Front-end <Web /> Development

Lesson 4: CSS (Part 2)

University of Tehran

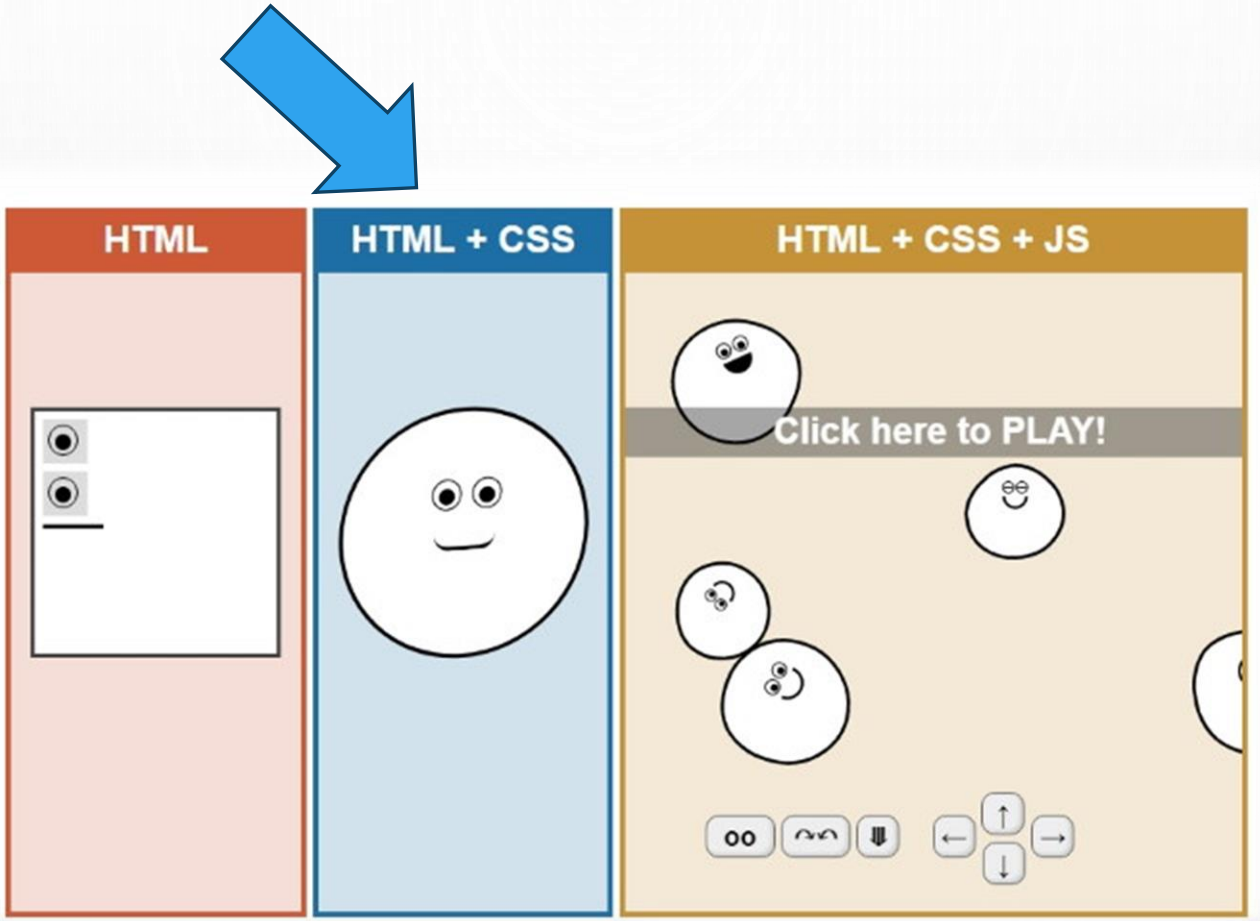
ACM
Summer of Code 2024

Lecturer:
Misagh Mohaghegh

A Short Review…

- Cascading Style Sheets
- How to include CSS (external, internal, inline)
- Simple selectors (tag, #id, .class)
- Attribute selectors ([attr], p[attr="test"], …)
- Combinator selectors (h1 p, p>span)
- Colors (RGB, Hex, HSL)
- Text Formatting (alignment, direction, decoration, transformation, spacing, shadow)
- Display type (block, inline)
- Box model (padding, border, margin, box-sizing)

