The CSS language

CSS stands for 'Cascading Style Sheets'. For now, do not worry about what the 'Cascading' part means and just focus on the 'Style Sheets'.

Using CSS, we can determine the visual appearance of our HTML elements independent of the HTML itself.

Recall the metaphor we used for HTML with the journalist and the publisher. Where HTML represents the author's work, CSS corresponds to the work the designer does: deciding how things look.

In the early days, there was no CSS, so any control over what the page looked like was done with tags that controlled the form of the Web page. Tags like `<font>` to choose a font, `<b>` for bold, `<i>` for italic were added to have some control, and that let your page be at the mercy of whatever browser the reader was using. There are several problems with this approach. First, it violates our paradigm of HTML containing only content. Second, and more practically, the tags only applied where they were used.

For instance, if you originally wrote your document with all the paragraphs indented with a certain amount and then later you were decided to change the indentation, then you would have to modify every single paragraph in your document. It would be nice if there were a central way to set such rules, i.e. one place that said "I want all my paragraphs to be indented this much", much like master sheets in a word processor. CSS helps to solve this problem.

The W3C CSS Working Group

https://www.w3.org/Style/CSS/Overview.en.html

The **CSS Working Group** (Cascading Style Sheets Working Group) is a working group created by the W3C in 1997 to tackle issues that had not been addressed with CSS level 1. The number of members reaches 126 in December 2017!

The CSS WG members are working on a whole range of specifications, but their core document is CSS snapshot 2017. This document collects together into one definition all the specs that together form the current state of Cascading Style Sheets (CSS) as of 2017. The primary audience is CSS implementers, not CSS authors, as this definition includes modules by specification stability, not Web browser adoption rate.

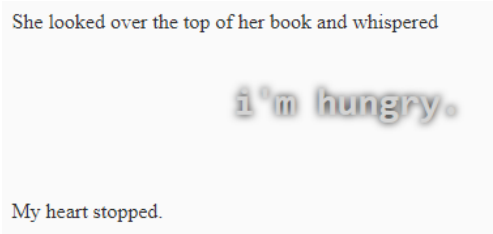
Let's see CSS in action. Below, we see two identical copies of HTML, however, styled differently.

Here is the HTML:

She looked over the top of her book and whispered <q>I'm hungry.</q> My heart stopped.

And now two very different looks:

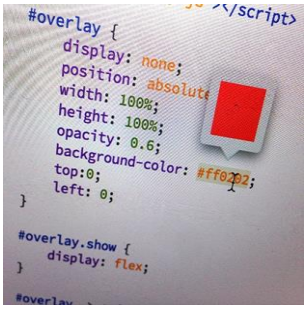
She looked over the top of her book and whispered I'm hungry. My heart stopped.



Both of these use the exact same HTML. It is the CSS that makes them so different. So let's get started.

Style and link tags

<style> tag



The best practice when working with CSS is to keep it in an external file using the `<link>` tag, however, when starting, it is simpler to merely place it directly into the document under edit.

To place CSS directly into an HTML document, we use the `<style>` tag. This tag can appear anywhere in an HTML document, however, the most common practice is to place it in the `<head>` section. Such as:

```
<!DOCTYPE html>
<html lang="en">

 <head>
<meta charset="UTF-8">
<title>Style and link tags</title>
<style>
/* CSS will go in this area */
</style>
</head>
<body>
</body>
</html>
```

<link> tag

While `<style>` is convenient, the better practice is to put the CSS into a separate file. One of the key advantages of using a separate file is that the CSS styles can easily be re-used between your different .html pages. Many authors further divide their CSS up into different files (for example: one for text styles and another one for layout).

Simply put your CSS into a separate file. This file does not need any HTML markup (i.e., no `<style>` tag required). Use the `.css` file extension and use a `<link>` tag to bind it in. The `<link>` tag must appear in the `<head>` section. By convention, css files are kept in a directory named *css*.

