04 – Cascading Style Sheets

Web Technology Project (International Computer Science)
Summer semester 2025
Prof. Dr. Felix Schwägerl

Key facts

- Stylesheet language used to describe the *presentation* of a *document*
  - e.g., HTML, SVG, or XML in general
- Syntax is not XML-like, but based on *selectors* and *property-value* pairs.
- Basic idea: Every HTML element carries additional visual properties that can be controlled by stylesheets.
- CSS is declarative and relies on sophisticated precedence rules to decide about the actual value for a property of an individual document element.
  - Important mechanisms: *inheritance* and *colliding styles*
- CSS controls the *layout* of a webpage.
  - The layout may vary for different devices
  - CSS is the basis of *responsive web design*.
  - Long-term goal of the standard: Require no JavaScript for the purpose of being responsive.
- CSS can be used to build reusable libraries (e.g., Bootstrap, PicoCSS)
- Recommended reference: https://developer.mozilla.org/docs/Web/CSS

History of CSS

- **1996 - CSS1**: Introduced basic styling features, including fonts, colors, margins, padding, and text alignment.
- **1998 - CSS2**: Added advanced capabilities like absolute and relative positioning, z-index, and media queries for different devices.
- **2005 - CSS3** Development: CSS3 split into modules, allowing for incremental updates and new features like border-radius and text-shadow.
- **2007 - CSS Transitions**: Enabled smooth changes between states of an element, enhancing user experience with animations.
- **2011 - CSS3 Animations**: Introduced keyframe-based animations for more complex and interactive visual effects.
- **2012 - Media Queries**: Improved responsive design support, allowing websites to adapt layouts based on screen size and resolution.
- **2017 - Flexbox**: Introduced flexible layout design, allowing for dynamic alignment and distribution of elements within a container.
- **2018 - CSS Grid**: Enabled two-dimensional layouts, offering precise control over both rows and columns in a grid structure.

[ChatGPT1]

Hello, CSS!

```
body {
    font-family: sans-serif;
    background-color: lavender;
}

h1 {
    color: royalblue;
}

p {
    font-size: 110%;
}

strong {
    text-decoration: underline;
    font-style: italic;
}
```

cascading style sheet

Hello, World!

This is my first HTML page with *CSS*. 👍

I love stylesheets.

They help convey **strong** messages.

Hello, World!

This is my first HTML page with *CSS*. 👍

I love stylesheets.

They help convey ***strong*** messages.

Including a CSS from a HTML document

External (recommended)

May be included from multiple documents

```html
<html lang="en">
    <head>
        <meta … />
        <link rel="stylesheet" href="hello-style.css">
    </head>
    <body> … </body>
</html>
```

```css
body {
    …
}
h1 {
    …
} …
```

Physically connected to current document

Internal

```html
<html lang="en">
    <head> … </head>
    <style>
        body {
            …
        }
        h1 {
            …
        } …
    </style>
    <body>
        <h1>Hello, World!</h1>
        …
    </body>
</html>
```

Inline

Linked to specific elements; duplication required

```html
<html lang="en">
    <head>
        <meta charset="utf-8" />
        <meta name="viewport" content="width=device-width"/>
        <title>Hello, World!</title>
    </head>
    <body style="font-family: sans-serif; background-color: lav
        <h1 style="color: royalblue">Hello, World!</h1>
        <p style="font-size: 110%">This is my first HTML page w
        <h2>I love stylesheets.</h2>
        <p style="font-size: 110%">They help convey <strong>str
    </body>
</html>
```

04_01_basic_css/hello-style.css
hello.html
hello-internal.html
hello-inline.html

CSS basic syntax

Selector
Determines to which element(s) the styling applies.

```
body {
    font-family: sans-serif;
    background-color: lavender;
}
h1 {
    color: royalblue;
}
p {
    font-size: 110%;
}
strong {
    text-decoration: underline;
    font-style: italic;
}
```

Property
A visual aspect of the selected element to be styled

Value
A value that defines or overrides the existing value for a property of a selected element.

IDs and classes address specific HTML elements.

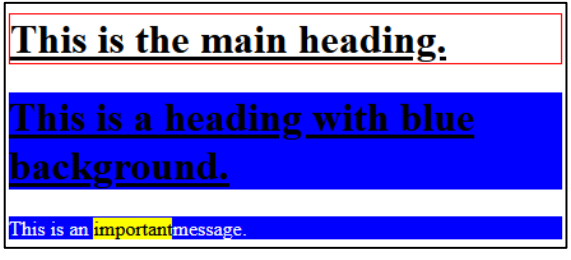
- ID: unique element identifier (must exist at most once per HTML document!)
- Class: an element may have multiple classes, a class may be used by multiple elements.

```
h1  {
    text-decoration: underline;
}
h1#main-heading {
    border: 1px solid red;
}
.blue-background {
    background-color: blue;
}
p.white-font {
    color: white;
}
.black-font {
    color: black;
}
span.yellow-background {
    background-color: yellow;
}
```

```
<body>
    <h1 id="main-heading">This is the …</h1>
    <h1 class="blue-background">This is a
        heading with blue background.</h1>
    <p class="white-font blue-background">
        This is an <span class="black-font
        yellow-background">important</span>
        message.</p>
</body>
```

IDs are referenced with #

Classes are referenced with .



