browsers are optimized to redraw only parts of pages that are needed
- Change a set of rules at once, instead of changing rules one-by-one
  - Browsers aggregate successive changes before redrawing page
- In scripts do not interleave read of the calculated layout properties with setting CSS rules
  - Reading of a calculated layout property requires reflow if CSS is modified

TIPS FOR FAST PAGE RENDERING

- Change the class for a DOM element instead of changing associated CSS rules
  - Browsers keep precomputed data structures associated with CSS, changes in CSS require rebuilding these structures
- Do not change page-wide rules unless really needed
  - Browsers are optimized to redraw only parts of pages that are needed
- Change a set of rules at once, instead of changing rules one-by-one
  - Browsers aggregate successive changes before redrawing page
- In scripts do not interleave read of the calculated layout properties with setting CSS rules
  - Reading of a calculated layout property requires reflow if CSS is modified
- Use CSS features instead of scripts if possible
  - E.g., for animations, hovering etc.

TIPS FOR FAST PAGE RENDERING

- And, what is the most important:
- Verify rendering bottleneck of your web page and gains from your improvements in a web page profiler
  - Currently all major browsers have profilers in their developer tools, e.g.,:

Elements Console Sources Network Timeline Profiles Resources Security Audits HTTPS Everywhere

7 2

Capture: JS Profile Memory Paint Screenshots

500 ms 1000 ms 1500 ms 2000 ms 2500 ms 3000 ms 3500 ms 4000 ms 4500 ms 5000 ms

FPS

CPU

NET

1500 ms 2000 ms 2500 ms 3000 ms 3500 ms 4000 ms 4500 ms 5000 ms

585.8 ms 446.2 ms 509.6 ms 1216.9 ms 309.3 ms 319.3 ms

Main Thread

P...) Evaluate ...min.js:1) Timer Fired L...

E...) Function Call

CompositorTileWorker1/8260

CompositorTileWorker2/5472

Summary Bottom-Up Call Tree Event Log

Group by Category

Self Time		Total Time		Activity	
663.7 ms	61.2 %	663.7 ms	61.2 %	Scripting	
330.3 ms	30.4 %	330.3 ms	30.4 %	Rendering	
130.9 ms	12.1 %	130.9 ms	12.1 %	Layout	
47.6 ms	4.4 %	47.6 ms	4.4 %	Recalculate Style	desktop.min.js:10
38.8 ms	3.6 %	38.8 ms	3.6 %	Recalculate Style	adblock_start_common.js:105
36.2 ms	3.3 %	36.2 ms	3.3 %	Recalculate Style	
26.0 ms	2.4 %	26.0 ms	2.4 %	Layout	desktop.min.js:10
16.8 ms	1.5 %	16.8 ms	1.5 %	Recalculate Style	sdk.js:132
13.7 ms	1.3 %	13.7 ms	1.3 %	Update Layer Tree	
13.1 ms	1.2 %	13.1 ms	1.2 %	Recalculate Style	www.facebook.com/v2.5/plugins/like.php?action=like&app_id=1381568178819875&...button_count&locale=pl...
3.5 ms	0.3 %	3.5 ms	0.3 %	Hit Test	
2.3 ms	0.2 %	2.9 ms	0.3 %	Layout	sdk.js:132
1.0 ms	0.1 %	1.0 ms	0.1 %	Recalculate Style	www.facebook.com/v2.5/plugins/like.php?action=like&app_id=1381568178819875&...button_count&locale=pl...
0.3 ms	0.0 %	0.3 ms	0.0 %	Recalculate Style	widgets.js:9
0.3 ms	0.0 %	0.3 ms	0.0 %	Layout	www.facebook.com/v2.5/plugins/like.php?action=like&app_id=1381568178819875&...button_count&locale=pl_PL&sdk=...
0.0 ms	0.0 %	0.0 ms	0.0 %	Layout	adblock_start_common.js:105
0.0 ms	0.0 %	0.0 ms	0.0 %	Layout	widgets.js:9

Heaviest stack

Total Time		Activity
663.7 ms	100.0 %	Scripting
614.5 ms	92.6 %	Evaluate S
154.6 ms	23.3 %	Parse HTM

