

HTML & CSS

HTML



CSS



IF3110 – Web-based Application Development
School of Electrical 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/>

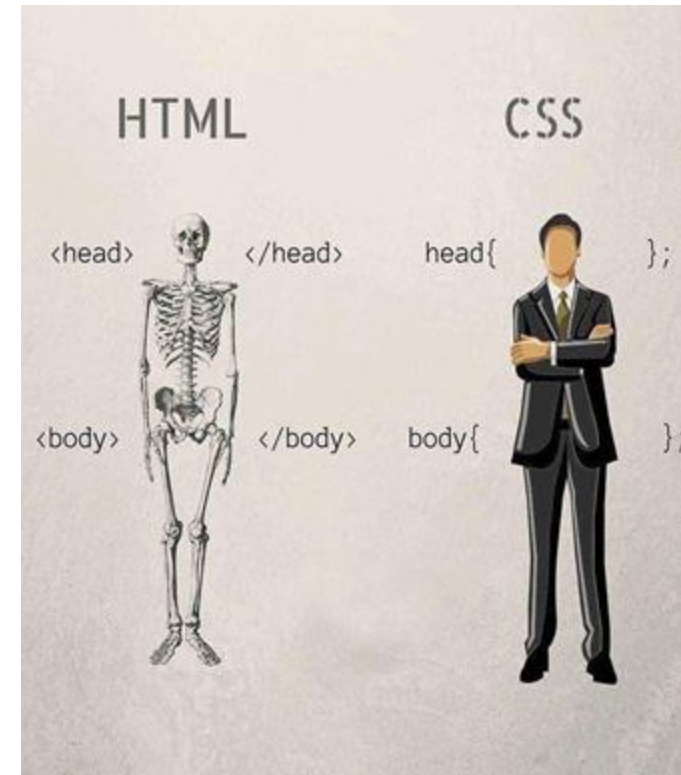
<https://weblab.mit.edu/schedule>

HTML

Hypertext Markup Language

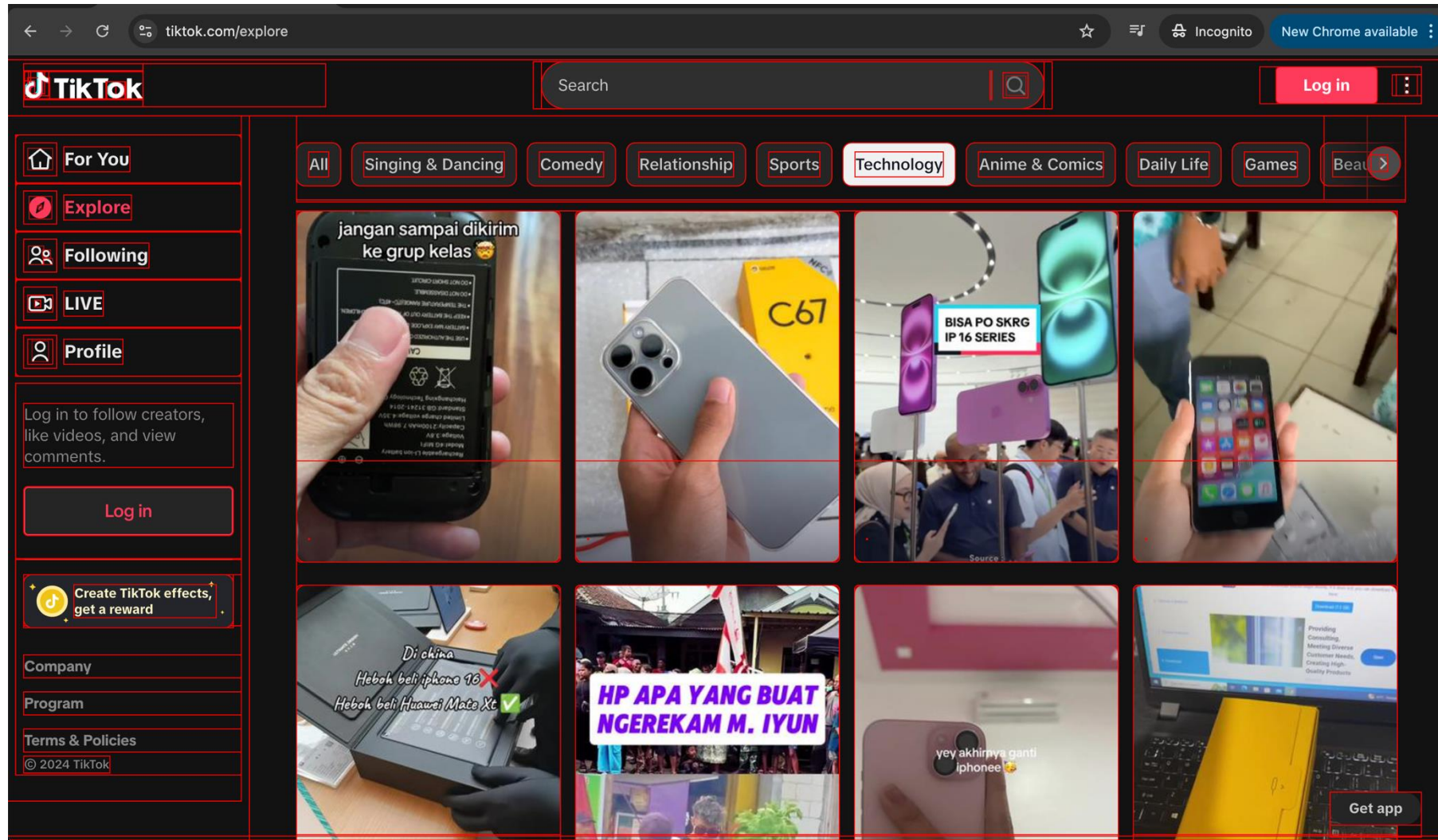
The language your web browser uses to describe the content and structure of web pages