Pseudo-classes reflect context-dependent pre-defined states.

```
h1:hover {
    color: darkorange;
}
li:first-child {
    color: darkgreen;
}
li:last-child {
    color: darkred;
}
a:visited {
    color: gray;
}
a:link {
    color: darkorange;
}
button:enabled {
    background-color: greenyellow;
}
button:disabled {
    background-color: coral;
}
button:active {
    background-color: deepskyblue;
}
```

Applied while mouse hovers over element

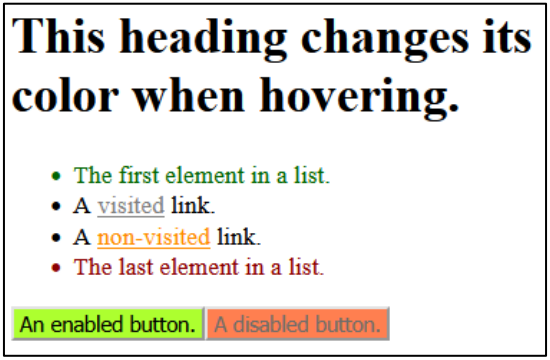
First/last sub-element

Links (not) visited yet

Enablement status

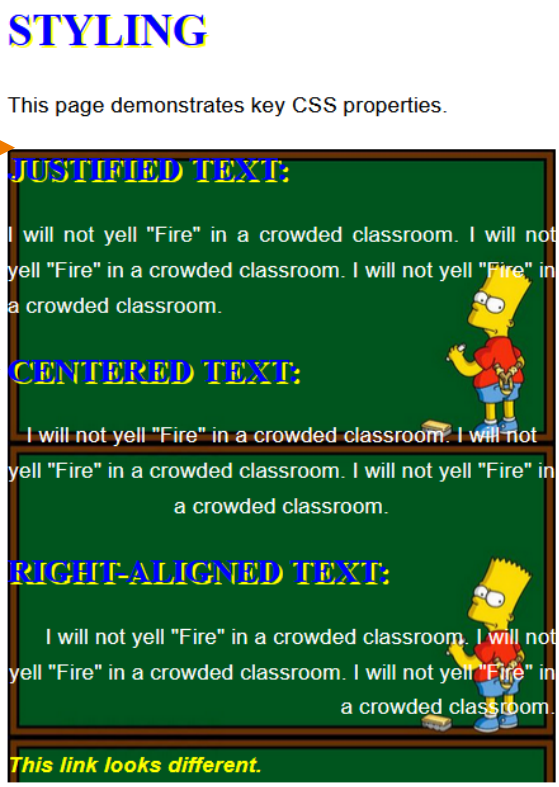
Applied while mouse button is pressed

```
<body>
    <h1>This heading …h1>
    <ul>
        <li>The first …</li>
        <li>A <a
href="http://visited.com">visited</a> link.</li>
        <li>A <a href="http://non-visited.com">non-
visited</a> link.</li>
        <li>The …</li>
    </ul>
    <button>An enabled button.</button>
    <button disabled>A disabled …</button>
</body>
```



Important properties for styling text

```
body {
    font-family: Arial, sans-serif;
}

p.outside {
    color: black;
}
section.bart {
    background-image: url("img/bart.jpg");
    background-size: contain;
}
h1 {
    font-family:  "Times New Roman", serif;
    color: blue;
    text-transform: uppercase;
    text-shadow: yellow 2px 2px;
}
p {
    font-size: 16px;
    line-height: 1.6;
    text-align: justify;
    color: white;
}
.center {
    text-align: center;
}
.right {
    text-align: right;
}
a {
    text-decoration: none;
    font-weight: bold;
    font-style: italic;
}
```



Pre-defined colors

- For most frequently used colors, a standard palette with keywords is provided

```
.red {
    color: red;
}

.blue {
    color: #1111dd;
}

.green {
    color: rgb(22, 255, 11);
}

.yellow {
    color: rgba(255, 255, 0, 127);
}
```

Hex colors

- Format: `#rrggbb`
- Every color has a double hex digit (0-f)
- 24 bits color depth

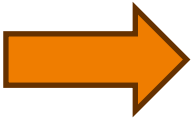
RGB colors

- Format: `#rgb(R, G, B)`
- Each color component ranges between 0 and 255
- Rgba mode supports transparency through alpha channel

Selector groups

Comma-separated selectors apply styles to multiple elements.

```
h1 {
    color: red;
}
h2 {
    color: red;
}
h3 {
    color: red;
}
h4 {
    color: red;
}
```



```
h1, h2, h3, h4 {
    color: red;
}
```

I want to be red

I want to be red, too.

Me too.

Can I be red, too?

- Goal: reduce redundancy for defining similar styles

Hierarchical selectors

Descendant selector

- Syntax: `sel1 sel2`
- Semantics: select every `sel2` that is direct or indirect sub-element of `sel1`
- More „greedy“

Child selector

- Syntax: `sel1 > sel2`
- Semantics: select every `sel2` that is direct sub-element of `sel1`
- More specific

```
article.r ul li ul li span {
    font-style: italic;
}

article.r > ul > li > ul > li > span {
    font-weight: bold;
}

article.r ul li ul li ul li span {
    text-decoration: underline;
}

article.r > ul > li > ul > li > ul > li > span {
    text-transform: uppercase;
}
```

All items having two lists and one list item as ancestors

Specifically selects level-2 items

All items having three lists and two list items as ancestors

Specifically selects level-3 items

- Germany
  - *Bavaria*
    - *UPPER PALATINATE*
      - *Regensburg*

The comma operator has higher precedence!