Continue…



Cascading Style Sheets

Fonts

Font Types

Fonts are generally one of the 5 types below:

- Sans-serif
- Serif
- Monospace
- Cursive
- Fantasy

Arial (Sans-serif)

Times New Roman (Serif)

Consolas (Monospace)

Lucida Handwriting (Cursive)

Papyrus (Fantasy)

Font Family

To set the font of an element, we use the **font-family** property.
We can specify multiple fonts as fallback fonts (to be used when the font is not found).
It is good practice to end the list with a generic font type to let the browser choose if everything else fails.

```
.paragraph {
    font-family: Arial, Helvetica, sans-serif;
}
```

If a font has space in its name, it should be quoted:

```
.paragraph {
    font-family: "Times New Roman", serif;
}
```

Font Properties

The **font-size** property sets the font size (units such as px, em, rem, %, vw).
The **font-style** property can set the text to be italic.
The **font-weight** property sets the weight of the font (100-900). 400 is normal, 700 is bold.

```
<head>
    <title>Text Font</title>
    <style>
        p.large { font-size: 32px; }
        p.italic { font-style: italic; }
        p.bold { font-weight: bold; }
        p.weight800 { font-weight: 800; }
    </style>
</head>

<body>
    <p class="large">Large size.</p>
    <p class="italic">Italic style.</p>
    <p class="bold">Bold.</p>
    <p class="weight800">More weight.</p>
</body>
```

The rem & em Units

We can set the default font size in the browser settings. This is usually 16px by default.
To have our text font size change with respect to the browser settings, we should use the **rem** unit.
By default, 1rem = 16px but the user can change the browser settings and 1rem will be larger or smaller.

```
.paragraph {
    font-size: 2.5rem; /* 40px */
}
```

There is also a **em** unit which is relative to the font size of the parent element.
(rem is 'root em' which means the font size is relative to the root element: <html>)

```
.half-size-text {
    font-size: 0.5em; /* 10px */
}
```

```
<section style="font-size: 20px;">
  <p class="half-size-text">Text here.</p>
</section>
```

The ch, vw, vh, and % Units

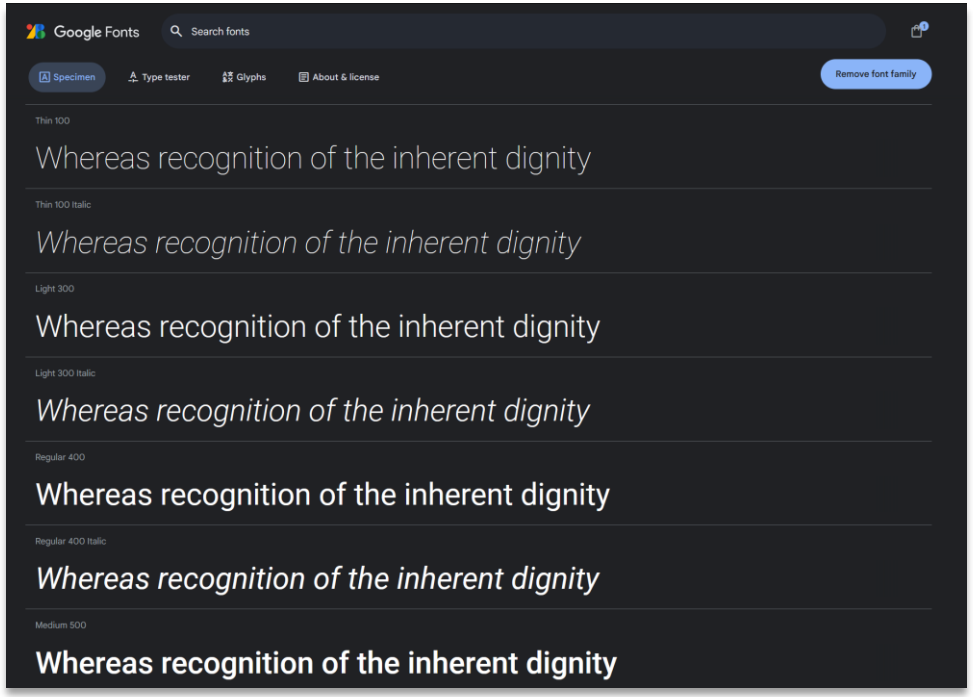
The **ch** unit is equal to the width of a '0' character in our current font.

