HTML & CSS

IF3110 – Web-based Application Development
School of Electical Engineering and Informatics
Institut Teknologi Bandung

Reference

It is a living document

https://html.spec.whatwg.org/multipage/

Additional reference:

https://developer.mozilla.org/en-US/
https://weblab.mit.edu/schedule

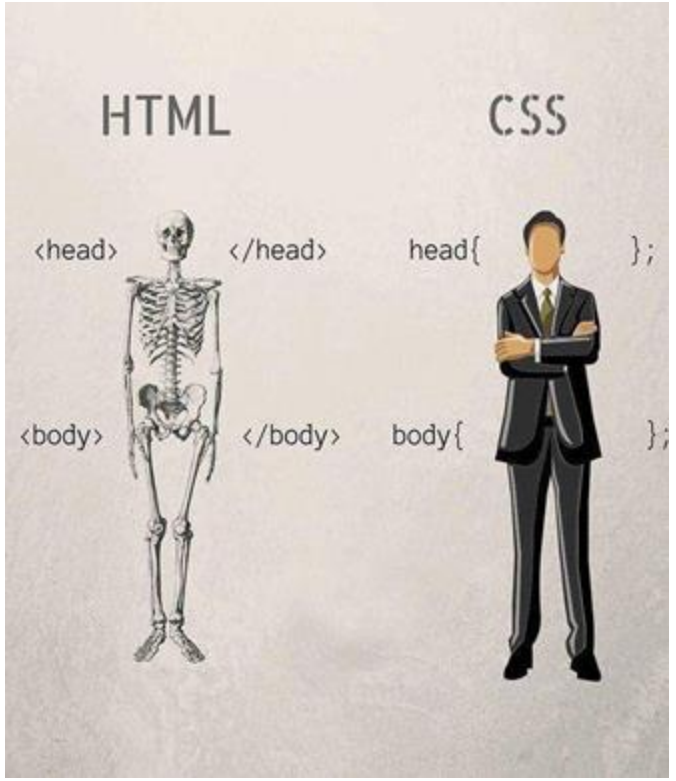
Definition

HTML

Hypertext Markup Language

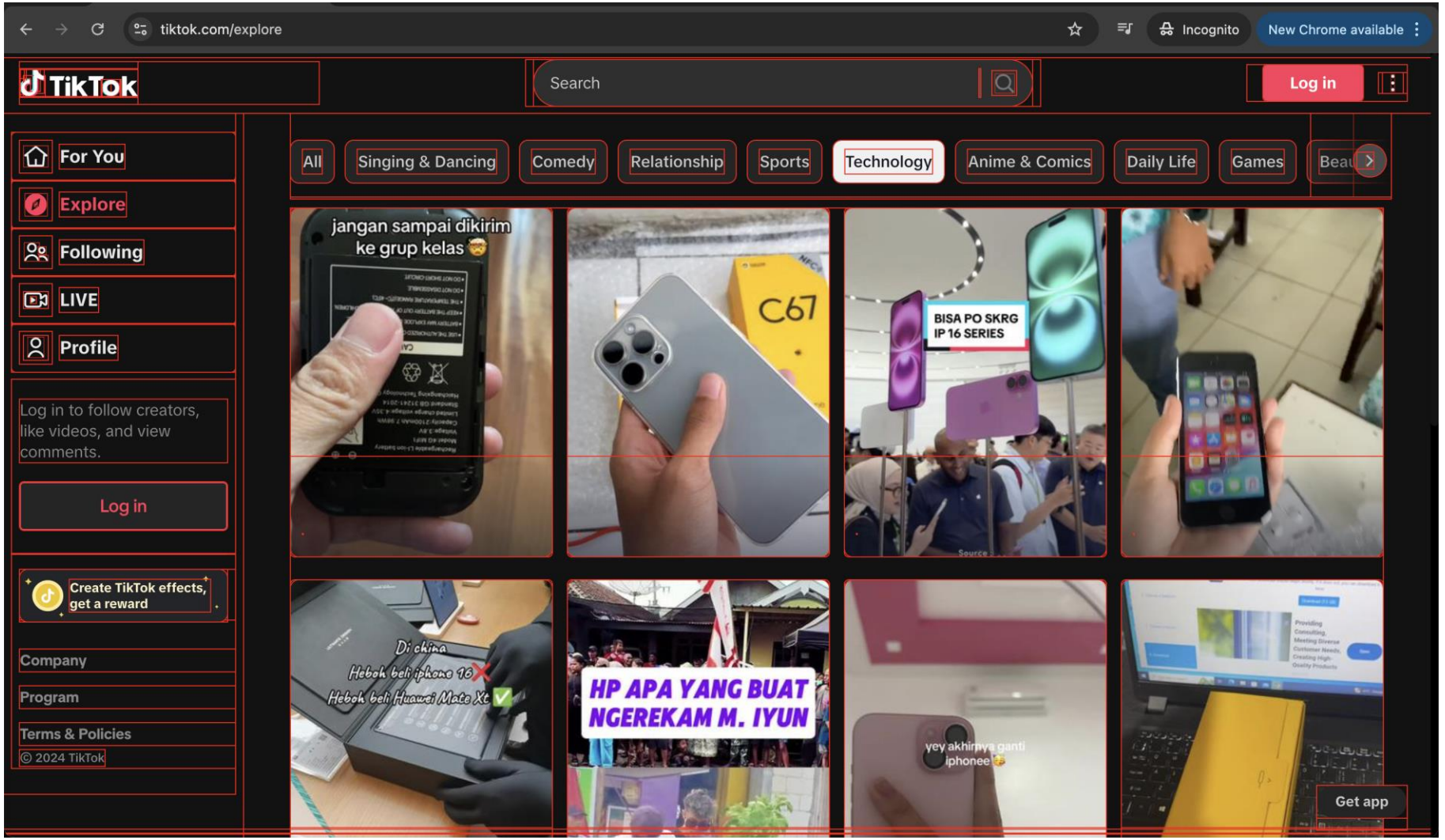
The language your web browser uses to describe the content and structure of web pages