This selects any h2 and h3 element (not only those descending from articles):

```
article h1, h2, h3 {
    color: green;
}
```

I want to be green

I want to be green, too.

Me too.

I'd stay red if you don't mind

To select all headings of an article, we need:

```
article h1, article h2, article h3 {
    color: green;
}
```

Or more specifically:

```
article > h1, article > h2, article > h3 {
    color: green;
}
```

Combining hierarchical selectors and pseudo classes

Goal: Format first sub-item of last item of a nested list blue.

```
<ul>
    <li class="special">Nothing special.</li>
    <li class="special">Parent of a normal item:
        <ul>
            <li class="special">Nothing special.</li>
        </ul>
    </li>
    <li class="special">Parent of a special item:
        <ul>
            <li class="special">I'm blue! (First special
sub-item of the last special item in an unordered list)</li>
            <li class="special">Nothing special.</li>
        </ul>
    </li>
</ul>
```

```
ul li.special:last-child ul li.special:first-child {
    color: blue;
}
```

- Nothing special.
- Parent of a normal item:
  - Nothing special.
- Parent of a special item:
  - I'm blue! (First special sub-item of the last special item in an unordered list)
  - Nothing special.

Universal selector

The universal selector * selects *all* elements.

```
<article class="blue-contents">
    <h1>I'm blue</h1>
    <ul>
        <li>da</li>
        <li>ba</li>
        <li>dee</li>
    </ul>
    <p>Parts of this are
      <span class="no-blue">not</span> blue.</p>
</article>
```

I'm blue

- da
- ba
- dee

Parts of this are not blue.

```
.blue-contents * {
    color: blue;
}
```

* Here matches h1, li, p

```
.blue-contents span.no-blue {
    color: black;
}
```

Specific selector overrides universal selector

Next-neighbor selector

The + selector selects a successor element.

Parts of this are not blue.

I'm the next sibling of an existing paragraph and therefore orange.

I'm another sibling.

Number four.

```
.blue-contents p + p {
    color: orange;
}
```

Selects all paragraphs that immediately follow a paragraph.

```
.blue-contents p + p + p {
    color: blue;
}
```

Reverts to blue for third paragraph and after.

Colliding styles

- When styles collide, the *most specific* style wins.
- If elements have same specificity, the later rule wins.
- These rules are applied individually for every property.

```
h1 {
    color: blue;
    text-decoration: underline;
}
.colorful {
    color: green;
}
h1 .colorful {
    color: red;
}
article > h1 {
    color: orange;
}
article > h1 {
    color: purple;
}
article > h1 .colorful {
    color: black;
}
```

Least specific selector, but property is not overridden individually.

Increasing level of specificity

Later rule wins


```
body>
    <h1>What color am I?</h1>
    <article>
        <h1>What color am I?</h1>
        <h1 class="colorful">What color am I?</h1>
    </article>
/body>
```

Some (but not all) CSS properties are inherited.

- *Inherited* properties are set to the *computed* value of the *parent* element.
  - Example: `color`, `font-weight`, `font-style`, …
- Non-inherited properties are set to a *default* value (defined by the user-agent stylesheet)
  - Example: `size`, `border`, `margin`, …

```html
<article class="bold">
    <h1>Header of a bold article</h1>
    <p>Bold text with <em>emphasis.</em></p>
    <p>This text contains <span
       class="defaulted">a non-bold span.</span></p>
    <p class="inherited-border">This text contains
       an inherited border.</p>
</article>
```

```css
article.bold {
    font-weight: bold;
    border: 1px solid blue;
}

.defaulted {
    font-weight: initial;
}

.inherited-border {
    all: inherit;
}
```

Ignore inheritance

Apply to all properties (including border)

Enforce inheritance

Header of a bold article

Bold text with *emphasis.*

This text contains a non-bold span.

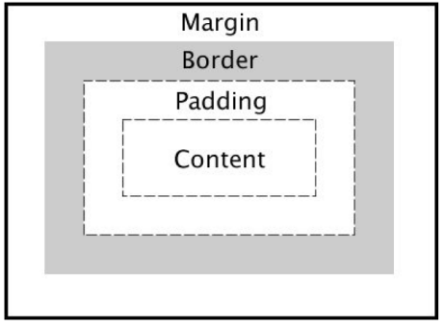
This text contains an inherited border.

Variables allow reusability of, e.g., color palettes.

```
:root {
    --bg-primary: beige;
    --bg-secondary: #ffddff;
    --fg-primary: blue;
    --fg-secondary: red;
    --input-border: 1px solid black;
}

article {
    background-color: var(--bg-primary);
    color: var(--fg-primary);
}

article form {
    background-color: var(--bg-secondary);
}

article form legend {
    color: var(--fg-secondary);
}

article form input, article form button {
    background-color: var(--bg-primary);
    border: var(--input-border);
}
```

- Declare once, use everywhere
- Variable declarations are typically located in the global (`:root`) namespace.
- Usage of variables requires the `var` operator.