HTML & CSS Analogy



Source: weblab.mit.edu

HTML = Nested Boxes

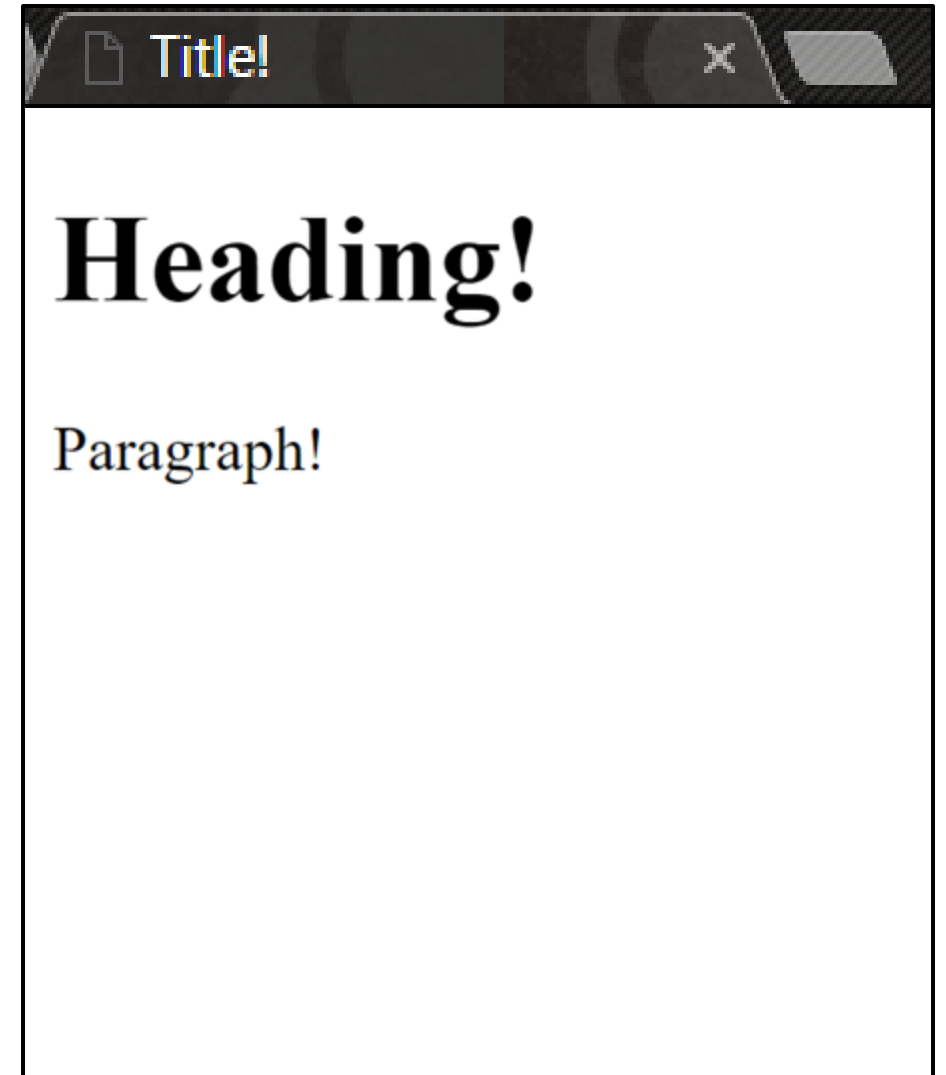


HTML Document Structure

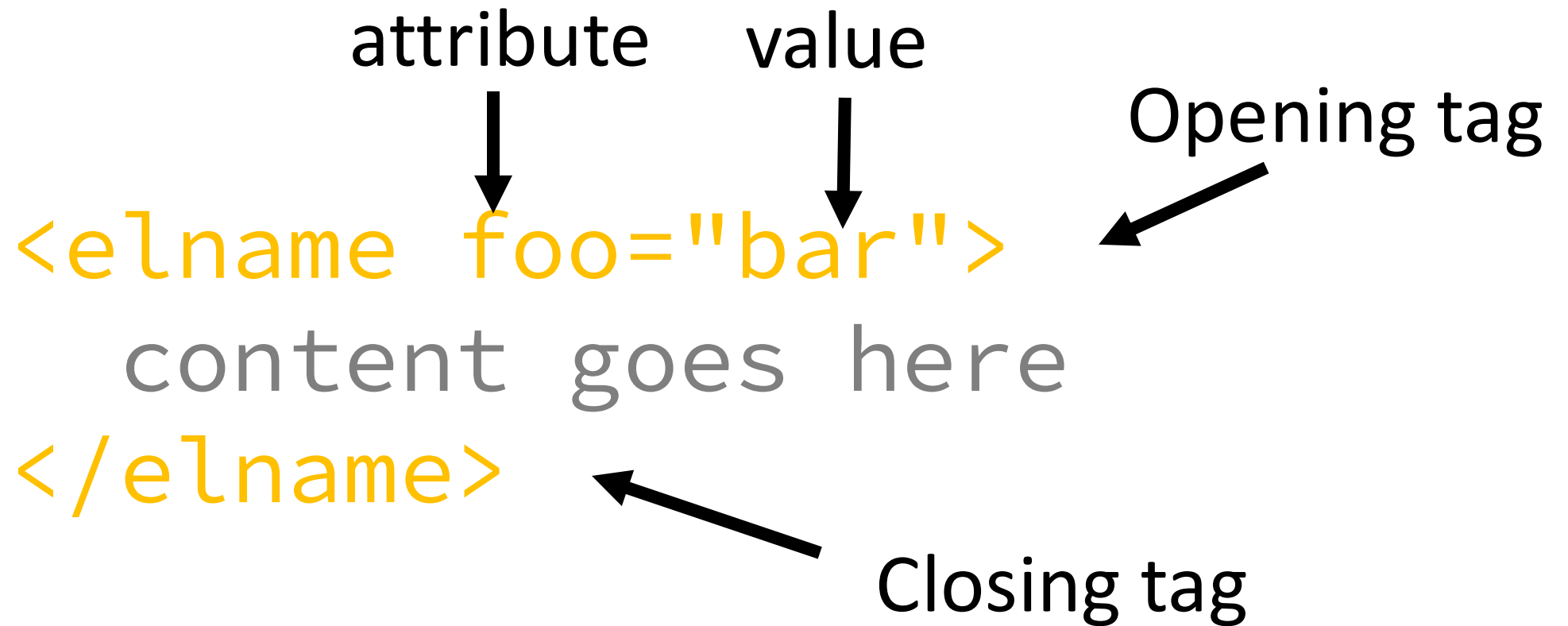
hello.html

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>
```



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

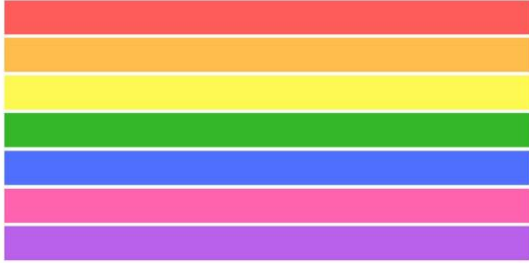
Generic block section

span

Generic inline section

Types: Block vs Inline Element

BLOCK-LEVEL ELEMENTS:



INLINE ELEMENTS:



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

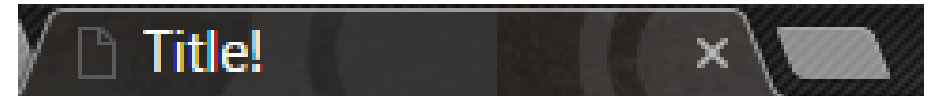
An inline element does not start on a new line.

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>
```



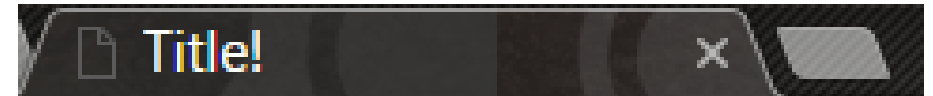
[Link to itb.ac.id!](https://itb.ac.id)