HTML & CSS Analogy



Source: weblab.mit.edu

HTML = Nested Boxes



HTML Document Structure

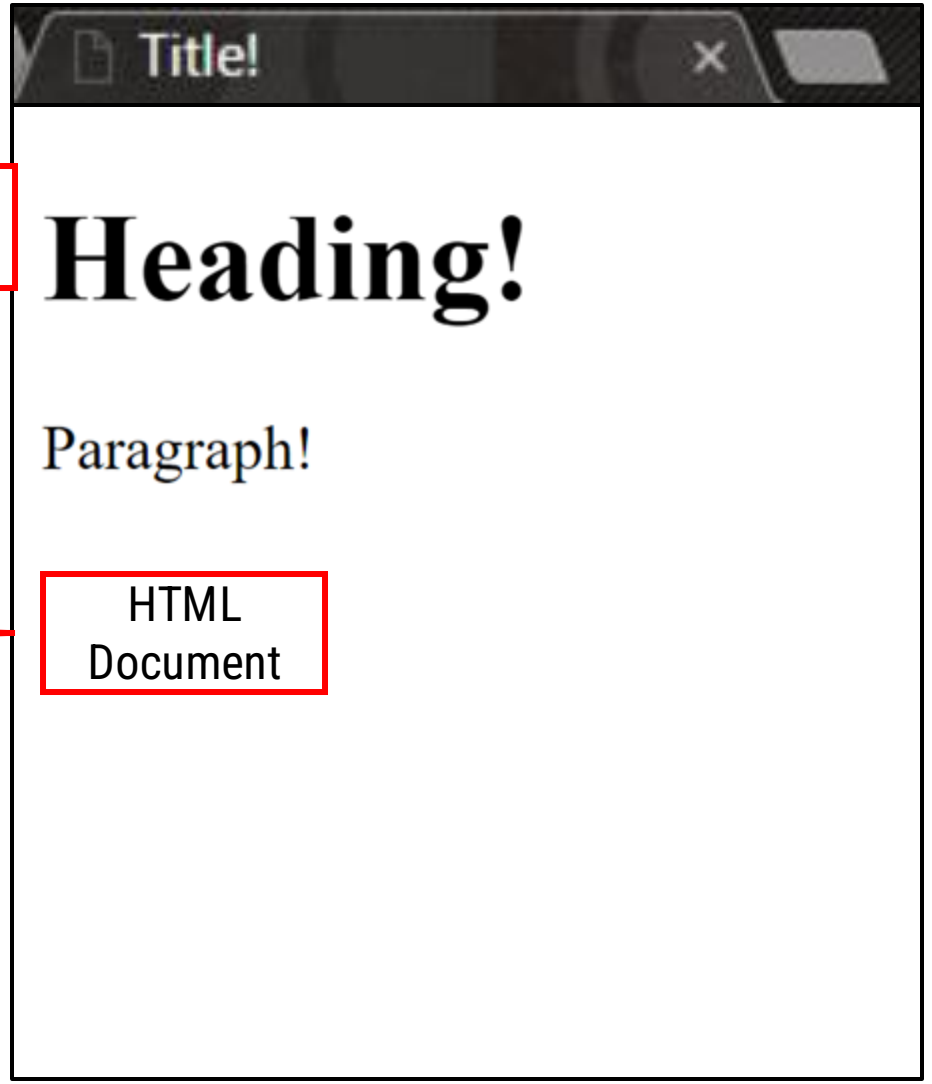
hello.html

this makes sure page uses latest html and not some random fallback version

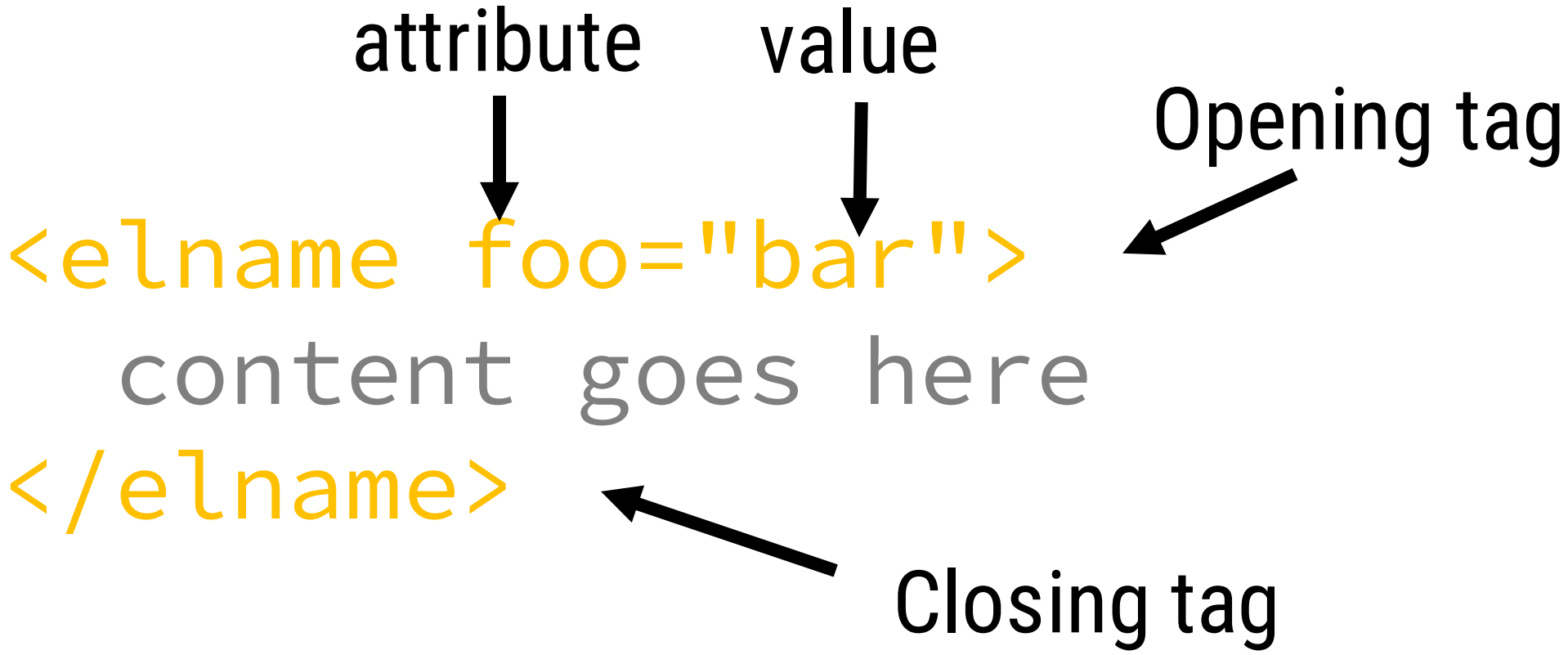
```
<!DOCTYPE html>
<html>
    <head>
        <title>Title!</title>
    </head>
    <body>
        <h1>Heading!</h1>
        <p>Paragraph!</p>
    </body>
</html>
```

Document Type Declaration

HTML Document



Anatomy of HTML Element

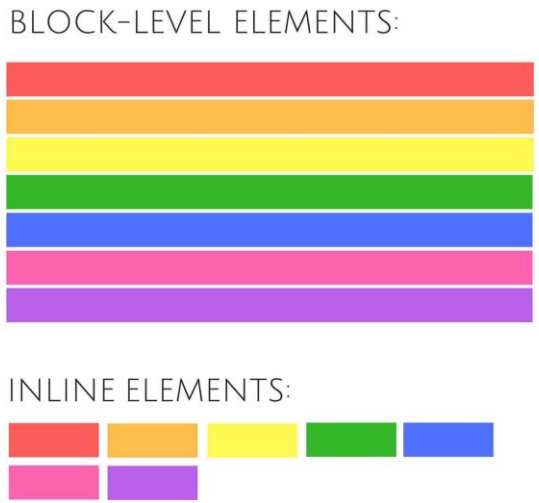


Elements and tags are *not* the same things. Tags begin or end an element in source code, whereas elements are part of the DOM, the document model for displaying the page in the browser.

Basic HTML Element

html	Root of HTML Document
head	Info about Document
body	Document Body
h1, h2, h3, …	Header
p	Paragraph
div	Generic block section
span	Generic inline section

Types: Block vs Inline Element



Block Level Elements

A block-level element always starts on a new line, and the browsers automatically add some space (a margin) before and after the element.

A block-level element always takes up the full width available (stretches out to the left and right as far as it can).

Two commonly used block elements are: `<p>` and `<div>`.

Inline Elements

An inline element does not start on a new line.

An inline element only takes up as much width as necessary.

Note: An inline element cannot contain a block-level element!

Example: Inserting Link

```
<a href="https://itb.ac.id">
Link to itb.ac.id!</a>
```

Link to itb.ac.id!