Use this `<link>` as a template:

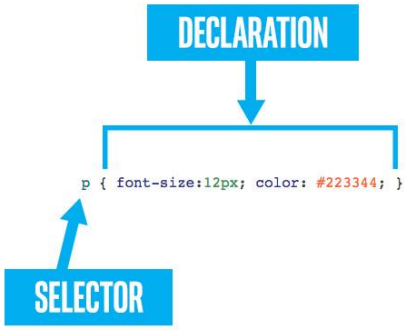
```
<link rel="stylesheet" href="css/my_styles.css">
```

Here is an example HTML document.

```
<!DOCTYPE HTML>
<html lang="en">
<head>
        <meta charset="utf-8">
         <title>CSS text columns</title>
        <link rel="stylesheet" href="css/attributes.css" type="text/css">
</head>
<body>
</body>
</html>
```

Rules: selectors and declarations

At its simplest, CSS is just a list of *rules*. Each *rule* consists of a *selector* and a *declaration*. Here is an example:



Selector

In the above, the *selector* is **p.** When a selector appears unprefixed by any punctuation, then it is assumed to match to an HTML tag. Thus, the **p** selector will apply the CSS rule to all `<p>` tags in the document.

We will cover more selector possibilities in the future.

Declaration

The *declaration* part of a CSS rule opens and closes with curly braces: **{ }**
And between them, you can put any number of *property value* pairs.

Properties and Values

There are hundreds of different visual properties that may be set via CSS. And each property has a range of possible values that it can be set to. Syntactically, property value pairs are simple. Each pair consists of a *property*, followed by a colon **:** followed by a *value* and terminated by a semi-colon **;**

```
font-size: 12px;
```

Best practice

In the example above, the entire CSS rule is written on one line. This is not uncommon when the declaration of the CSS rule only has one property. If a CSS rule has several properties, then it should be written to use one line per property value pair. For example:

```
p {
font-size: 12px;
line-height: 15px;
color: #223344;
}
```

Comments

CSS can include "comments" as well, by which you, the developer today, can leave notes and reminders to you, a different developer tomorrow. Or to others who might read your CSS.

Comments begin with `/*` and **must** end with `*/` and they can span several lines. But they **cannot** be nested.

```
/* Hoja de estilos básica */

body {
        color: gray; background-color: white;
      }

/* Un mismo estilo puede servir para dos o más etiquetas */
h1, h2  {
         color: red
         }
/* dos párrafos con distinto estilo */
.parrafo_especial  {
                        color: green;
                        font-size: 10pt;
                        font-style: italic;
                    }

.parrafo_azul  {
                        color: blue;
                        font-size: 16pt;
                }
```

```
p {
        font-size: 8px; /* client insists small text makes them more 'professional'. */
        /* I hope his idea of 'professional' includes paying on time. */

        line-height: 24px; /* see above */

        /* none of the stuff below is working. I don't know why.

        margin-top: 5%;
        margin-bottom:6%;
        */
        }
```

Activity

https://codepen.io/paqui-molina/pen/ZEEeVzg

Write styles for the above code

Common CSS properties

There are hundreds of CSS properties for you to use. The complete list is available on the W3C Web site (or also, see the CSS reference page on the MDN Web site).

Below we've gathered a more manageable list of the most useful and common CSS properties: `font-size`, `line-height`, `text-align`, `text-decoration`, `font-weight`, `font-style` and `font-family`.

Con respecto al texto:
`font-size`, `line-height`, `text-align`, `text-decoration`, `font-weight`, `font-style y font-family`.

Additionally, `font-size` supports a more readable set of values that many authors prefer: xx-small, x-small, small, medium, large, x-large, xx-large
and relative sizing (relative to the text of the parent): larger, smaller. For example:

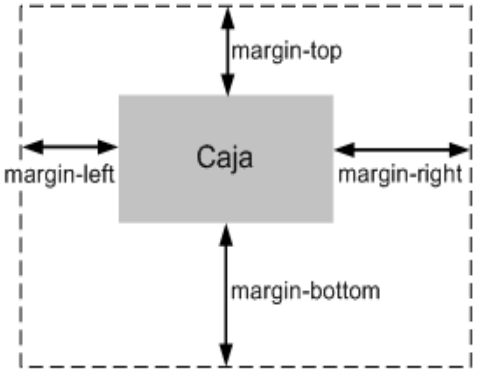
```
p {font-size:medium; }
q{font-size:small; }
blockquote{font-size:larger; }
```

https://codepen.io/paqui-molina/pen/BaaWvoP

line-height

Whereas `font-size` may drive the size of the text itself, the `line-height` property drives the height of the space it is drawn into. A large `line-height` will give the text more spacing. A small `line-height` will smash the text lines together.

https://www.w3schools.com/css/css_font.asp
https://www.w3schools.com/css/css_text.asp



