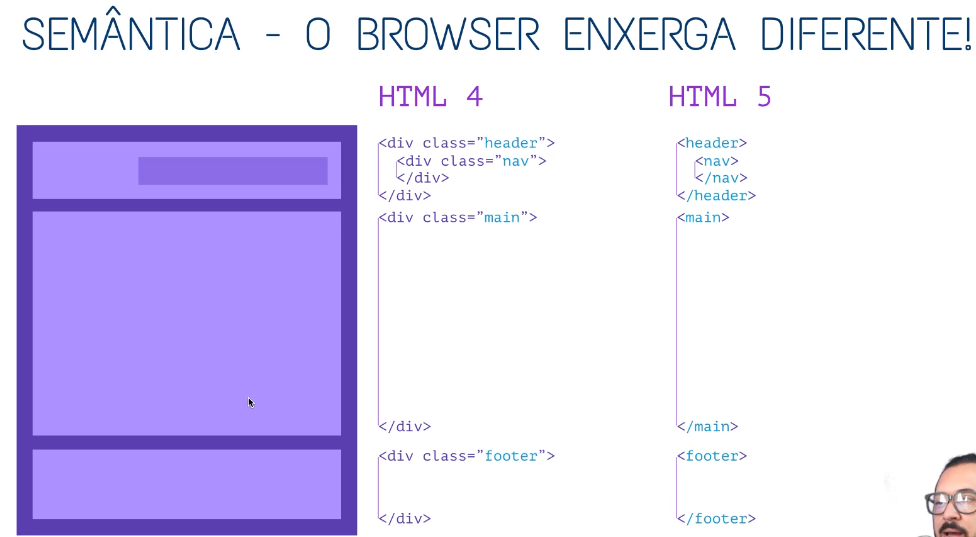
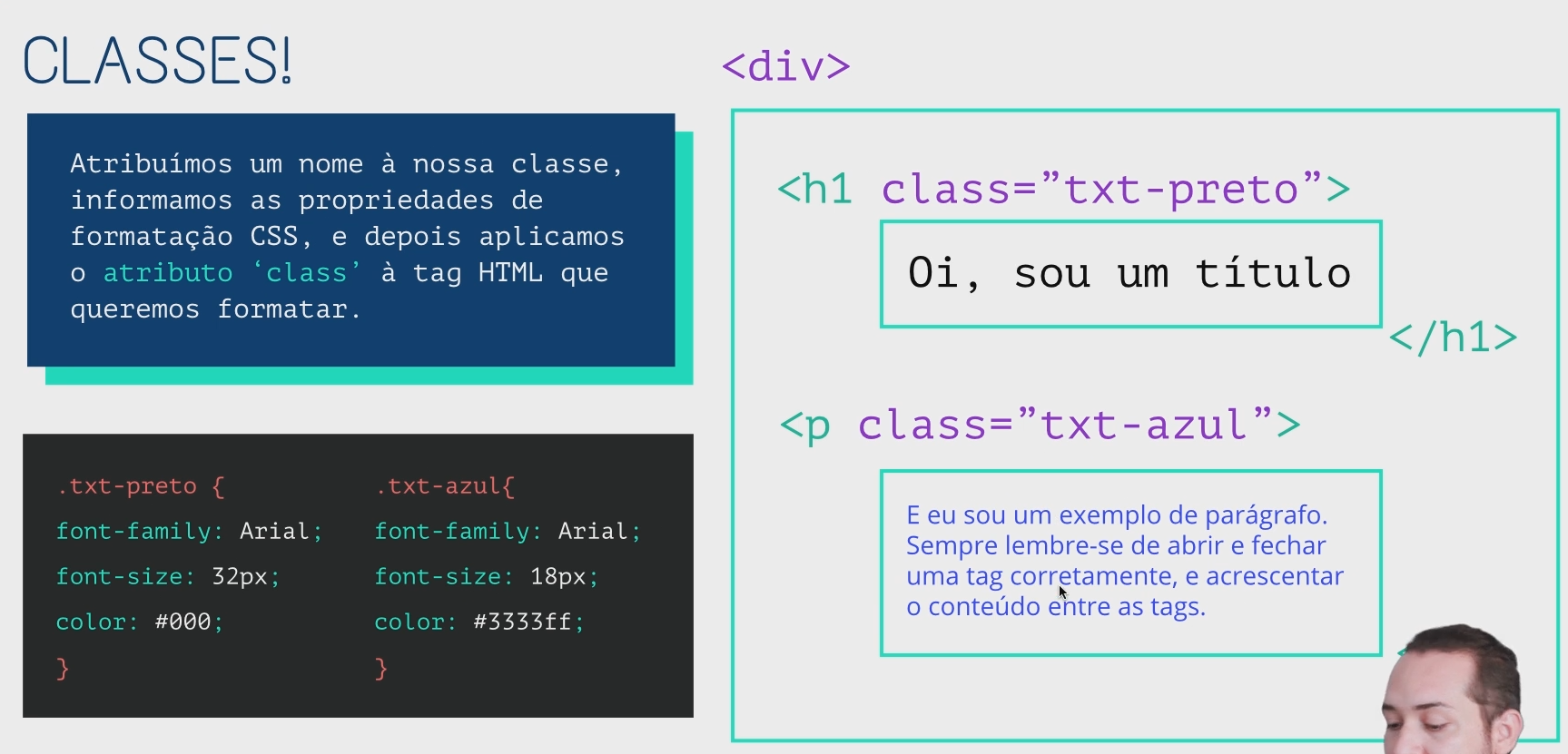
