HTML and CSS: Part II

Pau Fernández   Rafa Genés   Jose L. Muñoz

Universitat Politècnica de Catalunya (UPC)

Table of Contents

Embedded content

The `image` tag

The `img` tag embeds an image into the page.

- `src` : URL of the image to embed into the page.
- `alt` : text description of the image (for accessibility).

```html
<img src="https://imgs.xkcd.com/comics/tags.png"
     alt="XKCD Comic strip about web developers" />
```

The **picture** tag

The `picture` tag contains `source` elements and an `img` element to offer alternative versions of an image.

`source` attributes:

- `srcset` : URL of the source.
- `media` : conditions to show this version (media query).

```
<picture>
  <source srcset="logo-big.png" media="(min-width: 1200px)">
  <source srcset="logo-medium.png" media="(min-width: 600px)">
  <img src="logo.png" alt="Company Logo">
<picture>
```

How the `picture` chooses the `source`

The browser will consider each `source` and choose the best match among them.

If…

- no matches are found, or
- the browser doesn't support the `picture`

the URL of the `img` is selected.

The image is presented in the space occupied by the `img`.

The **video** tag

The `video` tag embeds a video player in the browser. It uses `source` elements to provide different versions.

Attributes

- `height`, `width`: Size of the element.
- `autoplay`: Play as soon as the page is loaded.
- `controls`: Show UI to control the video.
- `muted`: Start with muted audio.
- `poster`: Image to show while the video downloads.

```
<video controls muted>
  <source src="/videos/cat-jump.webm" type="video/webm">
  <source src="/videos/cat-jump.mp4" type="video/mp4">
  Your browser does not support embedded videos
</video>
```

The **audio** tag

The `audio` tag embeds an audio track in the browser. It uses `source` elements to provide different versions.

Attributes

- `autoplay` : Play as soon as the page is loaded.
- `controls` : Show UI to control the video
- `muted` : Start with muted audio.
- `loop` : Play in a loop.
- `preload` : Load in advance (hint to the browser).

```html
<audio controls>
  <source src="audio/chiquito.mp3">
  Your browser does not support the <code>audio</code> element
</audio>
```

Without `controls`, nothing is shown on the screen

The `iframe` tag

The `iframe` tag represents an *nested browsing context*, embedding a different page.

Attributes

- `height` , `width` : Size of the element.
- `src` : URL of the page to embed.
-

```html
<iframe id="bcn-map" title="Map of Barcelona"
  width="600" height="400"
  src="https://www.openstreetmap.org/export/embed.html?bbox=...">
</iframe>
```

Scalable Vector Graphics

XML-based format to represent vector 2D images.

Scalable: images of any size (good for high-resolution displays).

Compressed: the vector representation is much smaller than the pixels.

```
<svg xmlns="http://www.w3.org/2000/svg"
     version="1.1" baseProfile="full" width="300" height="200">
  <rect width="100%" height="100%" fill="red" />
  <circle cx="150" cy="100" r="80" fill="green" />
  <text x="150" y="125" font-size="60"
        text-anchor="middle" fill="white">
    SVG
  </text>
</svg>
```

Using SVG in the **img** tag

To show an SVG, it is enough to put it in an `img`:

```html
<img src="/img/logo.svg" alt="HTML5 Logo" />
```

Embedding SVG

But you can directly **embed** SVG in HTML.

```
<html>
  <body>
    <p>Below is an SVG image:</p>
    <svg xmlns="http://www.w3.org/2000/svg"
      version="1.1" baseProfile="full" width="300" height="200">
      <rect width="100%" height="100%" fill="red" />
      <circle cx="150" cy="100" r="80" fill="green" />
    </svg>
  </body>
</html>
```

Embedding an SVG lets you *style* it with CSS.

Table of Contents

Lists

The **ol** tag

The `ol` stands for ”ordered list”, and shows a **numbered** list.