Margin and color:

https://www.w3schools.com/css/css_margin.asp
https://developer.mozilla.org/es/docs/Web/CSS/margin

Color
https://www.w3schools.com/css/css_colors.asp
HSL color
https://www.w3schools.com/colors/colors_hsl.asp

Units:
px, em, rem, %, vh, vw
https://www.w3schools.com/css/css_units.asp

Unidades vw, vh, vmin y vmax
https://www.yunbitsoftware.com/blog/2017/06/22/viewport-units-css-que-es/

Example
https://codepen.io/paqui-molina/pen/ExxWXEx?editors=1100
https://codepen.io/paqui-molina/pen/VwwbyPW
https://codepen.io/paqui-molina/pen/gOOWoLv
https://codepen.io/paqui-molina/pen/WNNjdpL

CSS-tricks
New CSS3 Units
From the W3C specification

Which is the most recommended? (rem or em)
the most recommended is the rem. it makes our website accessible to people with visual impairment, since browsers allow the option to increase the base size of the root element by increasing the rest of the elements accordingly.

Accessible typography

With great power comes great responsibility

The CSS rules with which we've started are fun and easily understandable. They are mostly concerned with typography. Later, we will see how to use CSS to include decorative images, look at other decorative properties, and take up the topic of layout.

But even with our modest start we must, once again, take up the topic of accessibility. Before, we learned that using the correct tag with the best semantic meaning is very important for a variety of reasons, one of which included visitors who may have a disability. If you clearly put your page navigation in a `<nav>` block, and use the header tags and others (like `<article>` or `<main>`), then this can greatly enhance the page experience for certain disabled visitors, like the blind who might be having the page read aloud to them with a screen reader.

Accessibility concerns are important for CSS usage as well. Perhaps doubly so. As page authors, if we don't use CSS, then the page visitor just sees the page with the default typography, and perhaps assisted by tools that can help zoom in on pages, make text bigger, invert colors for the light-sensitive, etc. But as we start to customize the look of the page with CSS, we may unintentionally thwart those tools or make the reading experience less comfortable for those with vision problems.

Guidelines

For accessible typography, there are really just a few things to avoid:

1. do not make text too small
2. do not make lines of text too tight (alto línea)
3. do not use foreground and background colors that are too close to one another, in other words, ensure there is good color contrast
4. do not irregularly space text or make it jump around

Look at those four guidelines. Can you match each guideline to one or more CSS property from earlier? Take a moment and think about it. We'll touch on specific rules below.

Properties

Font-size

Using the wrong font size can make the text too small. So be careful with that. Also, in the past, the gold standard practice was to use em units instead of px. Now, we have other options: rem and vw, vmin and vmax as responsive units.

Line-height

An overly small `line-height` will cause lines to become cramped and difficult to read. Even the largest text can be rendered unreadable by a too small `line-height`. Generally, your `line-height` should always be at least one and a half times the `font-size` (ie, if Font-size is 1em, `line-height` should be greater than `1.5em`).

https://developer.mozilla.org/es/docs/Web/CSS/line-height

Color

Color contrast can be easily undone by misuse of the `color` property. The exact rules for contrast are rather advanced. For example, "wide stroke" text is allowed to have less contrast than narrow stroke text. But, regardless of rules, the overall concept is easy to understand: **keep your text high contrast to the background**. There are further color guidelines concerning certain combinations (like bright blue text on a bright red background), but the rule of thumb is that, **if the text is at all hard for you to read, then** just assume **it is unreadable to someone with a visual disability**. If you are interested, there are tools that can help such as Tanaguru Contrast-Finder or Juicy Studio Luminosity Colour Contrast Ratio Analyser.

Text-align

Any long passage of text should have its alignment match its reading order. Which means, **if the language is English or Spanish, which are read left to right, then** any **long passage of text should be aligned left**. Right aligned or center aligned text can be very hard for dyslexics.

Obviously, a header or perhaps a menu might be exempt, because they are not typically long passages of text. So this guideline doesn't mean an end to good page layout and typography.

Summary

So now, we've seen how typography can affect the accessibility and approachability of your page. It is not so very difficult. **Common sense and awareness are good companions and will serve you well**.

If you are interested in accessibility, there is much more to learn. These simple guidelines merely scratch the surface.