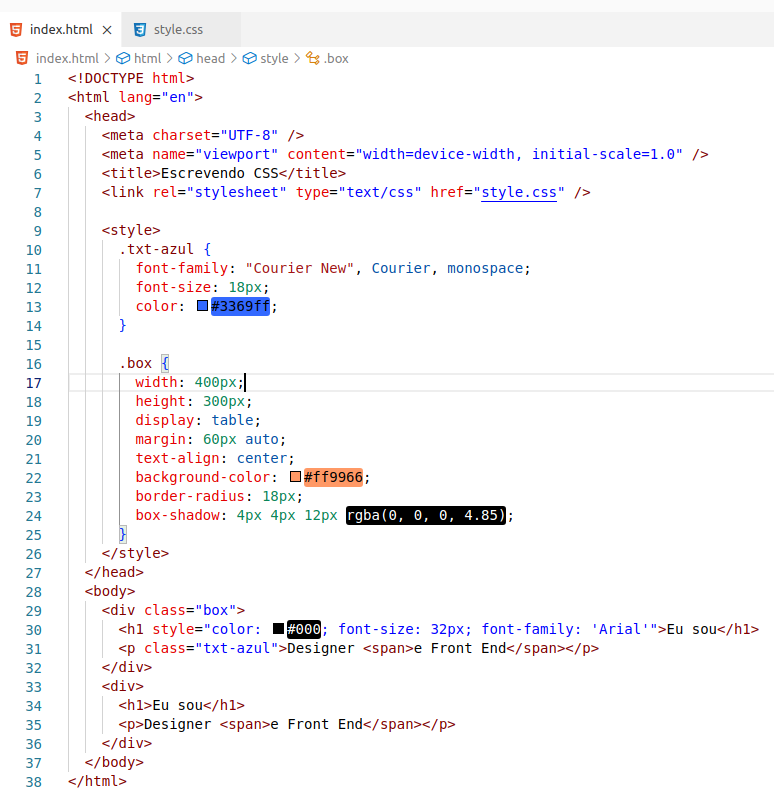
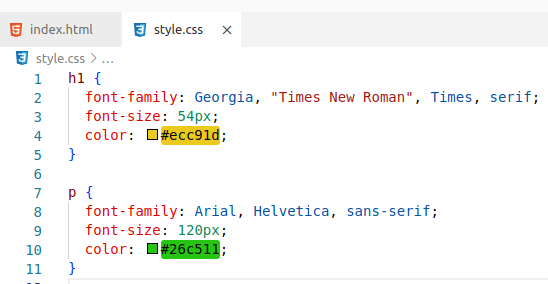
Extensions VC Code