The `li` stands for ”list item”.

```
<h1>The Three Musketeers</h1>
<ol>
  <li>Athos</li>
  <li>Porthos</li>
  <li>Aramis</li>
</ol>
```

`ol` tag attributes

- `reversed` : To reverse the order of the list.
- `start` : Starting value of the list.
- `type` : Kind of list marker
  - `1` - decimal
  - `a` - lower alpha
  - `A` - upper alpha
  - `i` - lower roman
  - `I` - upper roman

`li` attributes

`value` : Set a specific value (an integer) to the item.
(Only within an `ol` .)

```
<ol>
  <li>First</li>
  <li value="3">Second?</li>
</ol>
```

The **ul** tag

The `ul` tag stands for ”unordered list”, which shows a **bulleted** list.

(The `li` tag is the ”list item”.)

```
<h1>Coldplay</h1>
<ul>
  <li>Chris Martin</li>
  <li>Will Champion</li>
  <li>Jonny Buckland</li>
  <li>Guy Berryman</li>
</ul>
```

List styles

`list-style` : *<type> <position> <image>*

`list-style-type`	none disc square decimal decimal-leading-zero lower-roman upper-roman lower-greek upper-greek lower-latin upper-latin armenian georgian lower-alpha upper-alpha
`list-style-position`	inside outside
`list-style-image`	url(<image-url>)

Table of Contents

Tables

The `table` tag

A `table` shows data in two dimensions.

It contains:

- (`caption`)
- (`colgroup` s)
- (`thead`)
- `tr` : Several **t**able **r**ows (better inside a `tbody`)

  Each contains:

  - `th` : Several **t**able **h**eader elements (header cells).
  - `td` : Several **t**able **d**ata elements (cells).
- (`tfoot`)

A **table** example

```
<table>
  <tr>
    <th>Fruit</th>
    <th>Price</th>
  </tr>
  <tr>
    <td>Kiwi</td>
    <td>5.5€</td>
  </tr>
  <tr>
    <td>Apple</td>
    <td>1.7€</td>
  </tr>
</table>
```

thead, tbody and tfoot

```
<table>
  <thead>
    <tr><th>Fruit</th><th>Price</th></tr>
    <tr><th></th><th>in €</th></tr>
  </thead>
  <tbody>
    <tr><td>Kiwi</td><td>5.5</td></tr>
    <tr><td>Apple</td><td>1.7</td></tr>
    <tr><td>Blackberries</td><td>2.7</td></tr>
  </tbody>
  <tfoot>
    <tr><td>Total</td><td>9.9</td></tr>
  <tfoot>
</table>
```

If using `thead`, `tbody` and `tfoot`, no `tr` should be direct children of the table.

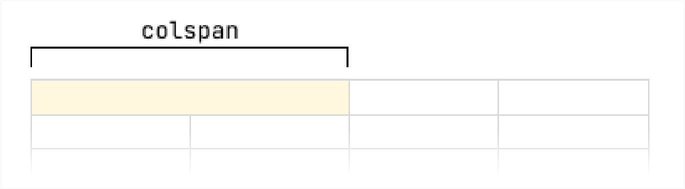
Tag omission

Within a `table`, we can omit closing tags for:
`thead`, `tbody`, `tr`, `th` and `td`

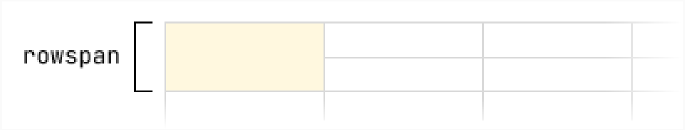
```
<table>
  <thead>
    <tr> <th> Fruit <th> Price
    <tr> <th> <th>in €
  <tbody>
    <tr> <td> Kiwi <td> 5.5
    <tr> <td> Apple <td> 1.7
    <tr> <td> Blackberries <td> 2.7
</table>
```

Joining cells

`colspan` : number of columns that the cell occupies.



`rowspan` : number of rows that a cell occupies.