Exercise

Which of the following is **NOT** an acceptable value for the `font-size` property?
Medium; x-small; .4page; 1.2em; .9rem; larger; 14px

Activity-units

https://codepen.io/paqui-molina/pen/vYYxMBJ

With the HTML below, please size the text using different units:

- Use `px` units to set the root size of the text for the document. (Etiqueta HTML)
- Use `rem` units to size the `h1` and `li` tags.
- Change the text size of the root CSS rule. You should observe all the text of the document adjusting appropriately.
- Change the `h1` so that it uses `px` units. As the root CSS rule is changed, the `h1` will no longer adjust with the rest of the document.

Activity

Modify any of the above codes (codepen) so that a different image appears in each section. Three different ways:

- the section covers the entire view
- the section covers half horizontally
- the section occupies the upper left

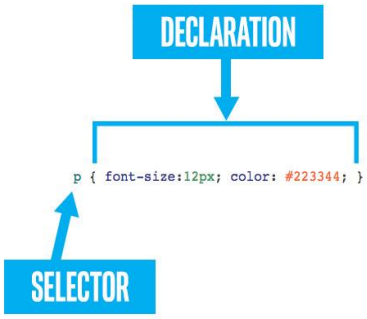
in addition, a section title, h1, responsive size will appear

Styling lists

https://www.w3schools.com/css/css_list.asp

https://developer.mozilla.org/es/docs/Web/CSS/list-style-type

Selectors



Earlier, we learned that a CSS rule is made up of two parts: the selector and the declaration. We've seen quite a few different declarations, but the only selector we've learned is the tag selector. There are other choices, and they can be composed together in interesting and useful ways. So let's learn some more CSS selectors.

Tag selector

We've already seen this one. A CSS selector that consists solely of a single tag (without punctuation or spacing) will be applied to any matching tag on the page.

```
Li {list-style-type:circle; }
```

id selector

You may remember the `id` attribute (short for "identifier"). This attribute can be applied to an HTML tag to uniquely identify the element. Recall that the value for any given `id` attribute can only appear once in a document. No two tags are allowed to have the same id. You may also recall that the id cannot contain spaces, nor most punctuation, nor begin with numbers.

In the HTML below, there are two paragraph tags. So, to style them individually, we can apply unique `id` attributes to the paragraphs (`id="p18"` and `id="p19"`). In the CSS, we will use the id selector. The id selector is simply a hash sign (#) followed directly by the id.

CSS:

```
#p18 {color:blue; }
#p19 {color:green; }
```

HTML:

```
<p id="p18">He is Ulysses, a man of great craft, son of Laertes. He was born in rugged Ithaca, and excels in all manner of stratagems and subtle cunning.</p>
<p id="p19">Madam, you have spoken truly.</p>
```

Result

He is Ulysses, a man of great craft, son of Laertes. He was born in rugged Ithaca, and excels in all manner of stratagems and subtle cunning.

Madam, you have spoken truly.

Class selector

The `class` attribute is similar to the `id`. However, whereas the `id` must be unique and singular, the values of the `class` attribute can be shared by multiple tags. And, multiple classes can be assigned to a tag by simply separating them with spaces.

The class selector is simply a period (.) followed by the class name itself.

`<ul>`
`<li class="bird flying">eagle</li>`
`<li class="bird">ostrich</li>`
`<li class="insect">ant</li>`
`<li class="insect flying">moth</li>`
`</ul>`
	`.bird {color:blue; }`
	`.insect{color:green; }`
	`.flying{ text-decoration:underline; }`
		eagle
		ostrich
		ant
		moth

Combining selectors

Comma separated selectors,

separate	joined

blockquote{
color:red;
font-style:italic;
}
q{
color:red;
font-style:italic;
}
.speech{
color:red;
font-style:italic;
	blockquote,
	q,
	.speech{
	color:red;
	font-style:talic;
	}

The joined version on the right is much easier to read and maintain.

If the "speech" items need to also be bold, that can simply be added by an additional rule:

```
blockquote,
q,
.speech{
color:red;
font-style:italic;
}
.speech{font-weight:bold; }
```

Apply a id to a specific HTML tag

```
a#superenlace{
        color:red;
        font-weight: bold;
        text-decoration: underline;
}
```

With pseudoclass

```
a#superenlace:hover{
        text-decoration: underline;
}
```

https://codepen.io/paqui-molina/pen/EdEqgL

Apply a class to a specific HTML tag

```
p.presentacion{
        color : grey;
        text-align:center
}
```

Nesting class

https://codepen.io/paqui-molina/pen/PyRvRE

Specialized selectors

If two selectors of different types (like tag and class) appear next to each other with no spacing separating them, then they form a specialized selector. To match, a candidate must match **both** rules. If a tag selector is used, it must appear first.