* Prettier - Code Fornatter
* Color HighLight









There are several rules ( applied in this order ) :

1. inline css ( html style attribute ) overrides css rules in style tag and css file
2. a more specific selector takes precedence over a less specific one
3. rules that appear later in the code override earlier rules if both have the same specificity.
4. A css rule with !important always takes precedence.

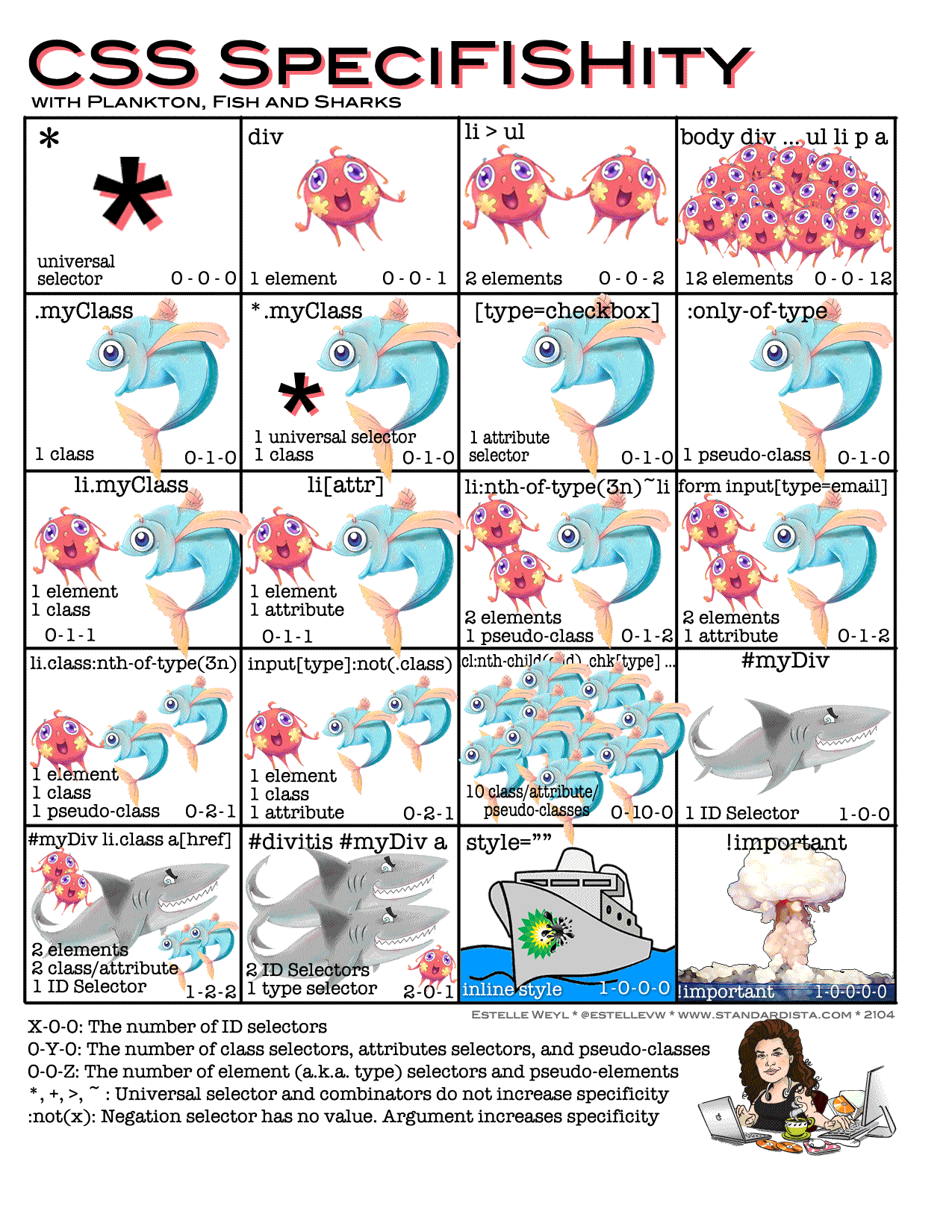
In your case its rule 3 that applies.

Specificity for single selectors from highest to lowest:

* ids (example: #main selects <div id="main">)
* classes (ex.: .myclass), attribute selectors (ex.: [href=^https:]) and pseudo-classes (ex.: :hover)
* elements (ex.: div) and pseudo-elements (ex.: ::before)

To compare the specificity of two combined selectors, compare the number of occurences of single selectors of each of the specificity groups above.





