

# Cascading Style Sheets (CSS)

Péter Jeszenszky

Faculty of Informatics, University of Debrecen

December 15, 2024

# Terms of Use

This work is licensed under a [Creative Commons](#) “[Attribution 4.0 International](#)” license.



# What is CSS?

- A style sheet language for describing the rendering of structured documents (such as HTML and XML).
  - Supports rendering on different devices, such as screens, printers, and Braille devices.
- Separates the presentation style of documents from the content of documents.
  - Thus, simplifies web authoring and site maintenance.
- Website: <https://www.w3.org/Style/CSS/>

# History

- Originally, CSS was proposed by Håkon Wium Lie, a Norwegian researcher in 1994 while he was working with Tim Berners-Lee at CERN.
  - Website: <https://www.wiumlie.no/>
- For more information, see: [20 Years of CSS \(W3C\)](#)
  - [A brief history of CSS until 2016](#)

# Development

Developed by the [CSS Working Group](#) of W3C.

# Levels

- CSS does not have versions in the traditional sense, instead, it has levels:
  - CSS Level 1
  - CSS Level 2
  - CSS Level 3
- Each level of CSS builds on the previous one, refining definitions and adding features.
  - See: [Levels, snapshots, modules...](#) (W3C)

# CSS Level 1

- Specification: [Cascading Style Sheets, level 1 \(Superseded W3C Recommendation\)](#)
- Considered to be obsolete.

# CSS Level 2

- Defined by the CSS2.1 specification (i.e., a single document):
  - [Cascading Style Sheets Level 2 Revision 1 \(CSS 2.1\) Specification \(W3C Recommendation, 7 June 2011\)](#)
- Its revision is currently under development:
  - [Cascading Style Sheets Level 2 Revision 2 \(CSS 2.2\) Specification \(W3C First Public Working Draft, 12 April 2016\)](#)

# CSS Level 3 (1)

- Currently under development.
- Has a modular structure:
  - Broken into modules, where each module defines a part of CSS.
  - Each module adds functionality and/or replaces part of the CSS2.1 specification.
- Modules also have levels.
  - Modules with no CSS Level 2 equivalent start at Level 1.
  - Modules that update features that existed in CSS Level 2 start at Level 3.

## CSS Level 3 (2)

- Different CSS modules are at different levels of stability.
- The list of all CSS specifications:  
<http://www.w3.org/Style/CSS/current-work.en.html>
- A few modules are the following:
  - Selectors Level 3 (W3C Recommendation, 6 November 2018)
  - CSS Values and Units Module Level 3 (W3C Candidate Recommendation Draft, 22 March 2024)
  - CSS Transforms Module Level 1 (W3C Candidate Recommendation, 14 February 2019)
  - CSS Transforms Module Level 2 (W3C Working Draft, 9 November 2021)
  - ...

# CSS Level 3 (3)

## CSS Snapshot 2023 (W3C Group Note, 7 December 2023)

- Collects together all specifications that together form the current state of CSS.
- The primary audience is CSS implementers.
- See the section titled [Cascading Style Sheets \(CSS\) – The Official Definition](#).

# CSS Level 4 and Beyond

- There is no CSS Level 4.
- Independent modules can reach level 4 or beyond, but CSS the language no longer has levels.
- “CSS Level 3” as a term is used only to differentiate it from the previous monolithic versions.

# Browser Support

- Modern web browsers support most of the features of CSS 2.1.
- They support some features of CSS3:
  - <https://caniuse.com/?search=CSS3>
  - The CSS3 Test <https://github.com/LeaVerou/css3test>

# Coding Styles

- [Google HTML/CSS Style Guide](#)
- [Nicolas Gallagher. Principles of writing consistent, idiomatic CSS.](#)

# Editors (1)

Free and open source software:

- Visual Studio Code (platform: Linux, macOS, Windows; license: MIT License) <https://code.visualstudio.com/>  
<https://github.com/Microsoft/vscode>
  - See: <https://code.visualstudio.com/docs/languages/css>

## Editors (2)

Non-free software:

- WebStorm (platform: Linux, macOS, Windows)  
<https://www.jetbrains.com/webstorm/>
  - See: <https://www.jetbrains.com/help/webstorm/style-sheets.html>

## Editors (3)

Emmet (written in: JavaScript; license: MIT License) <https://emmet.io/>  
<https://github.com/emmetio/emmet>

- A set of text editor plugins for boosting HTML and CSS code writing.
- Available for many text editors such as: Eclipse, NetBeans, Notepad++, Visual Studio Code, IntelliJ IDEA, ...
- See: <https://emmet.io/download/>
- Documentation: <https://docs.emmet.io/>
  - Customization: <https://docs.emmet.io/customization/>
- See also: [Emmet in Visual Studio Code](#)

## Editors (4)

Examples of Emmet abbreviations:

Abbreviation	Expanded to
bd	<code>border: 1px solid #000;</code>
bdt	<code>border-top: 1px solid #000;</code>
bgc	<code>background-color: #fff;</code>
fwb	<code>font-weight: bold;</code>
p10	<code>padding: 10px;</code>
p1e	<code>padding: 1em;</code>
mla	<code>margin-left: auto;</code>
mra	<code>margin-right: auto;</code>
pt.5e	<code>padding-top: 0.5em;</code>
tac	<code>text-align: center;</code>
tar	<code>text-align: right;</code>
df	<code>display: flex;</code>
w50%	<code>width: 50%;</code>

# CSS Frameworks

Minimalistic CSS frameworks:

- Milligram (license: MIT License) <https://milligram.io/>  
<https://github.com/milligram/milligram>
  - Examples: <https://milligram.io/#examples>
- Pico CSS (license: MIT License) <https://picocss.com/>  
<https://github.com/picocss/pico>
  - Examples: <https://picocss.com/examples>
- Simple.css (license: MIT License) <https://simplecss.org/>  
<https://github.com/kevquirk/simple.css>
  - Examples: <https://simplecss.org/demo>

For more, see: <https://github.com/troxler/awesome-css-frameworks>

# Other Tools

## Minifier tools:

- cssnano (written in: CSS/JavaScript; license: MIT License)  
<https://cssnano.github.io/cssnano/>  
<https://github.com/cssnano/cssnano>
- CSSO (written in: CSS/JavaScript; license: License MIT)  
<https://css.github.io/csso/> <https://github.com/css/csso>
- Visual Studio Code:
  - MinifyAll (written in: TypeScript; license: GPLv3) <https://marketplace.visualstudio.com/items?itemName=josee9988.minifyall>  
<https://minifyall.jgracia.es/> <https://github.com/Josee9988/MinifyAll>

# Online CSS Sandboxes

- [CodePen](#)
- [JSFiddle](#)
- [Liveweave](#)

# File Properties

- File extension: `.css`
- IANA media type: `text/css`

# Characters

CSS uses the Unicode character set.

