CSS

Written by [Febogamedev]

CSS, an acronym for Cascading Style Sheets, in Italian fogli di stile a cascata, is the language used to describe the presentation of structured documents, such as *HTML* pages. (W3C CSS Working Group, 2026b)
A file containing CSS code uses the .css extension.

CSS allows control over, among other aspects:

- Colours
- Typography
- Dimensions
- Spacing
- Borders and backgrounds
- Element layout
- Adaptation to different screens
- Interactive states
- Transitions and animations
- Presentation on screen or in print

HTML primarily describes the structure and meaning of content, while *CSS* establishes how that content should be represented. (W3C CSS Working Group, 2026b)
We can summarise the relationship as follows:

- *HTML* → what the content represents.
- *CSS* → how the content is presented.

This separation allows the appearance of the page to be changed without rewriting its structure.

For example, the same *HTML* document can be presented:

- With different colours and typefaces.
- In one or more columns.
- On a smartphone or a large screen.
- In light or dark mode.
- On screen or in print.

A stylesheet is a set of *CSS* rules. (W3C CSS Working Group, 2021b)
Each rule establishes:

- Which elements should be selected.
- Which properties should be applied.
- Which values should be assigned to these properties.

For example:

```
a {
```

```
  color: darkblue;
  font-weight: 700;
  text-decoration-thickness: 2px;
}
```

This rule selects all *HTML* <a> elements and assigns the text the colour darkblue, a font weight of 700 and a thickness of 2px for any decoration lines, such as underlines, overlines and strikethroughs.

As we have seen from the rule, CSS does not change the meaning of the element (for example, a link remains a link), but changes its presentation.

An external stylesheet can be associated with the *HTML* document through the <link> element, normally inserted inside <head>. (WHATWG, n.d.)

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

- The “rel” attribute with the value "stylesheet" indicates that the linked resource is a stylesheet. (WHATWG, n.d.)
- The “href" attribute with the value “style.css" indicates the *URL* of the *CSS* file (in this example, the *URL* is relative and is resolved against the base address of the *HTML* document). (WHATWG, n.d.)

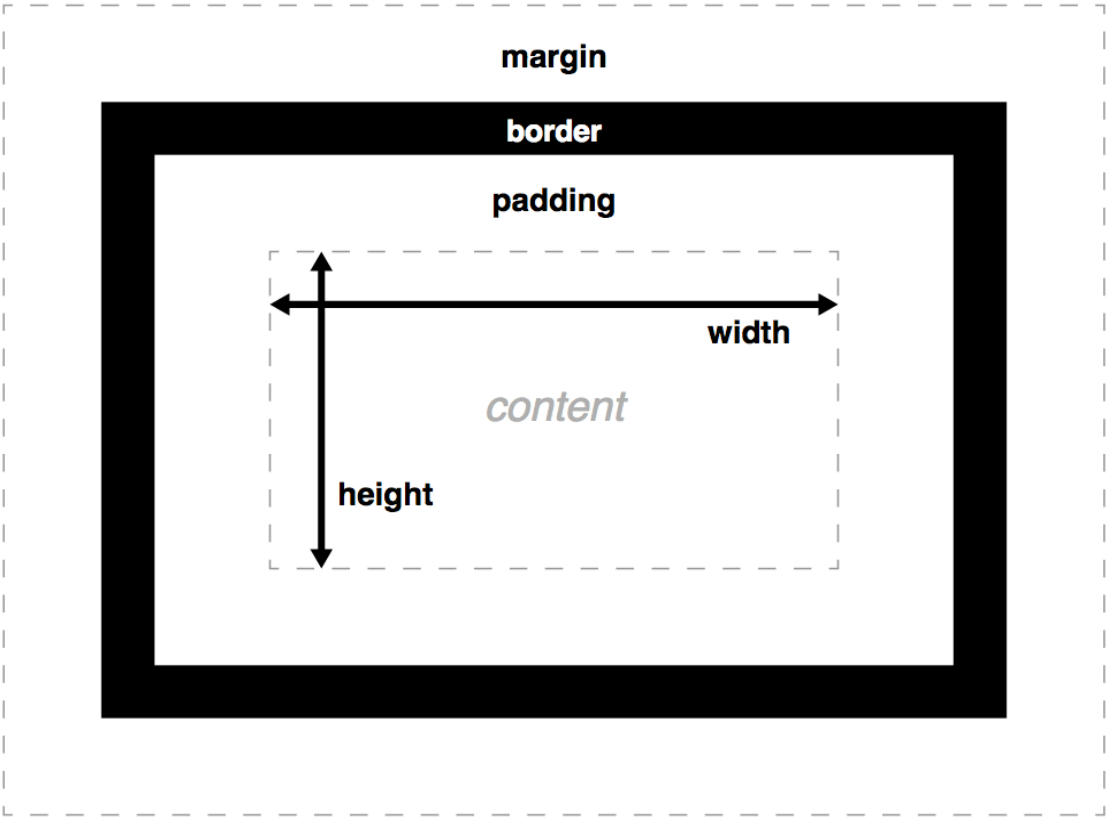
Using an external file to “decorate” the *HTML* file allows the same visual system to be applied to many pages.
A change to the stylesheet can therefore update the appearance of an entire website simultaneously.
CSS can also be inserted directly into the *HTML* document, but external stylesheets generally facilitate organisation, reuse and maintenance. (W3C CSS Working Group, 2026b)