DISPLAY PROPERTY

- `display` property specifies how **lay out** and **paint** an DOM element
- Default value: `inline`
  - Overridden by HTML specification and browser styles for some elements
  - E.g., `div` → `block`

DISPLAY PROPERTY VALUES

CSS level	Value	Meaning
1	none	No display, element not included in layout calculation
1	inline	The element generates one or more inline element boxes
1	block	The element generates a block element box.
1	list-item	The element generates a block box for the content and a separate list-item inline box.
2.1	inline-block	The element generates a block element box that will be flowed with surrounding content as if it were a single inline box
2.1	table, inline-table, table-caption, table-cell, table-column, table-column-group, table-footer-group, table-header-group, table-row, table-row-group	Represent layout behavior of HTML tables, but can be applied to other elements than tables
1 flexbox	flex	The element behaves like a block element and lays out its content according to the flexbox model.
1 flexbox	inline-flex	The element behaves like an inline element and lays out its content according to the flexbox model.
1 grid	grid	The element behaves like a block element and lays out its content according to the grid model.
1 grid	inline-grid	The element behaves like an inline element and lays out its content according to the grid model.

VISIBILITY PROPERTY

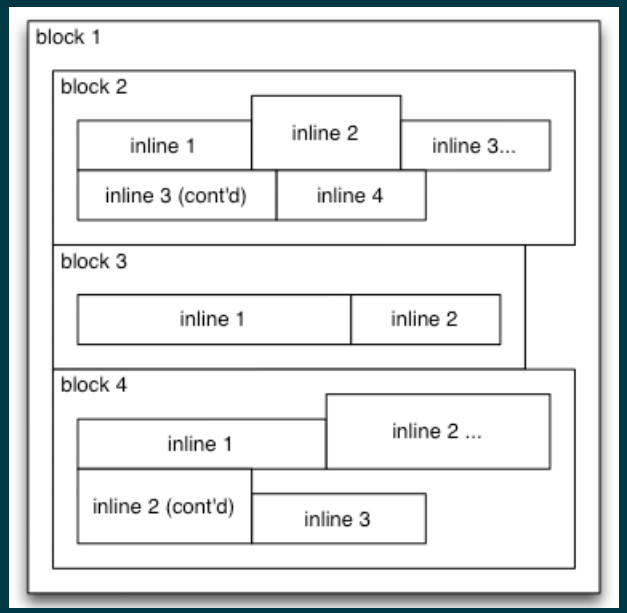
- `visibility` property controls whether an element is painted
  - The element is always laid out
- Values
  - `visible` – the element is drawn
  - `hidden` – the element is not drawn. Descendants of the element are visible if they have `visibility: visible`
  - `collapse` – for table rows, columns, column groups, and row groups the row(s) or column(s) are hidden and the space they would have occupied is removed. However, the size of other rows and columns is still calculated as though the cells in the collapsed row(s) or column(s) are present. This was designed for fast removal of a row/column from a table without having to recalculate widths and heights for every portion of the table.

POSITIONING OF ELEMENTS

- CSS2.1 and its extensions (CSS3) define several ways of positioning a DOM element
  - Static positioning
  - Absolute and relative positioning
  - Fixed positioning
  - Float positioning
  - Flexbox positioning
  - Grid positioning
- `position` property allows us to choose positioning mode between
  - `static` (default)
  - `absolute`
  - `relative`
  - `fixed`