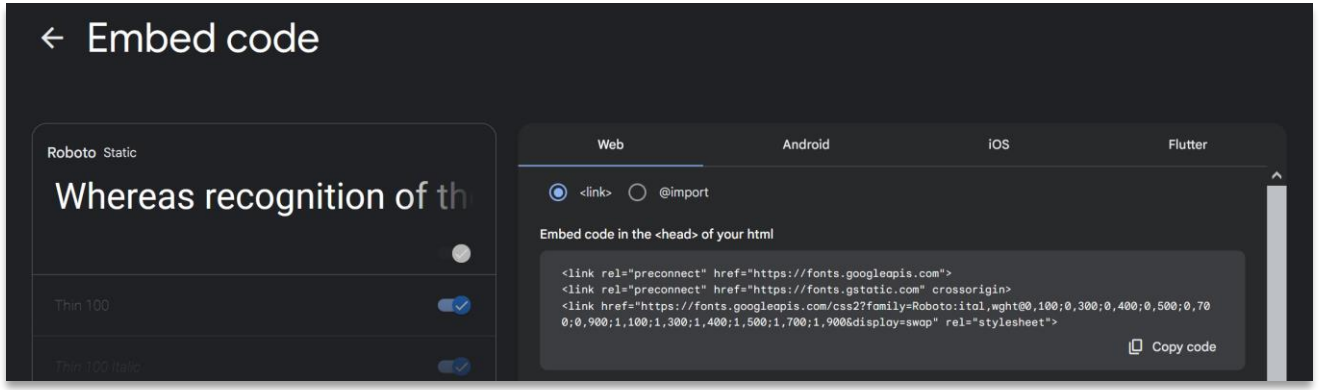
```
.paragraph {
    font-size: 4ch;
}
```

Usually, rem is used for font sizes but other options such as px, em, and ch are also used.
Other units like %, vw and vh are less used.
vw (view width) is equal to 1% of the browser page width (100vw is the entire view width).
vh (view height) is equal to 1% of the browser page height.
These are different from the '1%' unit which is 1% of the parent element width or height.

Google Fonts

Google Fonts is a good selection of fonts that we can easily use in our websites.
For example, Roboto is a very popular font.
We can use the given <link> elements to add them to our website.
After the font is added, we can use the font-family property to set Roboto as a font for our elements.





Custom Fonts

To import custom fonts (using font files) we use the **@font-face** CSS **at-rule**.
At-rules are CSS rules that start with a @ and define various things.

```
@font-face {
    font-family: 'Roboto';
    font-style: normal;
    font-weight: 400;
    src: url('../fonts/roboto-v30-latin-regular.woff2') format('woff2');
}
```

Common font formats include: .ttf (TrueType Font), .otf (OpenType Font), .woff (Web Open Font Format)
Some fonts can be 'variable' which means different font weights are embedded in a single file.
(This is different from font collections which include multiple fonts in a single file)

Pseudo Selectors

More Selectors

We discussed multiple ways to select HTML elements in CSS:

- Simple selectors (tag, #id, .class, *, grouping, and their combination)
- Attribute selectors ([attribute name], [attr="value"], and more equality options)
- Combinator selectors (h1 p, h1>p, and sibling selectors + and ~)

There are two more selector categories we need to talk about:

- Pseudo-class selectors
- Pseudo-element selectors

Pseudo-classes

Pseudo-classes specify a special state of the selected element.
Pseudo-classes start with a colon : character.
For example, the **:hover** pseudo-class specifies the CSS to use when the mouse is hovering over the element.

```
#my-button {
    background-color: hsl(240 80% 60%);
}

#my-button:hover {
    background-color: hsl(240 80% 40%);
}
```