Box model

To visually represent a document, the browser generates rectangles, called boxes, associated with the *HTML* elements that participate in the page layout. (W3C CSS Working Group, 2026a)

The fundamental model used to describe these rectangles is called the **box model**. (W3C CSS Working Group, 2024b)
Each box consists of four areas nested around one another. Moving from the outside inwards, we find: (W3C CSS Working Group, 2024b)

- Margin
- Border
- Padding
- Content



Content
The content area is the innermost part of the box. Text, images, videos and the boxes generated by any descendant elements are placed within it. It does not include padding, border or margin.

Its width and height are not always specified directly, but are calculated by the browser taking several factors into account.

- The content: when dimensions are automatic, the content can help determine them. For example, the height of a paragraph increases when the text occupies more lines, while an image can start from its natural dimensions. (W3C CSS Working Group, 2021a)
- The width and height properties: allow the width and height of the element to be specified respectively. With the default value box-sizing: content-box, a declaration such as width: 300px assigns a width of 300px to the content area. Padding and border are added outside this measurement. (W3C CSS Working Group, 2021a)
- The containing block: when a dimension is expressed as a percentage, the browser must calculate it relative to a reference rectangle called the containing block. In normal flow, this rectangle often coincides with the content area of the parent element. If the containing block is 800px wide, a child element with width: 50% will have a width of 400px. (W3C CSS Working Group, 2021a)
- The layout system: the way elements are organised can influence their dimensions.
    - In normal flow, block elements with automatic width generally occupy the available horizontal space.
    - With Flexbox, child elements can grow or shrink to share space along a row or a column. (W3C CSS Working Group, 2025a)

    - With Grid, on the other hand, they are placed in a structure of rows and columns whose dimensions influence those of the elements they contain. (W3C CSS Working Group, 2025b)
- The minimum and maximum limits: the min-width, max-width, min-height and max-height properties prevent the element from becoming smaller or larger than the established limits. (W3C CSS Working Group, 2021a)

These factors can act simultaneously. For example, an element can have width: 50% and max-width: 600px: it will occupy half the width of the containing block, without exceeding 600 px.

It is important to note that the box-sizing property determines which areas should be included in the dimensions assigned through width and height. (W3C CSS Working Group, 2021a)

- With the default value content-box, width and height refer only to the content area:
- With border-box, on the other hand, the measurement includes content, padding and border:

Padding

Padding is the internal space between the content and the border. Increasing it moves the content away from the element's border. The background is normally also drawn in the padding area. (W3C CSS Working Group, 2024a)

The four sides can be controlled separately through:

- padding-top
- padding-right
- padding-bottom
- padding-left

Border

The border surrounds the content and padding. It can be controlled through its thickness, style and colour. The border-radius property also allows its corners to be rounded. (W3C CSS Working Group, 2024a)

Margin

The margin is the space outside the border that separates the box from adjacent elements. (W3C CSS Working Group, 2024b)

It is transparent: the element's background does not cover it. (W3C CSS Working Group, 2024b)

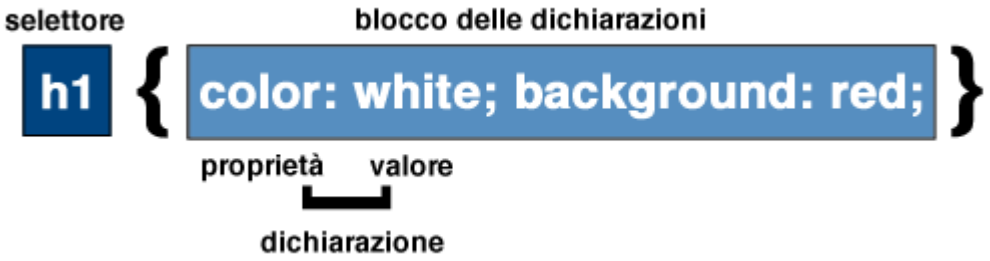
As with padding, its four sides can be controlled separately using:

- margin-top
- margin-right
- margin-bottom
- margin-left

Rules

A *CSS* style rule consists of: (W3C CSS Working Group, 2021b)

- a selector
- a declaration block



Selectors indicate which *HTML* elements are affected by the rule. (W3C CSS Working Group, 2026d)
Selectors can also refer to classes, identifiers, attributes, relationships between elements and interface states.