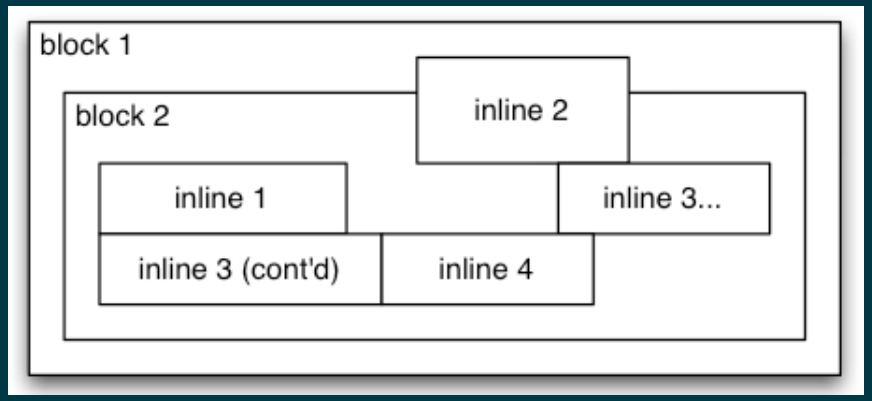
position: static

- **Inline** items are treated like letters in word in text
  - Items are placed one after another until there is no available space left
  - Then a new line starts
  - Size of the inline item cannot be set absolutely, it depends on the content
- **Block** items stack vertically, like paragraphs
  - By default the block items occupy whole width of the parent container
  - Size of the block items can be set by `width` and `height` properties



position: absolute AND position: relative

- **Absolute** positioning is used to place an element absolutely w.r.t. the position of the nearest ancestor element with `position` not `static`
  - Use properties `left, top, bottom, right` to set position
- **Relative** positioning is used to place an element relatively to its position as it would be in static positioning
  - Use properties `left, top, bottom, right` to set offset from normal position
  - Relative positioning does not change layout, thus the space occupied by normal position of an element would remain blank



POSITION: FIXED

- Elements are positioned relative to viewport (usually a browser window)
  - Use properties `left, top, bottom, right` to set offset from viewport
- This means that if page is scrolled, the element will remain in the same location in the screen

FLOATING POSITIONING

- In floating positioning, an element is moved (floated) to the leftmost or the rightmost position in its container
  - Without overlapping any other floated content
  - Possibly with overlapping other content
- Use `float` property to set mode of float positioning to
  - `none` (default)
  - `left`
  - `right`
- Floated elements are treated as it would be `display: block` set on them

FLOATING POSITIONING

- In floating positioning, an element is moved (floated) to the leftmost or the rightmost position in its container
  - Without overlapping any other floated content
  - Possibly with overlapping other content
- Use `float` property to set mode of float positioning to
  - `none` (default)
  - `left`
  - `right`
- Floated elements are treated as it would be `display: block` set on them
- Use `clear` property to indicate that an element must be wrapped to the next line of successive float elements
  - `none` (default)
  - `left` – for left floats
  - `right` – for right floats
  - `both` – for all floats

FLEXBOX POSITIONING

- Flexbox positioning is a flexible positioning mechanism that adjusts layout to the size of the viewport and allows us to, e.g.:
  - Center an element inside the middle of a page
  - Put a set of elements in a vertical flow
  - Create a row of elements that collapses vertically on small screen

- To create flexbox container use **`div`** element and set:

```
div.myflex {
    display: flex;
    /* Set direction of flow: row or column */
    flex-flow: row;
    /* Set if elements may wrap (wrap, nowrap, wrap-reverse) */
    flex-wrap: nowrap;
}
```

- Set on each child element flex property:

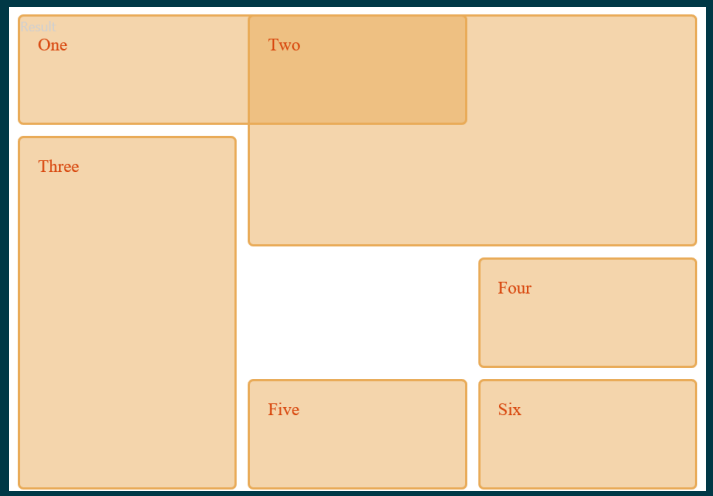
```
div.myflex div {
    /* Use original size of this element */
    flex: none;
    /* Use a specific size for layout calculation */
    flex: 0 0 60px;
    /* Distribute space using proportions: flex-grow | flex-shrink | flex-basis */
    flex: 1 2 10%;
    /* Fill available space, share space equally between elements of the same type */
    flex: auto;
}
```