# Escape Sequences (1)

- Unicode characters can be specified with escape sequences of the form `\hhhhh` where `hhhhh` is a sequence of one to six hexadecimal digits representing the code point of the Unicode character.
  - If the number of hex digits is less than six and a character in the range `[0-9a-fA-F]` follows the hexadecimal number, then a whitespace character must end the escape sequence.
  - A whitespace character that immediately follows an escape sequence will be ignored.
- Examples:
  - `\9`, `\09`, ..., `\000009` denote the vertical tab character.
  - `\A9`, `\0A9`, ..., `\0000A9` denote the copyright symbol (©).

## Escape Sequences (2)

- Special characters can be escaped with a `'\'` character.
- For example, it is required when a selector contains an element name that contains `'.'` characters.
  - For example, an element named `given.name` matches the selector `given\.name` and does not match the selector `given.name`.

# Character Encoding

- Character encoding is determined by the following:
  - A byte order mark (BOM).
  - Otherwise, the value of the `charset` parameter in the `Content-Type` HTTP header field.
  - Otherwise, the `@charset "encoding";` at-rule at the very beginning of the stylesheet.
    - Example: `@charset "UTF-16";`
  - Otherwise, UTF-8 is the default character encoding.
- See: [Declaring character encodings in CSS \(W3C\)](#)

# Comments

- Comments can be placed between the `/*` and `*/` delimiters.
  - Example: `/* Style sheet for index.html */`
- Can occur anywhere but in tokens.
- Cannot be nested.

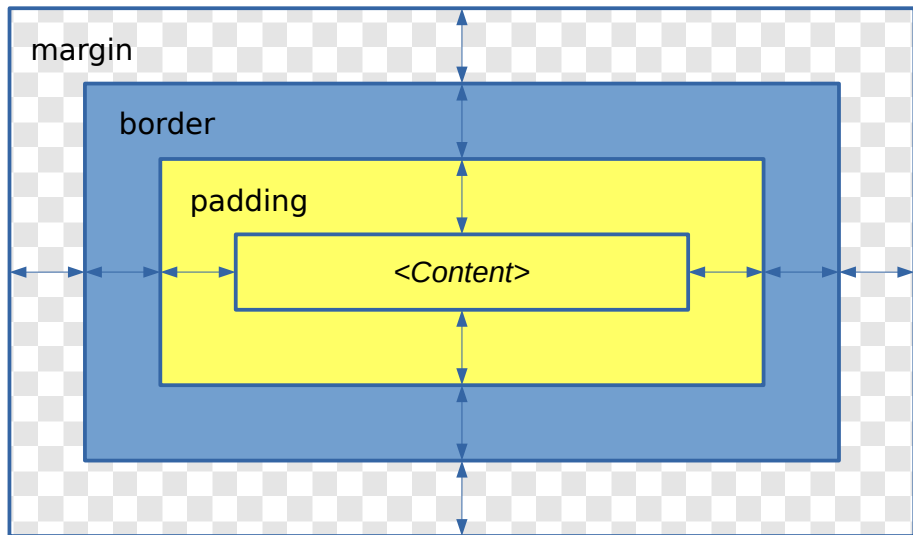
# Box Model (1)

Relevant specification: [CSS Display Module Level 3 \(W3C Candidate Recommendation Snapshot, 30 March 2023\)](#)

## Box Model (2)

- CSS takes a source document, organized as a tree, and renders it onto a canvas (such as the screen) generating an intermediary structure, the **box tree**, which represents the formatting structure of the rendered document.
- Each box in the box tree represents a corresponding element (or pseudo-element) from the document in space and/or time on the canvas.
- For each element, CSS generates zero or more boxes as specified by that element's `display` property.
  - Typically, an element generates a single box.

## Box Model (3)



# Properties (1)

- Parameters defined by CSS for controlling the rendering of documents.
- Properties have names and values.
- The list of CSS properties: [List of CSS properties \(W3C\)](#)
  - The total number of distinct property names is 602.

## Properties (2)

Shorthand property:

- A property that allows authors to specify the values of several properties simultaneously.
- For example, the `margin` property is a shorthand property for setting the value of the `margin-top`, `margin-right`, `margin-bottom` and `margin-left` properties.

# Declaration Blocks

- Start with a '{' character and end with a '}' character, in between there must be a list of zero or more declarations.
- Declarations are of the form *name: value*, where whitespace characters are allowed before and after tokens.
- In a declaration *name* is an identifier..
- Declarations must end with a ';' character.
- The ';' character after the last declaration can be omitted.

## Rule Sets (Style Rules)

Consist of a selector (or a list of selectors separated with a ' , ' character) followed by a declaration block.

# At-rules

- Define special processing rules or values.
- Start with a '@' character, followed by an identifier and includes everything up to the next ';' character, or the next declaration block.
- Examples: @charset, @import, @namespace, @media
- See: <https://developer.mozilla.org/en-US/docs/Web/CSS/At-rule>

# Values (1)

- Relevant specification: [CSS Values and Units Module Level 3 \(W3C Candidate Recommendation Draft, 22 March 2024\)](#)
- See also: [CSS values and units \(MDN\)](#)

## Values (2)

A property value can have one or more components:

- Identifiers (e.g., `none`, `inherit`)
- Strings (e.g., `'Hello, world!'`)
- URLs (e.g., `url(images/item.png)`)
- Numbers (e.g., `123`, `3.141593`)
- Percentages (e.g., `150%`)
- Dimensions (e.g., `10px`, `0.5em`, `45deg`)
- Colors (e.g., `fuchsia`, `#f0f`, `#ff00ff`, `rgb(255,0,255)`)
- Functional notations (e.g., `attr(name)`, `calc(100% - 50px)`, `counter(chapter-number)`)
- ...

## Values (3)

- For each property, CSS specifications define the type, number, and ordering of components allowed in property values.
  - Examples:
    - `text-transform: none | capitalize | uppercase | lowercase | full-width`
    - `letter-spacing: normal | <length>`
    - `border-color: <color>{1,4}`
    - `border-top: <line-width> || <line-style> || <color>`

# Identifiers

- Keywords: are ASCII case-insensitive (i.e., the [a-z] and [A-Z] characters are equivalent).
- Author-defined identifiers:
  - Used, for example, for names of counters.
  - Are case-sensitive.
- Syntax:
  - See: <https://www.w3.org/TR/css-syntax-3/#ident-token-diagram>

# Strings

- Sequences of characters delimited by `'` or `"` characters.
- Cannot contain the delimiter character directly.
  - Delimiters in strings must be escaped with a `\` character.
    - Example: `'It\'s So Cool'`, `"\"Good morning, Frank,\" said Hal."`
- Newlines can be specified only with the `\A` escape sequence.

