UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

WEB TECHNOLOGIES — LECTURE 03

CSS: CASCADING STYLE SHEETS

Luigi Libero Lucio Starace, PhD

luigiliberolucio.starace@unina.it
https://luistar.github.io
https://www.docenti.unina.it/luigiliberolucio.starace

PREVIOUSLY, ON WEB TECHNOLOGIES

- We have learned how to write HTML documents
- HTML is concerned with **structure** and **semantics** of documents
- HTML is saying nothing at all on the **appearence** of documents

- An **<em>** element specifies that its content should be emphasized
- It's not saying **how** the emphasizing part should be done
  - Emphasis might be conveyed using *italics*, different colors or backgrounds.

CSS: CASCADING STYLE SHEETS

- A **rule-based, declarative** language for specifying how documents should be presented to users.
- A **stylesheet** is a set of **Rules**, each defined as follows
- The `selector` specifies which HTML elems are affected by the rule
- Rules contain a set of **declarations**, in the form of **property-value pairs**, which specify the style to apply

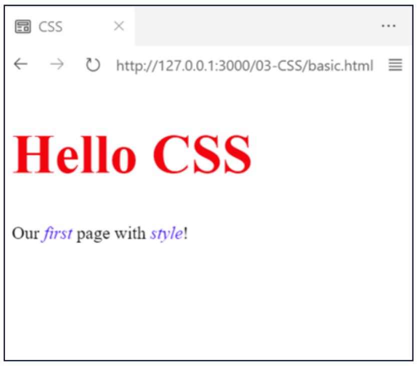
```
selector {
  property: value;
  property: value;
}
```

CSS: FIRST EXAMPLE

```
<h1>Hello CSS</h1>
<p>
  Our <em>first</em> page
  with <em>style</em>!
</p>
```

```
h1 {
  color: red;
  font-size: 50px;
}

em {
  color: blue;
}
```



Hello CSS

Our *first* page with *style*!

DEFAULT USER AGENT STYLES

- But the first web pages we developed have some styling!
  - Headings are bigger and shown with a bold face…
  - **<p>** starts on new lines, **<em>** are displayed in italic, **<strong>** in bold, **<a>** are underlined and blue, **<ul>** have bullets, and so on…

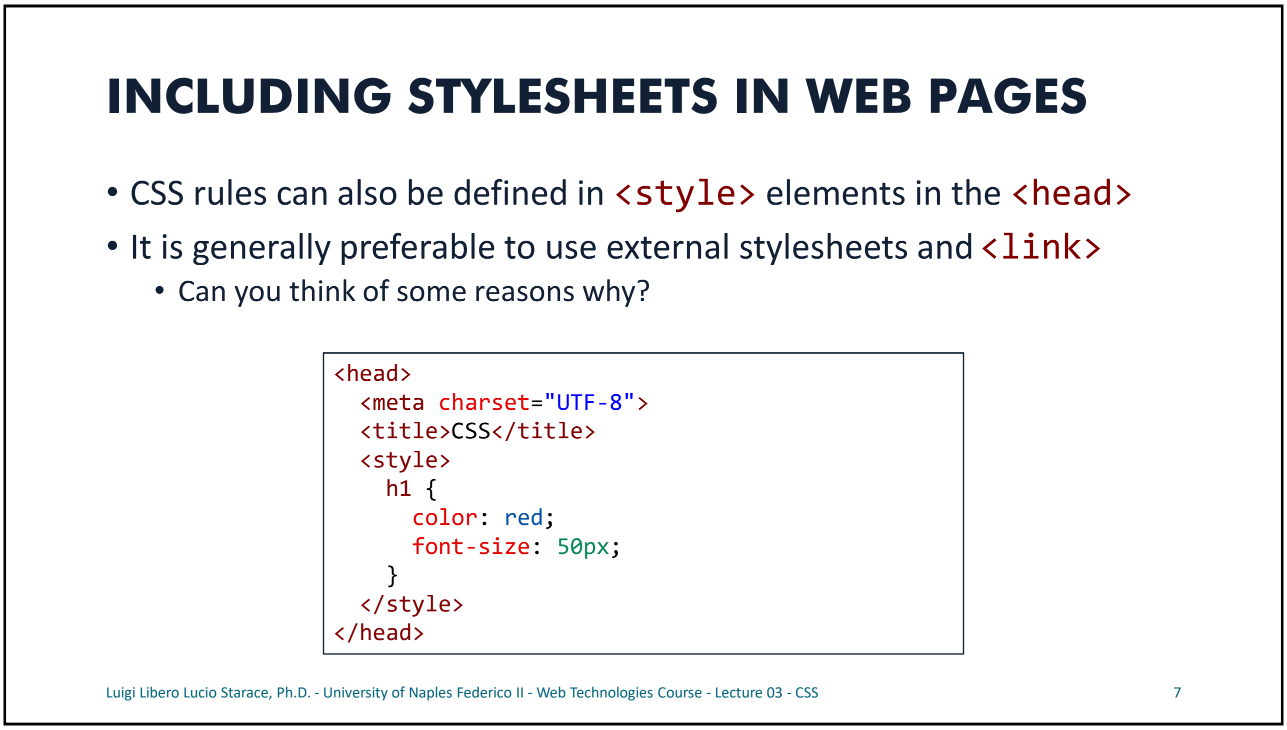
- That's because browsers apply their own, basic styles to every page!
- They are often referred to as **user agent styles**
- These defaults are roughly the same across different browsers, but some **differences** exist (and we'll get back to that!)

INCLUDING STYLESHEETS IN WEB PAGES

Styling can be included in HTML documents in different ways

- Using `<link>` elements in the `<head>` of the document
  - The `rel="stylesheet"` attribute specifies the relation between the current document and the linked document
  - The `href="style.css"` attrib. specifies the URL of the stylesheet to load
  - Same mechanism as `<img>`: browser will make an additional HTTP request to fetch the stylesheet before rendering the page

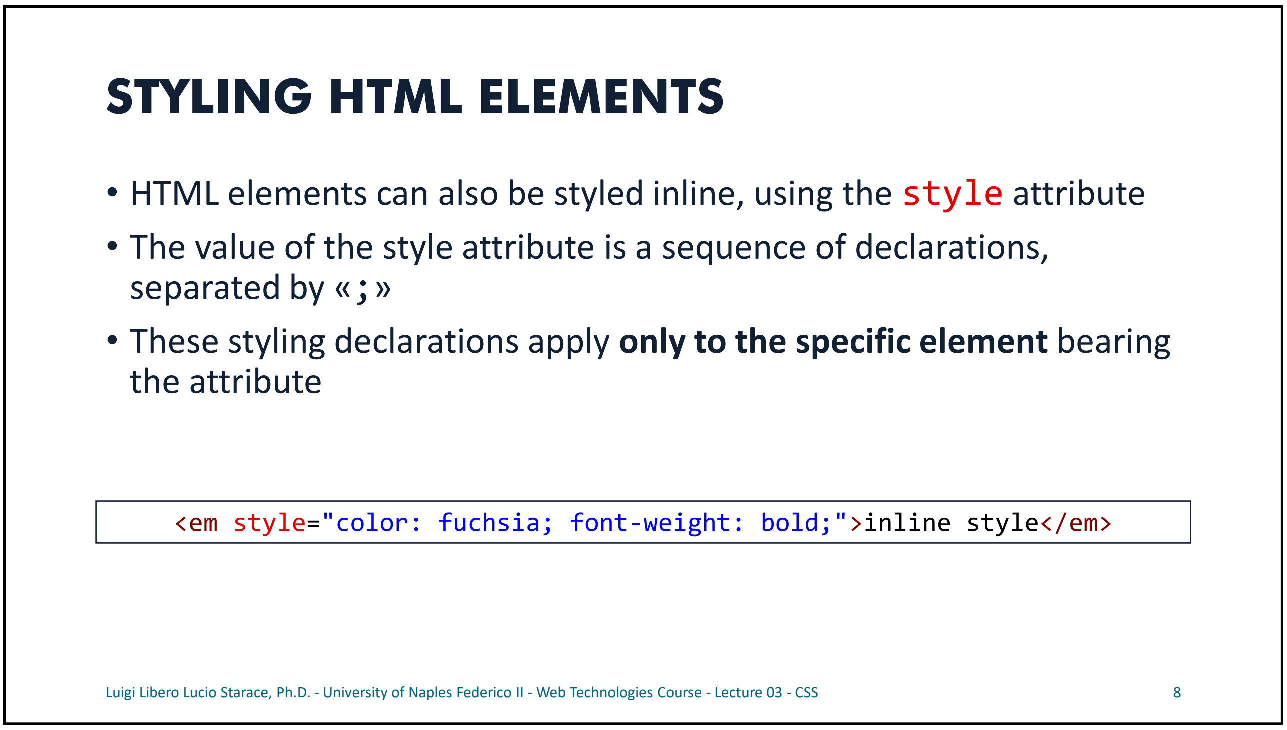
```html
<head>
  <meta charset="UTF-8">
  <title>CSS</title>
  <link rel="stylesheet" href="style.css">
</head>
```