Example: Inserting Image

```
<img src="pusheen.gif"></img>

<img src="pusheen.gif" />
```

Example: Inserting Lists

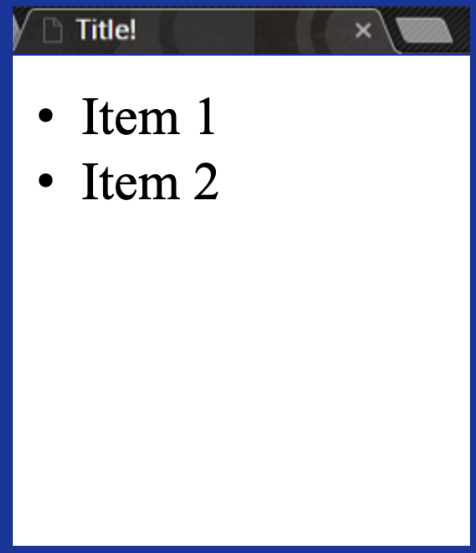
 Ordered List (1, 2, 3…)

 Unordered List (bullets)

 List Item

hello.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Title!</title>
  </head>
  <body>
    <ul>
      <li>Item 1</li>
      <li>Item 2</li>
    </ul>
  </body>
</html>
```

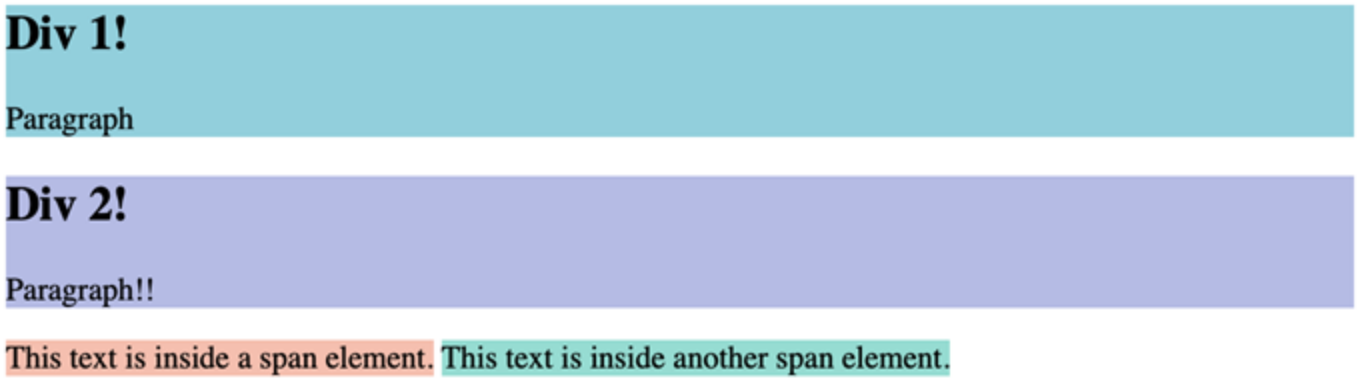


div & span

 Block Section in Document

 Inline Section in Document

```
<div id="div-0">
  <h2>Div 1!</h2>
  <p>Paragraph<p>
</div>

<div id="div-1">
  <h2>Div 2!</h2>
  <p>Paragraph!!</p>
</div>

<span id="span-0">This text is inside a span element.</span>
<span id="span-1" span>This text is inside another span element.</span>
```

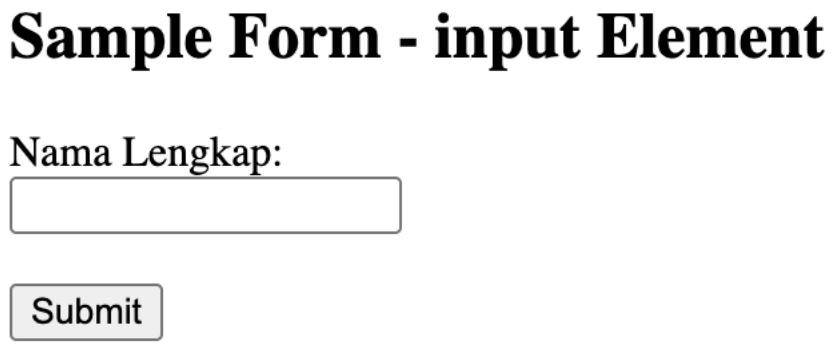


<form> Elements

Elements

<input>
<label>
<select>
<textarea>
<button>
<fieldset>
<legend>
<datalist>
<output>
<option>
<optgroup>

```
<!DOCTYPE html>
<html>
<body>

<h2>Sample Form - input Element</h2>

<form action="/action_page.php">
  <label for="fname">Nama Lengkap:</label><br>
  <input type="text" id="name" name="fname"><br><br>
  <input type="submit" value="Submit">
</form>

</body>
</html>
```



so many elements...

HTML elements reference

https://html.spec.whatwg.org/multipage/#toc-semantics
https://developer.mozilla.org/en-US/docs/Web/HTML/Element

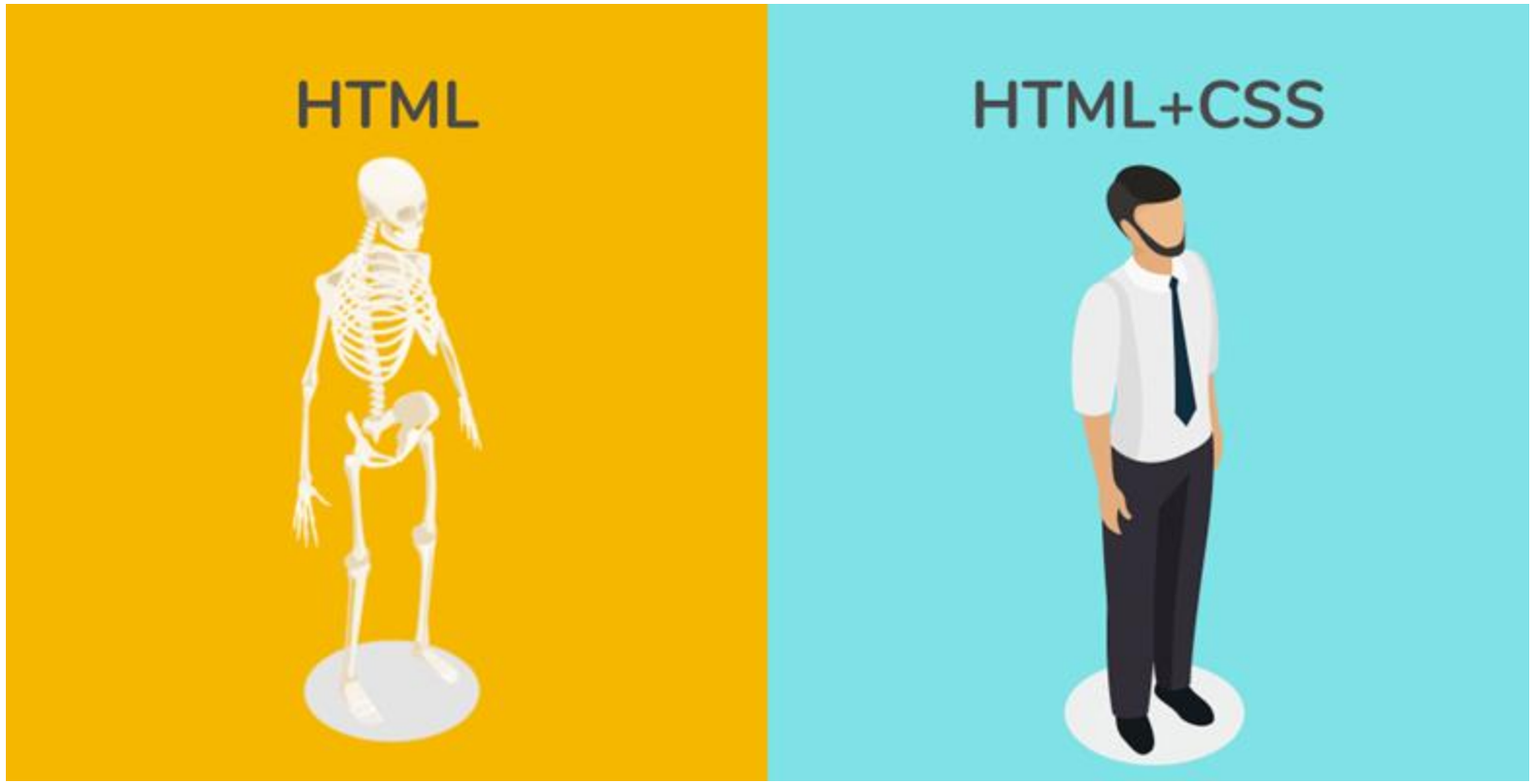
Definition

CSS

Cascading Style Sheets

The rules that tell your web browser how stuff looks

CSS = A list of descriptions

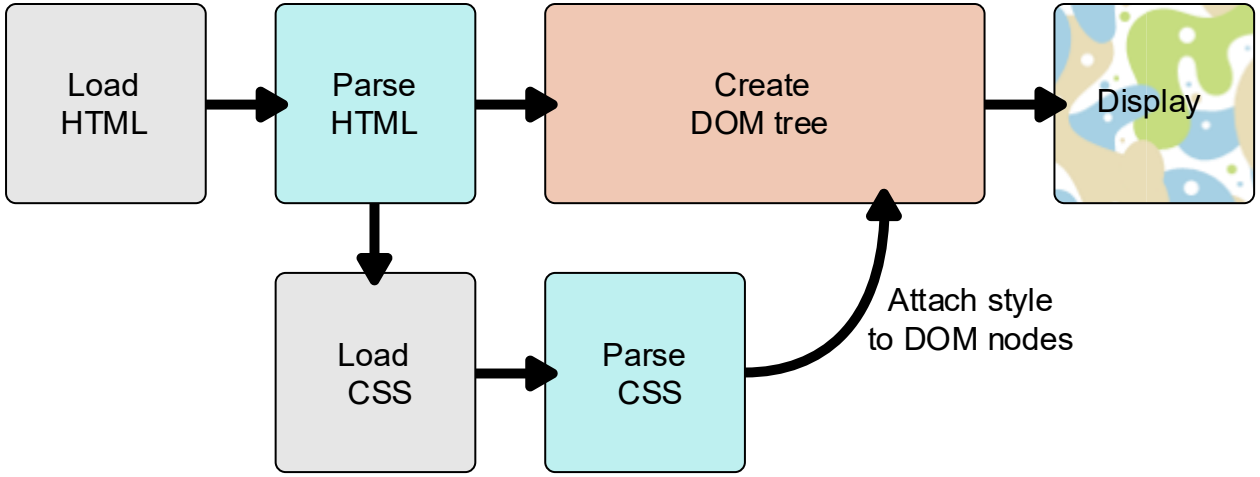


making things look pretty

CSS is used to provide descriptions or rules on how HTML elements should be displayed in the browser.

How does CSS actually work?

When a browser displays a webpage, it merges the content with its styling information. The browser goes through several key steps, and different browsers may handle the process slightly differently. The following diagram provides a simplified illustration of the process.



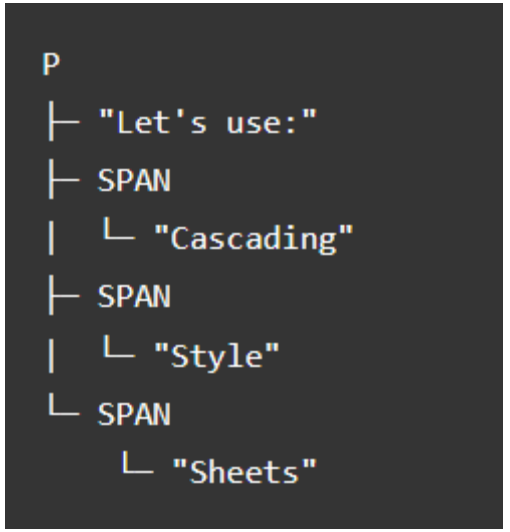
CSS & DOM

- A DOM has a tree-like structure. Each element, attribute, and piece of text in the markup language becomes a DOM node in the tree structure. The nodes are defined by their relationship to other DOM nodes. Some elements are parents of child nodes, and child nodes have siblings.

- When CSS is applied, the styles are associated with the DOM nodes, determining how each element will look on the page, such as its color, size, and position, without altering the DOM structure itself. This styling process helps shape the visual presentation of the document.

Example:

```
<p>
  Let's use:
  <span>Cascading</span>
  <span>Style</span>
  <span>Sheets</span>
</p>
```



Applying CSS to the DOM

Example:

```
<p>
  Let's use:
  <span>Cascading</span>
  <span>Style</span>
  <span>Sheets</span>
</p>

span {
border: 1px solid black;
 background-color: lime;
}
```

Result:

Let's use: Cascading Style Sheets

The CSS Cascade

The cascade is the algorithm for solving conflicts where multiple CSS rules apply to an HTML element.

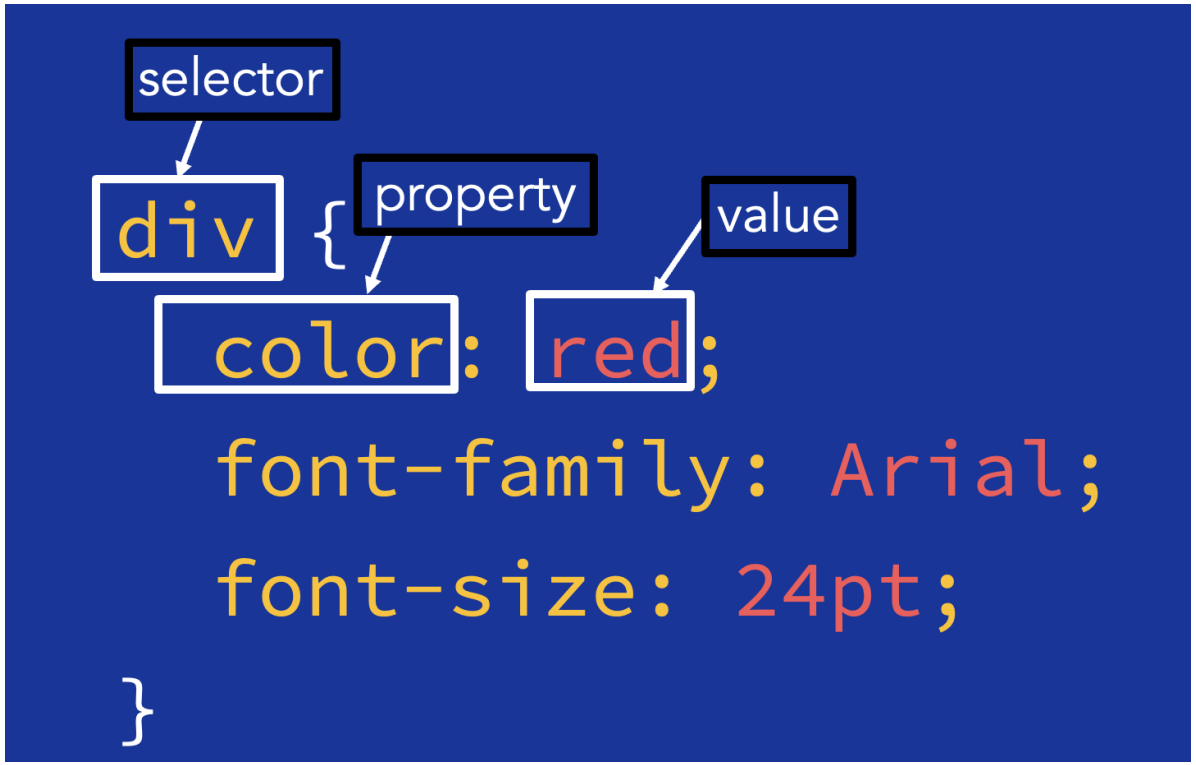
```
button {
  color: red;
}