# URL-s

- Examples:

- `list-style-image: url("http://eg.com/images/item.png")`
- `list-style-image: url('http://eg.com/images/item.png')`
- `list-style-image: url(http://eg.com/images/item.png)`

- Relative references can be used and are resolved against the URL of the stylesheet.

- Example: `list-style-image: url("images/item.png")`

- In some contexts, e.g., in `@import` at-rules, the `url()` function can be omitted and a bare string can be used instead.

- Example: `@import "default.css";`

# Numbers

- CSS uses two types of numbers:
  - **Integers**: consist of an optional sign and one or more decimal digits.
    - Examples: 0, 255, -1, +10946
  - **Real numbers**: consist of an optional sign, one or more decimal digits, an optional dot, optionally followed by an 'e' or 'E' character and an integer.
    - At least one digit is required after the dot.
    - Integers are special cases of real numbers.
    - Examples: 1.5, -100.0, +.25, 1E-10
- Properties may restrict numeric values to some range.
  - For example, they can exclude negative numbers.

# Percentages (1)

- Consist of an integer or a real number immediately followed by a '%' character.
- Percentage values are always relative to another quantity.
  - Each property that allows percentages also defines the quantity to which the percentage refers.
  - This quantity can be a value of another property for the same element, the value of a property for an ancestor element, or something else.

## Percentages (2)

Examples:

```
sup {  
  font-size: 75%;  
}
```

```
nav {  
  width: 40%;  
}
```

# Dimensions

- Consist of an integer or a real number immediately followed by a unit.
  - Units are ASCII case-insensitive identifiers.
- Each dimension represents a quantity, such as length, or angle.
- For zero lengths the unit can be omitted.

# Lengths (1)

- There are two types of length units:
  - **Relative:** specify a length relative to another length.
    - em, ex, ch, rem, vw, vh, vmin, vmax
  - **Absolute:** fixed in relation to each other and anchored to some physical measurement.
    - cm, mm, Q, in, pc, pt, px
- Examples: 0.5em, 1.5cm, 0px, 0, -1in
- See: <https://developer.mozilla.org/en-US/docs/Web/CSS/length>

## Lengths (2)

Relative units:

- `em`: equals to the computed value of the `font-size` property of the element on which it is used (in case of the `font-size` property itself it refers to the computed `font-size` of the parent element).
- `ex`: equals to the x-height of the element's font (in case of the of the `font-size` property itself it refers to the x-height of the parent element).
  - It is so called because it is often equal to the height of the lowercase `x` character.

# Colors (1)

Relevant specification: [CSS Color Module Level 3 \(W3C Recommendation, 18 January 2022\)](#)

## Colors (2)

- The 16 basic color keywords: aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, purple, red, silver, teal, white, yellow.
- Modern web browsers generally support several other color keywords, see:
  - <https://www.w3.org/TR/css-color-3/#svg-color>
  - <https://developer.mozilla.org/en-US/docs/Web/CSS/named-color>

## Colors (3)

RGB color model:

- For example, all of the following specify the same color:
  - `lightslategrey`
  - `#778899`
  - `#789` (each hexadecimal digit is doubled)
  - `rgb(119,136,153)`

## Colors (4)

RGB color model:

- The `rgb()` functional notation also accepts three percentages as arguments, where the integer value 255 corresponds to 100%.
- For example, the following colors are equivalent:
  - `#ffffff`
  - `#fff`
  - `rgb(255,255,255)`
  - `rgb(100%,100%,100%)`
  - `white`

# Selectors (1)

- Relevant specification: [Selectors Level 3 \(W3C Recommendation, 6 November 2018\)](#)
- Selectors express pattern matching rules.
- They determine which style rules apply to elements.

## Selectors (2)

- Selectors are case-insensitive within the ASCII range (i.e., the characters [a-z] and [A-Z] are equivalent), except for parts that are not under the control of CSS (element names, attribute names and values).
  - The case sensitivity of document language element names, attribute names, and attribute values in selectors depends on the document language
- For example, element names are case-sensitive in XHTML and XML.

## Selectors (3)

- Simple selector: either a type selector, universal selector, attribute selector, class selector, ID selector, or pseudo-class.
- Examples:
  - `div`
  - `[rel=stylesheet]`
  - `.copyright`
  - `#logo`
  - `:lang(hu)`

## Selectors (4)

Combinator: any of the following characters:

- *whitespace*
- '>'
- '+'
- '~'

## Selectors (5)

Sequence of simple selectors:

- A chain of simple selectors that are not separated by a combinator.
- Begins with a type selector or a universal selector. No other type selector or universal selector is allowed in the sequence.
- Examples:
  - `div`
  - `h2#status`
  - `link[rel=stylesheet] [type=text/css]`
  - `p.copyright`
  - `*:lang(hu)`
  - `tr:nth-child(odd)`
  - `li:not(:last-child)`

# Selectors (6)

Selector:

- A chain of one or more sequences of simple selectors separated by combinators.
- One pseudo-element may be appended to the last sequence of simple selectors.
- Examples:
  - `p`
  - `a img`
  - `h1 ~ table`
  - `div#main > h1`
  - `div > h1 + p`
  - `q::before`

# Type Selector

- Written as a CSS qualified name, generally, an identifier.
- Matches the elements of that name.
- Examples:

```
p { color: red }
```

```
a { text-decoration: none }
```

# Universal Selector

- Written as `*`.
- Matches all elements.
- May be omitted from a simple selector, if it is not the only component.
  - Thus, for example, the following selectors are equivalent:
    - `*#nav` and `#nav`
    - `*.important` and `.important`
    - `*[title]` and `[title]`
  - It is recommended not to omit the `*`, because it improves readability.
    - For example, the selector `div *:first-child` is more readable than the selector `div :first-child`, because it is less likely to confuse with the selector `div:first-child`.

# Attribute Selectors (1)

- `[att]`: matches elements with the `att` attribute.
- `[att=val]`: matches elements with the `att` attribute whose value is exactly *val*.
- `[att~=val]`: matches elements with the `att` attribute whose value is a whitespace-separated list of words, one of which is exactly *val*.