Joining cells example

```
<table>
  <thead>
    <tr> <th rowspan="2"> Fruit <th colspan="2"> Price
    <tr> <td> in EUR <td> in USD
  <tbody>
    <tr> <td> Kiwi <td> 5.5 <td> 6.1
    <tr> <td> Apple <td> 1.7 <td> 2.0
    <tr> <td> Blackberries <td> 2.7 <td> 2.5
</table>
```

Table styles

`border-collapse`	`separate | collapse`
`border-spacing`	*`<length>`*
`vertical-align`	`baseline | sub | super | text-top | text-bottom | middle | top | bottom`
`white-space`	`normal | nowrap | pre | pre-wrap | pre-line`

https://cssreference.io/property/white-space/

Table of Contents

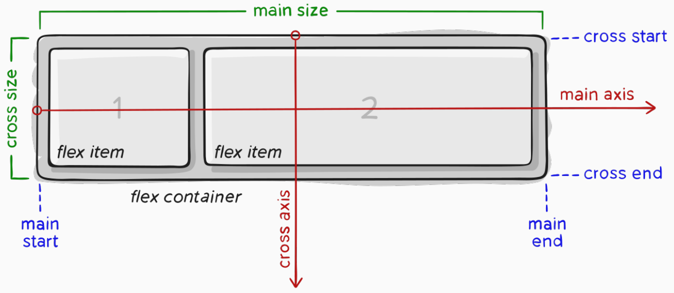
CSS Flexbox

Flexbox

Flexbox is a new mechanism to align items within its parent.

To use the Flexbox algorithm you have set **`display`** to **`flex`**:

```
div.carousel { display: flex; }
```



Flexbox Main Parameters

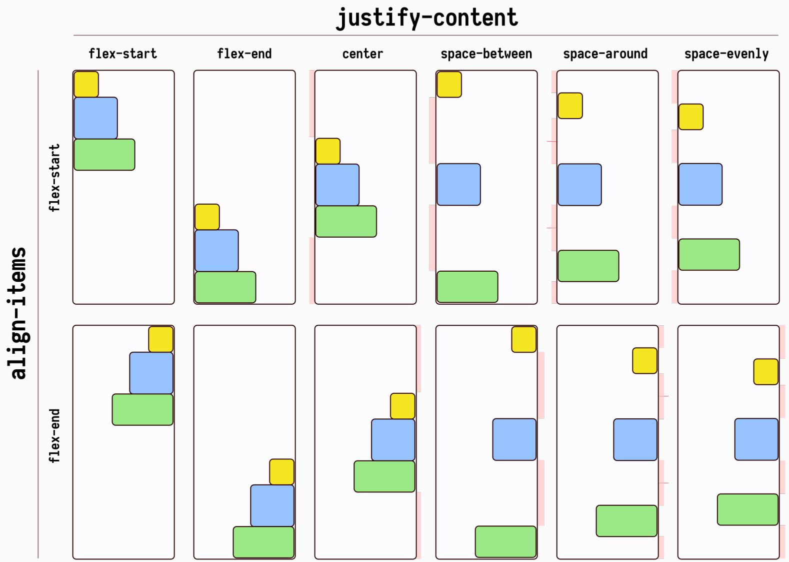
A component can specify the *layout of its children* with 3 main parameters.

`flex-direction`
Specify the **`primary axis`**
`column` (default) `row`

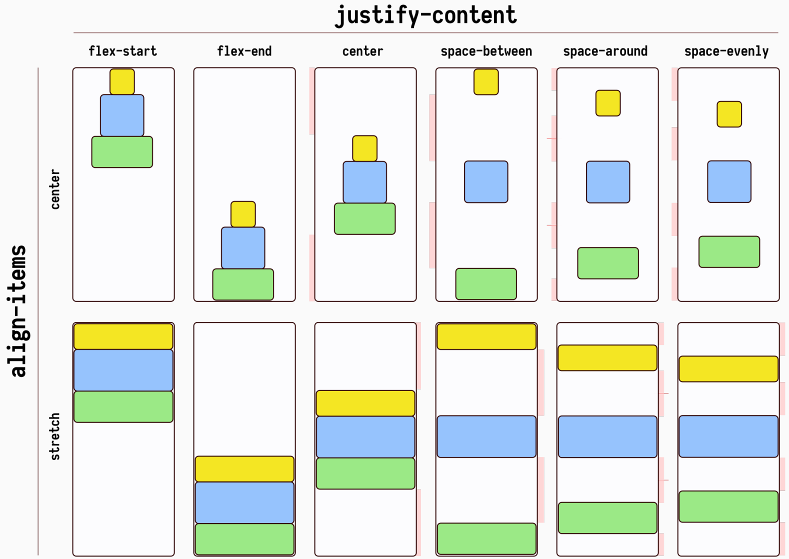
`justify-content`
Distribution of children along the **primary axis**
`flex-start` `center` `flex-end`
`space-around` `space-between` `space-evenly`