# CSS Selector’s priority

# <https://dev.to/one-beyond/css-selectors-priority-1cjh>

[#css](https://dev.to/t/css) [#webdev](https://dev.to/t/webdev) [#html](https://dev.to/t/html) [#beginners](https://dev.to/t/beginners)

Almost all the CSS style conflicts and style overrides have to do with the css selector's priority. In this article I will talk about how the css selector's priority works.

CSS selectors have a strict hierarchy. I have divided them into 6 levels (from 0 to 5).

**Level 0 :**

* [Universal selector](https://developer.mozilla.org/en-US/docs/Web/CSS/Universal_selectors) \*

\* { color: #000 }

* [CSS combinators](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Selectors#combinators), such as adjacent sibling selector +, child selector >, general sibling selector ~, descendant selector space

div + div { padding: 8px; }

div > p { text-decoration: underline; }

p ~ p { margin-bottom: 8px; }

header h1 { font-size: 18px; }

* [Logic pseudo-class](https://developer.mozilla.org/en-US/docs/Web/CSS/Pseudo-classes), such as :not(), :is(), :where() , these logic pseudo classes won't affect the priority, what will affect the priority is the selectors that pass to the pseudo-class function. Pay attention that here I only list the logic pseudo class, not including the other non-logic pseudo class, such as :hover, :first-child etc.

p:not(.text) { color: red; }

div:is(li) {font-size: 8px; }

main:where(h1, h2, h3) { color: orange; }

**Level 1:**

* [Type selector](https://developer.mozilla.org/en-US/docs/Web/CSS/Type_selectors), such as p div header span

header { color: #fff }

p { background: red }

**Level 2:**

* [Class selector](https://developer.mozilla.org/en-US/docs/Web/CSS/Class_selectors) .className

.text { color: red }

* [attribute selector](https://developer.mozilla.org/en-US/docs/Web/CSS/Attribute_selectors) [attribute]

[data-text] { color: blue }

* [pseudo class](https://developer.mozilla.org/en-US/docs/Web/CSS/Pseudo-classes) such as :hover :link

:hover { color: green }

* [pseudo-element](https://developer.mozilla.org/en-US/docs/Web/CSS/Pseudo-elements) such as :first-child:

:first-child {margin-left: 8px}

**Level 3**

* [ID selector](https://developer.mozilla.org/en-US/docs/Web/CSS/ID_selectors) #id

#text { color: #fff }

**Level 4**

* Inline style in html tag:

<p style= "color:#000" >some text<p/>

**Level 5**

* The key word !important

.text { color: #fff !important }

The key word !important has the highest priority, usually it’s used to override the javascript style, otherwise I don’t recommend using it, it will be difficult to maintain the code if we abuse the !important key word.

We can see that from 0 to 3 level is all about css selectors’ priority, it’s also the most important part. If we can fully understand it , it will be much easier for us to override the style using css selector.

Although now we know all the css selectors’ priority, it’s still very abstract for us to compare them when there are multiple selectors together. You may ask, how can we compare div .foo[bar] { color:#fff } with body > div p.foo { color: #000 } ?

Don’t worry, I’ll also explain how to calculate the css selectors’ priority with actual values. Let’s say if you encounter one **level 0** priority selector, then you add **0** value to the total priority , if you encounter one **level 1** priority selector, then you add **1** value to the total priority, if you encounter one **level 2** priority selector, then you add **10** value to the total priority, and finally if you encounter one **level 3** priority selector, you add **100** value to the total priority. In summary:

* Level 0: **+0**
* Level 1: **+1**
* Level 2: **+10**
* Level 3: **+100**

Let’s see some examples:

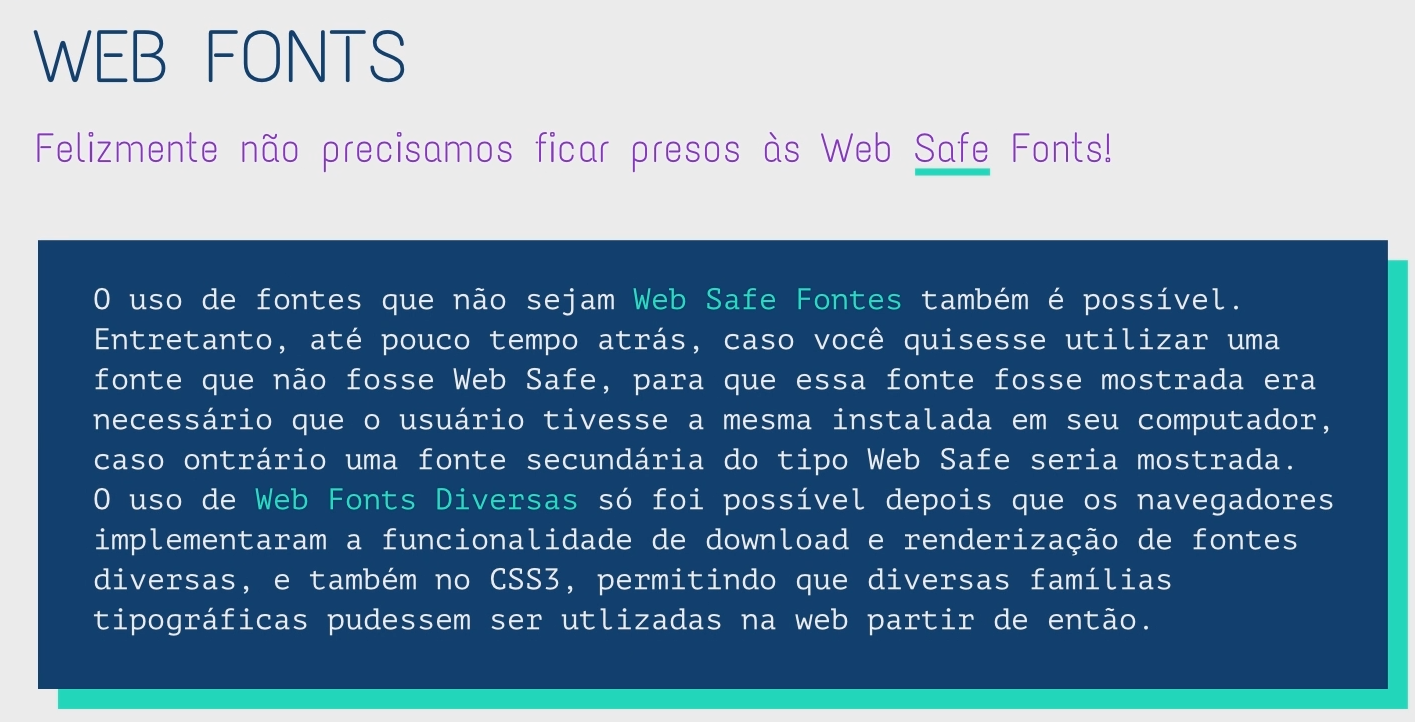
1. \* > div {} : 1 general selector (**level 0**), 1 css combinator (**level 0**), 1 type selector (**level 1**), so the total value is **0 + 0 + 1 = 1**
2. div > p .text:not(.content): 2 type selector (**level 1**), 1 *css combinator* (**level 0**), 2 *class selector* (**level 2**), 1 logic pseudo-class (**level 0**), so the total value is **2 \* 1 + 0 + 2 \* 10 + 0 = 22**
3. p[content="story"]: 1 type selector (**level 1**), 1 attribute selector (**level 2**), so the total value is **1+ 10 = 11**
4. #list > ol: 1 id selector (**level 3**), 1 css combinator (**level 0**), 1 type selector (**level 1**), so the total value is **100 + 0 + 1 = 101**

That’s how you calculate the priority value in css. Now you may ask what if I add 11 type selector, then the value is 11, so can it override one class selector, which has a value 10 ? The truth is the value here is only symbolic, I use 1, 10 ,100 here because no one will really use 10 selectors in the code, and the real fact is that the lower priority level selector can’t ever override the higher priority level. So even if you have 100 type selectors, you can’t override even one class selector.

If you happen to write 2 sets of selectors to target one element, and they have the same priority value, then the subsequent selectors will override the previous one. That’s how css is rendered in the web.

And that’s all you have to know about the css selectors’ priority.







Ir no <https://www.dafont.com/pt/> --> escolher a fonte e baixar.

Ir no <https://www.fontsquirrel.com/tools/webfont-generator> --> upload fonts --> convert pacote compatível html --> Download your kit --> usar no

fontpair --> sugere combinação de fonts