INCLUDING STYLESHEETS IN WEB PAGES

- CSS rules can also be defined in `<style>` elements in the `<head>`
- It is generally preferable to use external stylesheets and `<link>`
  - Can you think of some reasons why?

```
<head>
  <meta charset="UTF-8">
  <title>CSS</title>
  <style>
    h1 {
      color: red;
      font-size: 50px;
    }
  </style>
</head>
```



STYLING HTML ELEMENTS

- HTML elements can also be styled inline, using the `style` attribute
- The value of the style attribute is a sequence of declarations, separated by « ; »
- These styling declarations apply **only to the specific element** bearing the attribute

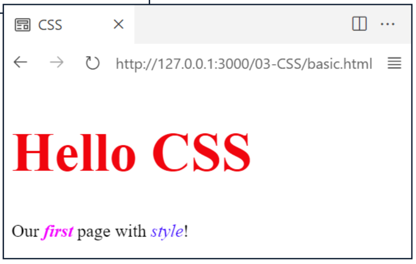
```
<em style="color: fuchsia; font-weight: bold;">inline style</em>
```

CSS: INLINE STYLES

```html
<h1>Hello CSS</h1>
<p>
  Our <em style="color:fuchsia;font-weight: bold;">first</em>
  page with <em>style</em>!
</p>
```

```css
h1 {
  color: red;
  font-size: 50px;
}

em {
  color: blue;
}
```



Hello CSS

Our *first* page with *style*!

SELECTORS

SELECTORS

- Selectors are a **key** part of CSS
- They specify to which elements a CSS rule applies
- CSS selectors are not only used for styling!
  - When using JavaScript to make web pages dynamic, they can be used to select which elements to interact with
  - When doing automated web testing, they can be used to determine which elements the test needs to interact with
  - When doing scraping/crawling, they can be used to select the elements that contain the information we want to extract

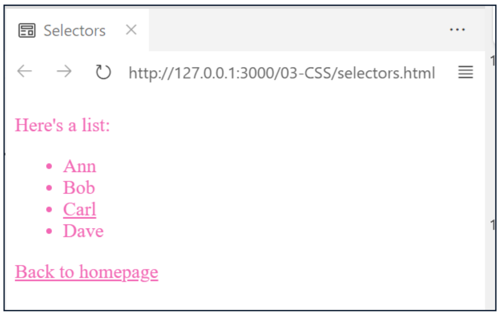
SIMPLE CSS SELECTORS

There exist **five** kinds of simple selectors:

- **Universal** selector (a.k.a **wildcard**). Matches any element.

```css
* {
  color: hotpink;
}
```

```html
<p>Here's a list:</p>
<ul>
  <li>Ann</li>
  <li>Bob</li>
  <li><a href="/car/">Carl</a></li>
  <li>Dave</li>
</ul>
<a href="/">Back to homepage</a>
```



Here's a list:

- Ann
- Bob
- Carl
- Dave

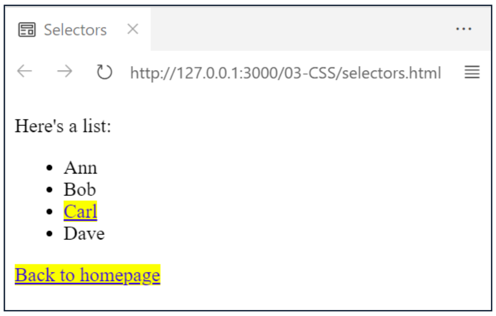
Back to homepage

SIMPLE CSS SELECTORS

- **Type** selector. Matches all element of a given type (i.e., tag name)
- The selector is simply the name of the tag to match

```
a {
  background: yellow;
}
```

```
<p>Here's a list:</p>
<ul>
  <li>Ann</li>
  <li>Bob</li>
  <li><a href="/car/">Carl</a></li>
  <li>Dave</li>
</ul>
<a href="/">Back to homepage</a>
```



Here's a list:

- Ann
- Bob
- Carl
- Dave

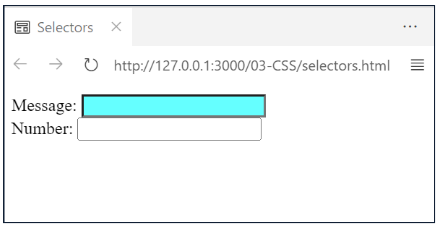
Back to homepage

SIMPLE CSS SELECTORS

- **Id** selector. Matches the element with the given **`id`** attribute.
- Selector has the form ***`#ElementId`***

```
#msg {
  background: cyan;
}
```

```
<form>
  <label for="msg">Message: </label>
  <input id="msg" type="text" name="msg"><br>
  <label for="num">Number: </label>
  <input id="num" type="number" name="num">
</form>
```



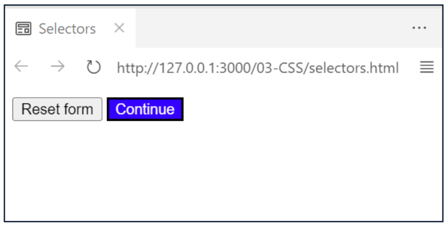
Message:
Number:

SIMPLE CSS SELECTORS

- **Class** selector. Matches the element with the given `class` attribute.
- Selector has the form `.classname`

```
.primary {
  background: blue;
  color: white;
}
```

```
<button>Reset form</button>
<button class="primary btn">Continue</button>
```

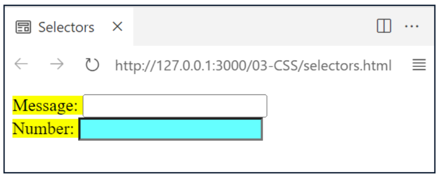


SIMPLE CSS SELECTORS

- **Attribute** selector. Matches the element with a certain attribute.
- Selector has the form `[attribute]` or `[attribute='value']`

```
[for]{ /*all elems with a for attribute*/
  background: yellow;
}
[type='number']{ /*all elems with type=number*/
  background: cyan;
}
```

```
<form>
  <label for="msg">Message: </label>
  <input id="msg" type="text" name="msg"><br>
  <label for="num">Number: </label>
  <input id="num" type="number" name="num">
</form>
```

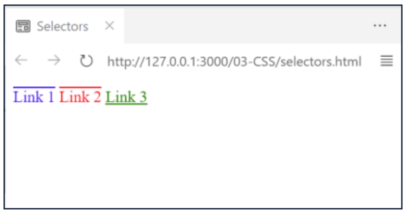


SIMPLE CSS SELECTORS

- Additional operators (***=, ^=, $=**) allow **partial matching** with attribute values

```
[href*='programming']{ /*contains 'programming'*/
  text-decoration: overline;
}
[href^='https']{ /*start with 'https'*/
  color: red;
}
[href$='.it/']{ /*ends with '.it/'*/
  color: green;
}
```

```
<a href="http://bookofprogramming.com/">Link 1</a>
<a href="https://programming.net/">Link 2</a>
<a href="http://webtechnologies.it/">Link 3</a>A
```