`align-items`
Alignment of children along the **secondary axis**
`flex-start` `center` `flex-end` `stretch`

Flexbox Table 1



Flexbox Table 2



Component Dimensions

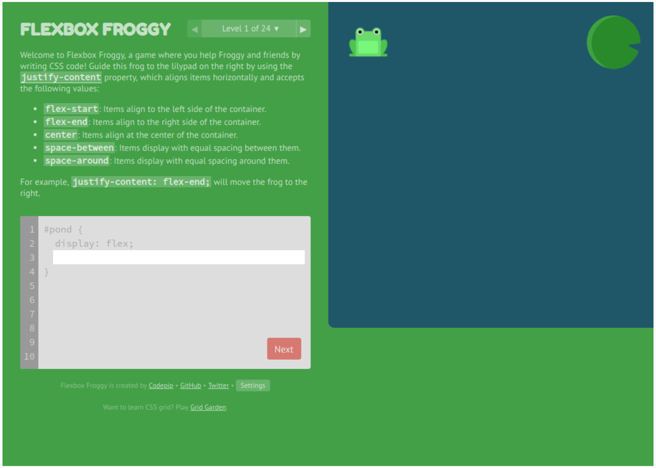
A component can either have fixed dimensions, or flex dimensions.

Fixed Dimensions
Specified using `width` and `height`, both in DIP (Device Independent Pixels).

Flex Dimensions
Specified using `flex`, which is a ”stretch factor”. The element will expand and shrink dynamically based on available space, according to the value of `flex`.

A Flexbox Game

A game for learning flexbox!



Welcome to Flexbox Froggy, a game where you help Froggy and friends by writing CSS code! Guide this frog to the lilypad on the right by using the `justify-content` property, which aligns items horizontally and accepts the following values:

- `flex-start`: Items align to the left side of the container.
- `flex-end`: Items align to the right side of the container.
- `center`: Items align at the center of the container.
- `space-between`: Items display with equal spacing between them.
- `space-around`: Items display with equal spacing around them.

For example, `justify-content: flex-end;` will move the frog to the right.

```
#pond {
  display: flex;

}
```

Flexbox Froggy is created by Codepip • GitHub • Twitter • Settings

Want to learn CSS grid? Play Grid Garden.

https://flexboxfroggy.com/

Table of Contents

CSS Element Positioning

Static Positioning

`position` : `static` is the position computed by the browser's layout algorithm.

```css
.element {
  position: static;
}
```

This is the **natural position** of the element, included in the document flow.

Layout and Positioning

Layout includes the element
`static`, `relative` (and `sticky`)

vs

Layout doesn't include the element
`absolute`, `fixed` (and `sticky`)

Relative Positioning

`position` : `relative` allows us to position an element relative to its static position.

```
.element {
  position: relative;
  top: 50px;
  left: -50px;
}
```

`top`, `right`, `bottom`, `left` are distances to the parent's edges, respectively.

Absolute Positioning

`position` : `absolute` removes the element from normal flow and positions it with respect to the closest ”relative” parent.

```
header {
  position: absolute;
  top: 0;
  right: 0;
  left: 0;
  height: 40px;
}
```

`fixed` is the same, with respect to the document (unaffected by scrolling).

Sticky Positioning

`position` : `sticky` is a combination of `static` and `fixed`.

When the element is visible in its position, it is `static`. When we scroll and make it disappear, it becomes `fixed`.

```css
header {
  position: sticky;
  top: 0;
  right: 0;
  left: 0;
  height: 40px;
}
```

Not 100% implemented yet but usable: `https://caniuse.com/css-sticky`

Table of Contents

Global Attributes

Global attributes

All tags have a certain number of *global attributes*, common to all elements.