Example

blockquote.speech{font-color:green; -> todas los blockquote con la clase speech

Html	css	result

<li class="bird flying">parrot
<li class="bird">ostrich
<li class="insect">ant
<li class="insect flying">wasp
<li class="insect flying">moth
<li class="flying">airplane
 	
	.insect.flying {
	text-decoration: underline;
	font-weight: bold;
	}
		parrot
		ostrich
		ant
		wasp
		moth
		airplane

Descendant selectors

```
#intro a{color:red;}
#guideline a{color:#00FF00; }
```

https://codepen.io/paqui-molina/pen/zmjONV

Direct descendant selectors (>)

```
#intro > a{font-size:large;}
```

Everything selector (*)

The asterisk (*) can be used to match **any** tag. By itself, this is only marginally useful. But combined with other selectors into a descendant selector, it can be pretty useful.

```
body > *{margin-left:10px; } /*all the _direct_ children of the body receive the margin*/
p *{text-decoration:underline; } /*the text of the paragraph will be normal, but any children anywhere inside it will be underlined*/
```

What if we wanted all the links in the introductory section to be red, but all the link in the guideline section to be green? That is what descendant selectors are for. Here is an example for the problem we are facing:

```
#intro a { color: red; }
#guideline a { color: #00FF00; }
```

We merely separate the tag, identifier, or class selectors by a space.

Adjacent sibling combinator (+)

The + combinator selects adjacent siblings. This means that the second element directly follows the first, and both share the same parent.

General sibling combinator (~)

The ~ combinator selects siblings. This means that the second element follows the first (though not necessarily immediately), and both share the same parent.

https://codepen.io/paqui-molina/pen/pooWNoY

Todos los selectores

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

https://www.w3schools.com/cssref/css_selectors.asp

Cascading: inheritance and precedence

Inheritance

Now that we've covered several ways of defining CSS selectors, we need to understand what happens when multiple selectors resolve to the same element, and how an element can get inherit rules from its parent.

Most CSS rules once applied to an element are also applied to all the children of that element, and to their children, and theirs ad infinitum. There are exceptions, notably the layout properties (margin, padding, position, width, etc.) and the decorative properties (border, background, etc.) do not cascade. This cascading of a CSS property from parent to child is also called "inheritance".

Generally, inheritance is a good thing. Do you want the whole page to use your corporate approved Web-font? body { font-family: "Soulless", serif; } is all you need. There is no need to apply the same font-family property to each and every tag used within the page. Thank you, Cascading!

However, sometimes inheritance can be a bad thing. An element may suddenly display in a way that you weren't expecting and you can't find any relevant CSS rule for that element. In this case, one likely culprit is a CSS rule that has been inherited from a parent. Thanks, Cascading!

Inheritance can be explicitly leveraged. Many CSS properties accept the value of `inherit`, which means to inherit the value from the parent. By smartly leveraging `inherit`, you can reduce repetition in your CSS rules and make your project easier to maintain.

Which rules are inheritable?

There is no reliable rule for which CSS properties are inheritable by default and which are not. However, generally, the properties associated with positioning and layout are *not* inherited. Likewise, the decorative properties (borders, background images, etc.) do not inherit. Most properties that begin with `text-` or `font-`inherit.

Precedence

It is possible, and easy, to have several different CSS rules all applying to the same element. This is often advantageous because most CSS properties are orthogonal to one another, meaning they do not interfere with each other. This gives us freedom to organize the CSS properties in rules in ways that make sense to us as developers, knowing that they can compose nicely. For example, a bit of text can be made italic by one rule, bold by another, and underlined by a third. We do not have to put all those properties into one place if that is not convenient for us.

However, what happens when there are different rules competing to set different values for the same property? This is where CSS precedence comes into play. When rendering CSS, the browser has some guidelines it follows for resolving conflicting rules. Here is rough summary, in order:

1 - Most specific rule

A more specific rule takes precedence over a less specific rule. A rule that more tightly matches a particular element than a general rule will be applied.

```
span { color: blue; }
ul li span { color: red; }
```

In the example above, both rules are attempting to set a span color for a span inside a list item. However, the second rule will "win" when there is a conflict (like color in this case).