button {
  color: blue;
}
```

Understanding the cascade algorithm helps you understand how the browser resolves conflicts like this. The cascade algorithm is split into 4 distinct stages.

1. **Position and order of appearance**: the order of which your CSS rules appear
2. **Specificity**: an algorithm which determines which CSS selector has the strongest match
3. **Origin**: the order of when CSS appears and where it comes from, whether that is a browser style, CSS from a browser extension, or your authored CSS
4. **Importance**: some CSS rules are weighted more heavily than others, especially with the !important rule type

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

CSS ruleset (rule)



List of selectors, properties :
https://developer.mozilla.org/en-US/docs/Web/CSS/Reference

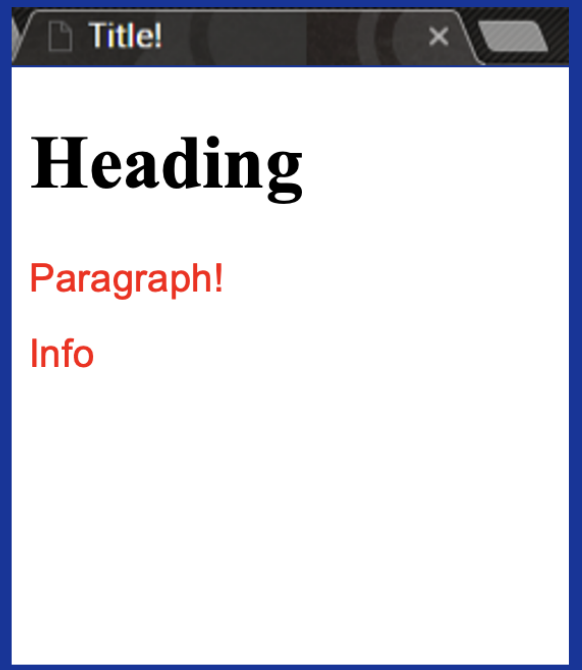
Example 1: Using Element Type Selector

hello.html

```
<h1>Heading!</h1>
<div>Paragraph!</div>
<div>Info</div>
```

style.css

```
div {
  color: red;
  font-family: Arial;
  font-size: 24pt;
}
```



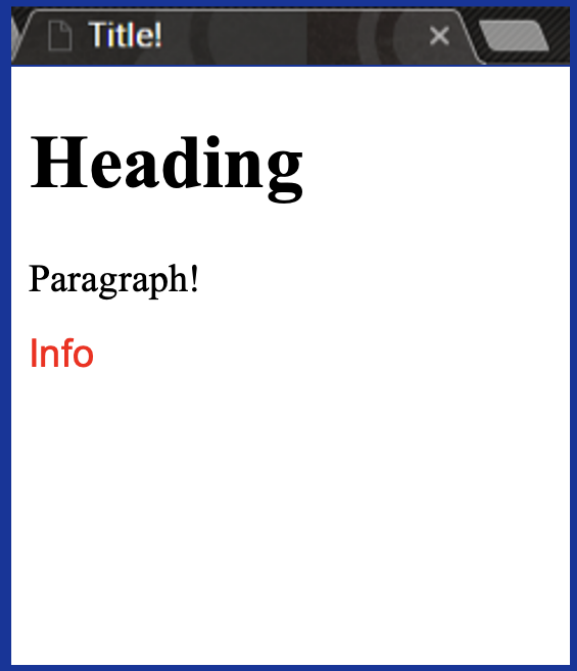
Example 2: Using Class Selector

hello.html

```
<h1>Heading!</h1>
<div>Paragraph!</div>
<div class="info">Info</div>
```

style.css

```
.info {
  color: red;
  font-family: Arial;
  font-size: 24pt;
}
```



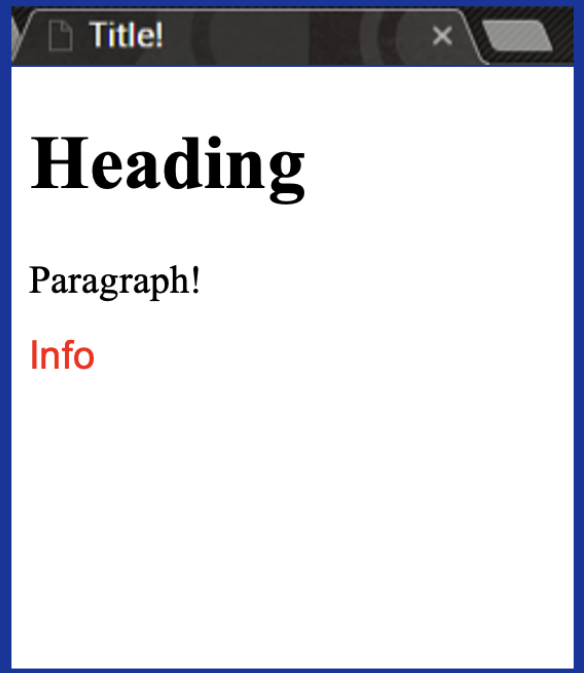
Example 3: Using ID Selector

hello.html

```
<h1>Heading!</h1>
<div>Paragraph!</div>
<div id="unique">Info</div>
```

style.css

```
#unique {
  color: red;
  font-family: Arial;
  font-size: 24pt;
}
```



ID vs Class

ID

- An element can have only one ID
- IDs must be unique in any given HTML document

```
#id {
    …
}
```

Class

- An element can have multiple classes
- Can use the same class on multiple elements

```
.classname {
    …
}
```

CSS Specificity

If there are two or more CSS rules that point to the same element, the selector with the highest specificity value will "win", and its style declaration will be applied to that HTML element.

Think of specificity as a score/rank that determines which style declaration is ultimately applied to an element.

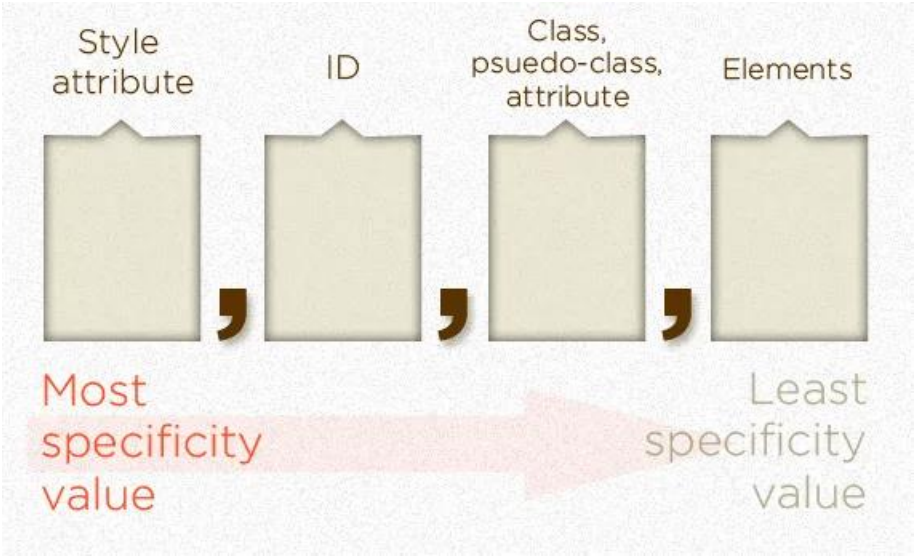
Example 1:

```
<html>
<head>
 <style>
   p {color: red;}
 </style>
</head>
<body>

<p>Hello World!</p>

</body>
</html>
```

Example 2:

```
<html>
<head>
 <style>
   .test {color: green;}
  p {color: red;}
 </style>
</head>
<body>