Example: Inserting Image

```
</img>
```

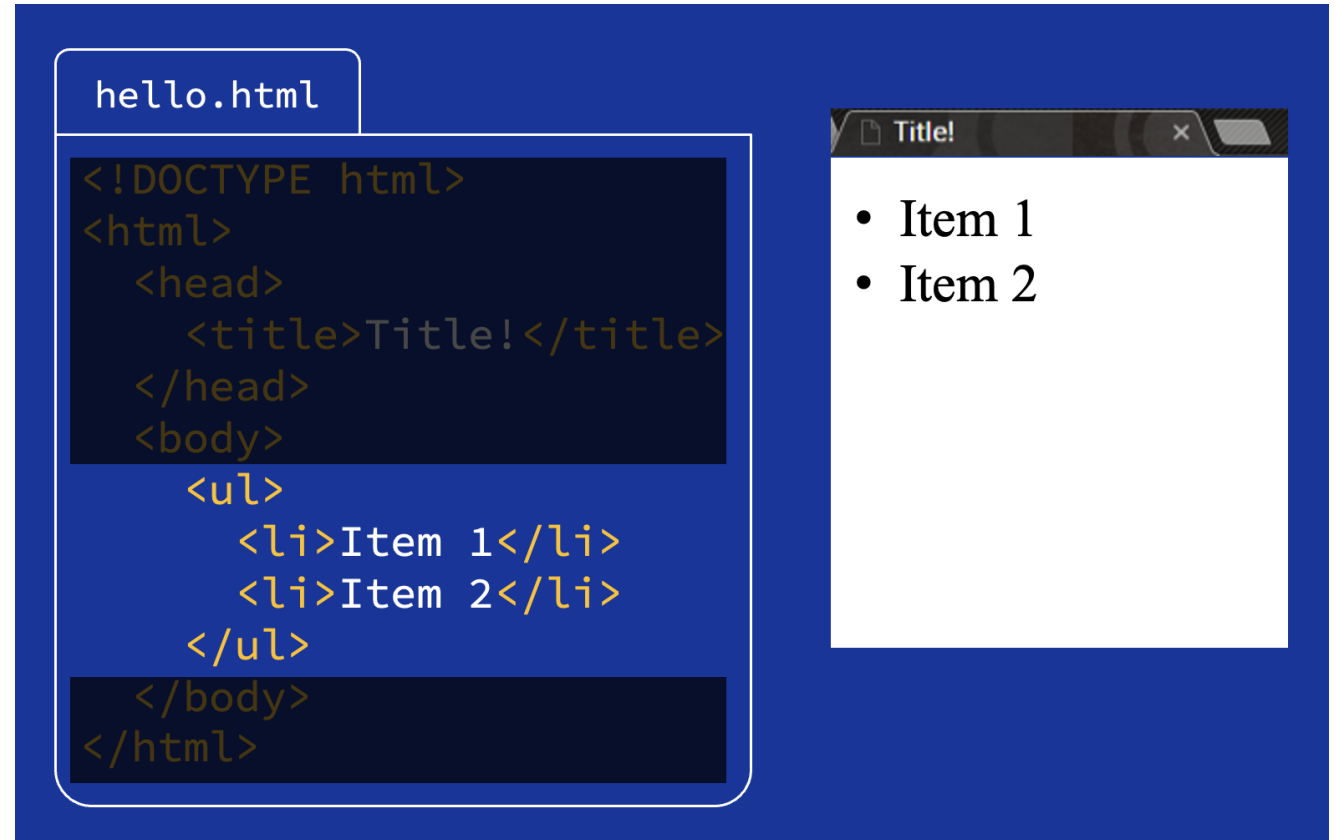
```

```



Example: Inserting Lists

- `` Ordered List (1, 2, 3...)
- `` Unordered List (bullets)
- `` List Item



div & span

`<div>` 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>
```

Div 1!

Paragraph

Div 2!

Paragraph!!

This text is inside a span element. This text is inside another span element.

so many elements...

HTML elements reference

<https://developer.mozilla.org/en-US/docs/Web/HTML/Element>

HTML 5

HTML5 is a markup language used for structuring and presenting content on the **World Wide Web**. It is the fifth and last major HTML version that is a **World Wide Web Consortium (W3C)** recommendation.

HTML5 is the next major revision of the HTML standard superseding HTML 4.01, XHTML 1.0, and XHTML 1.1.

HTML5 standard is a cooperation between the **World Wide Web Consortium (W3C)** and the **Web Hypertext Application Technology Working Group (WHATWG)**.

HTML 5 New Features

- **New Semantic Elements** – These are like <header>, <footer>, and <section>.
- **Forms 2.0** – Improvements to HTML web forms where new attributes have been introduced for <input> tag.
- **Persistent Local Storage** – To achieve persistence locally without resorting to third-party plugins.
- **WebSocket** – A next-generation bidirectional communication technology for web applications.
- **Server-Sent Events** – HTML5 introduces events which flow from web server to the web browsers and they are called Server-Sent Events (SSE).

HTML 5 New Features

- **Canvas** – This supports a two-dimensional drawing surface (bitmap) that you can program (manipulate) with JavaScript.
- **Audio & Video** – You can embed audio or video on your webpages without resorting to third-party plugins.
- **Geolocation** – Now visitors can choose to share their physical location with your web application.
- **Microdata** – This lets you create your own vocabularies beyond HTML5 and extend your web pages with custom semantics.
- **Drag and drop** – Drag and drop the items from one location to another location on the same webpage.

HTML5 Semantic Elements

- **<section>** – This tag represents a generic document or application section. It can be used together with h1-h6 to indicate the document structure.
- **<article>** – This tag represents an independent piece of content of a document, such as a blog entry or newspaper article.
- **<aside>** – This tag represents a piece of content that is only slightly related to the rest of the page.
- **<header>** – This tag represents the header of a section.

HTML5 Semantic Elements

- **<footer>** – This tag represents a footer for a section and can contain information about the author, copyright information, et cetera.
- **<nav>** – This tag represents a section of the document intended for navigation.
- **<dialog>** – This tag can be used to mark up a conversation.
- **<figure>** – This tag can be used to associate a caption together with some embedded content, such as a graphic or video.

HTML5 Semantic Tags

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset = "utf-8">
    <title>...</title>
  </head>
  <body>
    <header>...</header>
    <nav>...</nav>
    <article>
      <section>
        ...
      </section>
    </article>
    <aside>...</aside>
    <footer>...</footer>
  </body>
</html>
```

Note: <head> primarily holds metadata information for machine processing, not human-readability. For human-visible information, like top-level headings and listed authors, see the <header> element.

Web Forms 2.0

Web Forms 2.0 is an extension to the forms features found in HTML4. Form elements and attributes in HTML5 provide a greater degree of semantic mark-up than HTML4 and free us from a great deal of tedious scripting and styling that was required in HTML4.

The <input> types element in HTML5

- **datetime** - a date and time (year, month, day, hour, minute, second, fractions of a second) encoded according to ISO 8601 with the time zone set to UTC
- **datetime-local** - a date and time with no time zone information
- **date** - a date (year, month, day)
- **month** - a date consisting of a year and a month
- **week** - a date consisting of a year and a week number
- **time** - a time (hour, minute, seconds, fractional seconds)

The <input> types element in HTML5

- **number** - accepts only numerical value. The step attribute specifies the precision, defaulting to 1
- **range** - used for input fields that should contain a value from a range of numbers
- **email** - It accepts only email value. This type is used for input fields that should contain an e-mail address.
- **url** - It accepts only URL value. This type is used for input fields that should contain a URL address.

Web Storage

HTML5's **web storage** - feature to store some information locally on the user's computer, similar to cookies, but it is faster and much better than cookies.

HTML5 introduces two mechanisms, similar to HTTP session cookies, for storing structured data on the client side:

- **Session Storage** - designed for scenarios where the user is carrying out a single transaction, but could be carrying out multiple transactions in different windows at the same time.
- **Local Storage** - designed for storage that spans multiple windows, and lasts beyond the current session

Local Storage

```
1 <script>
2 // Check if the localStorage object exists
3 if(localStorage) {
4     // Store data
5     localStorage.setItem("first_name", "Peter");
6
7     // Retrieve data
8     alert("Hi, " + localStorage.getItem("first_name"));
9 } else {
10     alert("Sorry, your browser do not support local storage.");
11 }
12 </script>
```

- **localStorage.setItem(key, value)** stores the value associated with a key.
- **localStorage.getItem(key)** retrieves the value associated with the key.

WebSockets

WebSockets is a next-generation bidirectional communication technology for web applications which operates over a single socket and is exposed via a JavaScript interface in HTML 5 compliant browsers.

WebSockets send data from browser to server by calling a **send()** method, and receive data from server to browser by an **onmessage** event handler.

WebSocket Events

Event	Event Handler	Description
open	Socket.onopen	This event occurs when socket connection is established.
message	Socket.onmessage	This event occurs when client receives data from server.
error	Socket.onerror	This event occurs when there is any error in communication.
close	Socket.onclose	This event occurs when connection is closed.

WebSockets

```
1  let socket = new WebSocket("wss://javascript.info/article/websocket/chat/ws");
2
3  // send message from the form
4  document.forms.publish.onsubmit = function() {
5      let outgoingMessage = this.message.value;
6
7      socket.send(outgoingMessage);
8      return false;
9  };
10
11 // message received - show the message in div#messages
12 socket.onmessage = function(event) {
13     let message = event.data;
14
15     let messageElem = document.createElement('div');
16     messageElem.textContent = message;
17     document.getElementById('messages').prepend(messageElem);
18 }
```

Server-Sent Events

- Using **Server-Sent Events (SSE)** you can push DOM events continuously from your web server to the visitor's browser.
- The event streaming approach opens a persistent connection to the server, sending data to the client when new information is available, eliminating the need for continuous polling.
- It allows a web page to hold an open connection to a web server so that the web server can send a new response automatically at any time, there's no need to reconnect, and run the same server script from scratch over and over again.