Global attributes:

- `class`
- `id`
- `style`
- `title`
- ...

The `class` attribute

A `class` represents a set of elements in a document.

Many elements can be put in the same class.

A single element can belong to many classes.

The `class` attribute just lists all the classes of an element separated by spaces.

Setting classes

Adding one class

```
<h1 class="title">A Humble Title</h1>
```

Adding **many** classes:

```
<div class="box important left">Heads Up!</div>
<nav class="special menu left">
   <a href="#cart">Your Cart</a>
</nav>
```

(different `class` names are just separated by a space)

The **id** attribute

The `id` attribute identifies elements uniquely.

There can be no two items with the same **id**.

```
<h1 id="main-title">The Main Title</h1>
<div id="menu">
  <a href="/">Index</a>
  <a href="/help">Help</a>
</div>
```

Links to specific elements

If an element has an attribute **id**, links can point to that specific element

```html
<section id="first">
  <p>The first section</p>
</section>
<section>
  <p>
    The second section <br>
    <a href="#first">Go to first</a>
  </p>
</section>
```

Why do we need **`class`** and **`id`**?

To refer to elements from CSS and Javascript, we can:

- Ask for the element with a certain `id`, and apply specific styles to it, or access it programmatically.
- Ask for the *set* of elements with a certain `class`, and apply the same style to all of them, or manipulate them at once.

The `contenteditable` attribute

Setting `contenteditable` to `true`, the browser turns into an editor!

When you start editing, the browser changes the underlying DOM.

The `document.execCommand` is made available (to issue commands that manipulate the content).

Drawbacks:

- Different browsers implement edition in different ways.

https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Editable_content

Table of Contents

CSS Selectors 2

ID selectors

`#id` : specific element with ID `id`.

```html
<html>
  <body>
    <p id="abstract">Lorem ipsum...</p>
    <p>dolor sit amet</p>
  </body>
</html>
```

We style the paragraph with ID `abstract` with

```css
#abstract {
  border: 1px solid blue;
}
```

Class selector

`.class` : all elements having class `class`.

```
<body>
  <p>Lorem ipsum...</p>
  <p class="special">dolor sit amet</p>
  <p>consectetur adipiscing elit</p>
  <p>sed <span class="special">do eiusmod</span> tempor</p>
</body>
```

We style the set of elements with class `special` with

```
.special {
  background-color: orange;
}
```

Child selector

`tag1 > tag2` : all `tag2` which are direct children of `tag1` .

```html
<body>
  <p>First <span>paragraph</span></p>
  <p>Second paragraph</p>
  <p><a href="/third">Third <span>paragraph</span></a></p>
</body>
```

We can target only the first span with

```css
p > span {
  color: blue;
}
```

Tag and class

`tag.class` : all `tag` s which have a certain `class`.

```html
<body>
  <p>Lorem ipsum...</p>
  <p class="special">dolor sit amet</p>
  <p>consectetur adipiscing elit</p>
  <p>sed <span class="special">do eiusmod</span> tempor</p>
</body>
```

We style the `span` with class `special` with

```css
span.special {
  background-color: orange;
}
```

Attribute selector

`tag[attr]` : selects a `tag` with an attribute `attr` .

```
<body>
  <ol>
    <li value="2">First</li>
    <li>Second</li>
    <li value="5">Third</li>
  </ol>
</body>
```

Style `li` s that *have* the `value` attribute:

```
li[value] {
  color: gray;
}
```

Attribute and value selector

`tag[attr=value]` : selects a `tag` with `attr` equal to `value`.

```
<nav>
  <a href="/home">Home</a>
  <a href="/blog">Blog</a>
  <a href="/about">About</a>
</nav>
```

We can style `a` elements which link to "/home".

```
a[href="/home"] {
  color: pink;
}
```

Table of Contents

CSS Pseudo Classes and Elements

Pseudo-class selectors

Pseudo-classes refer to classes that we don’t have to declare, and which represent stuff that the browser already knows about:

- If an element is empty.
- Which position an element is (first, last, ...).
- The state of a link (enabled, disabled, clicked, visited, focused, ...).
- The validity of form controls (valid, invalid).
- Negation of other selector (not).