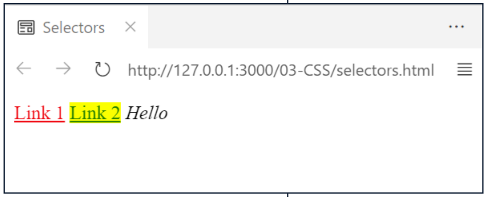


COMPLEX CSS SELECTORS: COMPOUNDS

- It is possible to combine selectors to get fine-grained control
- This is done by concatenating selectors
- Basically select the **intersection** of the involved selectors

```
a[target='_blank'] {
  color: red;
}
a.my-class{
  color: green;
}
a[href*='programming'].my-class {
  background: yellow;
}
```

```
<a href="http://bookofprogramming.com/" target="_blank">Link 1</a>
<a class="my-class" href="https://programming.net/">Link 2</a>
<em class="my-class">Hello</em>
```



COMPLEX CSS SELECTORS: COMBINATORS

- Combinators are used to select elements based on their position in the document (remeber that HTML documents can be seen as **trees**!)
- Syntax is: `selector1` **`combinator`** `selector2`

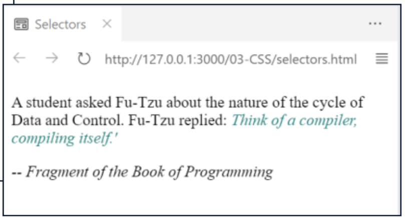
- Four different combinators exist in CSS:
  - Descendant selector (space)
  - Child selector (**>**)
  - Adjacent sibling selector (**+**)
  - General sibling selector (**~**)

COMBINATORS: DESCENDANT SELECTOR

- Syntax: `selectorA selectorB`
- Semantics: match all elements that match `selectorB` and are a contained within (i.e., are a descendant of) an element matching `selectorA`

```
section em {
  color: teal;
}
```

```
<section>
  <p>
    A student asked Fu-Tzu about the nature of
    the cycle of Data and Control. Fu-Tzu replied:
    <em>Think of a compiler, compiling itself.'</em>
  </p>
</section>
<em>-- Fragment of the Book of Programming</em>
```



A student asked Fu-Tzu about the nature of the cycle of Data and Control. Fu-Tzu replied: *Think of a compiler, compiling itself.'*

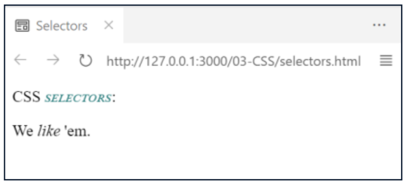
-- Fragment of the Book of Programming

COMBINATORS: CHILD SELECTOR

- Syntax: `selectorA > selectorB`
- Semantics: match all elements that match `selectorB` and are a **directly** contained within (i.e., are a direct child of) an element matching `selectorA`

```
main > em {
  color: teal;
  font-variant: small-caps;
}
```

```
<main>
  CSS <em>selectors</em>:
  <p>We <em>like</em> 'em.</p>
</main>
```



CSS *SELECTORS*:

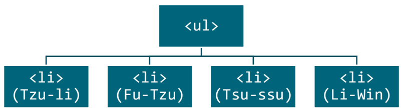
We *like* 'em.

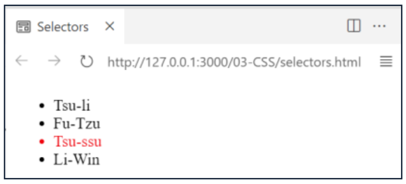
COMBINATORS: ADJACENT SIBLINGS

- Syntax: `selectorA + selectorB`
- Semantics: match all elements that match `selectorB` and are a **next adjacent siblings** of an element matching `selectorA`

```
.master + li {
  color:red;
}
```

```
<ul>
  <li>Tsu-li</li>
  <li class="master">Fu-Tzu</li>
  <li>Tsu-ssu</li>
  <li class="disciple">Li-Win</li>
</ul>
```





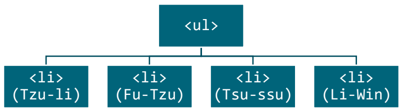
- Tsu-li
- Fu-Tzu
- Tsu-ssu
- Li-Win

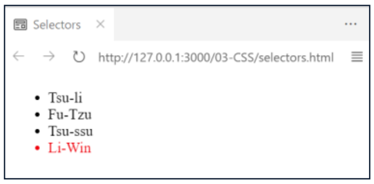
COMBINATORS: GENERAL SIBLINGS

- Syntax: `selectorA ~ selectorB`
- Semantics: match all elements that match `selectorB` and are a **subsequent siblings** of an element matching `selectorA`

```
.master ~ li.discipline {
  color:red;
}
```

```
<ul>
  <li>Tsu-li</li>
  <li class="master">Fu-Tzu</li>
  <li>Tsu-ssu</li>
  <li class="disciple">Li-Win</li>
</ul>
```





- Tsu-li
- Fu-Tzu
- Tsu-ssu
- Li-Win

CSS SELECTORS: PSEUDO–CLASSES

HTML elements can be in different **states**, for example because of **user interactions** or because of their relation with other elements.

Pseudo-classes selectors start with « **:** », and allow to style elements based on their **state:**

- **Interactive states** (resulting from user interaction)
- **Historic states** (used to «remember» which links were visited)
- **Form states** (specific of interaction with forms)
- States deriving from **relations with other elements**

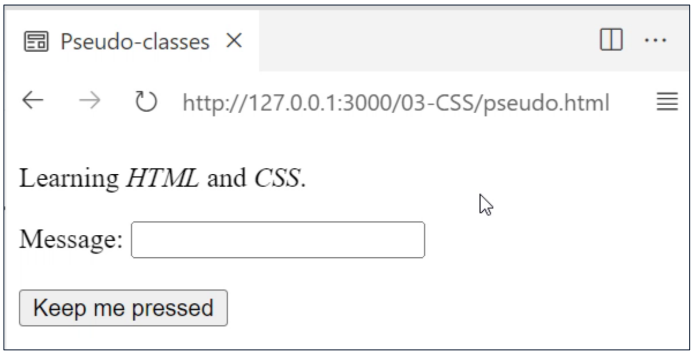
PSEUDO–CLASSES: INTERACTIVE STATES

- **:hover** selects the elements on which a pointing device (i.e.: mouse) is placed over
- :**active** matches the state in which an element is actively being interacted with (e.g.: button is being pressed)
- **:focus** matches the state in which an element (e.g.: a link or an input field) has focus (i.e.: is currently selected) in the web page

PSEUDO–CLASSES: INTERACTIVE STATES

```html
<p>Learning <em>HTML</em> and <em>CSS</em>.</p>
Message: <input type="text"><br><br>
<button>Keep me pressed</button>
```

```css
em:hover {
  background: yellow;
}
input[type='text']:focus{
  background: cyan;
}
button:active{
  background: blue;
  color: white;
}
```



Learning *HTML* and *CSS*.

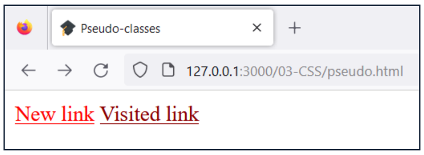
Message:

PSEUDO–CLASSES: HISTORIC STATES

- **:link** selects links that have not been visited yet
- **:visited** selects links that have already been visited

```
:link{
  color: red;
}
:visited{
  color: darkred;
}
```

```
<a href="./js/">New link</a>
<a href="./css/">Visited link</a>
```



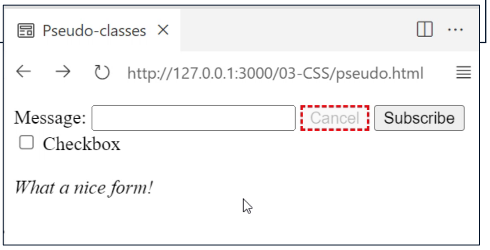
New link Visited link

Historic states are **not supported** in the *Live Preview* browser within Visual Studio Code! Open the page in Firefox to check them out.