## Attribute Selectors (2)

[att|=val]:

- Matches elements with the `att` attribute, its value either being exactly *val* or beginning with *val-*.
- Primarily intended to be used for attributes whose value is a language tag, for example, `en-US`, `en-GB`.
- For the `xml:lang` attribute and the HTML `lang` attribute the `:lang(C)` pseudo-class should be used.

## Attribute Selectors (3)

Additional attribute selectors introduced by CSS3:

- `[att^=val]`: matches elements with the `att` attribute whose value begins with the prefix *val*.
- `[att$=val]`: matches elements with the `att` attribute whose value ends with the suffix *val*.
- `[att*=val]`: matches elements with the `att` attribute whose value contains the substring *val*.

## Attribute Selectors (4)

- In each attribute selector, *val* must be a CSS identifier or a string.
- Thus, for example, the `[dir=rtl]`, `[dir='rtl']`, and `[dir="rtl"]` selectors are equivalent.

## Attribute Selectors (5)

Examples:

```
style[type=italic] {  
  font-style: italic;  
}
```

```
style[type=bold] {  
  font-weight: bold;  
}
```

```
style[type=normal] {  
  font-style: normal;  
  font-weight: normal;  
}
```

```
a[hreflang|=en] {  
  text-decoration: line-through;  
}
```

# Class Selector

- For HTML documents, the attribute selector `[class~=val]` is equivalent with the selector `.val`.
- Examples:

```
div.centered {  
  margin-left: auto;  
  margin-right: auto;  
}
```

```
.important {  
  color: red;  
  text-decoration: underline;  
}
```

# ID Selector (1)

- A selector of the form *#identifier* matches the element with that identifier.
- The identifier must be provided by an attribute of type ID in the document.
  - The name of such an attribute depends on the document language, for example, in HTML the attribute is named `id`.

## ID Selector (2)

- Examples:

```
div#main {  
  width: 50%;  
  margin-left: auto;  
  margin-right: auto;  
}
```

```
#footer {  
  text-align: center;  
}
```

# Pseudo-classes (1)

- Selectors of the form `:identifier` or `:identifier(value)`.
- Pseudo-class names are case-insensitive.
- Permit selection based on information that lies outside of the document or that cannot be expressed using the other simple selectors.
- Some pseudo-classes are mutually exclusive (e.g., `:link` and `:visited`).
- Some pseudo-classes are dynamic: an element may acquire or lose a dynamic pseudo-class while a user interacts with the document.
  - The two types of dynamic pseudo-classes: link pseudo-classes and user action pseudo-classes.

## Pseudo-classes (2)

Link pseudo-classes:

- `:link`: applies to links that have not yet been visited.
- `:visited`: applies to links that have been visited.
  - For privacy reasons, a very limited set of CSS properties can be set via the `:visited` pseudo-class.
  - Moreover, for the same reasons, modern web browsers lie about the values of the CSS properties of visited links when they are queried from JavaScript via the `Window.getComputedStyle()` method.
  - See:
    - [:visited \(MDN\)](#)
    - [Privacy and the :visited selector \(MDN\)](#)
    - [Window: getComputedStyle\(\) method \(MDN\)](#)

## Pseudo-classes (3)

User action pseudo-classes:

- `:hover`: applies while the user designates an element with a pointing device, but does not necessarily activate it.
- `:active`: applies while an element is being activated by the user.
  - For example, between the times the user presses the mouse button and releases it.
- `:focus`: applies while an element has the focus (accepts keyboard or mouse events, or other forms of input).

## Pseudo-classes (4)

Example for using dynamic pseudo-classes:

```
a:link {  
  color: indigo;  
  text-decoration: none;  
}
```

```
a:visited {  
  color: gray;  
  text-decoration: line-through; /* Won't work */  
}
```

```
a:hover {  
  border-width: medium;  
  border-style: solid;  
}
```

```
a:active {  
  font-weight: bolder;  
}
```

## Pseudo-classes (5)

The `:target` pseudo-class:

- When the URI of the document contains a fragment identifier, `:target` represents the element referred by the fragment.
- For example, if `index.html#copyright` is the URI of the document, then the selector `div:target` represents the `div` element with the ID `copyright`.

## Pseudo-classes (6)

The `:lang(C)` pseudo-class:

- Represents elements that are in language `C`, where `C` is a CSS identifier (language code).
- Examples:
  - `:lang(en)`, `:lang(en-GB)`, `:lang(en-US)`
  - `:lang(hu)`
- In XML documents, the language used is defined by the `xml:lang` attribute. In HTML documents, the language can be specified with both the `lang` and the `xml:lang` attributes.
  - The `lang` and `xml:lang` attributes are not necessarily occur directly on an element.

## Pseudo-classes (7)

Example for using the `:lang(C)` pseudo-class: using Hungarian quotation marks for quotations in Hungarian

```
q:lang(hu) {  
  quotes: "„" "”" "»" "«";  
}
```

## Pseudo-classes (8)

UI element states pseudo-classes:

- `:enabled`: represents user interface elements that are in an enabled state.
- `:disabled`: represents user interface elements that are in a disabled state.
- `:checked`: represents user interface elements that are in a “checked” state (for example, radio buttons, checkboxes).

# Pseudo-classes (9)

## Structural pseudo-classes:

- Only elements are counted when calculating the position of an element in its list of siblings.
  - For example, text, comments, and processing instructions are ignored.
- Index numbering starts at 1.

## Pseudo-classes (10)

### Structural pseudo-classes:

- `:root`: represents the root element of the document.
- `:only-child`: represents elements that have no siblings.
- `:only-of-type`: represents elements that have no siblings with the same element name.
- `:empty`: represents elements that have no children at all.
  - These elements do not contain any text or elements, but may contain comments and processing instructions.

## Pseudo-classes (11)

Structural pseudo-classes (continued):  $a$  and  $b$  are integers

- `:nth-child( $a$ + $b$ )`: represents elements that have  $a$  $n$  +  $b$  - 1 siblings before it in the document, for any non-negative integer value of  $n$ .
- `:nth-last-child( $a$ + $b$ )`: represents elements that have  $a$  $n$  +  $b$  - 1 siblings after it in the document, for any non-negative integer value of  $n$ .
- `:nth-of-type( $a$ + $b$ )`: represents elements that have  $a$  $n$  +  $b$  - 1 siblings with the same element name before it in the document, for any non-negative integer value of  $n$ .
- `:nth-last-of-type( $a$ + $b$ )`: represents elements that have  $a$  $n$  +  $b$  - 1 siblings with the same element name after it in the document, for any non-negative integer value of  $n$ .