Pseudo-elements

Pseudo-elements refer to parts of elements or special elements:

- First letter of line.
- Before/After content.

Empty elements

The `:empty` class refers to elements which have no content

```
<p>A paragraph with text</p>
<p></p>
```

```
p:empty { margin: 0; }
```

First/last child

The `:first-child` class applies to elements that occupy the first position in the list of children.

The `:last-child` is analogous for the last child.

```
li:first-child { border-top: 1px solid #ccc; }
li:last-child  { border-bottom: 1px solid #ccc; }
```

Child in position N

The `nth-child(n)` lets you use an index or a formula:

```css
li:nth-child(2) {
  background-color: yellow;
}
tr:nth-child(2) td:nth-child(3) {
  border-color: gray;
}
ul li:nth-child(2n) {
  text-transform: uppercase;
}
```

Link state classes

- `:link` : An unvisited link.
- `:visited` : An visited link.
- `:focus` : A link in focus.
- `:hover` : A link with cursor on top.
- `:active` : A link being clicked.

Negation pseudo-class

The `:not(sel)` class is true for elements that aren't `sel`.

`sel` *is a simple selector (tag or class)*

```
li:not(.more-info) {
  color: red;
}
:not(section) > table {
  display: none;
}
.link:not(li):not(p) {
  font-style: italic;
}
```

Before/After pseudo-elements

`::before` and `::after`, in combination with the `content` property, let you add prefixes and suffixes:

```
.story::before {
  content: "Once upon a time";
}
.story::after {
  content: "and they lived happily ever after.";
}
```

`content` can be a string, an image (no resizing), or a counter.

Similar pseudo-elements: `::first-line` *and* `::first-letter`

Table of Contents

CSS: The Cascade

Ambiguous rules

The **same element** can be the target of rules with different values

```
h1 { color: red; }
body h1 { color: green; }
```

```
h2.grape { color: purple; }
h2 { color: silver; }
```

```
html > body table tr[id="totals"] td ul > li { color: maroon; }
li#answer { color: navy; }
```

How do we know which one will win?

Specificity

A selector's specificity is determined by its components.

Specificity is a vector with 4 components: **(0, 1, 5, 2)**.

Components to the left of the vector weight more.
$(0, 0, 2, 0) > (0, 0, 1, 0)$
$(0, 1, 0, 0) > (0, 0, 1, 0)$

The actual specificity is determined **adding up** the different contributions:

For every element	add **(0, 0, 0, 1)**
For every class	add **(0, 0, 1, 0)**
For every ID	add **(0, 1, 0, 0)**
For every inline style	add **(1, 0, 0, 0)**

Specificity examples

```
h1 { color: purple; }                          /* 0, 0, 0, 1 */
p em { color: blue; }                          /* 0, 0, 0, 2 */
.grape { color: green; }                       /* 0, 0, 1, 0 */
*.bright { color: yellow; }                    /* 0, 0, 1, 0 */
p.bright e.dark { color: black; }              /* 0, 0, 2, 2 */
#id216 { color: orange; }                      /* 0, 1, 0, 0 */
div#sidebar *[href] { color: pink; }           /* 0, 1, 1, 1 */
nav#menu li#top { color: red; }                /* 0, 2, 0, 2 */
```

Importance

The `!important` keyword marks a rule as being above all others.

```css
p.dark {
  color: #333 !important;
  background: white;
}
```

`!important` *must go just before the semicolon.*

Important declarations have the same specificity but are considered separately to others. They always win against non-important declarations.

The `!` sign does not mean negation as in many programming languages!

Inheritance

Some rules apply to an element *and its descendants*:

- `color` ,
- `font-size` ,
- `font-family` …

Some rules apply just to the root element:

- `border` ,
- `margin` ,
- `margin` …

Which rules apply or not to descendants is down to common sense.

The Cascade

1. **Importance**
   If the rule was marked as `!important` (also transitions and animations).
2. **Origin**
   Where the rule was defined (website, user, browser defaults).
3. **Specificity**
   The specificity vector explained before.
4. **Order**
   The order of declaration (last rule overwrites a previous one).