PSEUDO–CLASSES: FORM STATES

```
<div>
  <label for="mail">Message:</label><input id="mail" type="email">
  <button disabled>Cancel</button><button>Subscribe</button>
</div>
<input type="checkbox"> Checkbox<br><br>
<em>What a nice form!</em>
```

```
:disabled {
  border: 2px dashed red;
}
:invalid {
  color: red;
}
:checked ~ em {
  color: deeppink; font-weight: bold;
}
```



What a nice form!

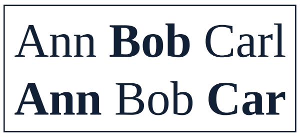
Technically, there can be email address without a dot. For example, user@localhost or user@com are valid addresses!

PSEUDO–CLASSES: POSITION RELATIONS

- **:first-child** and **:last-child** select the first (last) child among a set of siblings.
- **:only-child** can be used to select elements that have no siblings.
- **:first-of-type** and **:last-of-type** can be used to select elements that are the first (last) child among a set of sibling, considering only elements of the same type.
- **:nth-child(n)** and **:nth-of-type(n)** can be used to select elements that are in the n-th position among their siblings.
  - **Indexing in CSS starts at 1!**

PSEUDO–CLASSES: POSITION RELATIONS

```
<p>
  <em>Ann</em> <strong>Bob</strong> <em>Carl</em>
</p>
<p>
  <strong>Ann</strong> <em>Bob</em> <strong>Carl</strong>
</p>
```


Ann **Bob** Carl
Ann Bob **Car**

PSEUDO–CLASSES: POSITION RELATIONS

```
<p>
  <em>Ann</em> <strong>Bob</strong> <em>Carl</em>
</p>
<p>
  <strong>Ann</strong> <em>Bob</em> <strong>Carl</strong>
</p>
```

Ann **Bob** Carl
Ann Bob **Car**

```
em:last-child {
  color: red;
}
```

Ann **Bob** Carl
Ann Bob **Car**

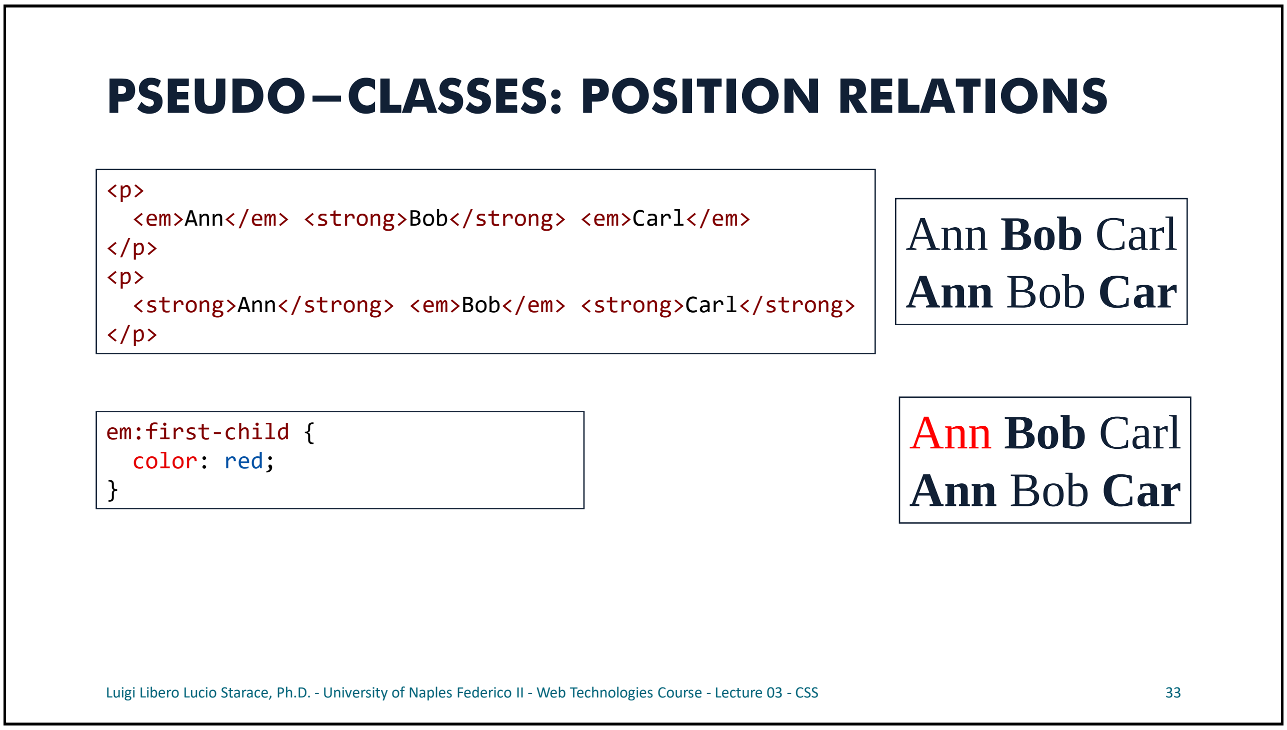
PSEUDO–CLASSES: POSITION RELATIONS

```
<p>
  <em>Ann</em> <strong>Bob</strong> <em>Carl</em>
</p>
<p>
  <strong>Ann</strong> <em>Bob</em> <strong>Carl</strong>
</p>
```

Ann **Bob** Carl
Ann Bob **Car**

```
em:last-of-type {
  color: red;
}
```

Ann **Bob** Carl
Ann Bob **Car**



PSEUDO–CLASSES: POSITION RELATIONS

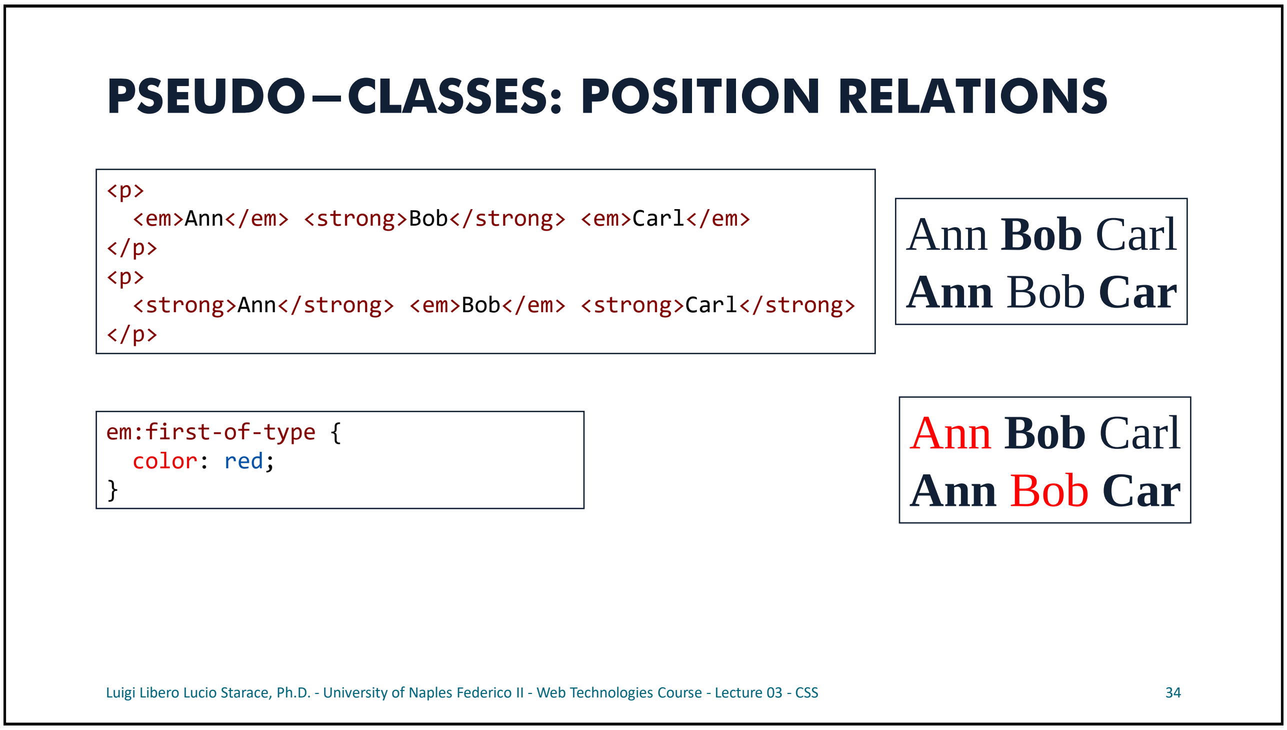
```html
<p>
  <em>Ann</em> <strong>Bob</strong> <em>Carl</em>
</p>
<p>
  <strong>Ann</strong> <em>Bob</em> <strong>Carl</strong>
</p>
```