Example

```
h1 {
  color: white;
  background-color: darkred;
}
```

This rule is applied to all <h1> elements.

The declaration block is delimited from the selector and the other rules by two curly braces, the first opening and the second closing. It can contain several declarations. (W3C CSS Working Group, 2021b)

A declaration consists of a pair:
- property
- value

Each declaration consists of a **property** and a **value**, separated by a colon. (W3C CSS Working Group, 2021b)
The property indicates the aspect to act on, while the value specifies how it should be set.

Example

```
h1 {
  color: white;
  background-color: darkred;
}
```

This declaration block has two declarations

- In the first declaration, color is the property and white is the value.
- In the second declaration, background-color is the property and darkred is the value.

Declarations, on the other hand, must be separated by a semicolon. (W3C CSS Working Group, 2021b)
It is not mandatory, but it is good practice to put a semicolon after the last declaration in the block as well.

Some properties, called **shorthand properties**, allow several related properties to be set simultaneously.

Example

```
div {
  margin: 10px 5px 20px 0;
}
```

margin is a shorthand that sets the top, right, bottom and left margins, in that order.
The rule is equivalent to:

```
div {
  margin-top: 10px;
  margin-right: 5px;
  margin-bottom: 20px;
  margin-left: 0;
}
```

Attribute selectors identify HTML elements based on the presence or value of one of their attributes. (W3C CSS Working Group, 2026d)
For example, [disabled] selects all elements that have the disabled attribute, while input[type="email"] selects only <input> elements whose type attribute has the value email.
It is also possible to select values that begin with, end with or contain a particular sequence. For example: (W3C CSS Working Group, 2026d)

- a[href^="https"] identifies links whose address begins with https.
- a[href$=".pdf"] those that end with .pdf and a[href*="wikipedia"] those that contain the word wikipedia.

Example

```
input[type="email"] {
    border: 2px solid blue;
```

```
}
```
This rule applies to <input> elements whose type attribute has the value email.

Pseudo-elements are components that can be used within a selector to represent a part or content associated with the selected element without adding new elements to the *HTML* code. (W3C CSS Working Group, 2025c)
They are generally written with two colons "::".
For example, "::first-letter" selects the first letter of a text, "::first-line" the first line, while "::before" and "::after" allow content to be generated before and after the element's content respectively. (W3C CSS Working Group, 2025c)
Pseudo-elements do not represent actual elements present in the HTML document, but parts or content created during the graphical representation of the page.

Example

```
p::first-letter {
    color: red;
    font-size: 40px;
    font-weight: bold;
}
```

This rule selects and formats only the first letter of the p element. That letter will be displayed larger, red and bold.

Several simple selectors can be combined to identify elements more precisely. (W3C CSS Working Group, 2026d)
When they are joined without a combinator, as in input[type="email"], the conditions refer to the same element.
A space, on the other hand, is a **descendant combinator** and requires the element specified on the right to be inside the one specified on the left. (W3C CSS Working Group, 2026d)
A pseudo-element is normally placed at the end of the selector and identifies a part of the selected element.

Example

```
form input[type="email"]::placeholder {
    color: blue;
    font-style: italic;
}
```

This selector identifies the ::placeholder pseudo-element of <input> elements with the attribute type="email" that are inside a <form> element.

So, to recap:

- *CSS* consists of rules.
- Each rule consists of a selector and a declaration block.
- Each declaration consists of a property and a value.

The hierarchical diagram is shown below:

```
Stylesheet
└── Style rule 1
    ├── Selector
    └── Declaration block
        └── Declaration
            ├── Property
            └── Value
└── Style rule 2
    ├── ....
```

IDs and classes

To apply styles to specific elements, CSS provides several types of selectors. Among the most commonly used are: (W3C CSS Working Group, 2026d)

- **Class selectors**, which identify elements based on the HTML class attribute;
- **ID selectors**, which identify an element based on the HTML id attribute;
- **Pseudo-classes**, which identify elements based on a condition, position or state, without requiring an attribute to be added to the HTML code.