<p class = “test”>Hello World!</p>

</body>
</html>
```

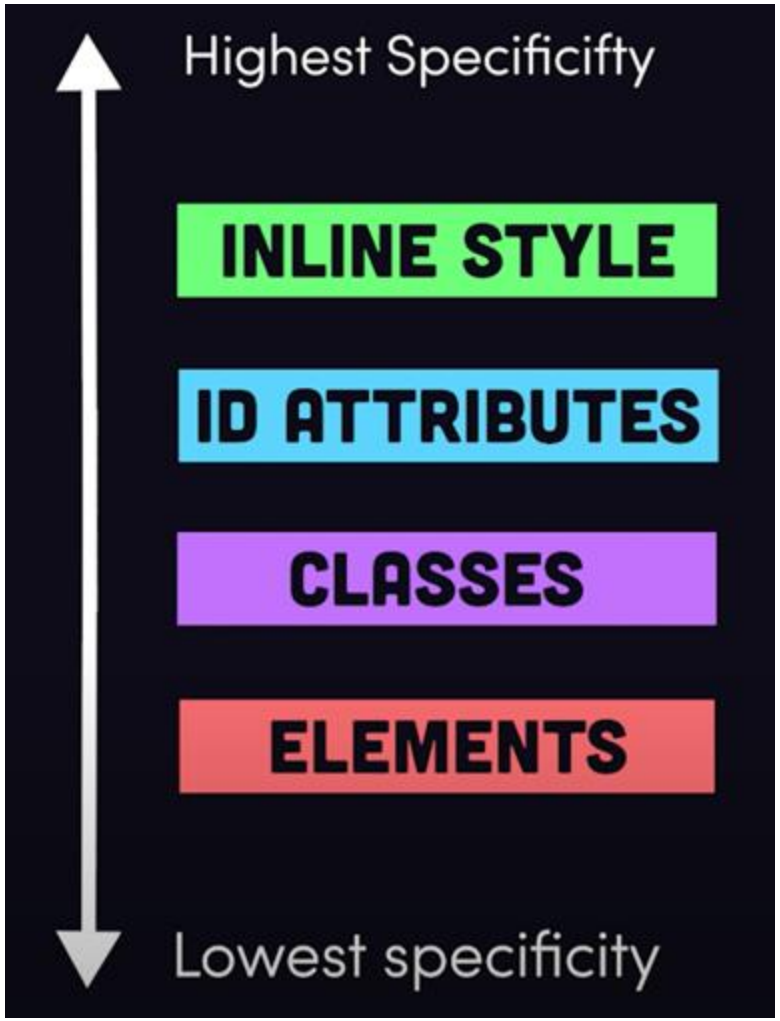
CSS Specificity (2)



Start at 0, add 100 for each ID value, add 10 for each class value (or pseudo-class or attribute selector), add 1 for each element selector or pseudo-element.

Note 1: Inline style gets a specificity value of 1000, and is always given the highest priority!
Note 2: There is one exception to this rule: if you use the `!important` rule, it will even override inline styles!

CSS Hierarchy



read more about **specificity** of selectors here

Only use classes for CSS styling!

CSS !important

- The **!important** rule in CSS is used to add more importance to a property/value than normal. In fact, if you use the **!important** rule, it will override ALL previous styling rules for that specific property on that element!
- The only way to override an **!important** rule is to include another **!important** rule on a declaration with the same (or higher) specificity in the source code

Example : Using !important

```
#myid {
  background-color: blue;
}

.myclass {
  background-color: gray;
}

p {
  background-color: red !important;
}
```

```
<body>
<p>Hello World!</p>
<p class="myclass">Hello World!</p>
<p id="myid“ >Hello World!</p>
</body>
```

```
#myid {
  background-color: blue; !important;
}

.myclass {
  background-color: gray; !important;
}

p {
  background-color: red !important;
}
```

```
<body>
<p>Hello World!</p>
<p class="myclass">Hello World!</p>
<p id="myid“ >Hello World!</p>
</body>
```

CSS Combinators

Specifies relationship between CSS selectors. Examples of selectors include HTML tags, such as div or p

We have 4 different CSS combinators:

- descendant selector (space)
- child selector (>)
- adjacent sibling selector (+)
- general sibling selector (~)

HTML Example

Take a look at an example skeleton in the HTML section, followed by some code in the CSS section.

HTML

```html
<div class="container">
  <section class="child" id="c1">
    <div>subchild 1</div>
  </section>
  <div class="child" id="c2">child 2</div>
  <div class="child" id="c3">child 3</div>
  <div class="child" id="c4">child 4</div>
</div>
```

CSS

```css
.container {

}
```

Descendant selector (space)

Matches all elements that are **descendants of the specified element**

- In the below example, we're selecting all the **`div`** elements that are descendants of the **`.container`** element → can be deeply nested (**subchild 1 is also selected**)

```css
.container div {
  font-size: 20px;
}
```

subchild 1

child 2

child 3

child 4

Child selector (>)

Matches all elements that are **direct children of the specified element**

- In the below example, only the elements denoted as **child** will be selected → further levels of nesting will not be selected

```css
.container > div {
  font-size: 20px;
}
```

subchild 1

child 2

child 3

child 4

Adjacent Sibling Selector (+)

Selects a *single* element that is **directly after another specific element**

- In the below example, the **element directly after the element with an id of "c1"** is selected

```css
#c1 + div {
  color: red;
}
```

subchild 1

child 2

child 3

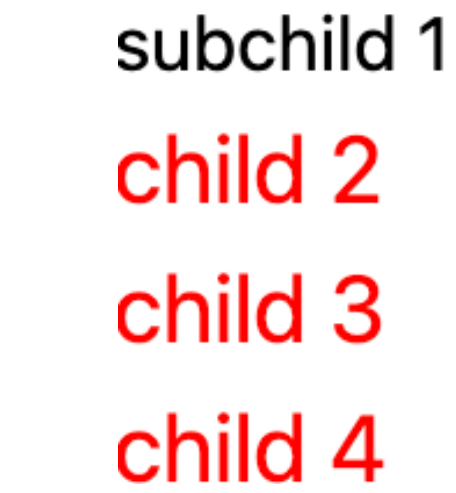
child 4

General Sibling Selector (~)

Selects *all* elements **after another specific element**

- In the below example, **all elements after the element with an id of "c1"** are selected