The box model



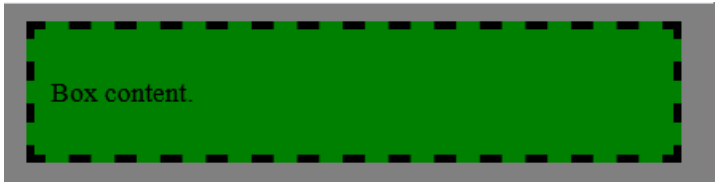
In CSS, elements are rendered as boxes with following properties:

- *Padding*: Spacing between border and contents
- *Margin*: Spacing between border and container
- *Border*: May consume additional space by itself.

```
<body>
    <div id="box">
        <p>Box content.</p>
    </div>
</body>

body {
    background-color: gray;
}

div#box {
    margin: 10px;
    border: 5px dashed black;
    padding: 10px 20px 10px 10px;
    background-color: green;
}
```

Same value for all directions

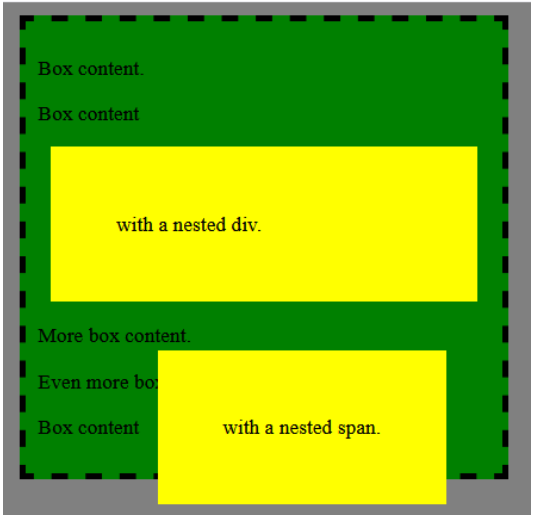
Individual value for top right bottom left



Block vs. inline elements

```
<body>
    <div id="box">
        <p>Box content.</p>
        <p>Box content <div class="nested">
with a nested div.</div></p>
        <p>Box content.</p>
        <p>Box content.</p>
        <p>Box content <span class="nested">
with a nested span.</span></p>
    </div>
</body>

.nested {
    margin-left: 10px;
    border: 0;
    padding: 50px;
    background-color: yellow;
}
```