GRID POSITIONING

- Grid layout enables an author to align elements into columns and rows
- A grid container's child elements could position themselves so they actually overlap and layer, similar to CSS positioned elements.

```html
<div class="wrapper">
    <div class="one">One</div>
    <div class="two">Two</div>
    <div class="three">Three</div>
    <div class="four">Four</div>
    <div class="five">Five</div>
    <div class="six">Six</div>
</div>
```

```css
.wrapper {
    display: grid;
    grid-template-columns: repeat(3, 1fr);
    grid-gap: 10px;
    grid-auto-rows: minmax(100px, auto);
}
.one { grid-column: 1 / 3; grid-row: 1; }
.two { grid-column: 2 / 4; grid-row: 1 / 3; }
.three { grid-column: 1; grid-row: 2 / 5; }
.four { grid-column: 3; grid-row: 3; }
.five { grid-column: 2; grid-row: 4; }
.six { grid-column: 3; grid-row: 4; }
```



Source: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout

AT-RULES

- At-rules (or @-rules) are a special type of rules that do not directly influence layout nor painting of the page, instead **they control interpretation** of a CSS file
- Examples

```
@charset "UTF-8";

/* Import other CSS file here */
@import "custom.css";
/* Import other CSS file for media type: print */
@import url("print.css") print;

/* Namespaces for XML content */
/* Default namespace */
@namespace "XML-namespace-URL";
/* Prefixed namespace */
@namespace prefix "XML-namespace-URL";
```

@FONT-FACE AT-RULE

- A way to specify a server-provided font to display text in a web browser
- Eliminates dependence on the limited number of fonts installed in the user's computer

```
/* Definition of a font */
@font-face {
  font-family: "Custom name of font";
  src: local("Custom-font-name-bold"), url(custom.woff), url(custom.ttf);
  /* Remaining properties are optional */
  font-variant: small-caps;
  font-stretch: normal;
  font-weight: normal;
  font-style: normal;
}

p {
    /* Use of the font defined above in paragraphs */
    font-family: 'Custom name of font';
}
```

WEB OPEN FONT FORMAT (WOFF)

- Developed by Mozilla and other organizations
  - v1 & v2 are W3C Recommendations
- WOFF has all features of TrueType, OpenType and Open Font Format fonts
- Additional features
  - Compression
  - Metadata and private vendor-specific data
  - Predefined fields to put licensing information
- Browser support
  - V1 & v2: all major browsers
- A web fonts repository:
  - https://fonts.google.com/

MEDIA QUERIES

- `@media` rule is used to indicate that a fragment of a CSS document applies only to:
  - A certain media
    - `all` – all suitable devices
    - `print` – for paged material and for documents viewed on screen in print preview mode
    - `screen` – for color computer screens
    - `speech` – for speech synthesizers
    - Other media types are deprecated as of CSS4 Media Queries
  - Under certain conditions
    - `width` – viewport width
    - `height` – viewport height
    - `aspect-ratio` – width-to-height aspect ratio of the viewport
    - `orientation` – orientation of the viewport
    - `resolution` – pixel density of the output device
    - `color` – number of bits per color component of the output device, or zero if the device is not color
    - `scripting` – the output device supports scripting (e.g., JavaScript)

MEDIA QUERIES

```
/* Print only rule */
@media print {
    body {
        background: #fff;
        color: #000;
    }
}

/* And condition */
@media print and (min-width: 20cm) and (orientation: landscape) {
    body {
        font-size: larger;
    }
}

/* Or condition */
@media (max-width: 700px), screen and (orientation: landscape) {
    .menu {
        height: 100px;
    }
}

/* Negation of whole condition ("not" refers to (all and (monochrome)))) */
@media not all and (monochrome) {
    body {
        background: url('rainbow.png');
    }
}
```