## Pseudo-classes (12)

### Structural pseudo-classes (continued):

- `:nth-child()`, `:nth-last-child()`, `:nth-of-type()` és `:nth-last-of-type()`:
  - The argument  $an+0$  can be specified as  $an$  in short.
  - The argument  $0n+b$  can be specified as  $b$  in short.
  - For a negative  $b$  the argument must be specified as  $an-b$ , the form  $an+-b$  is invalid.
  - The argument  $2n$  can be specified as `even`, the argument  $2n+1$  as `odd`, respectively.

## Pseudo-classes (13)

### Structural pseudo-classes (continued):

Selector	Meaning
<code>tr:nth-child(2n+0)</code> <code>tr:nth-child(2n)</code> <code>tr:nth-child(even)</code>	The even rows of a table
<code>tr:nth-child(2n+1)</code> <code>tr:nth-child(2n-1)</code> <code>tr:nth-child(odd)</code>	The odd rows of a table
<code>tr:nth-child(5)</code>	The the fifth row of a table
<code>tr:nth-child(5n+1)</code> <code>tr:nth-child(5n-4)</code>	The 1st, 6th, 11th, ... rows of a table
<code>tr:nth-child(-n+5)</code> <code>tr:nth-last-child(-n+5)</code>	The first five rows of a table The last five rows of a table

## Pseudo-classes (14)

### Structural pseudo-classes (continued):

- `:first-child`: has the same meaning as `:nth-child(1)`
- `:last-child`: has the same meaning as `:nth-last-child(1)`
- `:first-of-type`: has the same meaning as `:nth-of-type(1)`
- `:last-of-type`: has the same meaning as `:nth-last-of-type(1)`

## Pseudo-classes (15)

The negation pseudo-class:

- A selector of the form `:not(X)` , where the argument is a simple selector (excluding the negation pseudo-class itself).
- Matches elements that do not match the argument.
- Examples:
  - `:not(.important)`
  - `:not([title])`
  - `li:not(:last-child)`
  - `a:not(:hover)`

## Pseudo-classes (16)

The negation pseudo-class (continued):

- Only the argument is taken into account when calculating specificity!
- For example, the selector `h1:not(h2)` is practically equivalent to the selector `h1`, but it has a higher specificity:  $(a = 0, b = 0, c = 2)$ .

# Pseudo-elements (1)

- Pseudo-elements allow authors to refer to content in the document that is otherwise inaccessible.
  - This includes content that originally does not exist in the document (i.e., generated content).
- Only one pseudo-element may appear per selector.
- The following pseudo-elements are available:
  - `::first-letter`
  - `::first-line`
  - `::before`, `::after`
- CSS3 uses `::` instead of `:` for pseudo-elements to distinguish them from pseudo-classes.
- For compatibility `:` can be used also.

## Pseudo-elements (2)

`::first-line`: represents the first formatted line of an element.

- Example:

```
p::first-line {  
  text-decoration: underline  
}
```

## Pseudo-elements (3)

`::first-letter`: represents the first letter or digit of an element, if it is not preceded by any other content (such as images) on its line.

- Also applies to any punctuation character that precedes or follows the first letter.
- Example:

```
p::first-letter {  
  font-size: 2em;  
  font-weight: bold;  
}
```

## Pseudo-elements (4)

`::before` and `::after`: they can be used to describe generated content before or after an element's content.

- Example:

```
div.proof::before {  
  content: "Proof: ";  
  font-weight: bold;  
}
```

```
div.proof::after {  
  content: "\220E" /* End of proof */  
}
```

# Combinators (1)

Descendant combinator:

- Whitespace that separates two sequences of simple selectors.
- If P and Q are two sequences of simple selectors, the selector P Q represents elements matching Q that are descendants of elements matching P.
- Example:

```
thead th {  
  background-color: darkgrey;  
}
```

## Combinators (2)

Child combinator:

- A '>' character that separates two sequences of simple selectors.
- If P and Q are two sequences of simple selectors, the selector P > Q represents elements matching Q that are children of elements matching P.
- Example:

```
nav > div {  
  display: inline;  
}
```

```
p > img:only-child {  
  margin-left: 0;  
}
```

## Combinators (3)

Next-sibling combinator:

- A '+' character that separates two sequences of simple selectors.
- If P and Q are two sequences of simple selectors, the selector P + Q represents elements matching Q that immediately follow an element matching P.
  - The elements represented by the two sequences share the same parent in the document.
  - Non-element constructs (e.g., text and comments) are ignored when considering adjacency of elements.
- Example:

```
h1 + p {  
  text-indent: 0;  
}
```

## Combinators (4)

### Subsequent-sibling combinator (CSS3):

- A '~' character that separates two sequences of simple selectors.
- If P and Q are two sequences of simple selectors, the selector P ~ Q represents elements matching Q that (not necessarily immediately) follow an element matching P.
- The elements represented by the two sequences share the same parent in the document.
- Example:

```
img ~ span {  
  color: red;  
}
```

# Grouping Rules (1)

- Rules with the same declarations can be grouped together.
- The selector of the grouped rule contains the selectors of the original rules separated by ' , ' characters.

## Grouping Rules (2)

For example, the first three rules are equivalent to the fourth one:

```
h1 + p { text-indent: 0 }
```

```
h2 + p { text-indent: 0 }
```

```
h3 + p { text-indent: 0 }
```

```
h1 + p, h2 + p, h3 + p {  
  text-indent: 0;  
}
```

# Selectors Level 4 (1)

- Specification: [Selectors Level 4 \(W3C Working Draft, 11 November 2022\)](#)
- Changes since Level 3:  
<https://www.w3.org/TR/selectors-4/#changes-level-3>
- New features:
  - `:any-link` and `:local-link` pseudo-classes
  - `:has()` pseudo-class
  - `:is()` pseudo-class
  - `:not()`: takes a selector list as an argument
  - ...

## Selectors Level 4 (2)

`:is()` pseudo-class:

- A selector of the form `:is(X)` where the argument is a selector list.
- Also called matches-any pseudo-class.
- Matches any element that can be matched by any of the selectors in the list argument.
- Useful for representing long selector lists in a more compact form.
- Pseudo-elements are not valid within `:is()`.
- The specificity of the `:is()` selector is the specificity of selector with the highest specificity in the list argument.
- Browser support: <https://caniuse.com/css-matches-pseudo>

## Selectors Level 4 (3)

`:is()` pseudo-class: for example, the following two style rules are equivalent:

```
div:is(.note, .warning, .hint)::before {  
    /* ... */  
}
```

```
div.note::before,  
div.warning::before,  
div.hint::before {  
    /* ... */  
}
```

# Other Uses of CSS Selectors

## Web scraping:

- Free and open source software:
  - Scrapy (written in: Python; license: New BSD License)  
<https://scrapy.org/> <https://github.com/scrapy/scrapy>
  - Parsel (written in: Python; license: New BSD License)  
<https://parsel.readthedocs.io/> <https://github.com/scrapy/parsel>

# Specificity (1)

- For each selector and declaration a specificity is calculated, which is a vector  $(a, b, c)$  of length 3, where  $a$ ,  $b$ , and  $c$  are non-negative integers.
- The vectors are sorted in lexicographic order.