Server-Sent Events

```
1  <?php
2  header("Content-Type: text/event-stream");
3  header("Cache-Control: no-cache");
4
5  // Get the current time on server
6  $currentTime = date("h:i:s", time());
7
8  // Send it in a message
9  echo "data: " . $currentTime . "\n\n";
10 flush();
11 ?>
```

Server-Sent Events

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4  <title>Using Server-Sent Events</title>
5  <script>
6      window.onload = function() {
7          var source = new EventSource("server_time.php");
8          source.onmessage = function(event) {
9              document.getElementById("result").innerHTML += "New time received from
web server: " + event.data + "<br>";
10         };
11     };
12 </script>
13 </head>
14 <body>
15     <div id="result">
16         <!--Server response will be inserted here-->
17     </div>
18 </body>
19 </html>
```

Canvas

- HTML5 element `<canvas>` gives you an easy and powerful way to draw graphics using JavaScript. It can be used to draw graphs, make photo compositions or do simple (and not so simple) animations.

Canvas

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4 <meta charset="utf-8">
5 <title>Drawing a Rectangle on the Canvas</title>
6 <script>
7     window.onload = function() {
8         var canvas = document.getElementById("myCanvas");
9         var context = canvas.getContext("2d");
10        context.rect(50, 50, 200, 100);
11        context.stroke();
12    };
13 </script>
14 </head>
15 <body>
16     <canvas id="myCanvas" width="300" height="200"></canvas>
17 </body>
18 </html>
```

SVG

The Scalable Vector Graphics (SVG) is an XML-based image format that is used to define two-dimensional vector based graphics for the web. Unlike raster image (e.g. .jpg, .gif, .png, etc.), a vector image can be scaled up or down to any extent without losing the image quality.

SVG

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="utf-8">
5      <title>Embedding SVG in HTML</title>
6  </head>
7  <body>
8      <svg width="300" height="200">
9          <text x="10" y="20" style="font-size:14px;">
10             Your browser support SVG.
11          </text>
12          Sorry, your browser does not support SVG.
13      </svg>
14  </body>
15 </html>
```

Differences Canvas vs SVG

Canvas	SVG
Resolution dependent	Resolution independent
No support for event handlers	Support for event handlers
Poor text rendering capabilities	Best suited for applications with large rendering areas (Google Maps)
You can save the resulting image as .png or .jpg	Slow rendering if complex (anything that uses the DOM a lot will be slow)
Well suited for graphic-intensive games	Not suited for game applications

Audio & Video

The HTML5 <audio> and <video> tags make it simple to add media to a website. You need to set src attribute to identify the media source and include a controls attribute so the user can play and pause the media.

```
1 <video controls="controls">
2   <source src="media/shuttle.mp4" type="video/mp4">
3   <source src="media/shuttle.ogv" type="video/ogg">
4   Your browser does not support the HTML5 Video element.
5 </video>
```

Geolocation

The HTML5 **geolocation** feature lets you find out the geographic coordinates (latitude and longitude numbers) of the current location of your website's visitor.

It utilizes the three methods that are packed into the navigator.geolocation object – **getCurrentPosition()**, **watchPosition()** and **clearWatch()**.

Properties of position object

Property	Type	Notes
<code>coords.latitude</code>	<code>double</code>	Decimal degrees
<code>coords.longitude</code>	<code>double</code>	Decimal degrees
<code>coords.altitude</code>	<code>double</code> or <code>null</code>	Meters above the reference ellipsoid
<code>coords.accuracy</code>	<code>double</code>	Meters
<code>coords.altitudeAccuracy</code>	<code>double</code> or <code>null</code>	Meters
<code>coords.heading</code>	<code>double</code> or <code>null</code>	Degrees clockwise from true north
<code>coords.speed</code>	<code>double</code> or <code>null</code>	Meters/second
<code>timestamp</code>	<code>DOMTimeStamp</code>	Like a <code>Date ()</code> object

Geolocation

```
<script>
var x = document.getElementById("demo");
function getLocation() {
    if (navigator.geolocation) {
        navigator.geolocation.getCurrentPosition(showPosition);
    } else {
        x.innerHTML = "Geolocation is not supported by this browser.";
    }
}

function showPosition(position) {
    x.innerHTML = "Latitude: " + position.coords.latitude +
        "<br>Longitude: " + position.coords.longitude;
}
</script>
```


Microdata

- **Microdata** is a standardized way to provide additional semantics in your web pages.
- **Microdata** lets you define your own customized elements and start embedding **custom properties** in your web pages. At a high level, microdata consists of a **group of name-value pairs**.
- The groups are called **items**, and each name-value pair is a property. Items and properties are represented by regular elements.
- If a browser supports the HTML5 **microdata API**, there will be a **getItems()** function on the global **document** object.

Microdata Global Attributes

Attribute	Description
<i>itemscope</i>	Defines the scope of microdata item.
<i>itemprop</i>	Defines the name/value pairs of the microdata.
<i>itemtype</i>	A URL to define the vocabulary used for encoding the microdata.
<i>itemid</i>	To set an unique identifier for microdata item.
<i>itemref</i>	To include itemprop attributes outside the itemscope attribute.

Microdata

```
<div itemscope itemtype="http://schema.org/SoftwareApplication">
  <span itemprop="name">Angry Birds</span> -

  REQUIRES <span itemprop="operatingSystem">ANDROID</span><br>
  <link itemprop="applicationCategory" href="http://schema.org/GameApplication"/>

  <div itemprop="aggregateRating" itemscope itemtype="http://schema.org/AggregateRating">
    RATING:
    <span itemprop="ratingValue">4.6</span> (
    <span itemprop="ratingCount">8864</span> ratings )
  </div>