CSS TRANSFORMS

- CSS transform is a property that **changes coordinates of painting** of an DOM element **without actually changing its layout**
- Available transforms are
  - **Rotation –** `rotate(angle), rotate3d(x, y, z, angle)`
  - **Skewing –** `skew(anglex, angley)`
  - **Scaling –** `scale(factorx, factory), scale3d(factorx, factory, factorz)`
  - **Translation –** `translate(deltax, deltay), translate3d(x, y, z)`
  - All in 2D and 3D
- Typical usage involves two properties
  - `transform` – a space-separated list of transforms to apply
  - `transform-origin` – defines the origin of the coordinate system relative to the current element, by default it is center

```
/* Rotates all images by 90deg clockwise around bottom left corner */
img {
    transform: rotate(90deg);
    transform-origin: bottom left;
}
```

CSS TRANSFORMS

- CSS transforms can be combined, in which case they are evaluated from left to right, e.g.,
  HTML:
  ```
  <div>Transformed element</div>
  ```
  CSS:
  ```
  div {
      border: solid red;
      transform: translate(30px, 20px) rotate(20deg);
      width: 140px;
      height: 60px;
  }
  ```

Source: https://developer.mozilla.org/en-US/docs/Web/CSS/transform

CSS ANIMATIONS

- CSS animations enable smooth transitions between many CSS rule sets
- Each animation contains two components
  - A **style** describing the CSS animation
  - A set of **keyframes** that indicate the start, the end, and the intermediate animation states

CSS ANIMATIONS

- CSS animations enable smooth transitions between many CSS rule sets
- Each animation contains two components
  - A **style** describing the CSS animation
  - A set of **keyframes** that indicate the start, the end, and the intermediate animation states
- To apply an animation to an element, use the `animation` shorthand property or one of the simple properties:
  - `animation-delay` – the delay between loading an element and beginning the animation
  - `animation-direction` – sets whether the animation alternates direction on each run or resets to the start point and repeat itself
  - `animation-duration` – sets how long the animation should take to complete one cycle
  - `animation-iteration-count` – sets the number of times the animation repeats or `infinite`
  - `animation-name` – specifies name of `@keyframes` rule
  - `animation-play-state` – lets you pause and resume the animation sequence
  - `animation-timing-function` – sets how the animation transitions through keyframes
  - `animation-fill-mode` – sets what values are applied before and after the animation

DEFINING KEYFRAMES

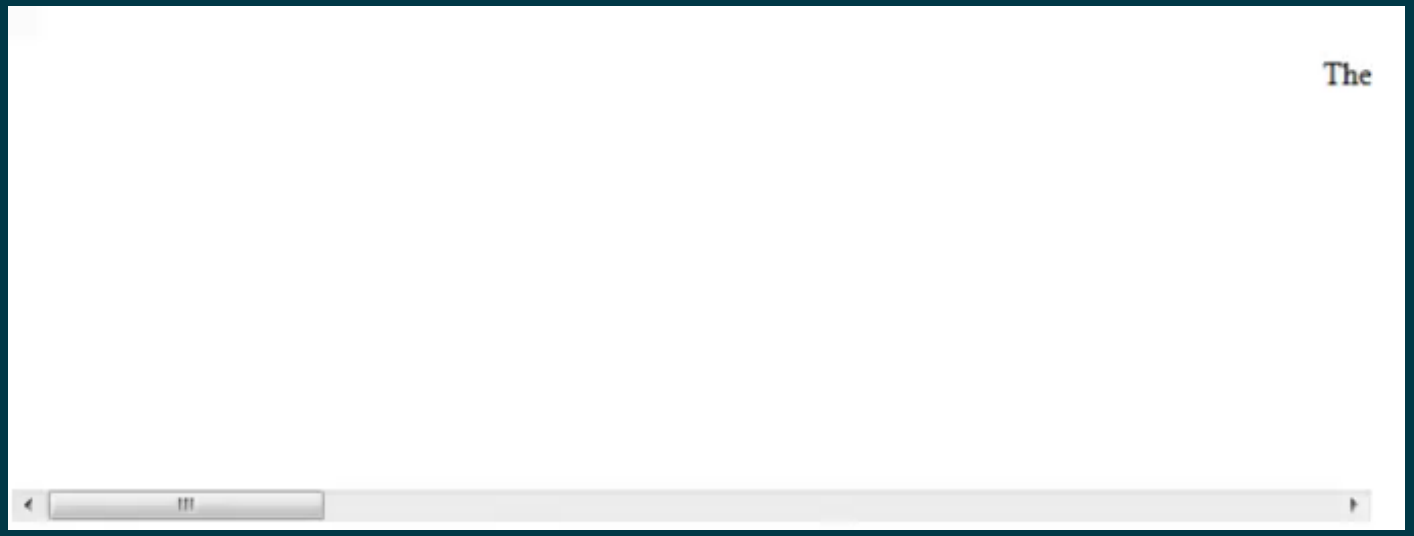
- Use `@keyframes` at-rule
  - At least one keyframe must be defined
    - If starting or ending keyframes are not defined, browser uses computed styles as fallback
  - Browser performs smooth transitions between keyframes
- Keyframes are specified by
  - Percentage of animation, e.g.,
    - `0%`, `50%`, `100%`
    - `from` and `to` are aliases for `0%` and `100%`, respectively
  - Set of CSS rules to be applied at this keyframe