Ann **Bob** Carl
Ann Bob **Car**

```css
em:first-child {
  color: red;
}
```

Ann **Bob** Carl
Ann Bob **Car**

Luigi Libero Lucio Starace, Ph.D. - University of Naples Federico II - Web Technologies Course - Lecture 03 - CSS 33



PSEUDO–CLASSES: POSITION RELATIONS

```html
<p>
  <em>Ann</em> <strong>Bob</strong> <em>Carl</em>
</p>
<p>
  <strong>Ann</strong> <em>Bob</em> <strong>Carl</strong>
</p>
```

Ann **Bob** Carl
Ann Bob **Car**

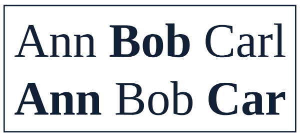
```css
em:first-of-type {
  color: red;
}
```

Ann **Bob** Carl
Ann Bob **Car**

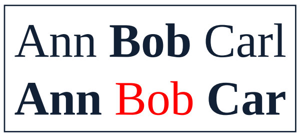
Luigi Libero Lucio Starace, Ph.D. - University of Naples Federico II - Web Technologies Course - Lecture 03 - CSS 34

PSEUDO–CLASSES: POSITION RELATIONS

```html
<p>
  <em>Ann</em> <strong>Bob</strong> <em>Carl</em>
</p>
<p>
  <strong>Ann</strong> <em>Bob</em> <strong>Carl</strong>
</p>
```


Ann **Bob** Carl
Ann Bob **Car**

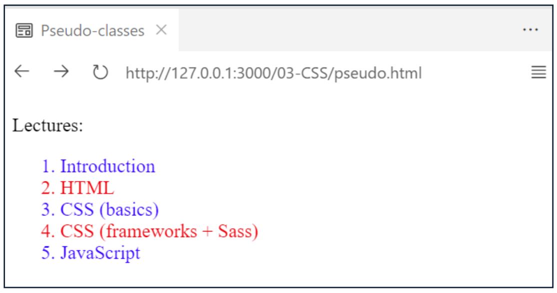
```css
em:nth-child(2) {
  color: red;
}
```


Ann **Bob** Carl
Ann Bob **Car**

PSEUDO–CLASSES: POSITION RELATIONS

```html
<p>Lectures:</p>
<ol>
  <li>Introduction</li>
  <li>HTML</li>
  <li>CSS (basics)</li>
  <li>CSS (frameworks + Sass)</li>
  <li>JavaScript</li>
</ol>
```

```css
li:nth-child(even){
  color: red;
}
li:nth-child(odd){
  color: blue;
}
```


Pseudo-classes
http://127.0.0.1:3000/03-CSS/pseudo.html

Lectures:

1. Introduction
2. HTML
3. CSS (basics)
4. CSS (frameworks + Sass)
5. JavaScript

PSEUDO–ELEMENTS

Pseudo-elements can be used to target specific parts of the content of a given HTML element, without adding extra HTML markup

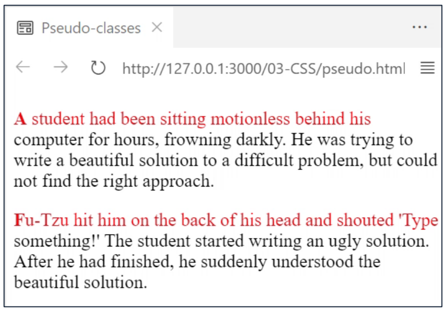
The sintax of pseudo-element selectors is **selector::*pseudo-element***

- **selector** is a CSS selector for the target element
- ***pseudo-element*** is one of the supported pseudo-element selectors:
  - **::first-letter:** targets the first letter of the content of a **block-level** element
  - **::first-line:** targets the first line of the content of a **block-level** element
  - **::selection:** targets the content that is currently selected by the user
  - **::before:** creates an element that is the **first child** of the selected element
  - **::after:** creates an element that is the **last child** of the selected element

PSEUDO–ELEMENTS: EXAMPLES

```
<p>A student had been sitting motionless behind  his computer for hours,
frowning darkly. He was trying to write a beautiful solution to a difficult
problem, but could not find the right approach.</p>
<p>Fu-Tzu hit him on the back of his head and shouted 'Type something!' The
student started writing an ugly solution. After he had finished, he suddenly
understood the beautiful solution.</p>
```

```
p::first-letter {
  font-weight:bold;
}
p::first-line{
  color: red;
}
p:last-child::selection {
  background: red;
  color: white;
}
```



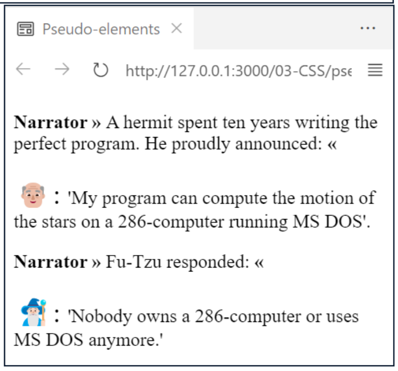
A student had been sitting motionless behind his computer for hours, frowning darkly. He was trying to write a beautiful solution to a difficult problem, but could not find the right approach.

Fu-Tzu hit him on the back of his head and shouted 'Type something!' The student started writing an ugly solution. After he had finished, he suddenly understood the beautiful solution.

PSEUDO–ELEMENTS: EXAMPLES

```html
<p class="narrator">A hermit spent ten years writing the perfect program. He
proudly announced: </p>
<p class="hermit">'My program can compute the motion of the stars on a 286-
computer running MS DOS'.</p>
<p class="narrator">Fu-Tzu responded:</p>
<p class="fu-tzu">'Nobody owns a 286-computer or uses MS DOS anymore.'</p>
```

```css
.narrator::before {
  content: "Narrator » "; font-weight: bold;
}
.narrator::after {
  content: " «"; font-weight: bold;
}
.hermit::before {
  content: "👴 : "; font-size: 24px;
}
.fu-tzu::before {
  content: "🧙 : "; font-size: 24px;
}
```



Narrator » A hermit spent ten years writing the perfect program. He proudly announced: «

👴 : 'My program can compute the motion of the stars on a 286-computer running MS DOS'.

Narrator » Fu-Tzu responded: «

🧙 : 'Nobody owns a 286-computer or uses MS DOS anymore.'

THE CASCADE

THE CASCADE IN CASCADING STYLE SHEETS

- Sometimes, two or more rules might apply to the same element
- These rules might be **conflicting**, i.e., assign different values to the same property (e.g.: `color`)
- **The cascade** is the algorithm used to **resolve** such **conflicts**
  - **Input:** a set of conflicting properties that apply to a given element
  - **Output:** a single, cascaded, property to actually apply
- The cascade considers **4 key aspects, in order:**
  1. **Origin and Importance**
  2. **Layers**
  3. **Specificity**
  4. **Position and order of appearence** of the rule

THE CASCADE: ORIGIN

- The CSS we write (a.k.a. **authored CSS**) is not the only one being applied to a web page
- We've already mentioned that **user agent styles** exist
  - The stylesheets that are included by browsers by default
- Other styles (a.k.a. **local user styles**) might be added by specific browser extensions or from the operating system level
  - For example, for accessibility purposes
  - Visually-impaired persons might want to use high-contrast color schemes, with larger fonts, etc.