  <div itemprop="offers" itemscope itemtype="http://schema.org/Offer">
    Price: $<span itemprop="price">1.00</span>
    <meta itemprop="priceCurrency" content="USD" />
  </div>
</div>
```

Drag & drop

- **Drag and Drop** (DnD) is powerful User Interface concept which makes it easy to copy, reorder and deletion of items with the help of mouse clicks.
- In HTML5, drag and drop is part of the standard: Any element can be draggable

Drag & drop

Event	Description
ondragstart	Fires when the user starts dragging an element.
ondragenter	Fires when a draggable element is first moved into a drop listener.
ondragover	Fires when the user drags an element over a drop listener.
ondragleave	Fires when the user drags an element out of drop listener.
ondrag	Fires when the user drags an element anywhere; fires constantly but can give X and Y coordinates of the mouse cursor.
ondrop	Fires when the user drops an element into a drop listener successfully.
ondragend	Fires when the drag action is complete, whether it was successful or not. This event is not fired when dragging a file to the browser from the desktop.

Web Worker

A web worker is a JavaScript that runs in the background, independently of other scripts, without affecting the performance of the page

It continues to do whatever you want: clicking, selecting things, etc., while the web worker runs in the background.

Offline Support for Web App

Use **Service Worker** to fetch contents/files and store data in **local storage**

A **Service Worker** is a script that executes in the background, in a separate thread from the browser UI. Service worker cache makes it possible for a web site to function offline. They are the technical powerhouse that levels up a website to a progressive web app. They enable deep platform integration, like rich caching, push notifications and background sync.

A **Service worker** sits between the browser and the network, acting like a proxy server, handling a collection of non-UI centric tasks. They are event driven and live outside the browser process, enabling them to work without an active browser session.

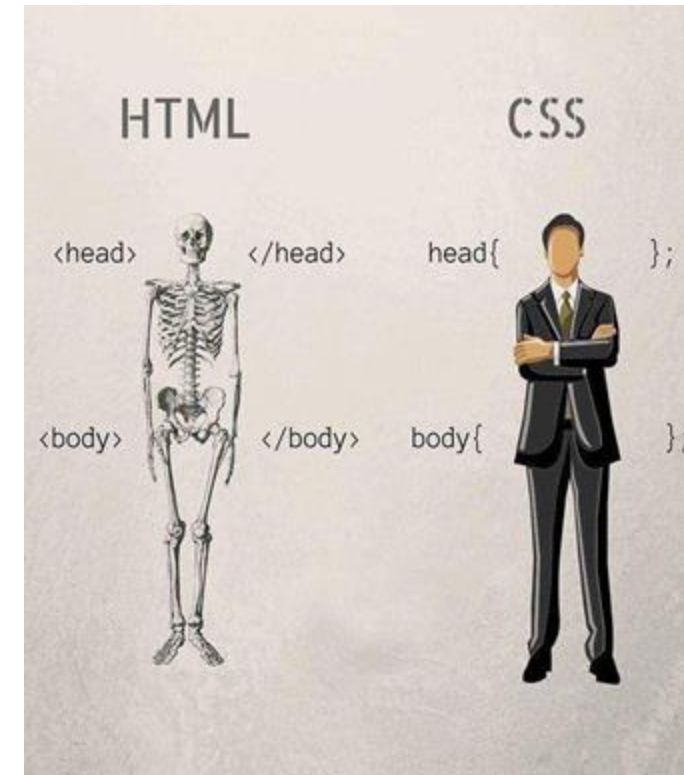
Definition

CSS

Cascading Style Sheets

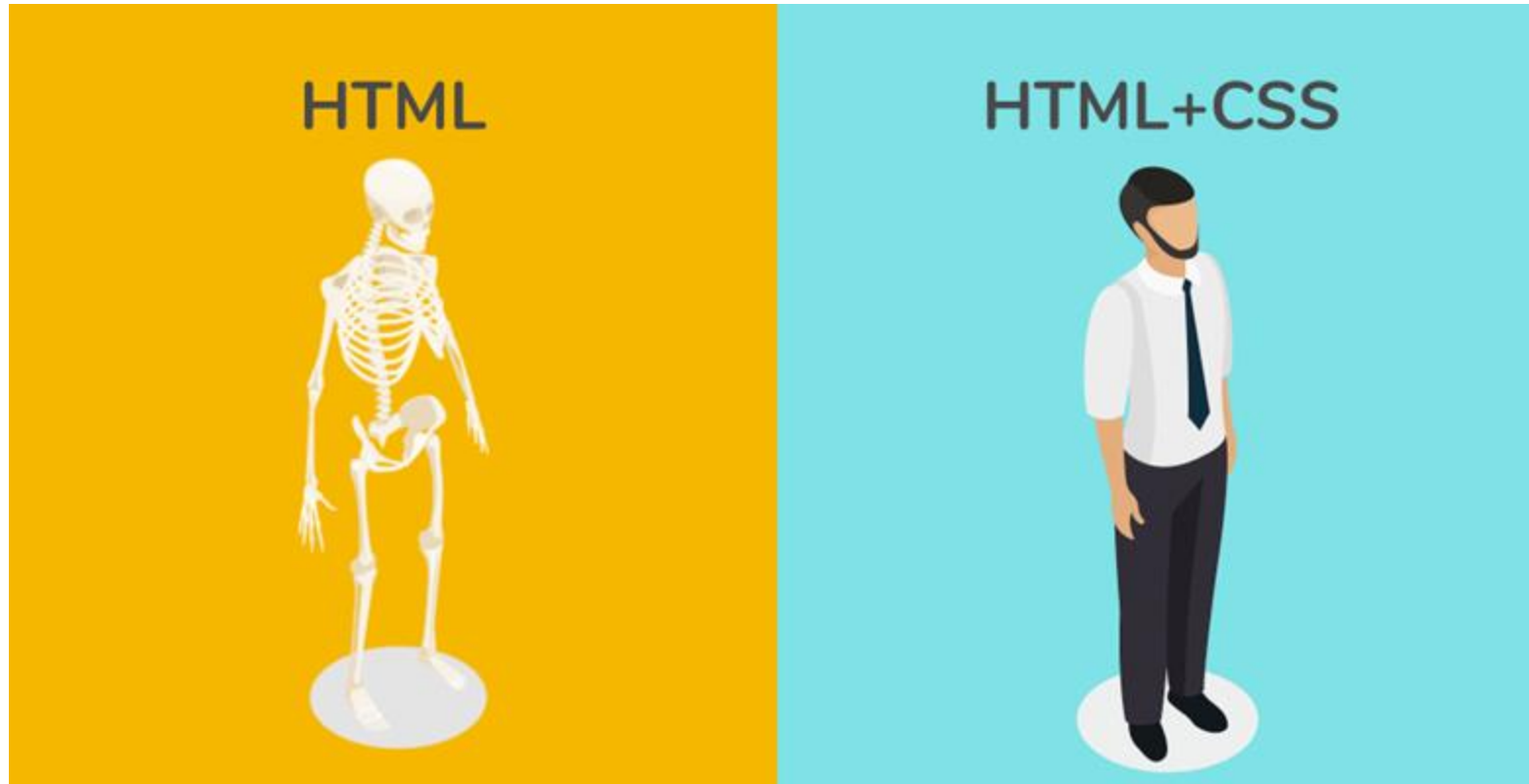
The **rules** that tell your **web browser** **how stuff looks**

HTML & CSS Analogy



Source: weblab.mit.edu

CSS = A list of descriptions

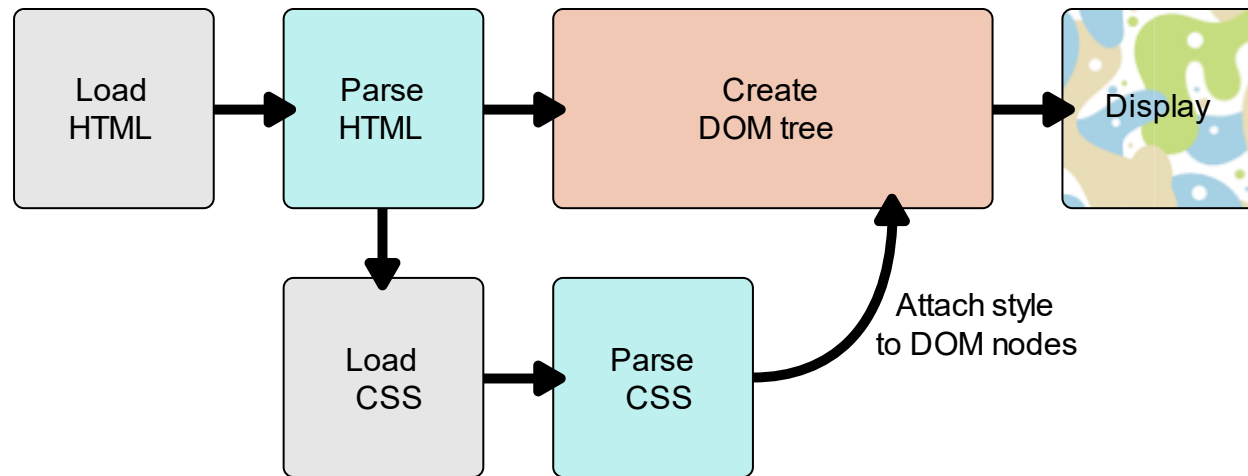


making things look pretty

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>
```

```
P  
├ "Let's use:"  
├ SPAN  
│   └ "Cascading"  
├ SPAN  
│   └ "Style"  
└ SPAN  
    └ "Sheets"
```

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;  
}
```

Hello, I have blue text

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

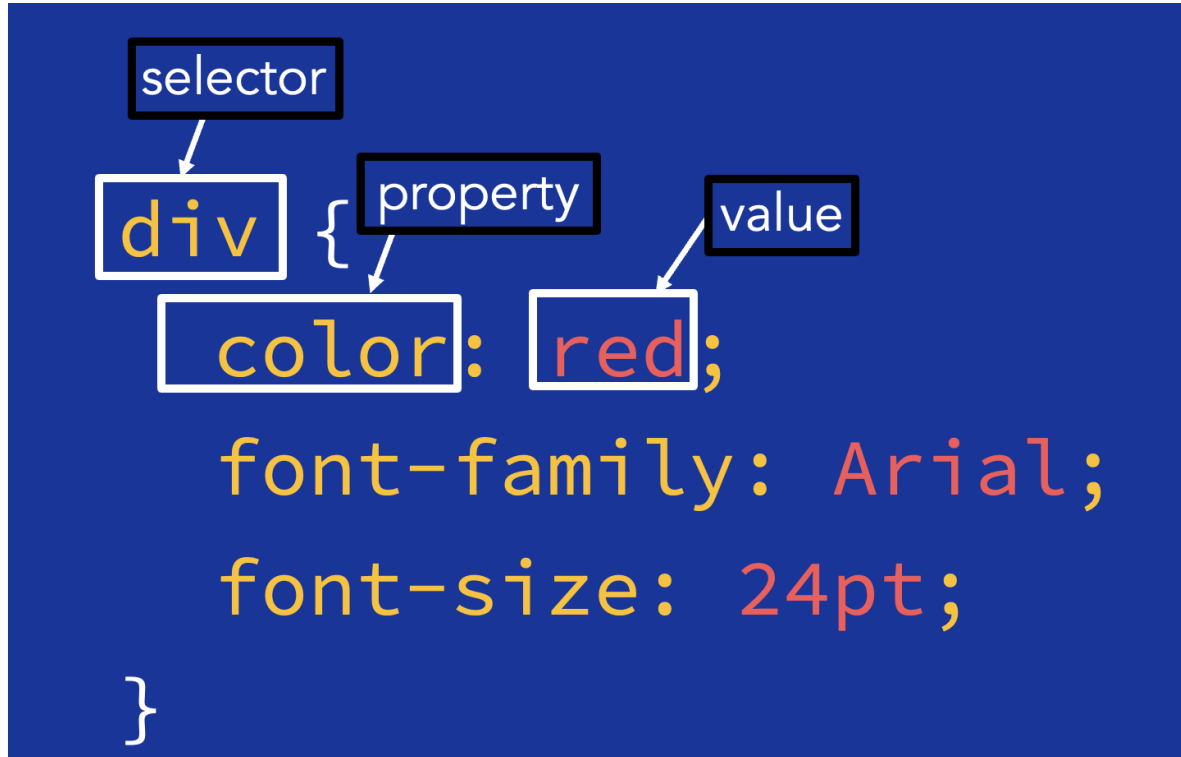
2.Specificity: an algorithm which determines which CSS selector has the strongest match

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 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;
}
```