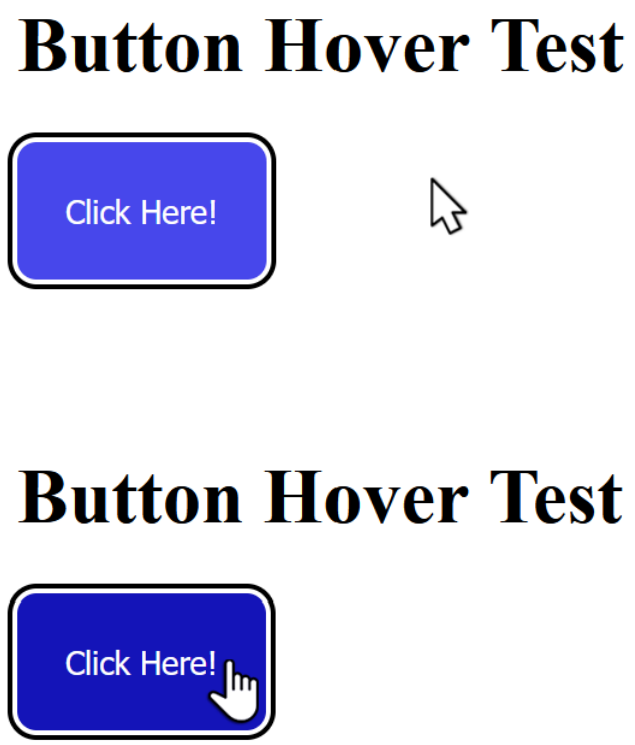
:hover pseudo-class

```html
<head>
    <title>Button Hover</title>
    <style>
        button {
            color: white;
            background-color: hsl(240 80% 60%);
            border: none;
            border-radius: 8px;
            padding: 20px;
            outline: 2px solid black;
            outline-offset: 2px;
            cursor: pointer;
            /* transition: background-color 0.2s; */
        }

        button:hover {
            background-color: hsl(240 80% 40%);

        }
    </style>
</head>

<body>
    <h1>Button Hover Test</h1>
    <button>Click Here!</button>
</body>
```



Input & Link pseudo-classes

Input pseudo-classes:

- :enabled – Selects enabled user inputs.
- :disabled – Selects disabled user inputs. (<input type="text" disabled>)
- :checked – Selects checked radio buttons or checkboxes.
- :valid – Selects user inputs with valid content.
- :invalid – Selects user inputs with invalid content. (normal text in a <input type="email">)

Link pseudo-classes:

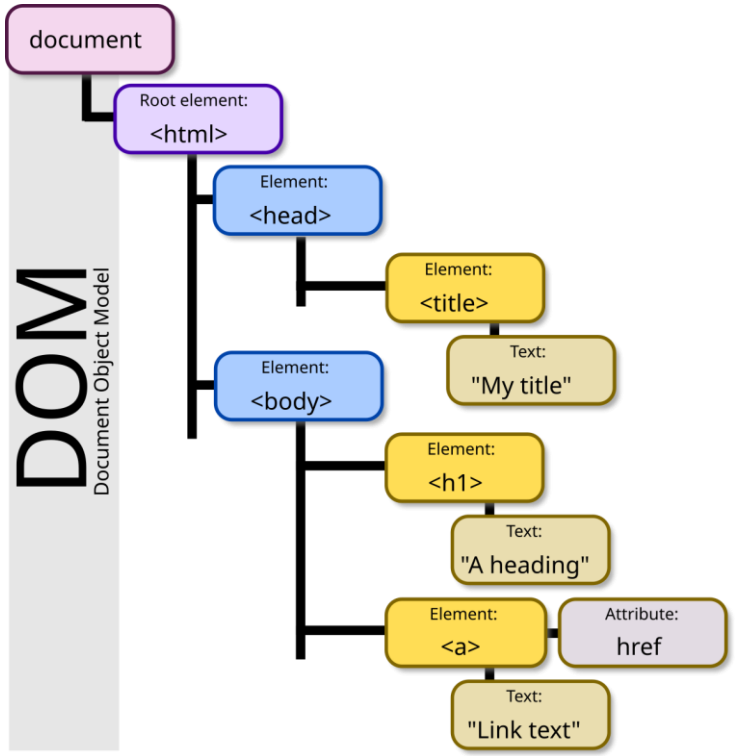
- :link – Selects not yet visited links.
- :visited – Selects visited links.

