## Applying CSS to HTML

let's examine three methods of applying CSS to a document:

* External Stylesheet

<link rel="stylesheet" href="styles/style.css">

* Internal Stylesheet

<style>

h1 {

color: blue;

background-color: yellow;

border: 1px solid black;

}

p {

color: red;

}

</style>

* Inline Stylesheet

Inline styles are CSS declarations that affect a single HTML element, contained within a style attribute

<h1 style="color: blue;background-color: yellow;border: 1px solid black;">Hello World!</h1>

Functions

A function consists of the function name, and brackets to enclose the values for the function

Example would be the various values for [transform](https://developer.mozilla.org/en-US/docs/Web/CSS/transform), such as rotate().

<div class="box"></div>

.box {

margin: 30px;

width: 100px;

height: 100px;

background-color: rebeccapurple;

transform: rotate(0.8turn)

}

## @rules

CSS [@rules](https://developer.mozilla.org/en-US/docs/Web/CSS/At-rule) (pronounced "at-rules") provide instruction for what CSS should perform or how it should behave. Some @rules are simple with just a keyword and a value. For example,  @import imports a stylesheet into another CSS stylesheet:

@import 'styles2.css';

One common @rule that you are likely to encounter is @media, which is used to create [media queries](https://developer.mozilla.org/en-US/docs/Web/CSS/Media_Queries). Media queries use conditional logic for applying CSS styling.

In the example below, the stylesheet defines a default pink background for the <body> element. However, a media query follows that defines a blue background if the browser viewport is wider than 30em.

body {

background-color: pink;

}

@media (min-width: 300px) {

body {

background-color: blue;

}

}

## Shorthands

Some properties like [font](https://developer.mozilla.org/en-US/docs/Web/CSS/font), [background](https://developer.mozilla.org/en-US/docs/Web/CSS/background), [padding](https://developer.mozilla.org/en-US/docs/Web/CSS/padding), [border](https://developer.mozilla.org/en-US/docs/Web/CSS/border), and [margin](https://developer.mozilla.org/en-US/docs/Web/CSS/margin) are called **shorthand properties.** This is because shorthand properties set several values in a single line.

For example, this one line of code:

padding: 10px 15px 15px 5px; // top right bottom left

In 4-value shorthands like padding and margin, the values are applied

in the order top, right, bottom, left (clockwise from the top). There are also other

shorthand types, for example 2-value shorthands, which set padding/margin

for top/bottom, then left/right

is equivalent to these four lines of code:

padding-top: 10px;

padding-right: 15px;

padding-bottom: 15px;

padding-left: 5px;

This One line css –

background: red url(bg-graphic.png) 10px 10px repeat-x fixed;

is equivalent to these five lines::

background-color: red;

background-image: url(bg-graphic.png);

background-position: 10px 10px;

background-repeat: repeat-x;

background-attachment: fixed;

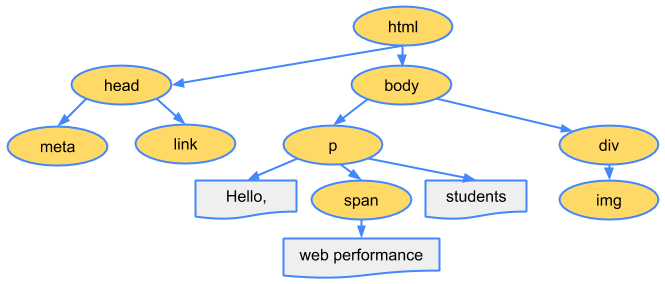
## How does CSS actually work?

When a browser displays a document, it must combine the document's content with its style information. It processes the document in a number of stages, which we've listed below.

1. The browser loads the HTML (e.g. receives it from the network).
2. It converts the [HTML](https://developer.mozilla.org/en-US/docs/Glossary/HTML) into a [DOM](https://developer.mozilla.org/en-US/docs/Glossary/DOM) (Document Object Model). The DOM represents the document in the computer's memory.
3. The browser then fetches most of the resources that are linked to by the HTML document, such as embedded images and videos ... and linked CSS! JavaScript is handled a bit later on in the process
4. The browser parses the fetched CSS, and sorts the different rules by their selector types into different "buckets", e.g. element, class, ID, and so on. Based on the selectors it finds, it works out which rules should be applied to which nodes in the DOM, and attaches style to them as required
5. The render tree is laid out in the structure it should appear in after the rules have been applied to it
6. The visual display of the page is shown on the screen (this stage is called painting).