CSS ANIMATION EXAMPLE

```html
<p>
    The Caterpillar and Alice looked at each other for some time in silence:
    at last the Caterpillar took the hookah out of its mouth, and addressed
    her in a languid, sleepy voice.
</p>
```

```css
p {
    animation-duration: 3s;
    animation-name: slidein;
    animation-iteration-count: infinite;
    animation-direction: alternate;
}

@keyframes slidein {
    from {
        margin-left: 100%;
        width: 300%;
    }

    to {
        margin-left: 0%;
        width: 100%;
    }
}
```

CSS ANIMATION EXAMPLE



CONTENT GENERATION

- The `content` CSS property replaces an element with a generated value. Objects inserted using the content property are anonymous replaced elements.
- `::before` and `::after` pseudo-elements are the first and the last child of the selected element, respectively. They are often used to add cosmetic content to an element with the content property. They are inline by default.

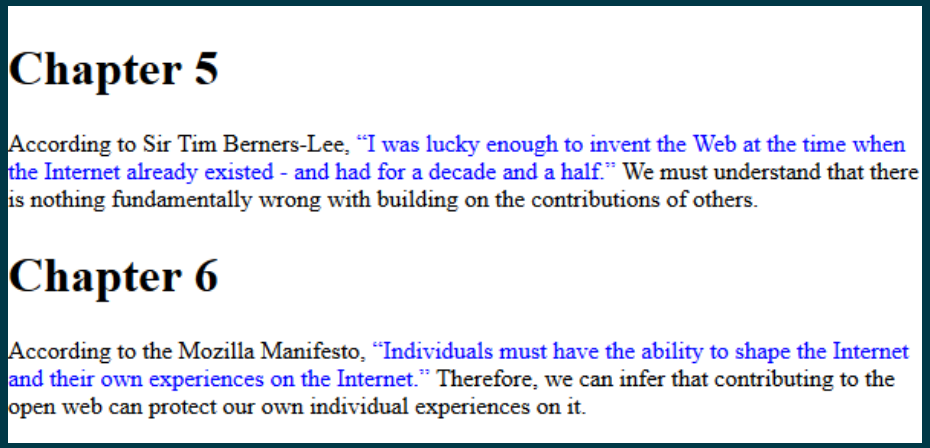
CONTENT GENERATION

- HTML:

```
<h1>5</h1>
<p>According to Sir Tim Berners-Lee,
<q cite="http://www.w3.org/People/Berners-Lee/FAQ.html#Internet">I was lucky
enough to invent the Web at the time when the Internet already existed - and had
for a decade and a half.</q> We must understand that there is nothing
fundamentally wrong with building on the contributions of others.</p>
<h1>6</h1>
<p>According to the Mozilla Manifesto,
<q cite="http://www.mozilla.org/en-US/about/manifesto/">Individuals must have the
ability to shape the Internet and their own experiences on the Internet.</q>
Therefore, we can infer that contributing to the open web can protect our own
individual experiences on it.</p>
```