## Specificity (2)

A selector's specificity is calculated as follows: the specificity of the selector is a vector  $(a, b, c)$ , where:

- $a$  is the number of ID selectors in the selector.
- $b$  is the number of class selectors, attribute selectors, and pseudo-classes in the selector.
  - Selectors inside the negation pseudo-class are counted like any other, but the negation itself does not count as a pseudo-class!
- $c$  is the number of type selectors and pseudo-elements in the selector.

## Specificity (3)

Specificity of a declaration:

- Each declaration has the same specificity as of the selector of the style rule it appears in.
- Declarations that do not belong to a style rule (such as the contents of a `style` attribute in HTML) are considered to have a specificity higher than any selector.

# Calculating a Selector's Specificity (1)

- The specificity of the selector `*` is  $(a = 0, b = 0, c = 0)$ .
  - Because it does not contain any ID selectors, class selectors, attribute selectors, pseudo-classes, type selectors, or pseudo-elements.
- The specificity of the selector `div` is  $(a = 0, b = 0, c = 1)$ .
- The specificity of the selector `p::first-letter`, `a img`, and `h1 + p` is  $(a = 0, b = 0, c = 2)$ .

## Calculating a Selector's Specificity (2)

- The specificity of the selector `div[class=nav]` and the equivalent selector `div.nav` is  $(a = 0, b = 1, c = 1)$ .
- The specificity of the selector `#main *:lang(en)` is  $(a = 1, b = 1, c = 0)$ .

## Calculating a Selector's Specificity (3)

The selectors from the examples in increasing order of specificity:

Selector	Specificity
*	$(a = 0, b = 0, c = 0)$
div	$(a = 0, b = 0, c = 1)$
p::first-letter	$(a = 0, b = 0, c = 2)$
a img	
h1 + p	
div[class=nav]	$(a = 0, b = 1, c = 1)$
div.nav	
#main *:lang(en)	$(a = 1, b = 1, c = 0)$

# Stylesheets of Different Origins

- Multiple stylesheets of different origins may belong to a document.
- Stylesheets may have three different origins:
  - User agent
  - User
  - Author
- Origins are listed in increasing order of precedence.

# User agent stylesheets (1)

- User agents must apply a default stylesheet.
- For example, they provide a style rule specifying that the content of the HTML `em` element is presented using an italic font.

## User agent stylesheets (2)

- **Firefox** : the default stylesheet can be accessed as follows:
  - Type the following in the address bar: <resource://gre-resources/>
  - As a result, a directory is presented, in which the `html.css` file is the default stylesheet.
  - See:  
<https://searchfox.org/mozilla-central/source/layout/style/res/html.css>
- **Chromium, Google Chrome** : the default stylesheet to render HTML documents: <https://chromium.googlesource.com/chromium/blink/+ /master/Source/core/css/html.css>

# User Stylesheets

- The user may be able to specify style information for a particular document.
- This is important to people with disabilities.
- Related concept: web accessibility <https://www.w3.org/WAI/>
- The Stylish extension for the Chromium, Google Chrome, Firefox, and Opera browsers provides such functionality: <https://userstyles.org/>

# Author Stylesheets (1)

- In HTML, the `link` element associates an external stylesheet with the document.

- Example:

```
<link rel="stylesheet" href="style.css">
```

- In HTML, the `style` element allows the direct embedding of style rules in the document.

- Example:

```
<style>  
  h1, h2, h3, h4, h5, h6 { font-variant: small-caps }  
</style>
```

## Author Stylesheets (2)

- In XML, the `xml-stylesheet` processing instruction associates an external stylesheet with the document.
- The processing instruction should occur before the root element of the document.
- Example:

```
<?xml-stylesheet type="text/css" href="style.css"?>
```

- See: [Associating Style Sheets with XML documents 1.0 \(Second Edition\)](#) (W3C Recommendation, 28 October 2010)

# Importing Stylesheets

- The `@import` at-rule allows users to import style rules from other stylesheets.
- The URL of the stylesheet can be specified with the `url()` function or as a string.
- If an `@import` at-rule refers to a valid stylesheet, user agents must treat the contents of the stylesheet as if they were written in place of the `@import` at-rule.
- Any `@import` at-rules must precede all other at-rules and style rules with the exception of the `@charset` at-rule.
- Example:

```
@import url(https://fonts.googleapis.com/css?family=Tangerine);  
@import "default.css";
```

# Important Declarations (1)

- A declaration can be followed by the token '!' token and keyword `important`.
- Example:  

```
color: red !important
```
- Such a declaration overrides any other normal declarations.
- See: [Specificity – The !important exception \(MDN\)](#)

## Important Declarations (2)

- By default, rules in author stylesheets have higher precedence than rules in user stylesheets.
- Both author and user stylesheets may contain `!important` declarations, and user `!important` rules override author `!important` rules.

## Important Declarations (3)

Example:

```
<html lang="en">
  <head>
    <title>!important</title>
    <meta charset="UTF-8">
    <style>
      p {
        background-color: cornsilk;
        color: green !important;
      }
    </style>
  </head>
  <body>
    <p style="background-color: aliceblue; color: red">
      Hello, World!
    </p>
  </body>
</html>
```

## Important Declarations (4)

Practical example:

```
table.chessboard > tbody > tr:nth-child(odd) > td:nth-of-type(odd) {  
    background-color: white;  
}
```

```
table.chessboard > tbody > tr:nth-child(odd) > td:nth-of-type(even) {  
    background-color: lightgray;  
}
```

```
table.chessboard > tbody > tr > td:hover {  
    background-color: salmon !important;  
}
```

Without `!important`, the third rule would never be used.

# Calculating Property Values

The final value of a CSS property for a given element or box is the result of a multi-step calculation:

- **Cascaded value:** the result of the cascade.
- **Specified value:** the result of the defaulting.
- **Computed value:** the result of transforming a relative value to an absolute one.
- **Used value:** the result of formatting the document.
- **Actual value:** the result of transforming the used value to be suitable for a display environment.