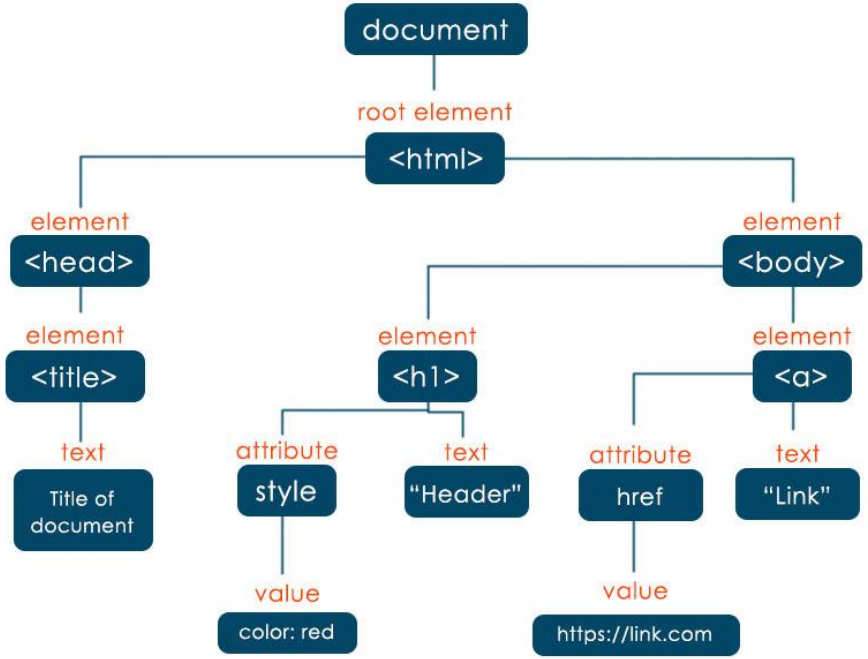
There is also a :not pseudo-class that negates what is written in it:

```
#container :not(p.small) {
    font-size: 1.2rem;
}
```

DOM

DOM (Document Object Model) is the tree structure made out of our HTML code.
Each element is a node in DOM and can have many children.





DOM pseudo-classes

Some of the DOM pseudo-classes include:

- :root – Selects the root element. (<html>)
- :first-child – Selects the first child of an element.
- :last-child – Selects the last child of an element.
- :nth-child(n) – Selects the n-th child of an element.
- :nth-last-child(n) – Selects the n-th child from the end.

DOM pseudo-classes

```html
<head>
    <title>DOM pseudo-classes</title>
    <style>
        :root {
            font-family: 'Consolas', monospace;
        }

        div#parent {
            border: 2px solid black;
            padding: 0 10px;
            width: 200px;
        }
        div#parent p:first-child {
            color: blue;
        }
        div#parent p:last-child {
            color: red;
        }

        ul li:nth-child(even) {
            color: green;
        }
        ol li:nth-child(-n + 2) {
            color: magenta;
        }
    </style>
</head>
```

First paragraph

Second paragraph

Third paragraph

- Item 1
- Item 2
- Item 3
- Item 4
- Item 5
- Item 6

1. Item 1
2. Item 2
3. Item 3
4. Item 4
5. Item 5
6. Item 6

Pseudo-elements

Pseudo-elements specify a specific part of the selected element. (pseudo-classes were specific states)
Pseudo-elements start with two colon :: characters.
For example, the **::first-line** pseudo-element specifies the CSS to use for the first line of an element.
There is also a **::first-letter** pseudo-element.

::first-letter & ::first-line

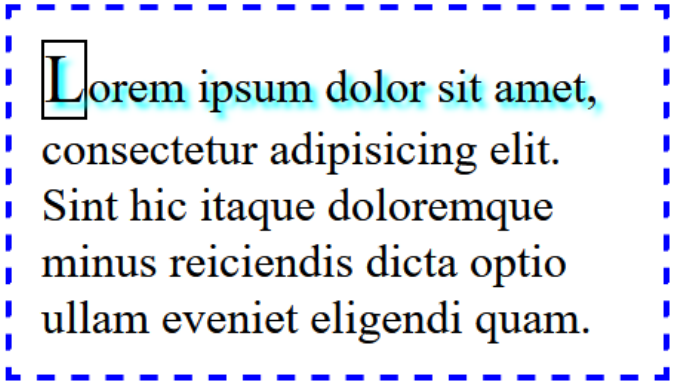
```html
<head>
    <title>Pseudo-elements</title>
    <style>
        #container {
            width: 200px;
            border: 2px dashed blue;
            padding: 10px;
        }

        p {
            margin: 0;
        }

        p::first-letter {
            font-size: 1.4rem;
            border: 1px solid black;
        }

        p::first-line {
            text-shadow: 2px 2px 2px cyan;
        }
    </style>
</head>

<body>
    <div id="container">
        <p>
            Lorem ipsum dolor sit amet, consectetur adipisicing elit.
            Sint hic itaque doloremque minus reiciendis dicta optio ullam eveniet eligendi quam.
        </p>
    </div>
</body>
```



Pseudo-elements

Other pseudo-elements include:

- ::selection – Specifies how selected text should look like.
- ::placeholder – Selects the placeholder text in inputs.
- ::marker – Selects the marker for list elements.
- ::before & ::after – Used to insert content before or after an element.