```css
#c1 ~ div {
  color: red;
}
```



CSS Layout

CSS page layout techniques enable precise control over the positioning of elements within a webpage. These elements can be positioned based on several factors, including their default position in the normal layout flow, their relation to surrounding elements, their parent container, and the main viewport or window

Some CSS layout techniques:

- Normal flow
- The display property
- Flexbox
- Grid
- Floats
- Positioning
- Table layout
- Multiple-column layout

Display Types

The **display** property allows us to tell the browser how to display an element and its children on the page.

We've looked at:

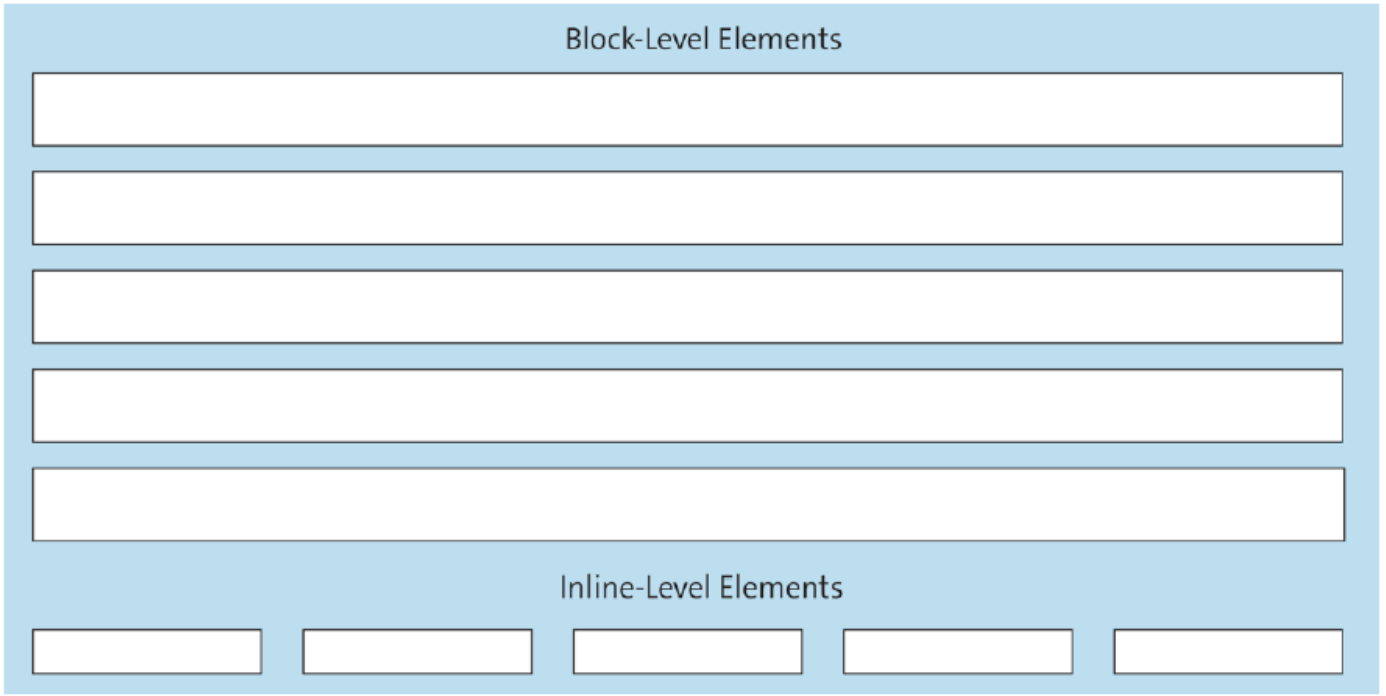
- block
- flex

Values we'll be looking at:

- grid
- none

Box model

- HTML categorizes elements as either block-level or inline-level. Block-level elements start on a new line, stacking vertically, while inline-level elements fit into the flow of text, lining up horizontally. With CSS's display property, we can alter these behaviors: setting display: **block** treats the element as block-level, and display: **inline** makes it inline-level.

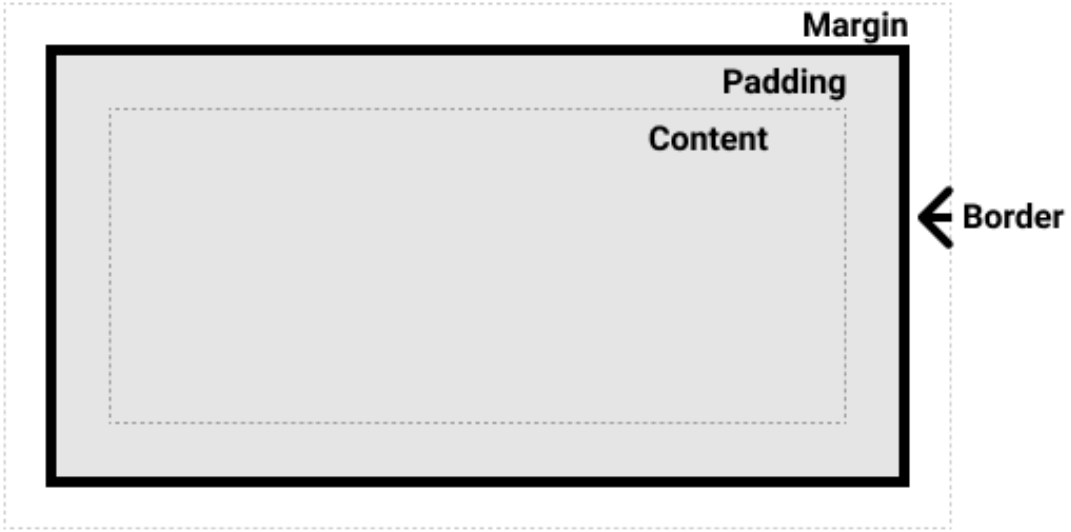


CSS box model

- The CSS box model as a whole applies to block boxes and defines how the different parts of a box — margin, border, padding, and content — work together to create a box that you can see on a page. Inline boxes use just some of the behavior defined in the box model.

Parts of a box in CSS:

- Content box: The area where the content is displayed. You can size it with **width**, **height**, or similar properties.
- Padding box: The space around the content. Set its size using **padding**.
- Border box: The layer around the content and padding. Size it with **border** properties.
- Margin box: The outermost space between the element and others. Adjust it using **margin**.



HTML Example

Take a look at an example skeleton in the HTML section, followed by some code in the CSS section.

HTML

```html
<div class="container">
  <div class="child" id="c1">child 1</div>
  <div class="child" id="c2">child 2</div>
  <div class="child" id="c3">child 3</div>
  <div class="child" id="c4">child 4</div>
</div>
```

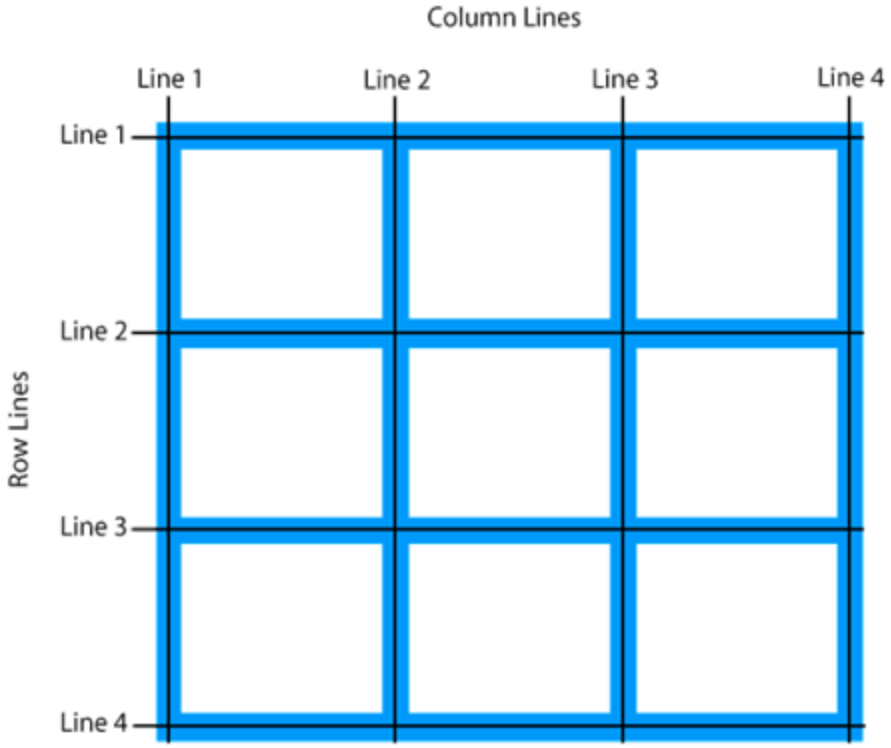
CSS

```css
.container {

}
```

display: grid

Tells the browser to display child elements in a two-dimensional grid layout

```css
.container {
  display: grid;
}
```



grid-auto-flow

auto-flow gives us the power to tell the browser how to automatically handle child elements that reach the end of our grid layout

row instructs the browser to prioritize adding rows

- By default, **grid** will prioritize rows

columns instructs the browser to prioritize adding columns

grid-auto-flow