Class selectors

The “class” attribute allows the same name to be assigned to several HTML elements. (WHATWG, n.d.)

Example:

```html
<article class="card">
  <h2>First project</h2>
</article>

<article class="card">
  <h2>Second project</h2>
</article>
```

Both elements belong to the card class.
In CSS, a class is selected by writing its name preceded by a dot:

```css
.card {
  padding: 1rem;
  border: 1px solid gray;
  border-radius: 0.5rem;
}
```

The rule is applied to all elements that have the card class.

Classes are therefore particularly suitable for defining reusable components or styles, such as cards, buttons, messages, headings, grids, colour variants, etc.

The same element can have several classes, separated by a space: (WHATWG, n.d.)

```html
<a class="pulsante pulsante-evidenziato" href="/contatti">
  Contact
</a>
```

In this case, the element belongs to both the “pulsante” class and the “pulsante-evidenziato” class.

Below we can see what these classes do:

```css
.pulsante {
  display: inline-block;
  padding: 0.75rem 1rem;
  text-decoration: none;
}

.pulsante-evidenziato {
  background-color: darkblue;
  color: white;
}
```

- The first class defines the characteristics common to all buttons.
- The second adds a particular visual variant.

This system allows different rules to be combined without having to create a specific class for every possible combination.

ID selectors

The “id” attribute assigns an identifier to an element. (WHATWG, n.d.)

```html
<header id="intestazione-principale">
  ...
</header>
```

The value of an “id” attribute must be unique within the same *HTML* document. (WHATWG, n.d.)
It is therefore incorrect to assign the same identifier to two elements.

In CSS, an “id” is selected by writing its name preceded by the # symbol:

```
#intestazione-principale {
  position: sticky;
  top: 0;
}
```

The rule is applied to the element whose “id” is intestazione-principale.

An “id” is not used only by *CSS*.
It can also serve as the destination of a *URL* fragment. (WHATWG, n.d.)
For example, this element has the “id” attribute:

```
<section id="contatti">
  <h2>Contact</h2>
</section>
```

The following hyperlink, which is a link, leads to the section whose id has the value contatti

```
<a href="#contatti">Go to contact details</a>
```

The *URL* of this link might look like this: https://example.com/#contatti
The #contatti part is the fragment *of the URL* and corresponds to the element whose id is “contatti”. (WHATWG, n.d.)

An id can also establish relationships between HTML elements.
For example, the value of a label’s for attribute must match the id of the field to which the label refers: (WHATWG, n.d.)

```
<label for="email">Email address</label>
<input id="email" name="email" type="email">
```

In this way, the browser explicitly associates the label with the field.

Pseudo-classes
Pseudo-classes are keywords added to a CSS selector to identify an element when it is in a particular condition, position or state, without having to add a class to the *HTML* code. (W3C CSS Working Group, 2026d)

Pseudo-classes are written by adding a single colon character (:) to the selector, followed by the name of the pseudo-class.

For example:

- :hover identifies an element when the pointer is over it;
- :focus identifies an element that has received focus;
- :checked identifies a selected checkbox or radio button;
- :first-child identifies an element that is the first child of its parent.

Unlike pseudo-elements, which represent a part of the element or generated content, pseudo-classes change whether a selector matches based on a particular condition. They can be dynamic, such as :hover, or structural, such as :first-child. (W3C CSS Working Group, 2026d)

Example

```
a:hover {
    color: red;
    text-decoration: underline;
}
```

The link (which is represented by the a element) becomes red and underlined when the user places the pointer over it.

Cascade

Several CSS declarations can assign different values to the same property of the same element. (W3C CSS Working Group, 2024c)
For example, one rule can assign the colour red to a paragraph and another the colour blue. The browser must therefore determine which declaration to apply.

The process through which these conflicts are resolved is called the **cascade**. (W3C CSS Working Group, 2024c)

For each property of each element, the browser first identifies the applicable declarations. The selector must match the element, and any conditions must be satisfied.
For example, a rule contained in @media participates in the cascade only when the condition is satisfied, for example when the window has certain dimensions. (W3C CSS Working Group, 2026c)

Next, the browser orders the declarations according to several priority criteria. In simplified terms, the main ones are:

Origin and importance
The browser first considers where a declaration comes from: (W3C CSS Working Group, 2024c)

- The browser’s default styles.

- The user's preferences or styles.
- The CSS written by the page's author.

Among normal declarations, the author's generally take precedence over the user's and the browser's.

A declaration can be marked with !important:

```
.avviso {
    color: blue !important;
}
```

Importance is evaluated before specificity. (W3C CSS Working Group, 2024c)
An !important declaration can therefore override a normal declaration associated with a more specific selector. However, when several !important declarations are compared, the other cascade criteria continue to be considered.