Title!

Heading

Paragraph!

Info

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>
```

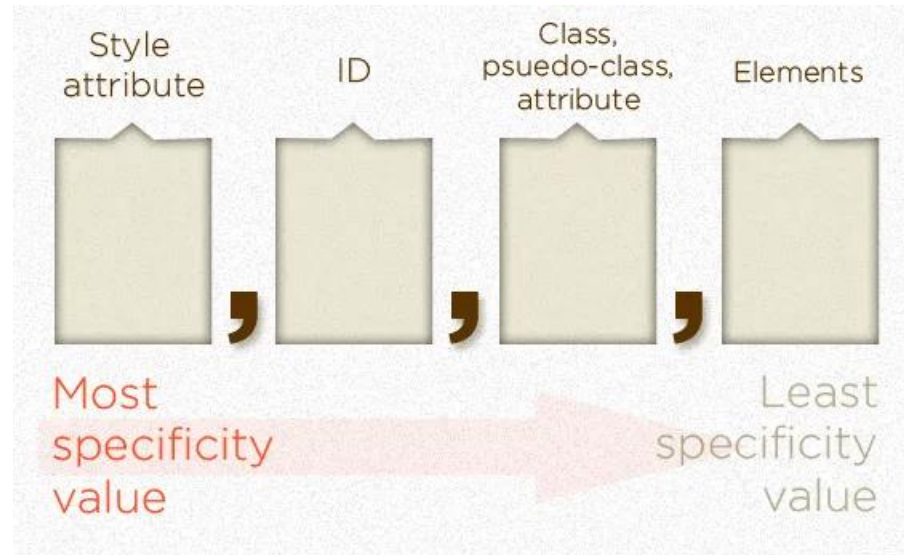
Example 2:

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

<p class = "test">Hello World!</p>

</body>
</html>
```

CSS Specificity (2)

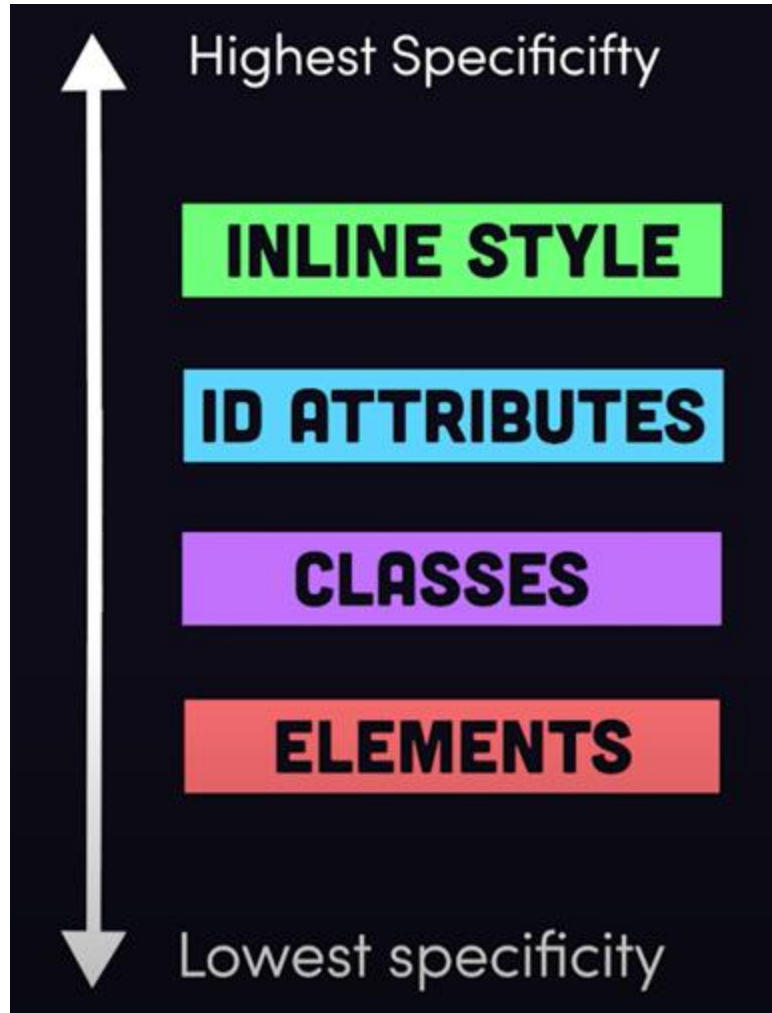


Start at 0, add 100 for each ID value, add 10 for each class value (or pseudo-class or attribute selector), add 1 for each element selector or pseudo-element.

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

<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


.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**)