THE CASCADE: IMPORTANCE

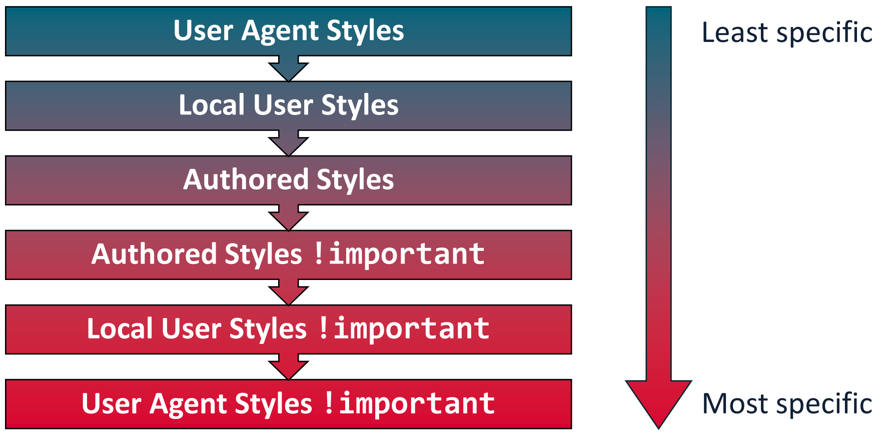
- The `!important` rule can be used to add more importance to a property inside a CSS rule
- `!important` is simply added at the end of the property declaration

```
h1 {
  color: red !important;
}
```

- Importance plays a significant role in the cascade

THE CASCADE: ORIGIN–IMPORTANCE

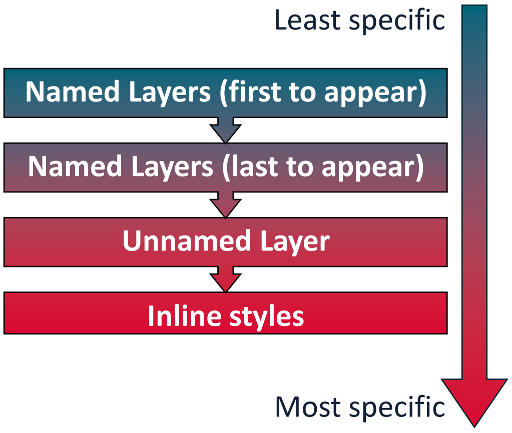
From the lest specific origin to the most specific one


User Agent Styles
Local User Styles
Authored Styles
Authored Styles !important
Local User Styles !important
User Agent Styles !important
Least specific
Most specific

THE CASCADE: LAYERS

Within each origin/importance bucket, there can be multiple cascade **layers**

- Within authored styles:
  - Custom layers can be defined using @layer rule (we won't see that)
    - The custom layers that are declared later have higher priority
  - All other CSS (in **<style>** or imported with **<link>**) belongs to a unnamed layer
  - Inline styles belong to a separate layer and have the highest priority



THE CASCADE: SPECIFICITY

When two conflicting rules:

- Belong to the same origin-importance bucket, and
- Belong the same layer

Specificity is considered.

- The idea is that the **most specific** selector should win

```
.primary {
  color: blue;
}
```

VS

```
h1 {
  color: red;
}
```

```
<h1 class="primary">Cascading!</h1>
```

THE CASCADE: SPECIFICITY

CSS defines how to calculate the specificity of a selector:

- Ignore the universal selector
- Count the number of id selectors (=A)
- Count the number of class, attribute and pseudo-classes selectors (=B)
- Count the number of type and pseudo-element selectors (=C)

The specificity is a numeric triple **(A, B, C)** computed as above

THE CASCADE: SPECIFITY EXAMPLES

- A: Number of id selectors
- B: Number of class, attribute and pseudo-classes selectors
- C: Number of type and pseudo-element selectors

Selector	Specificity (A, B, C)
`#id`	(1, 0, 0)
`em.master[target]`	(0, 2, 1)
`#navbar ul li a.nav-link[href*='/']`	(1, 2, 3)
`article.item section p::first-letter`	(0, 1, 4)
`a:hover`	(0, 1, 1)
`*`	(0, 0, 0)

THE CASCADE: COMPARING SPECIFICITIES

Comparisons are made by considering the three components in order:

- the specificity with a larger **A** is more specific;
- if the two **A** are tied, then the specificity with a larger **B** wins;
- if the two **B** are also tied, then the specificity with a larger **C** wins;
- if all the values are tied, the two specificities are **equal**.

THE CASCADE: SPECIFICITY

VS

Selector #1	Specif. #1	Selector #2	Specif. #2	Winner
`a[target]`	(0, 1, 1)	`.list a`	(0, 1, 1)	Draw
`#msg`	(1, 0, 0)	`input[type].inp`	(0, 2, 1)	#1
`#nav > #brd a.lk`	(2, 1, 1)	`em.foo.bar.light`	(0, 3, 1)	#1
`[id='nav'] a`	(0, 1, 1)	`#nav a`	(1, 0, 1)	#2

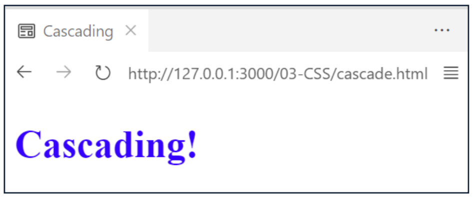
THE CASCADE: POSITION AND APPEARENCE

When two properties:

- Belong to the same origin/importance bucket
- Belong to the same layer
- Have the same specificity

The **last rule** to appear has the highest priority

```
h1 {
  color: red;
}
h1 {
  color: blue;
}
```


Cascading
http://127.0.0.1:3000/03-CSS/cascade.html

Cascading!

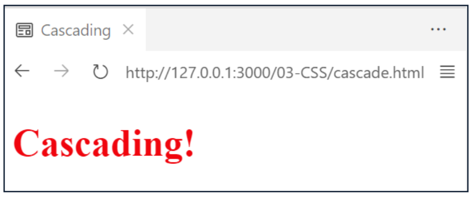
THE CASCADE: POSITION AND APPEARENCE

- This rule applies within the same stylesheet, and on the order in which stylesheets appear

```
h1 {
  color: blue;
}
```