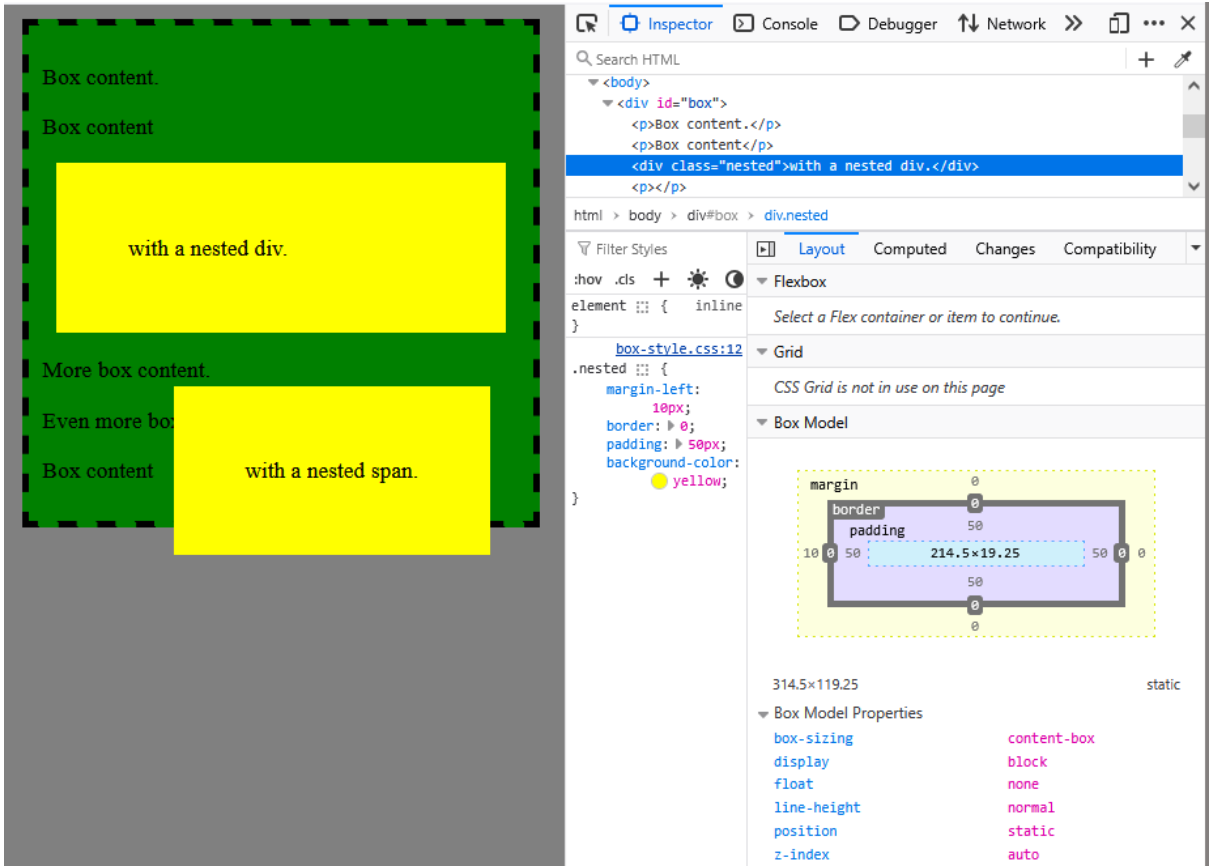
Individual value for single direction



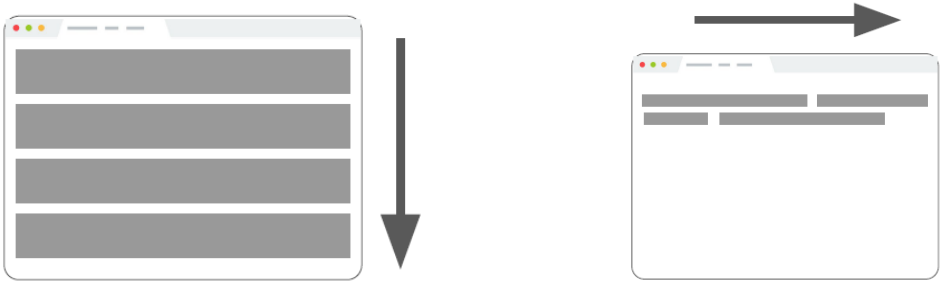
Why are div and span aligned differently?

- `div` is a *block-level element*: They start a new line and add keep a minimum distance defined by the margin.
- `span` is an *inline* element: They continue in the existing line if possible; minimum distance is kept only for left/right, but not for bottom/top directions.
- The rendering type can be adjusted manually, e.g., {`display: block;`} for a `span`

Inspecting CSS properties in the Browser



So far, we know two types of layouts, *block* and *inline*:



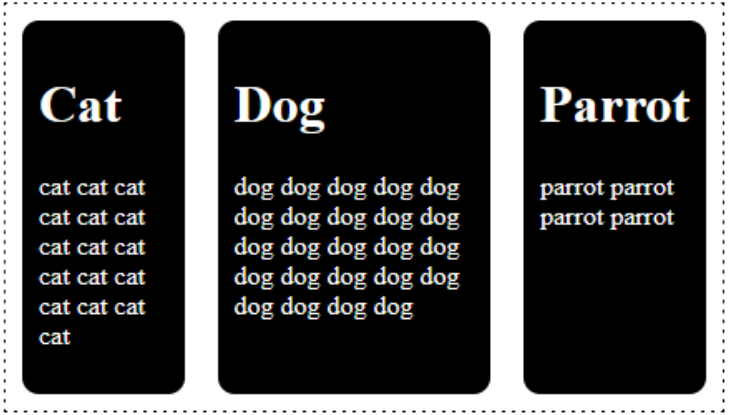
Modern CSS also supports the more advanced *flex* layout:

- We will consider simple *column layouts* as example.

Flex layout: Columns (1)

```html
<div class="flex-container">
    <div class="flex-content">
        <h1>Cat</h1><p>…</p>
    </div>
    <div class="flex-content">
        <h1>Dog</h1><p>…</p>
    </div>
    <div class="flex-content">
        <h1>Parrot</h1><p>…</p>
    </div>
</div>
```

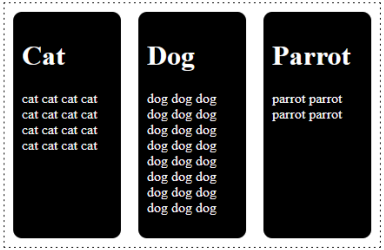
```css
.flex-container {
    border: 1px dashed black;
    display: flex;
}

.flex-content {
    margin: 10px;
    padding: 10px;
    color: white;
    background-color: black;
    border-radius: 10px;
}
```



Flex layout: Columns (2)

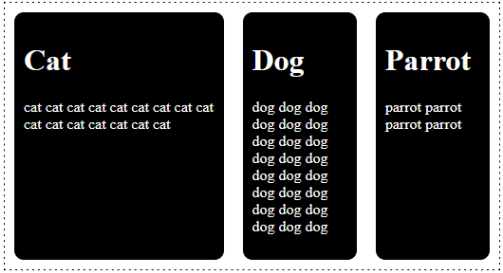
Equal column width:

```
.flex-content {
    flex: 1;
}
```

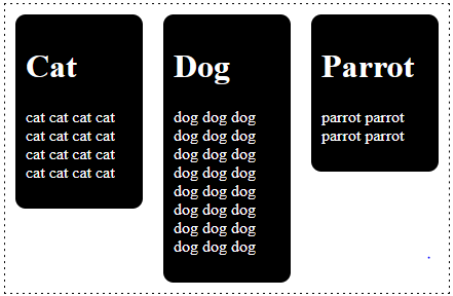


Individual column width:

```
.flex-container div:first-child {
    flex-grow: 2;
}
```