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

subchild 1

child 2

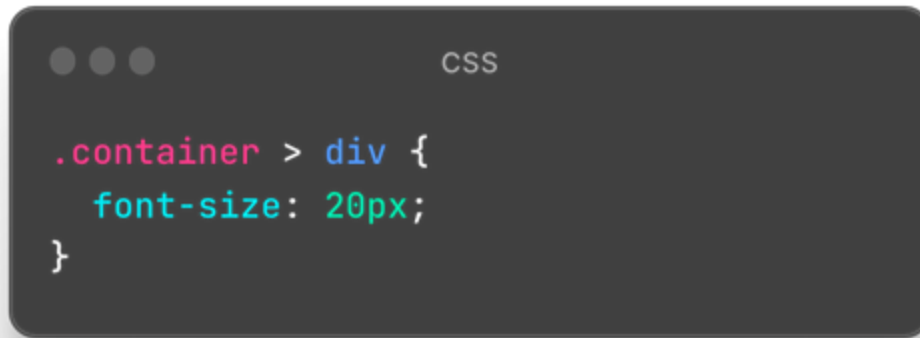
child 3

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



```
.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

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

subchild 1

child 2

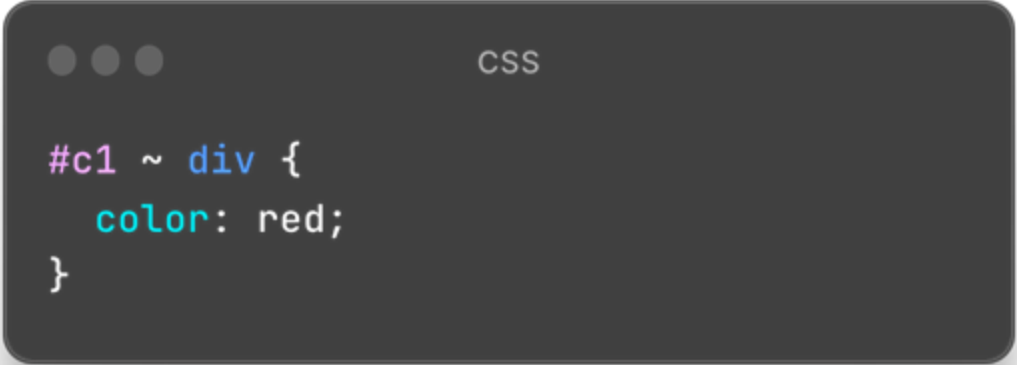
child 3

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



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

subchild 1

child 2

child 3

child 4

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:

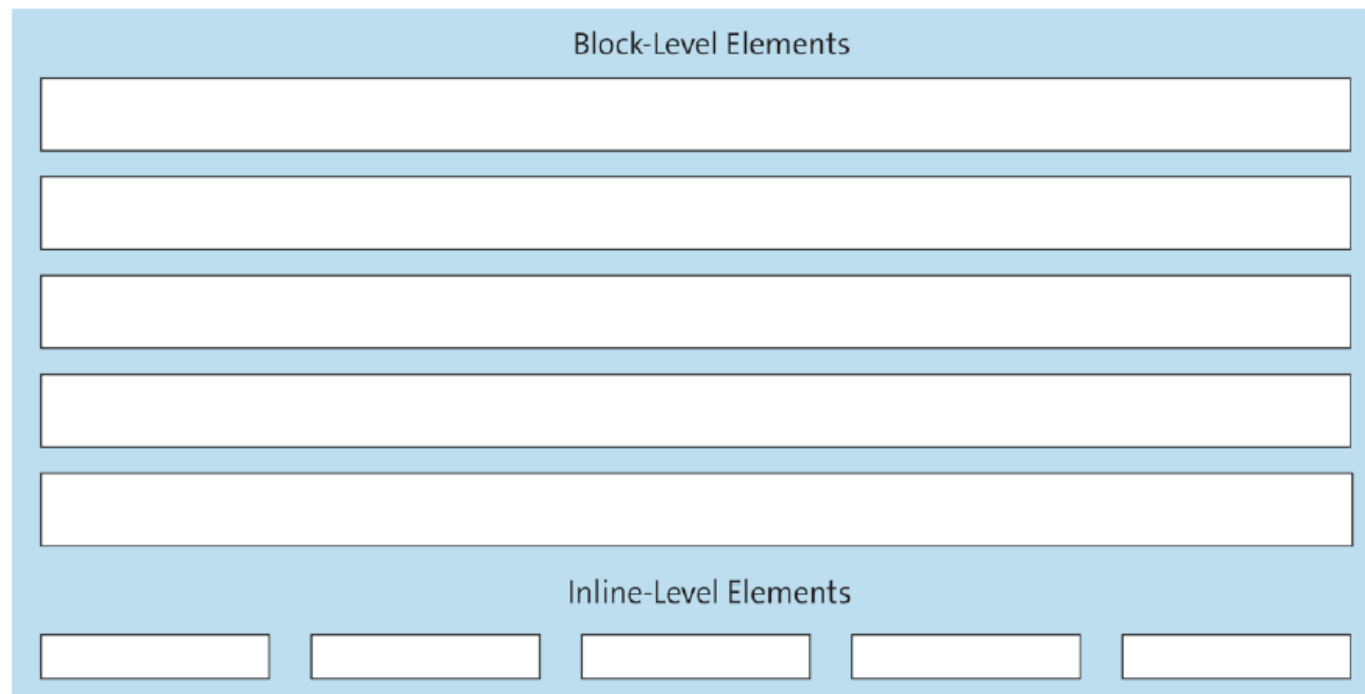
- block
- flex

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.

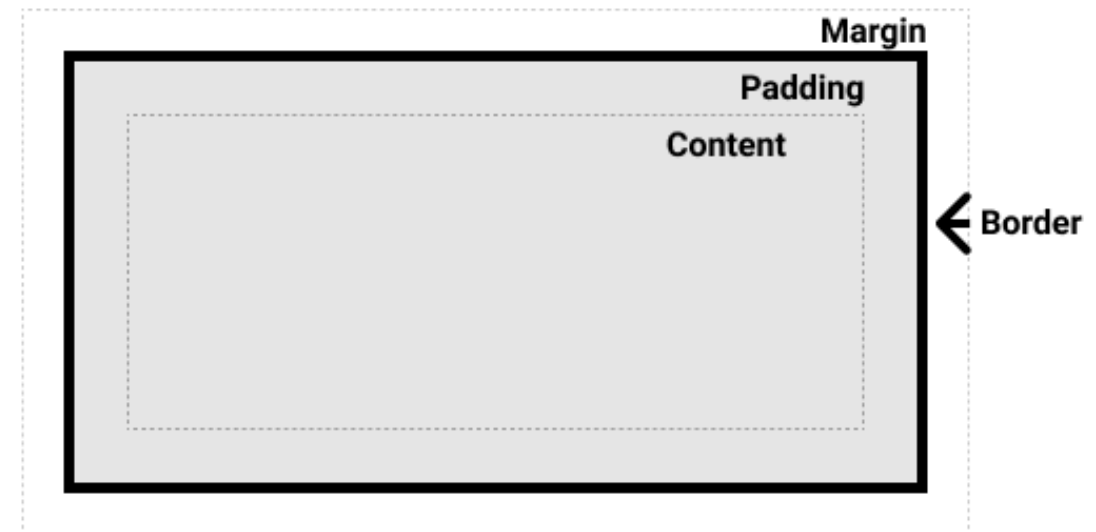


Css box model

- 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

<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

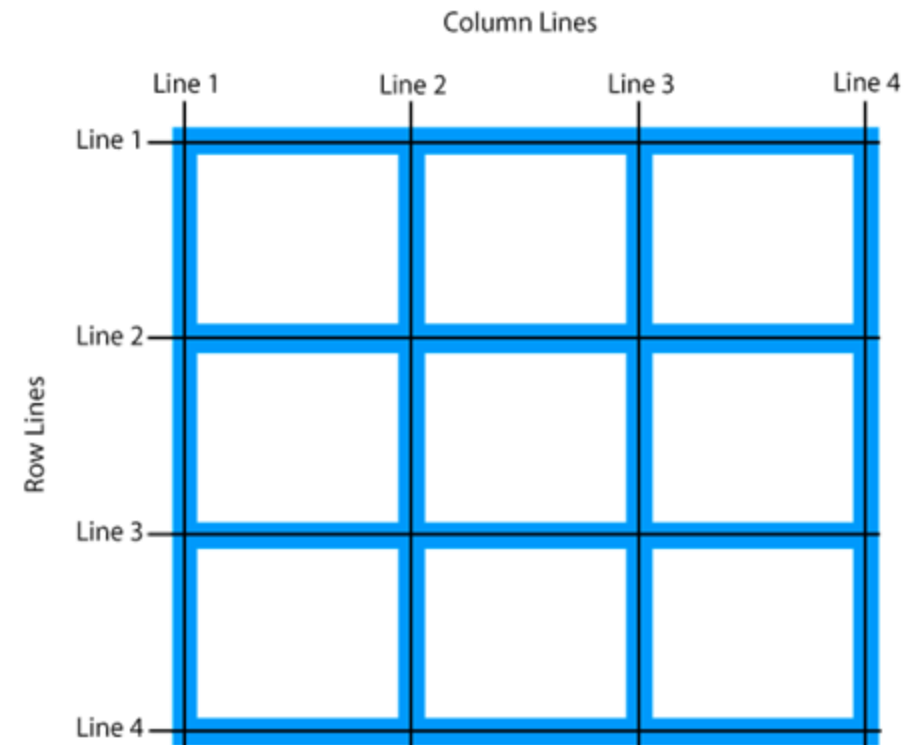
.container {

}
```

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;
}
```

child 1
child 2
child 3
child 4

```
CSS

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

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:

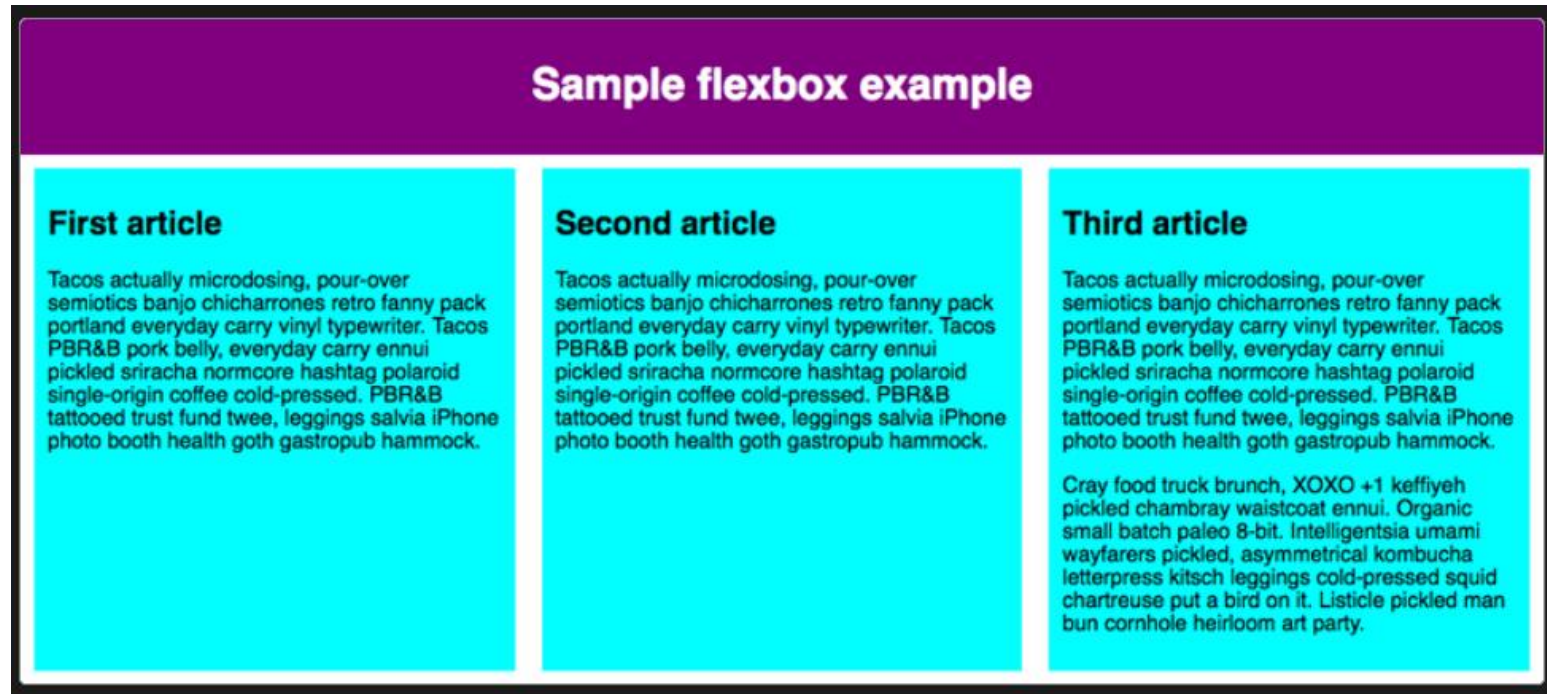
- Vertically center a block of content inside its parent.
- Make all the children of a container take up an equal amount of the available width/height, regardless of how much width/height is available.
- Make all columns in a multiple-column layout adopt the same height even if they contain a different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

Example : flexbox

```
section {
```

```
display: flex;
```

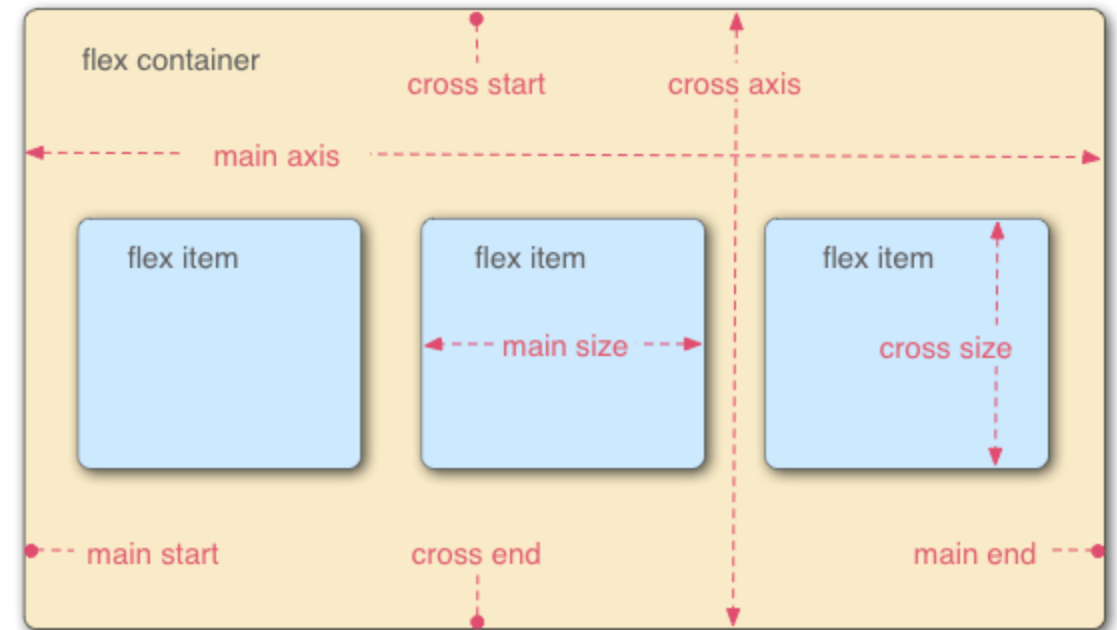
```
}
```



flex model

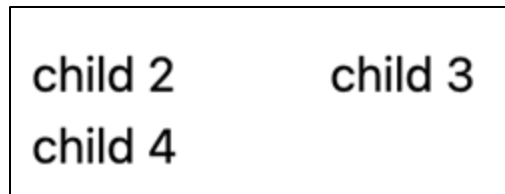
When elements are laid out as flex items, they are laid out along two axes:

- 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.

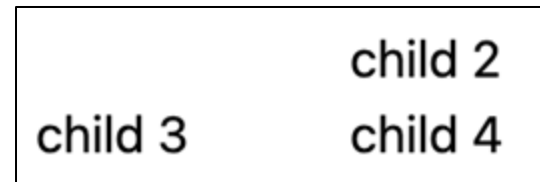


display: none

- 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



```
● ● ● CSS
#c1 {
  display: none;
}
```



```
● ● ● 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;
}
```

the
quick
brown
fox
jumps
over
the
lazy
dog

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;
}
```

the
quick

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. Desain Responsif
4. Kreativitas dalam Penggunaan Warna dan Tipografi

Bonus:

3. Desain yang Interaktif (without JavaScript, only HTML+CSS)
5. Aksesibilitas Web
6. Optimisasi Kinerja

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.

Exercise: Penjelasan Kriteria Lanjutan

Struktur HTML yang Semantik:

- Buatlah struktur halaman yang sesuai dengan standar HTML5, menggunakan elemen-elemen semantik seperti `<header>`, `<nav>`, `<section>`, `<article>`, `<aside>`, dan `<footer>`.
- Jelaskan alasan penggunaan elemen-elemen tersebut dalam konteks portfolio Anda (misalnya, mengapa Anda menggunakan `<section>` pada bagian tertentu dan bukan `<div>` biasa?).

Desain Responsif:

- Gunakan media queries untuk membuat halaman web yang responsif terhadap berbagai ukuran layar (desktop, tablet, dan smartphone).
- Mahasiswa harus menguji dan mempresentasikan hasil tampilan dari halaman yang telah mereka buat di tiga ukuran layar berbeda.
- Tantangan: Pastikan layout tetap menarik dan fungsional pada semua ukuran layar, serta mempertimbangkan kenyamanan pengguna, misalnya: 2 kolom pada desktop dan 1 kolom pada mobile.

Kreativitas dalam Penggunaan Warna dan Tipografi:

- Pilih kombinasi warna dan tipografi yang sesuai dengan tema portfolio pribadi Anda.
- Tantangan: Buatlah skema warna yang ramah mata pengguna dan selaras dengan estetika keseluruhan. Jelaskan bagaimana pilihan warna dan tipografi Anda mendukung citra pribadi yang ingin Anda tampilkan melalui portfolio ini.