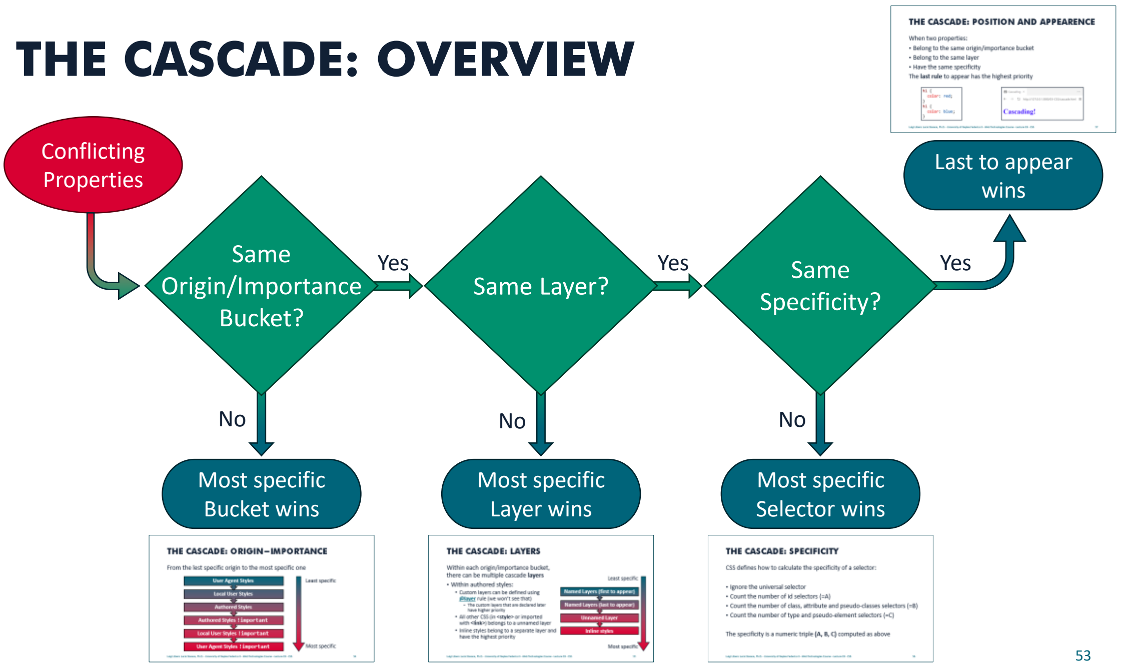
```
h1 {
  color: red;
}
```

```
<head>
  <meta charset="UTF-8">
  <title>Cascading</title>
  <link rel="stylesheet" href="blue.css">
  <link rel="stylesheet" href="red.css">
</head>
<body>
  <h1>Cascading!</h1>
</body>
```


Cascading
http://127.0.0.1:3000/03-CSS/cascade.html

Cascading!

THE CASCADE: OVERVIEW



THE CASCADE: POSITION AND APPEARENCE

When two properties:
- Belong to the same origin/importance bucket
- Belong to the same layer
- Have the same specificity

The **last rule** to appear has the highest priority

```
h1 {
  color: red;
}
h1 {
  color: blue;
}
```

Cascading!

THE CASCADE: ORIGIN–IMPORTANCE

From the lest specific origin to the most specific one

THE CASCADE: LAYERS

Within each origin/importance bucket, there can be multiple cascade **layers**
- Within authored styles:
  - Custom layers can be defined using @layer rule (we won't see that)
    - The custom layers that are declared later have higher priority
  - All other CSS (in <**style**> or imported with <**link**>) belongs to a unnamed layer
  - Inline styles belong to a separate layer and have the highest priority

THE CASCADE: SPECIFICITY

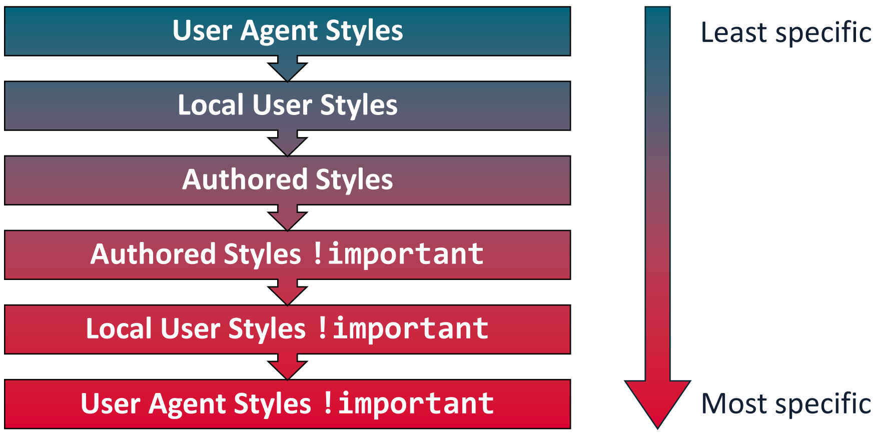
CSS defines how to calculate the specificity of a selector:

- Ignore the universal selector
- Count the number of id selectors (=A)
- Count the number of class, attribute and pseudo-classes selectors (=B)
- Count the number of type and pseudo-element selectors (=C)

The specificity is a numeric triple **(A, B, C)** computed as above

THE CASCADE: ORIGIN–IMPORTANCE

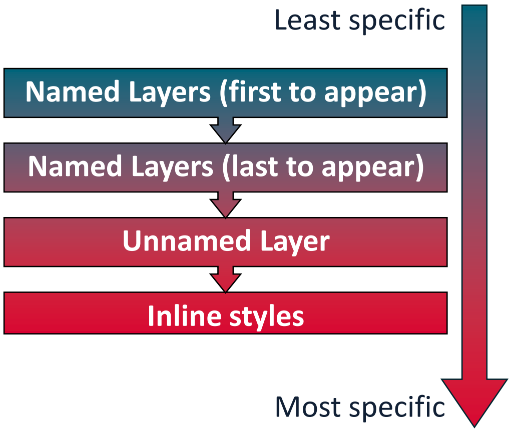
From the lest specific origin to the most specific one


User Agent Styles
Local User Styles
Authored Styles
Authored Styles !important
Local User Styles !important
User Agent Styles !important
Least specific
Most specific

THE CASCADE: LAYERS

Within each origin/importance bucket, there can be multiple cascade **layers**

- Within authored styles:
  - Custom layers can be defined using @layer rule (we won't see that)
    - The custom layers that are declared later have higher priority
  - All other CSS (in **<style>** or imported with **<link>**) belongs to a unnamed layer
  - Inline styles belong to a separate layer and have the highest priority



THE CASCADE: SPECIFICITY

CSS defines how to calculate the specificity of a selector:

- Ignore the universal selector
- Count the number of id selectors (=A)
- Count the number of class, attribute and pseudo-classes selectors (=B)
- Count the number of type and pseudo-element selectors (=C)

The specificity is a numeric triple **(A, B, C)** computed as above

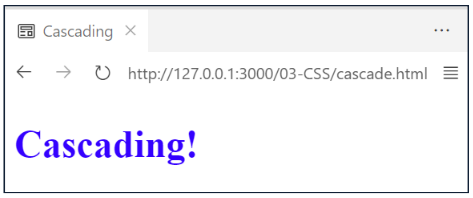
THE CASCADE: POSITION AND APPEARENCE

When two properties:

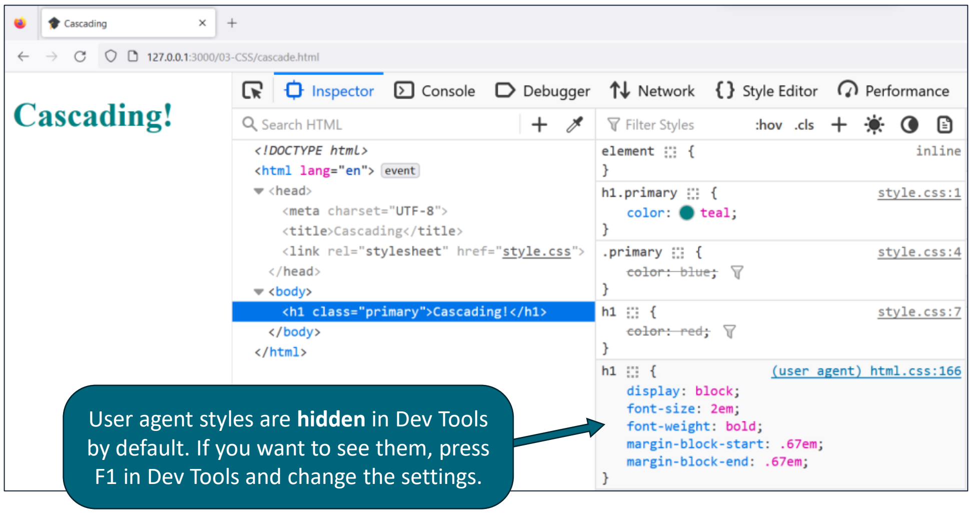
- Belong to the same origin/importance bucket
- Belong to the same layer
- Have the same specificity

The **last rule** to appear has the highest priority

```
h1 {
  color: red;
}
h1 {
  color: blue;
}
```



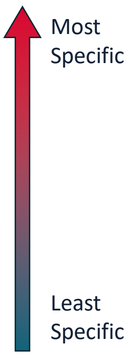
THE CASCADE IN BROWSER DEV TOOLS



```
<!DOCTYPE html>
<html lang="en"> event
  <head>
    <meta charset="UTF-8">
    <title>Cascading</title>
    <link rel="stylesheet" href="style.css">
  </head>
  <body>
    <h1 class="primary">Cascading!</h1>
  </body>
</html>
```

```
element {                                  inline
}
h1.primary {                           style.css:1
  color: teal;
}
.primary {                             style.css:4
  ~~color: blue;~~
}
h1 {                                   style.css:7
  ~~color: red;~~
}
h1 {                    (user agent) html.css:166
  display: block;
  font-size: 2em;
  font-weight: bold;
  margin-block-start: .67em;
  margin-block-end: .67em;
}
```

User agent styles are **hidden** in Dev Tools by default. If you want to see them, press F1 in Dev Tools and change the settings.