Cascade layer
Through @layer, the author's rules can be organised into layers with different priorities: (W3C CSS Working Group, 2024c)

```
@layer base, componenti;
```

Among normal declarations, those contained in layers declared later take precedence over those in earlier layers.
 Normal declarations not explicitly placed in a layer have higher priority than those placed in layers.

The layer is compared before specificity. A less specific selector placed in a higher-priority layer can therefore override a more specific selector belonging to a lower layer.

For !important declarations, the priority order of the layers is reversed.

Selector specificity
When two declarations belong to the same origin, have the same importance and are in the same layer, the browser compares the specificity of the selectors. (W3C CSS Working Group, 2024c)

In general terms:
- ID selectors, such as #introduzione, have higher specificity.
- Classes, attributes and pseudo-classes come next.
- Element selectors and pseudo-elements come next.

Consider this example:

```css
#messaggio {
    color: red;
}

.avviso {
    color: blue;
}
```

```html
<p id="messaggio" class="avviso">
    Warning
</p>
```

Both rules select the paragraph.
Since they have the same origin and importance and are in the same layer, their specificity is compared. The #messaggio selector is more specific than .avviso, so the text is red. (W3C CSS Working Group, 2024c)

The higher specificity of ID selectors can make styles more difficult to override and reuse. For this reason, many projects prefer classes for defining the presentation of components, while IDs are used mainly to identify unique elements, create destinations for URL fragments and establish relationships between HTML elements.

Order of appearance
If specificity is also equal, the declaration that appears last takes precedence: (W3C CSS Working Group, 2024c)

```css
.avviso {
    color: red;
}

.avviso {
    color: blue;
}
```

In this case, the text is blue because the two declarations have the same origin, importance, layer and specificity, but the second appears later in the stylesheet.

Order of appearance is therefore the final tie-breaking criterion. It is incorrect to say that “the last rule always wins”: this happens only when all the preceding criteria are equivalent. (W3C CSS Working Group, 2024c)

Note:
The description presented is a simplified account of the most common cases. The full cascade also considers styles applied directly through the style attribute, animations,

transitions, encapsulation contexts and, when used, @scope. (W3C CSS Working Group, 2024c)

Fluid layout and media queries

Web pages can be viewed in windows and on devices of very different sizes. CSS allows their presentation to be adapted to the available space through two main tools: (W3C CSS Working Group, 2026c)

- Fluid layouts
- Media queries

These two approaches are used together to create a Web page that is as suitable as possible for different screen sizes.
A **mobile firs**t approach is also often followed: rules for narrower screens are written first, and configurations for wider spaces are progressively added through media queries with min-width. (W3C CSS Working Group, 2026c)

Fluid layout
In a fluid layout, element dimensions change gradually in relation to the available space. Elements can also shrink, grow or redistribute themselves according to the rules defined through Flexbox or Grid. (W3C CSS Working Group, 2025a, 2025b)
Proportional values and flexible units, such as percentages and “fr”, are used together with properties such as max-width to prevent content from becoming excessively large. (W3C CSS Working Group, 2024d, 2025b)

An example is shown below:

```
.pagina {
    width: 90%;
    max-width: 1200px;
    margin: 0 auto;
}

img {
    max-width: 100%;
    height: auto;
}

.galleria {
    display: grid;
    grid-template-columns: 1fr;
    gap: 20px;
}
```

The element with class .pagina occupies 90% of the available width, up to a maximum of 1200px.
The declaration margin: 0 auto centres it horizontally within its container.

Images can shrink so that they do not exceed the width of the container, while height: auto preserves their original proportions. (W3C CSS Working Group, 2021a)

The element with class .galleria is arranged as a single-column grid. The column has a size of 1fr and occupies all the available space. (W3C CSS Working Group, 2025b)
The gap property establishes a distance of 20px between the grid elements.

Media queries
Media queries allow certain *CSS* declarations to be applied only when specific conditions are met, for example when the viewport reaches a certain width. (W3C CSS Working Group, 2026c)

A media query uses an **at-rule**, that is, a special *CSS* rule that begins with the @ symbol and provides an instruction to the CSS engine, defining the context in which the rules it contains should be applied. (W3C CSS Working Group, 2026c)

```
@media (min-width: 600px) {
    .galleria {
        grid-template-columns: repeat(2, 1fr);
    }
}

@media (min-width: 1000px) {
    .galleria {
        grid-template-columns: repeat(3, 1fr);
    }
}
```

The **@media** at-rule ensures that the declarations it contains are applied only when the specified condition is true. (W3C CSS Working Group, 2026c)

In this case, min-width: 600px requires the viewport width to be at least 600 CSS pixels.