- CSS:

```
q { color: blue; }
q::before { content: open-quote; }
q::after { content: close-quote; }
h1::before { content: "Chapter ";
/*The trailing space creates separation
between the added content and the rest
of the content*/ }
```



Source: https://developer.mozilla.org/en-US/docs/Web/CSS/content

REFER HTML ATTRIBUTES IN CSS

- It is possible to refer attributes of DOM elements matched by a CSS selector in a value of a CSS property using `attr()` function:

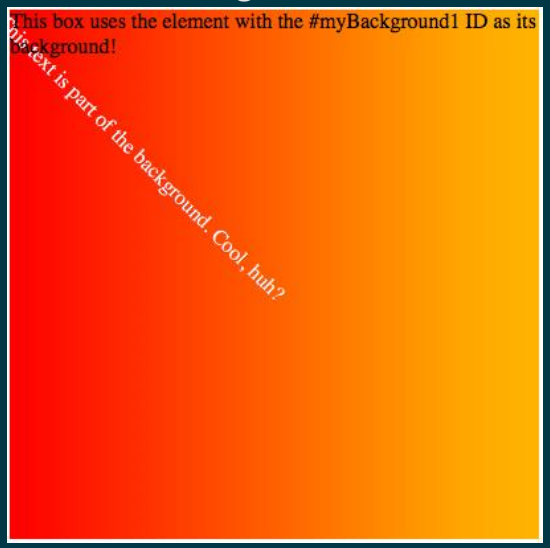
```
<p data-foo="hello">world</p>

[data-foo]::before {
        content: attr(data-foo) " ";
}
```

- Unfortunately, as of May 2020 this works only in the `content` property, although the CSS Values and Units Module Level 3 specification that allows for the use of `attr()` in any property is in Candidate Recommendation state

RENDERING HTML TO AN IMAGE

- With `element()` function it is possible to render a DOM element to an image and use this image in any image property of CSS

```
<div style="width:400px; height:400px; background:-moz-element(#myBackground1) no-repeat;">
    <p>This box uses the element with the #myBackground1 ID as its background!</p>
</div>
<div style="overflow:hidden; height:0;">
    <div id="myBackground1"
        style="width:1024px; height:1024px;
               background-image: linear-gradient(to right, red, orange, yellow, white);">
        <p style="transform-origin:0 0; transform: rotate(45deg); color:white;">
            This text is part of the background. Cool, huh?
        </p>
    </div>
</div>
```



Source: https://developer.mozilla.org/en-US/docs/Web/CSS/element

CONCLUSIONS

- Cascading Style Sheets are powerful technology for describing presentation of documents
  - CSS is primarily designed for use with HTML
  - However, it was adopted in other XML-related technologies, like SVG
- CSS supports
  - Different output devices and media: screen, print, speech
  - Layout and painting
  - Static and dynamic content
- CSS is still under active development
  - CSS2.1 specification is done
  - CSS2.2 specification is Working Draft
  - Many CSS3 and some CSS4 specifications are done
- Nowadays, when W3C publishes a Recommendation, it is usually already supported by all major browsers, due to active participation of browser developers in the process of establishing standards

REFERENCES

- CSS reference, Mozilla Developer Network https://developer.mozilla.org/en-US/docs/Web/CSS/Reference
- Using media queries, Mozilla Developer Network https://developer.mozilla.org/en-US/docs/Web/CSS/Media_Queries/Using_media_queries
- Using CSS animations, Mozilla Developer Network https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Animations/Using_CSS_animations
- Specifics on CSS Specificity https://css-tricks.com/specifics-on-css-specificity/
- Cascading Style Sheets, Wikipedia, the free Encyclopedia, https://en.wikipedia.org/wiki/Cascading_Style_Sheets
- Rendering Performance, Google Developers https://developers.google.com/web/fundamentals/performance/rendering/
- Matt Ryall, CSS layout fundamentals, part 4: positioning http://mattryall.net/blog/2008/08/css-layout-fundamentals-positioning
- Matt Ryall, CSS layout fundamentals, part 5: floats http://mattryall.net/blog/2008/09/css-layout-fundamentals-floats