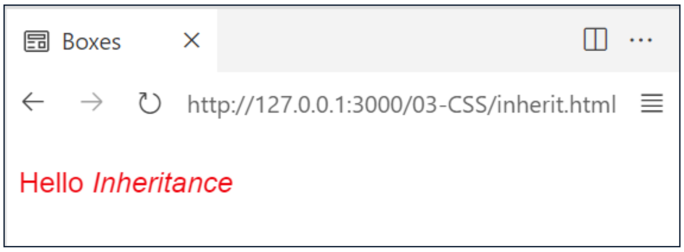
INHERITANCE

INHERITANCE IN CSS

- Some CSS properties can be inherited from ancestor elements, if no specific value is set
- Inheritable properties include **color**, **font-size**, **font-family**, **font-weight, font-style**

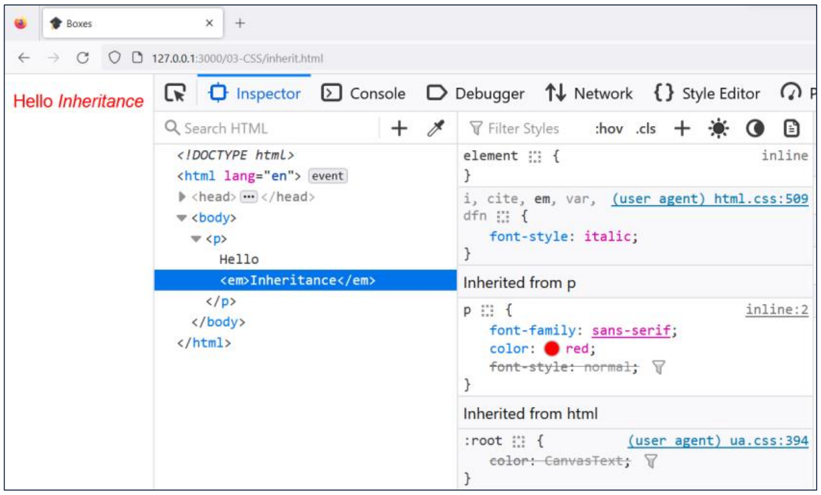
```
p {
  font-family: sans-serif;
  color: red;
  font-style: normal;
}
```

```
<p>
  Hello <em>Inheritance</em>
</p>
```



Hello *Inheritance*

INHERITANCE IN CSS



```
<!DOCTYPE html>
<html lang="en"> event
  <head> ··· </head>
  <body>
    <p>
      Hello
      <em>Inheritance</em>
    </p>
  </body>
</html>
```

```
element ::: {                                inline
}
i, cite, em, var,     (user agent) html.css:509
dfn ::: {
    font-style: italic;
}
Inherited from p
p ::: {                                    inline:2
    font-family: sans-serif;
    color: ● red;
    ~~font-style: normal;~~
}
Inherited from html
:root ::: {             (user agent) ua.css:394
    ~~color: CanvasText;~~
}
```

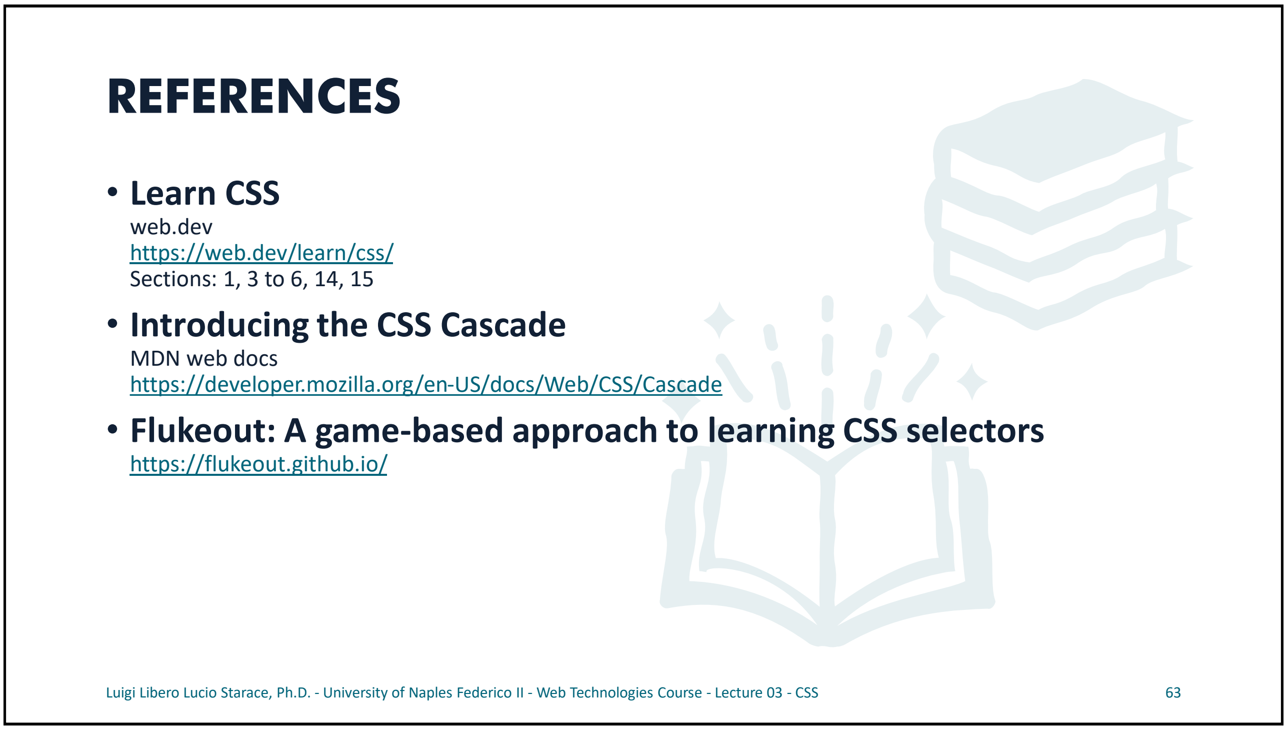
Inherited properties have the lowest specificity of all styling methods

ASSIGNMENT #2

Today’s lecture comes with **Assignment #2**! In this assignment, you will:

- Do some practice with basic CSS
- Write some tricky CSS rules
- Test your knowledge of the Cascade algorithm

Note: the live HTTP server we setup in **Exercise 1** of **Assignment #1** will be handy for this assignment! Unless you are already familiar with HTTP servers and already know what you’re doing, make sure you completed at least **Exercise 1** in **Assignment #1** before doing **Assignment #2**!



REFERENCES

- **Learn CSS**
  web.dev
  https://web.dev/learn/css/
  Sections: 1, 3 to 6, 14, 15
- **Introducing the CSS Cascade**
  MDN web docs
  https://developer.mozilla.org/en-US/docs/Web/CSS/Cascade
- **Flukeout: A game-based approach to learning CSS selectors**
  https://flukeout.github.io/