The values 600px and 1000px are called **breakpoints**, that is, points at which the layout organisation changes. The gallery uses one column in narrower viewports, two columns from 600px and three columns from 1000px. (W3C CSS Working Group, 2026c)

When the viewport reaches 1000px, both media queries are true.
Since the declarations have the same priority and specificity, the one that appears last takes precedence and the gallery is organised into three columns. Between breakpoints, the columns continue to adapt to the available space thanks to the flexible fr unit. (W3C CSS Working Group, 2024c, 2025b)

Another example:

```
.pagina {
```

```
    width: calc(100% - 32px);
    margin: 0 auto;
}

@media (min-width: 768px) {
    .pagina {
        width: 720px;
    }
}

@media (min-width: 1200px) {
    .pagina {
        width: 1120px;
    }
}
```

In this example, in viewports below 768px, the page occupies all the available width minus 32px, leaving a total of 32px of space at the sides. From 768px it takes on a fixed width of 720px, while from 1200px it changes to 1120px.

CSS example

Let us look at a very simple example of using *CSS*.
Let us take some code in *HTML*.

```
<!doctype html>
<html lang="it">

<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">

  <title>CSS example</title>

  <link rel="stylesheet" href="style.css">
</head>

<body>
  <main>
    <h1>My website heading</h1>
    <p>This is an example paragraph.</p>
  </main>
</body>

</html>
```

The <link> element connects an external resource declared as a stylesheet to the document. (WHATWG, n.d.)
Indeed, we can see that the HTML document, through the <link> element, links an external stylesheet called style.css

Let us look at the contents of "*style.css*"

```css
*,
*::before,
*::after {
  box-sizing: border-box;
}

body {
  margin: 0;
  font-family: system-ui, sans-serif;
  line-height: 1.5;
}

main {
  width: min(100% - 2rem, 60rem);
  margin-inline: auto;
  padding-block: 2rem;
}

h1 {
  color: darkblue;
  font-size: 2rem;
}

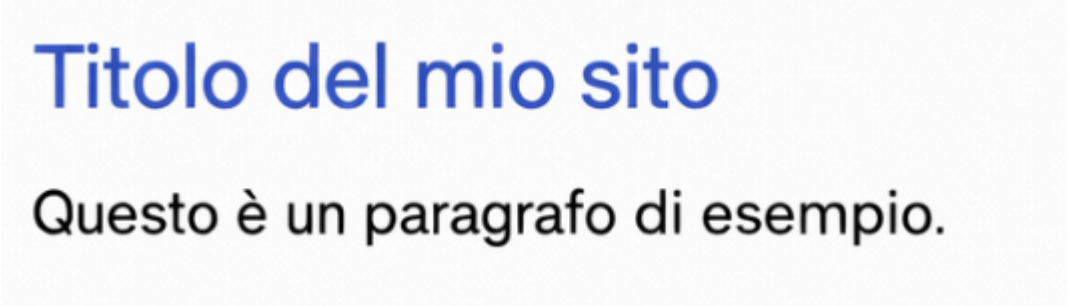
p {
  color: #222;
  font-size: 1rem;
}
```

In this .css we can see that formatting rules are also defined for two elements, "*h1"* and "*p".*
This means that the rules associated with the h1 and *p* selectors must comply with the formatting rules imposed by the .css document.
In this case:

- The contents of the h1 element will be darkblue and the font size will be 2rem.
- The contents of the p element will be #222 and the font size will be 1rem.

Note that, to display the page, the browser must retrieve the *HTML* document and, if available, the linked *CSS* stylesheet; if the *CSS* is not retrieved, the *HTML* document can still be displayed, simply without the styles defined in that stylesheet. (WHATWG, n.d.)

The Web page produced using these two documents is shown below.



Supplementary material

Exercises

Separating structure and presentation
Consider this HTML document:

```
<main>
    <h1>Photography portfolio</h1>
    <p>A selection of my work.</p>
    <a href="/portfolio">View the projects</a>
</main>
```

The designer wants to achieve these characteristics:

- The heading must be dark blue and large.
- The paragraph must be grey.
- The link must look like a button.
- All pages of the website must use the same visual system.