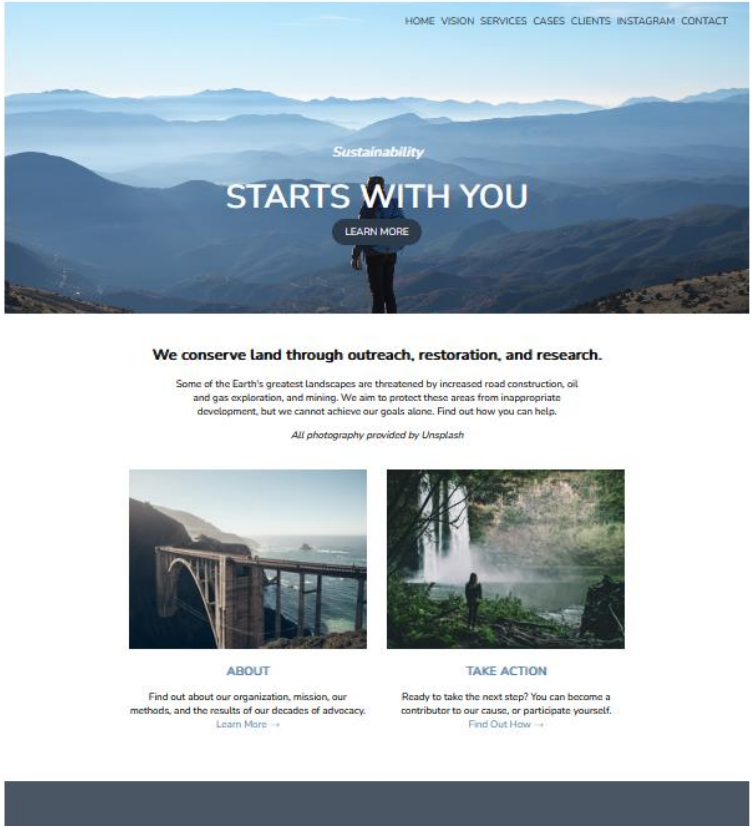
double width

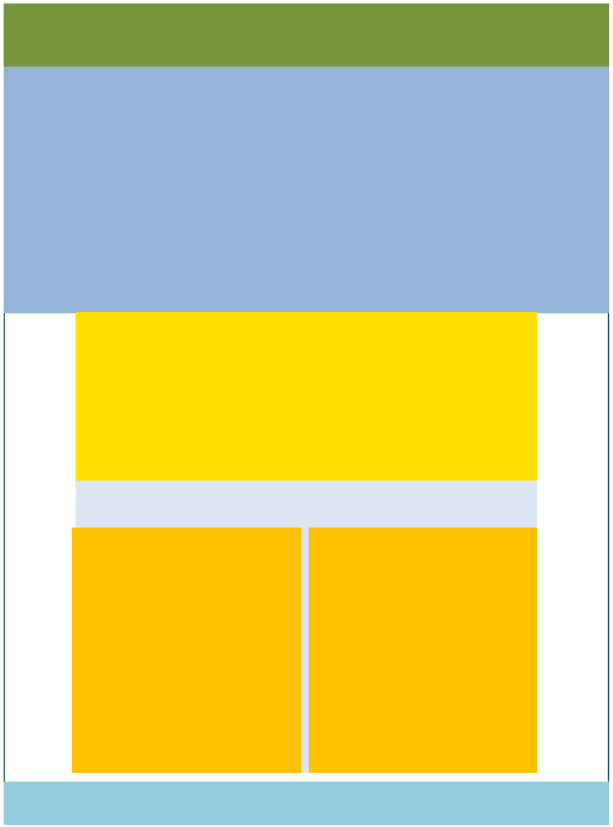


Flexible column height:

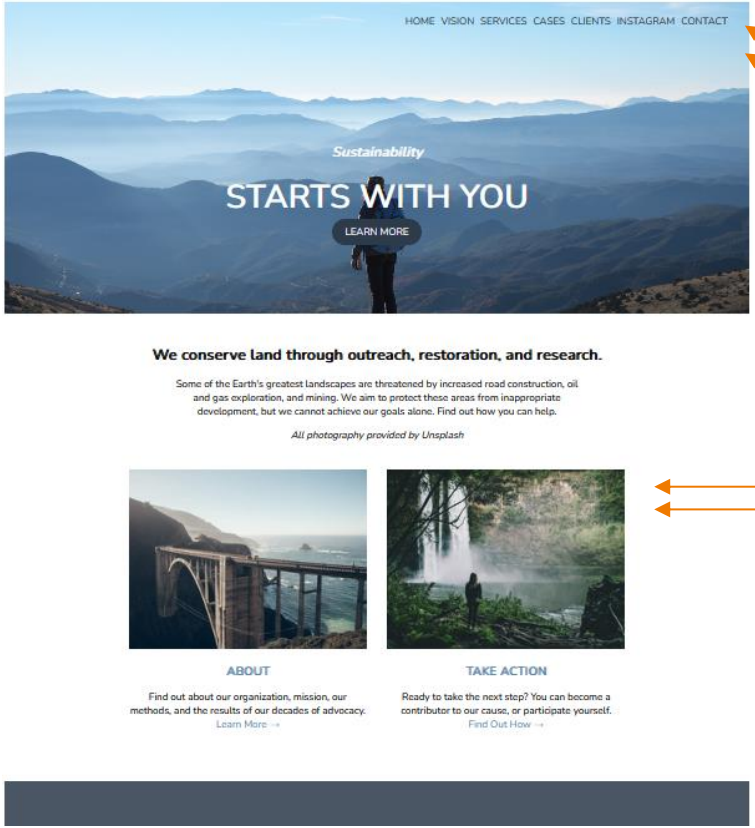
```
.flex-container {
    align-items: flex-start;
}
```