```css
.container {
  display: grid;
  grid-auto-flow: row;
}
```

```css
.container {
  display: grid;
  grid-auto-flow: column;
}
```

child 1
child 2
child 3
child 4

child 1 child 2 child 3 child 4

Flexbox

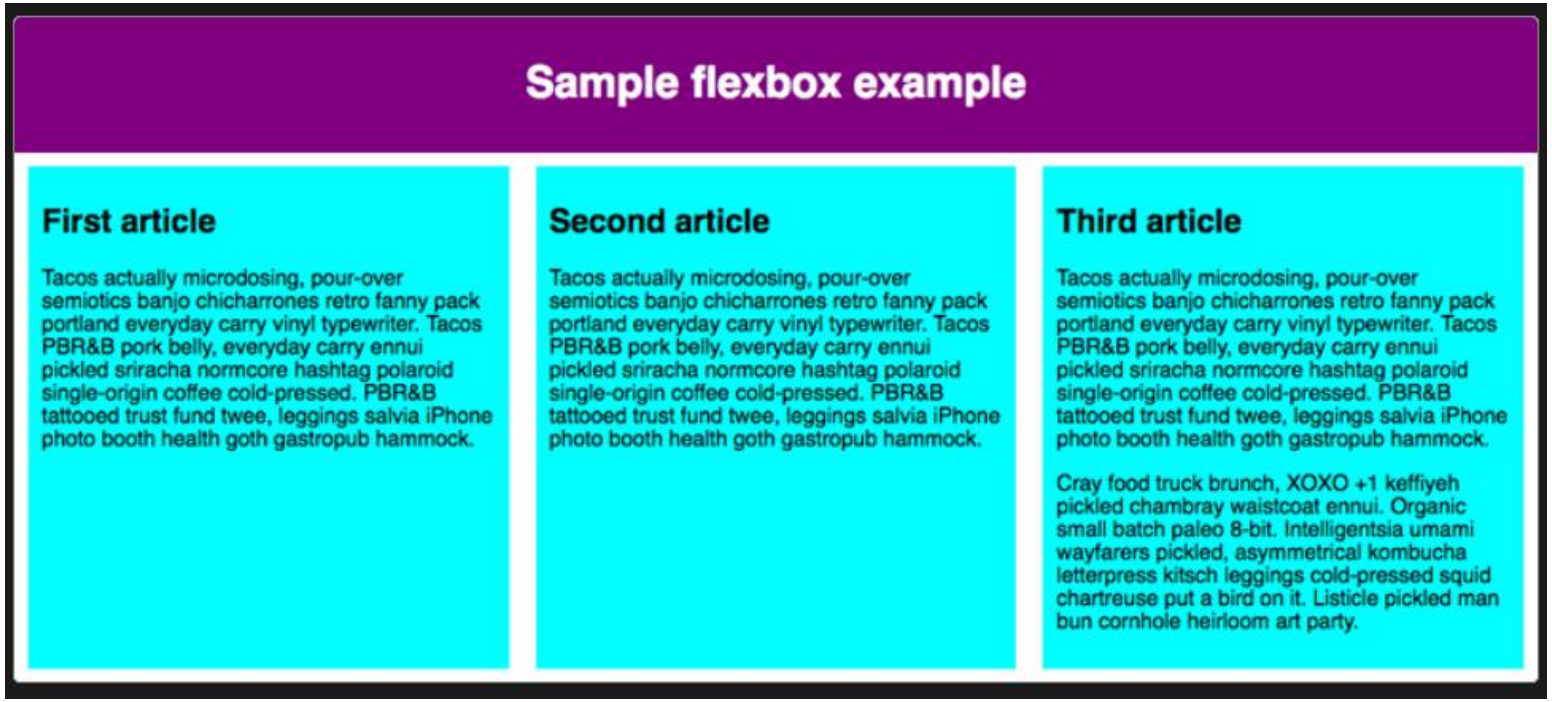
Flexbox is a one-dimensional layout method for arranging items in rows or columns. Items *flex* (expand) to fill additional space or shrink to fit into smaller spaces. This article explains all the fundamentals.

CSS flexible box layout enables you to:

- Vertically center a block of content inside its parent.
- Make all the children of a container take up an equal amount of the available width/height, regardless of how much width/height is available.
- Make all columns in a multiple-column layout adopt the same height even if they contain a different amount of content.

Example : flexbox

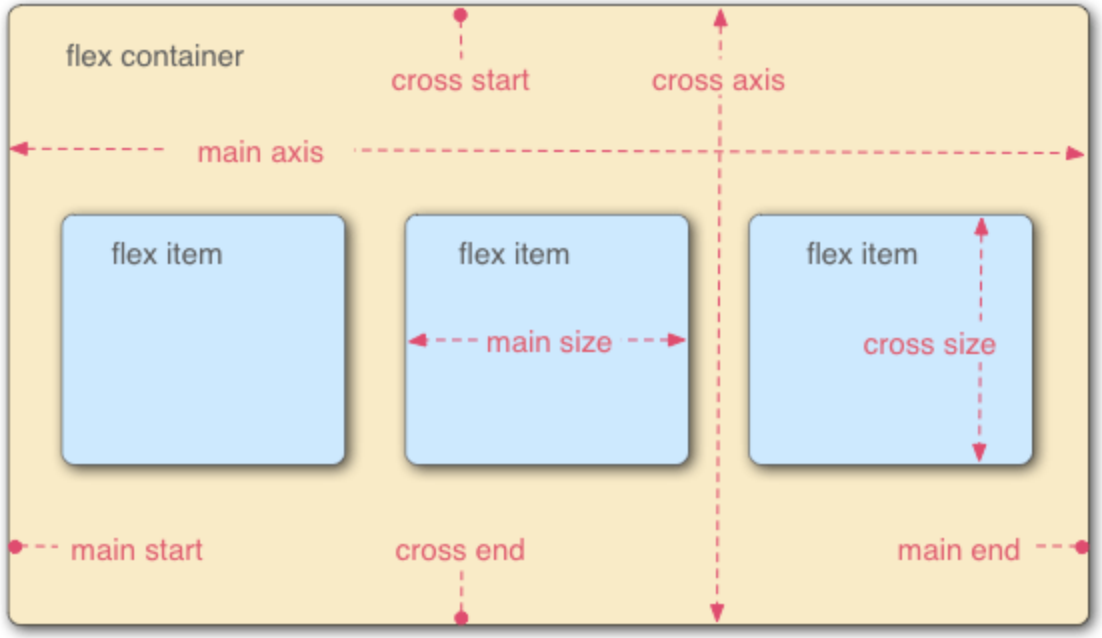
```
section {

display: flex;

}
```



flex model

When elements are laid out as flex items, they are laid out along two axes:

- The main axis is the direction in which the flex items are arranged (like a row or column). The start and end points of this axis are called main start and main end, and the length between them is the main size.
- The cross axis runs perpendicular to the main axis, with start and end points called cross start and cross end, and the length between them is the cross size.
- The element with display: flex is the flex container.
- The elements inside the flex container are called flex items.



display: none

Tells the browser to remove an element from the document
→ doesn't take up any space in the layout

- This is different from **`visibility: hidden`**, which hides the element but still takes up space in our layout

child 2 child 3
child 4

```css
#c1 {
  display: none;
}
```

child 2
child 3 child 4

```css
#c1 {
  visibility: hidden;
}
```

Content Overflow

The **overflow** property allows us to tell the browser how to handle child elements that may exceed the size of a parent element.

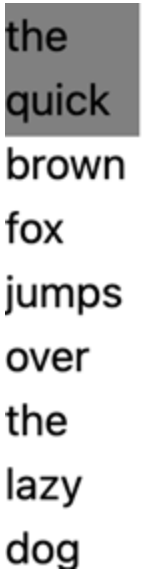
Values we'll be looking at:

- visible
- hidden
- scroll
- auto

overflow: visible

Default behavior → tells the browser to display the overflowing content

```css
.container {
  width: 50px;
  height: 50px;
  background-color: gray;
  overflow: visible;
}
```



overflow: hidden

Tells the browser to clip the overflowing content → cannot scroll within the parent element.

```css
.container {
  width: 50px;
  height: 50px;
  background-color: gray;
  overflow: hidden;
}
```

overflow: auto

Tells the browser to display a scrollbar for the parent element if needed → this scrollbar will only be present if there is any overflowing content.

```css
.container {
  width: 50px;
  height: 50px;
  background-color: gray;
  overflow: auto;
}
```

overflow content

no overflow content

Exercise: Membuat Web Portofolio Pribadi

Buat halaman web portpfolio pribadi sebagai representasi digital dari karya, keterampilan, dan identitas Anda menggunakan HTML dan CSS dari awal (from scratch), tidak diperbolehkan menggunakan template atau CSS framework apapun atau Genarative AI.

Kriteria dan Tantangan Utama:
1. Gunakan Struktur HTML yang Semantic
2. Kreativitas dalam Penggunaan Warna dan Tipografi

Additional:
3. Desain yang Interaktif (without JavaScript, only HTML+CSS)

Deploy web anda pada penyedian layanan hosting web statis (Netlify, Vercel, Static.app, Github Pages, etc…)

Link dikumpulkan melalui Edunex kelas Parent dan berikan alasan atas keputusan desain yang ada ambil untuk kriteria-kriteria tersebut.