Pseudo-elements

```html
<head>
    <title>Pseudo-elements</title>
    <style>
        ::selection,
        ::placeholder {
            color: rgb(200, 100, 180);
        }

        ::selection {
            background-color: lightgrey;
        }

        input {
            padding: 6px;
        }

        li::marker {
            content: "X: ";
        }

        h1::before {
            content: "[";
            margin-right: 2px;
        }

        h1::after {
            content: "]";
            margin-left: 2px;
        }
    </style>
</head>

<body>
    <h1>More pseudo-elements</h1>
    <label for="textinput">Sample:</label>
    <input id="textinput" type="text" placeholder="This is a placeholder">
    <ul>
        <li>Item 1</li>
        <li>Item 2</li>
    </ul>
</body>
```

Specificity

Specificity

Specificity is an algorithm used to determine the most relevant CSS declaration for an element.
When multiple selectors select the same element and apply styles to it, if the styles overlap, one of them has to override the others.
This is chosen based on the selector specificity.

Selector specificity is usually described as a 3 column value: a-b-c
The 3 weight categories are ID-CLASS-TYPE:

- Each #id in a selector adds 1 to the first column.
- Each .class, [attr] and :pseudo-class add 1 to the second column.
- Each tag and ::pseudo-element add 1 to the third column.

When two CSS rules have the same specificity, the one that comes later wins.
Inline styles win over all selectors.

Specificity Examples

```
[type="text"] { /* 0-1-0 */
    color: red;
}
.user-input { /* 0-1-0 but wins over the above */
    color: blue;
}
input:enabled { /* 0-1-1 */
    color: green;
}
:root #form-container input.user-input { /* 1-2-1 */
    color: black;
}
```

Important

CSS declarations can be marked as important using the **!important** keyword.
An important declaration does not change specificity, but overrides all other declarations.
It is often a bad practice to use !important.

```
p#my-elem.my-class { /* 1-1-1 */
    color: red;
}

p { /* 0-0-1 but the color property wins here */
    color: blue !important;
}
```

Reset & Normalize

Reset

Browsers may have different default styling for elements.
An example is the default margin for headings and paragraphs.
A **reset.css** removes all default stylings and makes everything 0.

The right code box includes 4 of the most used reset rules:

- All element margins are zeroed out.
- The box-sizing of all elements is border-box.
- Input elements not using the parent font by default is fixed.
- Images and media are block elements and have max-width.

While it is best to have your own custom reset for your project,
here are some reference reset CSS with explanation:
Eric Meyer's reset.css (quite old): Link
Andy Bell's reset: Link
Josh Comeau's reset: Link

```css
* {
  margin: 0;
}
*, *::before, *::after {
  box-sizing: border-box;
}
input, button, textarea, select {
  font: inherit;
}
img, picture, video {
  display: block;
  max-width: 100%;
}
```

Normalize

Normalize.css tries to fix inconsistencies between browsers and not just making everything 0.
It provides consistent defaults and does not remove stylings.

An example comparison can be something like the following:

```css
/* normalize css */
h1 {
    font-size: 2em;
    margin: 0.67em 0;
}
```

```css
/* reset css */
h1 {
    font-size: 1em;
    margin: 0;
}
```

Some reference normalize CSS:
csstools' normalize (consistency and bug fixes): Link
csstools' sanitize (normalize with opinionated styles): Link
Reboot (comes with the Bootstrap framework): Link

Images

Images

The following properties were discussed in the class:

- Object-fit
- Object-position
- Opacity
- Filter
- Transform

```html
<head>
    <title>Images</title>
    <style>
        #container {
            width: 400px;
            height: 400px;
            border: 4px solid red;
            background-color: lightblue;
            padding: 10px;
        }

        img {
            width: 100%;
            height: 100%;
            object-fit: cover;
            object-position: center;
            border-radius: 8px;
            filter: brightness(200%); /* blur, contrast, grayscale, ... */
            transform: scaleY(-1);
        }
    </style>
</head>

<body>
    <div id="container">
        <img src="pic.jpg" alt="sample">
    </div>
</body>
```

Thank you for your attention.