CSS layouting requires thinking in boxes

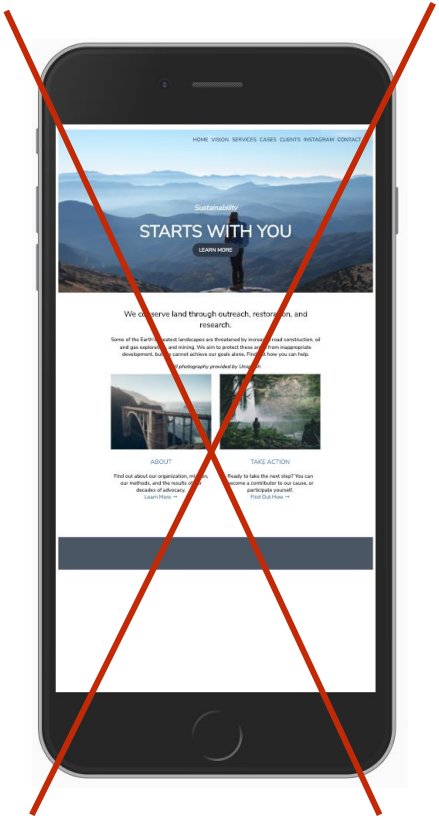




Squarespace example: Flexbox

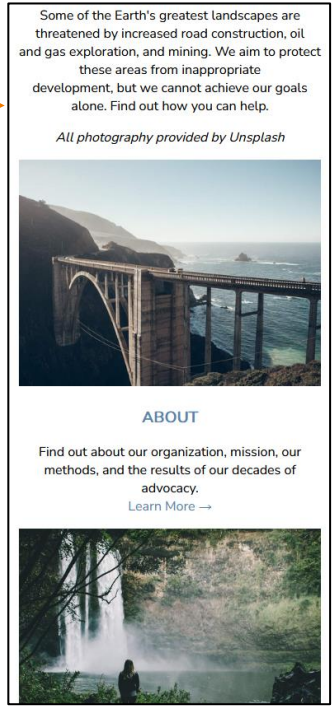



```
nav ul {
  list-style: none;
  display: flex;
  justify-content: flex-end;
  margin-right: 30px;
}

nav ul li {
  display: inline-block;
  color: #2E3B4B;
  margin:5px;
  margin-top: 20px;
}

#columns {
  display:flex;
  justify-content: space-between;
}

.section {
  height:400px;
  width:48%;
  text-align: center;
}
```

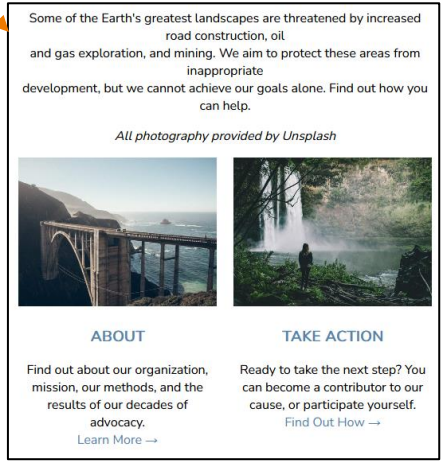


Responsive = adapting to the device properties (screen size, viewport)

Below 600 px device width:



From 600 px device width:



The *viewport* is the area of the window showing web content.

- The viewport is often not the same size as the rendered page.
- Some mobile devices render pages in a *virtual viewport* (wider than the screen)
  - This is the default strategy for webpages not optimized for *responsiveness*.
  - The HTML viewport tag allows to *override* the default strategy.
- HTML viewport properties:
  - `width` and `height`: Fixed sizes in pixels or `device-width`
  - `initial-scale`, `initial-scale`, `initial-scale`: relative values
- Example:

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

Disable virtual viewport (use device width as width)

Do not apply scaling initially, but allow to zoom in and out

Media queries allow to make CSS rules conditional to certain device properties (e.g., screen width)

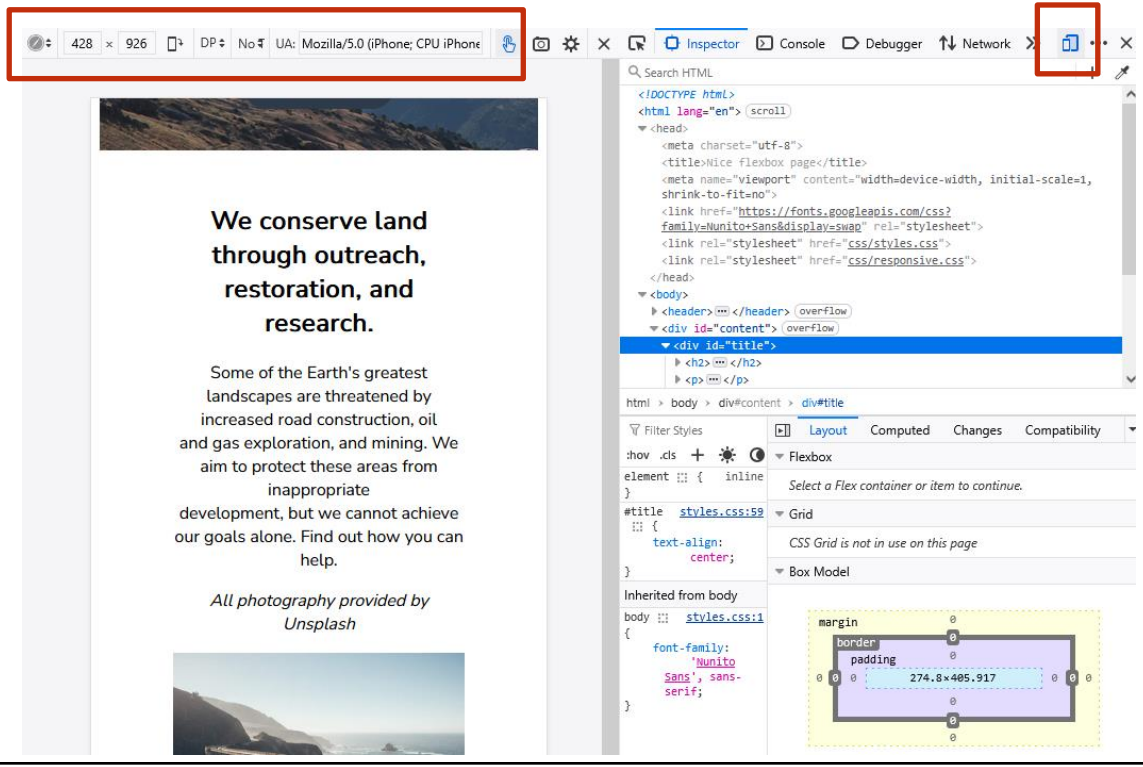
```
@media only screen and (max-width: 600px) {
    #columns {
        flex-direction: column;
        align-items: center;
    }

    .section {
        width: 100%;
        height: auto;
    }
}
```