Answer the following questions:

- Explain why it is more appropriate to use CSS than to modify the HTML structure to achieve these results.
- Write a possible CSS stylesheet that implements the required characteristics.
- For each rule, identify the selector, properties and their values.
- Explain why changing the link's graphical appearance does not change its meaning as a link.
- Indicate how you would link a style.css file to the HTML document.
- Explain why using an external stylesheet can be advantageous in a website consisting of fifty pages.
- Imagine that you want to change the main colour of the entire website. Explain why separating HTML and CSS makes this operation easier.

How large is the box really?
An element uses this CSS:

```
.card {
    width: 300px;
    padding: 20px;
    border: 5px solid black;
    margin: 30px;
}
```

By default, the following is used:

```
box-sizing: content-box;
```

- Indicate the width of the content area.

- Calculate the total width occupied by content, padding and border.
- Calculate the total space including the side margins.
- Explain the difference between padding and margin.
- Indicate which of the two areas is normally covered by the element's background.
- Rewrite the rule using box-sizing: border-box.
- Explain how the meaning of width: 300px changes when using border-box.
- A container is 800 px wide and a child element has width: 50%. Explain how wide its content area will be, assuming that the containing block corresponds to that container and that no other constraints apply.
- Add max-width: 300px and explain what would happen if the container became 1000 px wide.

Choosing the correct selector

Consider this HTML:

```
<form>
    <input type="text" placeholder="Name">
    <input type="email" placeholder="Email">
    <button disabled>Submit</button>
</form>

<a href="manuale.pdf">Download the manual</a>
<a href="https://example.com">Visit the site</a>
```

Write an appropriate CSS selector to:

- Select only the email field.
- Select all elements that have the disabled attribute.
- Select only links that lead to a PDF file.
- Select only links whose address begins with https.
- Change the colour of the email field's placeholder.
- Change a link's style when the user hovers over it with the mouse.
- Explain the difference between an attribute selector, a pseudo-class and a pseudo-element.
- Explain why :hover describes a state of the element, while ::placeholder identifies a specific part of its representation.
- Finally, write a selector that identifies the email field's placeholder only when it is inside a <form>.

Class or ID?

You need to design a page containing:

- Twelve project cards.
- Three main buttons.
- One of these buttons must have a highlighted variant.
- A single main header.
- A section with the identifier contatti, reachable through an internal link.

Design the structure of the identifiers and classes.

- Indicate which elements you would manage through classes.

- Indicate which element could be identified through an ID.
- Propose a reusable class for all the cards.
- Propose a general class for the buttons and a second class for the highlighted variant.
- Write an example of an HTML element that uses both classes.
- Explain why it is correct to assign the same class to several elements.
- Explain why, on the other hand, an id value must be unique within the document.
- Write the link that allows the contatti section to be reached.
- Explain why, according to the handout, classes are generally more suitable than IDs for defining reusable visual components.

Which rule wins?

Consider this code:

```
<p id="messaggio" class="avviso">
    Warning
</p>
```

and this CSS:

```
p {
    color: green;
}

.avviso {
    color: blue;
}

#messaggio {
    color: red;
}

.avviso {
    color: orange;
}
```

- Indicate which rules apply to the paragraph.
- Determine which colour will be displayed.
- Explain why the #messaggio selector takes precedence over the .avviso selectors.
- Explain why it is incorrect simply to say “the last rule always wins”.
- Remove the rule with #messaggio and determine which colour will be applied.
- Explain why orange takes precedence in this second case.
- Add the declaration color: blue !important; to the first .avviso rule and determine the result again.
- Explain the roles of importance, specificity and order of appearance in the cascade.
- Explain why excessive use of IDs and !important can make styles more difficult to maintain.

Designing a responsive gallery

You need to create a photography gallery that works well on smartphones, tablets and desktops.
The requirements are:

- There must be only one column on small screens.
- From 600 px, two columns must appear.
- From 1000 px, three columns must appear.
- The page must never exceed 1200 px in width.
- Images must not extend beyond their container.
- The layout must continue to adapt fluidly between breakpoints as well.

Write possible CSS using Grid, relative units, max-width and media queries.
Then:

- Explain why the solution can be described as responsive.
- Identify which parts of the solution constitute a fluid layout.
- Identify the breakpoints.
- Explain what happens at a viewport width of 500 px.
- Explain what happens at a viewport width of 800 px.
- Explain what happens at a viewport width of 1200 px.
- At 1200 px, both media queries are true. Explain why the three-column configuration is still used.
- Explain the meaning of the fr unit in the grid.
- Explain why max-width: 100% and height: auto can be useful for images.
- Explain why this setup can be considered mobile first.
- Conclude by explaining the difference between the continuous adaptation of a fluid layout and the conditional change introduced by media queries.

Bibliography

WHATWG. (n.d.). *HTML Standard*.