# Cascading, Inheritance, and Initialization

- **Cascading**: conflict resolution mechanism that is applied when different declarations set a value for the same element/property combination.
- **Inheritance, initialization**: mechanisms that are applied when no declarations try to set a the value for an element/property combination.
- Relevant specification: [CSS Cascading and Inheritance Level 3 \(W3C Recommendation, 11 February 2021\)](#)

# Cascade (1)

- Several different declarations can set the value of the same variable.
- These declarations can be of different origins.
- The cascade is the process during which it is determined which declaration sets the value of a given property of a given element.

## Cascade (2)

The cascade determines the cascaded value of a property for an element as follows:

- The declarations that set the value of a property for the element are identified and collected.
  - Remember that these declarations are provided by style rules or the `style` HTML attribute.
- These declarations are sorted by decreasing origin.
- Declarations of the same origin are sorted by decreasing specificity.
  - In case of equal specificity the last declaration in document order wins.
- The value of the property is set by the first declaration according to this order.

# Rule Ordering (1)

- The ordering of style rules may be significant.
- The ordering is significant when more than one style rule with the same specificity applies to an element.
- In this case, the last declaration in document order wins.

## Rule Ordering (2)

- For example, the first two rules will never be taken into account here:

```
a:active { color: red } /* (a = 0, b = 1, c = 1) */  
a:hover { color: green } /* (a = 0, b = 1, c = 1) */  
a:visited { color: black } /* (a = 0, b = 1, c = 1) */  
a:link { color: blue } /* (a = 0, b = 1, c = 1) */
```

- The correct order of these rules is the following (the first two rules are interchangeable):

```
a:visited { color: black }  
a:link { color: blue }  
a:hover { color: green }  
a:active { color: red }
```

# Specified Value

- The specified value is the value of a given property that the style sheet authors intended for that element.
- If the cascade results in a value, then that is the specified value.
- Otherwise, the specified value is determined by defaulting process.

# Defaulting

- When the cascade does not result in a value, the specified value must be found some other way.
  - Inherited properties draw their defaults from their parent element through inheritance.
  - All other properties take their initial value.
- Authors can explicitly request inheritance or initialization via the `inherit` and `initial` keywords.

# Computed Value (1)

- In general, a relative value is transformed to an absolute one when the computed value is determined.
- However, the computed value can be a percentage!
  - The result of determining the computed value is a percentage, when the specified value is percentage that is relative to a quantity that depends on the layout of the document.
    - Such percentages are transformed to absolute values when the used value is determined.
- The computed value is the value that is transferred from parent to child during inheritance.

## Computed Value (2)

Example:

- CSS:

```
section { font-size: 12px }  
h1 { font-size: 1.5em }
```

- HTML:

```
<section>  
  <h1>Introduction</h1>  
</section>
```

- The specified and the computed value of the `font-size` property of the `section` element is `12px`.
- The specified value of the `font-size` property of the `h1` element is `1.5em`, it's computed value is  $1.5 \times 12\text{px} = 18\text{px}$ .

# Used Value

- Computed values are determined without formatting the document.
- However, there are properties whose value may depend on the layout of the document (e.g., `height`, `width`).
  - For such properties the used value is determined from the computed value based on the layout of the document.
  - For all other properties, the used value is the same as the computed value.

# Actual Value

- The used value is transformed to the actual value based on constraints of the display environment.
- For example, the value of the `font-size` property may need adjustment based on the availability of fonts.

# Inheritance (1)

- Inheritance propagates property values from parent elements to their children.
- Some properties are inherited, this means that, unless the cascade results in a value, the value will be determined by inheritance.
- For each property, the relevant CSS specification defines whether it is inherited or not.
  - For example, the `font-family`, `font-style`, `font-variant`, `font-weight`, `font-size`, and `font` properties are inherited.
  - For example, the `margin-top`, `margin-bottom`, `margin-right`, `margin-left`, `margin`, and `width` properties are not inherited.
- If the cascaded value of a property is the `inherit` keyword, the inherited value becomes the property's specified and computed values.

## Inheritance (2)

Example:

When no rules apply to the `em` HTML element below that set the value of the `color` property, the element inherits the value of the property from the `span` element (i.e., `red`).

```
<span style="color: red">
```

```
  May the <em>force</em> be with you!
```

```
</span>
```

## Inheritance (3)

Example:

When the style rule below applies, a hyperlink inherits the value of the `color` property from its parent. Thus, for example, the value of the `color` property of the `a` element in the `div` element is green.

```
a:visited, a:link {  
  color: inherit;  
}
```

```
<p style="color: green">  
  Click here: <a href="https://www.w3.org/">W3C</a>  
</p>
```

# Initial Value

- Each property has an initial value, defined in the relevant CSS specification. The initial value can be defined to depend on the user agent.
  - For example, the initial value of the `background-color` property is the `transparent` keyword.
    - See: <https://www.w3.org/TR/css-backgrounds-3/#the-background-color>
  - For example, the initial value of the `color` property depends on the user agent.
    - See: <https://www.w3.org/TR/css-color-3/#foreground>
- If the property is not an inherited property, and the cascade does not result in a value, then the specified value of the property is its initial value.
- If the cascaded value of a property is the `initial` keyword, the property's initial value becomes its specified value.

# Assignment (1)

- Consider the following style rules in an author stylesheet associated with a HTML document:

```
:lang(hu) { color: red }  
div > p { color: blue }  
#main > p.foo { color: yellow }  
#main :lang(hu) { color: green }  
div:lang(hu) p.foo:first-child { color: black }
```

- What is the color of the text in the p element?

```
<div id="main" lang="hu">  
  <p class="foo">What is the color of this text?</p>  
</div>
```

## Assignment (2)

- Consider the following style rules in an author stylesheet associated with a HTML document:

```
.movies > ul > li:not(:first-child) { color: navy }  
h1 + ul li:first-child { color: gold }  
ul > :nth-child(2n + 1) { color: purple }  
#ot li + :last-child { color: tomato }  
*:not([class]) + ul:first-child > li:first-child { color: silver }
```

- What is the color of the text in each li element?

```
<section id="ot" class="movies">  
  <h1>The Original Trilogy</h1>  
  <ul>  
    <li>A New Hope</li>  
    <li>The Empire Strikes Back</li>  
    <li>Return of the Jedi</li>  
  </ul>  
</section>
```

# Error Handling (1)

See:

- [CSS Syntax Module Level 3 \(W3C Candidate Recommendation Draft, 24 December 2021\)](#)
  - For a high-level overview of error handling, see subsection [2.2. Error Handling](#).
- [CSS error handling \(MDN\)](#)

## Error Handling (2)

- When errors occur in CSS, the parser attempts to recover gracefully, throwing away only the minimum amount of content before returning to parsing as normal.
  - This is because new syntax looks like an error to an old parser.