When our device has 600px or lower …

… change flex direction from row to column and align items in center

… make sure that contents are displayed on full device width

Most browsers can simulate device widths and viewports of different mobile devices.



CSS frameworks offer re-usable and customizable styles.

- Many general problems (accessibility, responsiveness, etc) are solved by default.
- Provides pre-designed icons, fonts, colors, etc.
- Dark mode
- Custom CSS classes for simplified layouting
- Often, complicated *wrapper classes* need to be used: (e.g., Bootstrap CSS)

.col-xs-12 .col-md-8	.col-xs-6 .col-md-4
.col-xs-6 .col-md-4	.col-xs-6 .col-md-4	.col-xs-6 .col-md-4
.col-xs-6	.col-xs-6

```
<!-- Stack the columns on mobile by making one full-width and the other half-width -->
<div class="row">
  <div class="col-xs-12 col-md-8">.col-xs-12 .col-md-8</div>
  <div class="col-xs-6 col-md-4">.col-xs-6 .col-md-4</div>
</div>

<!-- Columns start at 50% wide on mobile and bump up to 33.3% wide on desktop -->
<div class="row">
  <div class="col-xs-6 col-md-4">.col-xs-6 .col-md-4</div>
  <div class="col-xs-6 col-md-4">.col-xs-6 .col-md-4</div>
  <div class="col-xs-6 col-md-4">.col-xs-6 .col-md-4</div>
</div>

<!-- Columns are always 50% wide, on mobile and desktop -->
<div class="row">
  <div class="col-xs-6">.col-xs-6</div>
  <div class="col-xs-6">.col-xs-6</div>
</div>
```

Home Help Dropdown

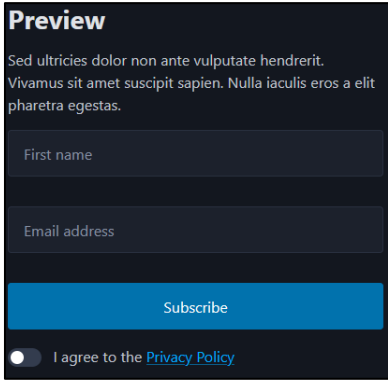
Default Primary Success Info Warning Danger

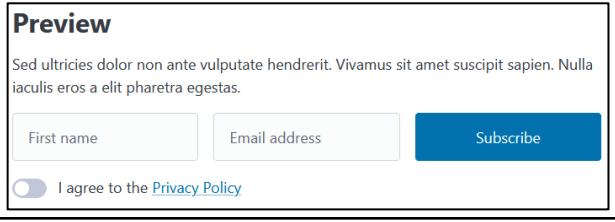
[Bootstrap 2025]

PicoCSS is a minimal CSS framework suited for semantic HTML

- Starting point: https://picocss.com/docs/version-picker
- „Classless“ HTML is rendered with reasonable default rules.
- For more advanced problems like responsive styles, simple CSS classes are pre-defined.
  - E.g. containers with class `grid` offer a three-column responsive layout.

```
<link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/@picocss/pico@2.1.1/css/pico.css">

<section id="preview">
    <h2>Preview</h2>
    <p>…</p>
    <form>
        <div class="grid">
            <input type="text" name="firstname" required>
            <input type="email" name="email" required>
            <button type="submit">Subscribe</button>
        </div>
        <fieldset>…</fieldset>
    </form>
</section>
```





[PicoCSS 2025]

References

- [Frain 2020] Ben Frain: Responsive web design with HTML5 and CSS, 3rd edition, Packt, 2020
- [Mozilla 2025] Mozilla Developer Network (MDN): HTTP web docs, https://developer.mozilla.org/en-US/docs/Web/HTTP
- [PicoCSS 2025] PicoCSS documentation, https://picocss.com/docs
- [Bootstrap 2025] Bootstrap documentation, https://getbootstrap.com/docs/3.4/css/
- [ChatGPT1] ChatGPT (https://chatgpt.com/) with prompt: “Generate a 8 bullet point summary about the history of CSS. It should fit one PowerPoint slide nicely. Omit browser-specific details and persons.”