W3C CSS Working Group. (2021a). *CSS Box Sizing Module Level 3*. World Wide Web Consortium.

W3C CSS Working Group. (2021b). *CSS Syntax Module Level 3*. World Wide Web Consortium.

W3C CSS Working Group. (2024a). *CSS Backgrounds and Borders Module Level 3*. World Wide Web Consortium.

W3C CSS Working Group. (2024b). *CSS Box Model Module Level 3*. World Wide Web Consortium.

W3C CSS Working Group. (2024c). *CSS Cascading and Inheritance Level 6*. World Wide Web Consortium.

W3C CSS Working Group. (2024d). *CSS Values and Units Module Level 4*. World Wide Web Consortium.

W3C CSS Working Group. (2025a). *CSS Flexible Box Layout Module Level 1*. World Wide Web Consortium.

W3C CSS Working Group. (2025b). *CSS Grid Layout Module Level 2*. World Wide Web Consortium.

W3C CSS Working Group. (2025c). *CSS Pseudo-Elements Module Level 4*. World Wide Web Consortium.

W3C CSS Working Group. (2026a). *CSS Display Module Level 3*. World Wide Web Consortium.

W3C CSS Working Group. (2026b). *CSS Snapshot 2026*. World Wide Web Consortium.

W3C CSS Working Group. (2026c). *Media Queries Level 4*. World Wide Web Consortium.

W3C CSS Working Group. (2026d). *Selectors Level 4*. World Wide Web Consortium.

Further reading

- Eric A. Meyer and Estelle Weyl, CSS: The Definitive Guide, 5th ed., O'Reilly Media, 2023 — Probably the most comprehensive reference for this handout. It systematically examines CSS syntax, selectors, pseudo-classes, pseudo-elements, the cascade, specificity, inheritance, the box model, positioning, layout systems, and responsive design. Its detailed explanations make it particularly valuable for understanding not only how CSS properties work, but also how browsers determine which rules apply and how those rules interact.
- Keith J. Grant, CSS in Depth, 2nd ed., Manning Publications, 2024 — One of the most directly relevant books for the chapter's practical and conceptual approach. It explains the cascade, specificity, inheritance, relative units, the box model, Flexbox, Grid, responsive design, and cascade layers, showing how these concepts work together when building real interfaces. It also explores CSS architecture, reusable components, maintainability, and modern features such as container queries and logical properties.
- Jennifer Niederst Robbins, Learning Web Design: A Beginner's Guide to HTML, CSS, JavaScript, and Web Images, 6th ed., O'Reilly Media, 2025 — The most accessible general introduction for students. It explains how CSS is connected to HTML and progressively introduces stylesheets, selectors, properties, values, typography, colours, backgrounds, spacing, the box model, and responsive layouts. Its practical exercises make it particularly suitable for readers who want to apply the theoretical concepts introduced in the handout.
- Ethan Marcotte, Responsive Web Design, 2nd ed., A Book Apart, 2014 — The most specialized reference for the section on fluid layouts and media queries. Marcotte develops the principles of responsive design through fluid grids, flexible images, and CSS media queries, explaining how layouts can adapt to different viewport dimensions without requiring separate websites for each device. Although some implementation details reflect an earlier generation of CSS, the underlying design principles remain highly relevant.
- Lea Verou, CSS Secrets: Better Solutions to Everyday Web Design Problems, O'Reilly Media, 2015 — A practical collection of advanced CSS techniques for solving common interface design problems. It covers backgrounds, borders, typography, visual effects, layout, transitions, and animations, demonstrating how combinations of CSS properties can produce sophisticated results without unnecessary complexity. It is particularly useful for progressing from understanding individual CSS rules to designing complete visual solutions.

Editorial notes

In producing these materials, I have used generative artificial intelligence tools to support the writing process, particularly to improve the form of the text, reorganize content, refine wording, and speed up certain editorial tasks.

As I work independently on the production of these materials, I try to automate any activities that can reasonably be automated, so that I can devote more time to research, design, and content development.

However, artificial intelligence does not determine the content of this work: the choice of topics, structure, ideas, interpretations, examples, and teaching approach are developed by me. AI is therefore used primarily as a tool to support production and formal revision, while authorship and responsibility for the design and content remain mine.

Unless otherwise indicated, this material is distributed under the Creative Commons Attribution–NonCommercial–ShareAlike 4.0 International license (CC BY-NC-SA 4.0).

You are therefore free to share, redistribute, adapt, and create derivative works based on this material, provided that appropriate credit is given, the material is not used for commercial purposes, and any modified or derivative versions are distributed under the same license.