## Error Handling (3)

- Each invalid construct (i.e., declaration, style rule, at-rule) is ignored by the user agent and parsing is continued.
  - When the selector list of a style rule contains an invalid selector, the entire style rule is ignored by the UA.
    - An exception is when the invalid selector occurs within an `:is` pseudo-class, in this case, only the invalid selector is ignored.
- Invalid, thus ignored, declarations can be seen in DevTools.

## Error Handling (4)

Example:

The style rule

```
p {  
  color: white  
  background color: navy;  
  font-family: sans-serif;  
  unknown: 0;  
  font-size: xxx;  
}
```

is effectively the same as the following style rule:

```
p {  
  font-family: sans-serif;  
}
```

## Error Handling (5)

Example:

- The following style rule is ignored because `:foobar` is not a valid pseudo-class:

```
a:hover, a:foobar {  
  font-weight: bold;  
}
```

- However, the following style rule is not ignored completely:

```
a:is(:hover, :foobar) {  
  font-weight: bold;  
}
```

# Vendor Prefix (1)

- The CSS2.1 specification reserves a prefixed syntax for proprietary and experimental extensions to CSS.
  - A CSS feature is a proprietary extension if it is meant for use in a closed environment accessible only to a single vendor's user agent(s).
  - A CSS feature is considered unstable until its specification has reached the Candidate Recommendation (CR) stage in the W3C process.
- See:
  - [CSS Snapshot 2023 – Implementations of Unstable and Proprietary Features \(W3C\)](#)
  - [https://developer.mozilla.org/en-US/docs/Glossary/Vendor\\_Prefix](https://developer.mozilla.org/en-US/docs/Glossary/Vendor_Prefix)

## Vendor Prefix (2)

- Keywords and property names beginning with '-' or '\_' are reserved for vendor-specific extensions.
  - An initial dash or underscore is guaranteed never to be used in a property or keyword by any current or future level of CSS.
- See: [CSS 2.1 Specification – Syntax and basic data types – Vendor-specific extensions](#)

## Vendor Prefix (3)

Once an unstable feature has stabilized (i.e., its specification reaches the Candidate Recommendation stage), support for the vendor-prefixed syntax should be removed from implementations.

## Vendor Prefix (4)

Notable vendor prefixes include the following:

- Mozilla (Firefox): `-moz-`
- Microsoft (Internet Explorer, Microsoft Edge): `-ms-`
- Apple (Safari): `-webkit-`

## Vendor Prefix (5)

Vendor prefix trend:

- Today, the use of vendor prefixes is considered to be a bad practise and browser vendors are moving away from using them.
- Instead, the recommended practise for unstable features is to put them behind a flag.

## Vendor Prefix (6)

- Chromium does not use vendor prefixes.
- Instead, a browser setting (i.e., `chrome://flags/#enable-experimentalweb-platform-features`) is provided to enable/disable experimental features.
- See: <https://www.chromium.org/blink/developer-faq/#will-we-see-a-chrome-vendor-prefix-now>

# Vendor Prefix (7)

Proprietary extensions to CSS:

- [Mozilla CSS extensions](#)
- [WebKit CSS extensions](#)

## Vendor Prefix (8)

An example for a proprietary extension: `::-moz-list-number`

- Example of use:

```
li::-moz-list-number {  
    font-style: italic;  
    font-weight: bold;  
}
```

## Vendor Prefix (9)

An example for a once unstable feature: `fit-content`

- Relevant CSS specification: [CSS Box Sizing Module Level 4 \(W3C Working Draft, 20 May 2021\)](#)
- Browser support:  
[https://caniuse.com/mdn-css\\_properties\\_width\\_fit-content](https://caniuse.com/mdn-css_properties_width_fit-content)
  - Firefox supports the unprefixed standard form since version 94 released in 2021.

## Vendor Prefix (10)

An example for a once unstable feature: `fit-content`

- A style rule that also works with earlier versions of Firefox:

```
.fit-content {  
  width: -moz-fit-content;  
  width: fit-content;  
}
```

# CSS Object Model (CSSOM)

- Specification: [CSS Object Model \(CSSOM\) \(W3C Working Draft, 26 August 2021\)](#)
- A set of APIs that allows access to and manipulation of style related state information and processes from JavaScript.
- Defines an object model is similar to the DOM.
- Further information: [https://developer.mozilla.org/en-US/docs/Web/API/CSS\\_Object\\_Model](https://developer.mozilla.org/en-US/docs/Web/API/CSS_Object_Model)

# Working with the CSSOM

Example:

Suppose that the

```
body {  
  font-size: 16px;  
}
```

style rule applies to the body element that has no style attribute.

```
console.log(document.body.style.fontSize); // ''  
console.log(window.getComputedStyle(document.body).fontSize); // '16px'  
document.body.style.padding = '1.5em';  
console.log(document.body.style.padding); // '1.5em'  
console.log(window.getComputedStyle(document.body).padding); // '24px'
```

# CSS Houdini (1)

- Houdini is a set of low-level APIs that exposes parts of the CSS engine, giving developers the power to extend CSS by hooking into the styling and layout process of a browser's rendering engine.
- See: <https://developer.mozilla.org/en-US/docs/Web/Houdini>

## CSS Houdini (2)

- Specifications:
  - CSS-TAG Houdini Editor Drafts
- Status and support:
  - Is Houdini ready yet?
  - <https://caniuse.com/?search=houdini>

# CSS Houdini (3)

Further useful links:

- [CSS Houdini Wiki](#)
- [Houdini.how](#)
- [Houdini Spellbook](#)

# Further Resources (1)

## References:

- [MDN Web Docs – CSS](#)
  - [CSS Reference](#)
- [CSS Reference – A free visual guide to CSS](#)
- [CSS Reference](#)
- [DevDocs CSS Reference](#)

## Further Resources (2)

### Surveys:

- [The State of CSS Survey](#)
  - [The State of CSS 2023](#)
- HTTP Archive. [Web Almanac – Part I Chapter 1: CSS](#). 2022.

## Further Resources (3)

Demos and games:

- [CSS Zen Garden](#)
- [CSSBattle](#)
- [CSS Diner](#)
  - Repository: <https://github.com/flukeout/css-diner>
- [Fun and Games with CSS](#)
  - Repository: <https://github.com/rupl/fun-games-css>
- [Unfolding the Box Model](#)
  - Repository: <https://github.com/rupl/unfold>