2 - `#id` selector is the most specific

Rules with an id selector (e.g. #someid) are considered more specific than rules without.

3- `.class` selector is more specific than a tag selector

Rules employing a class selector (e.g. `.someclass`) are considered more specific than rules without (but not as specific as an #id selector, which trumps everything).

4- Rules that come later override those that come earlier

This guideline is for two CSS rulesets with the same selector. Where there are conflicts, the rules from the later one apply.

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

In the example above, an element with the .hortense class will be underlined and its color will be **blue**, because that rule came later than when it was set red.

No fear

These guidelines seem fairly straightforward, but situations can quickly get rather knotty.

For example, what color should we expect in this situation?

Example:

arbol

```
p.forest span       { color: green; }
p        span.tree  { color: blue;  }
```

What if the order of the CSS rules were reversed? Would it make a difference?

If this problem seems difficult to figure out, don't worry about it. In the next section, we will be looking at Chrome Developer Tools. You'll see how you can use the tools in the browser itself to inspect your elements and see exactly what CSS rules and properties are being inherited, applied and what is their precedence.

Ejemplo:

https://codepen.io/paqui-molina/pen/ePrybr

Pseudoclass

A pseudo-class is a selector that selects elements that are in a specific state

:visited; :focus; :link; :hover; :active

https://codepen.io/paqui-molina/pen/pxKozG

pseudoclase :focus

When they receive the focus.

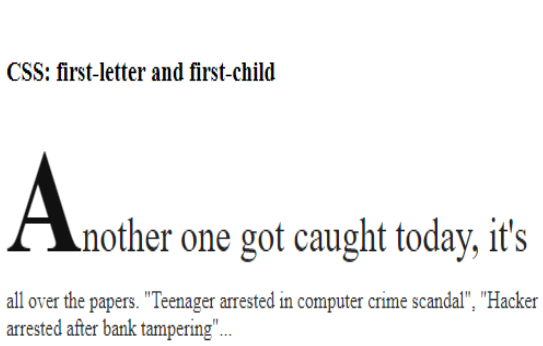
Example.

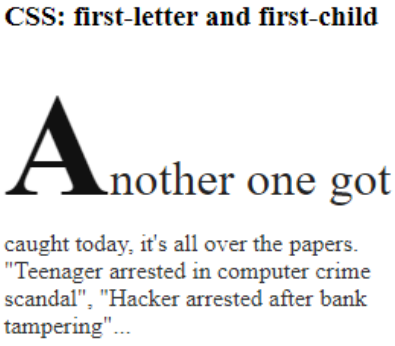
```
Input {
        background-color:white}
input:focus {
        background-color:lightgray
}
```

pseudoclases :first-line, first-letter

These are effects, common in publications such as press or magazines, to highlight the first character or the first sentence of text.

https://codepen.io/paqui-molina/pen/VEdwpz





pseudoclases :firstchild, lastchild

They apply to the first child element and the last child element of a label. Por ejemplo: element <li> of a list <ul>.

https://codepen.io/paqui-molina/pen/bmKGOy

Pseudoclase :nth-child

Useful for alternating styles in structures such as lists or rows of a table. It is a simple effect that allows you to create elegant tables without the need for borders.

Example: menu horizontal dentro

https://codepen.io/paqui-molina/pen/oayggd

Pseudoclase	Ejemplo	Descripción
:nth-child(n)	0	0
:nth-last-child(n)	0	0
:nth-of-type(n)	0	0
:nth-last-of-type(n)	p:nth-last-of-type(2)	Selects every <p> element that is the second <p> element of its parent, counting from the last child

https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/Selectors/Combinators

Pseudoclase :not

Allows styles to be applied to elements that do not comply with the application rule.

https://codepen.io/paqui-molina/pen/ReJNgO

Más sobre pseudoclases …

Attribute selector

From CSS3 you can apply attributes to tags:

```
a[title] {
     font-size=10pt;
}
```

This type of advanced selectors formed by the attribute selectors, allow to select HTML elements based on their attributes and / or values of those attributes.

https://www.w3schools.com/css/css_attribute_selectors.asp

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

Example: https://codepen.io/paqui-molina/pen/ExxQYKM