**DOM construction:** Finally, because the HTML markup defines relationships between different tags (some tags are contained within other tags) the created objects are linked in a tree data structure that also captures the parent-child relationships defined in the original markup: the *HTML* object is a parent of the *body* object, the *body* is a parent of the *paragraph* object, and so on.

<!DOCTYPE html>  
<html>  
  <head>  
    <meta name="viewport" content="width=device-width,initial-scale=1">  
    <link href="style.css" rel="stylesheet">  
    <title>Critical Path</title>  
  </head>  
  <body>  
    <p>Hello <span>web performance</span> students!</p>  
    <div><img src="awesome-photo.jpg"></div>  
  </body>  
</html>



**The final output of this entire process is the Document Object Model (DOM) of our simple page, which the browser uses for all further processing of the page.**

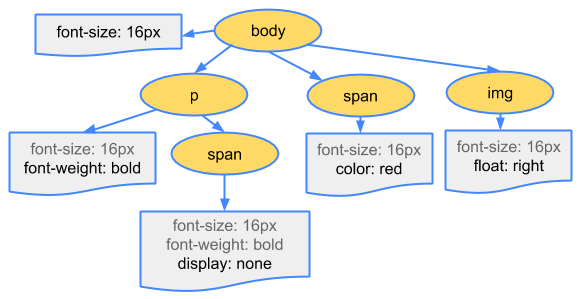
The DOM tree captures the properties and relationships of the document markup, but it doesn't tell us how the element will look when rendered. That’s the responsibility of the CSSOM.

## CSS Object Model (CSSOM)

While the browser was constructing the DOM of our simple page, it encountered a link tag in the head section of the document referencing an external CSS stylesheet: style.css. Anticipating that it needs this resource to render the page, it immediately dispatches a request for this resource, which comes back with the following content:

body { font-size: 16px }  
p { font-weight: bold }  
span { color: red }  
p span { display: none }  
img { float: right }

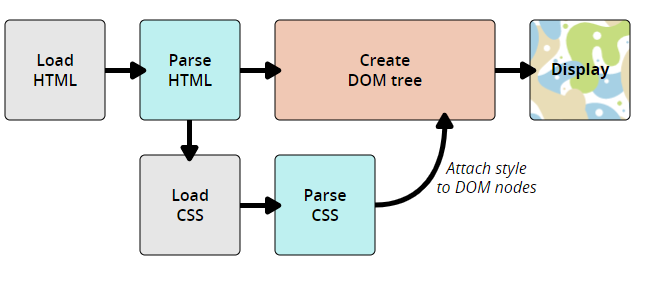
The CSS bytes are converted into characters, then tokens, then nodes, and finally they are linked into a tree structure known as the "CSS Object Model" (CSSOM):



Why does the CSSOM have a tree structure? When computing the final set of styles for any object on the page, the browser starts with the most general rule applicable to that node (for example, if it is a child of a body element, then all body styles apply) and then recursively refines the computed styles by applying more specific rules; that is, the rules "cascade down."

The CSSOM and DOM trees are combined into a render tree, which is then used to compute the layout of each visible element and serves as an input to the paint process that renders the pixels to screen. Optimizing each of these steps is critical to achieving optimal rendering performance.

* The DOM and CSSOM trees are combined to form the render tree.
* Render tree contains only the nodes required to render the page.
* Layout computes the exact position and size of each object.
* The last step is paint, which takes in the final render tree and renders the pixels to the screen.

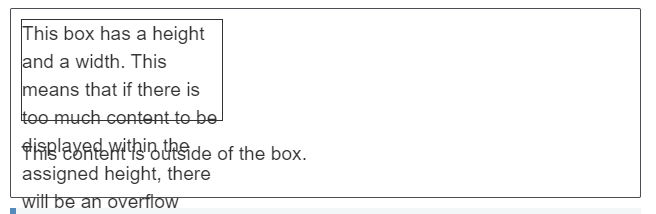


## What is overflow?

Everything in CSS is a box. You can constrain the size of these boxes by assigning values of [width](https://developer.mozilla.org/en-US/docs/Web/CSS/width) and [height](https://developer.mozilla.org/en-US/docs/Web/CSS/height) (or [inline-size](https://developer.mozilla.org/en-US/docs/Web/CSS/inline-size) and [block-size](https://developer.mozilla.org/en-US/docs/Web/CSS/block-size)). **Overflow happens when there is too much content to fit in a box.** CSS provides various tools to manage overflow. As you go further with CSS layout and writing CSS, you will encounter more overflow situations.

Css Tries to avoid the data loss

Ex-1The first example is a box that has been restricted by setting a height. Then we add content that exceeds the allocated space. The content overflows the box and falls into the paragraph below.



.box {

border: 1px solid #333333;

width: 200px;

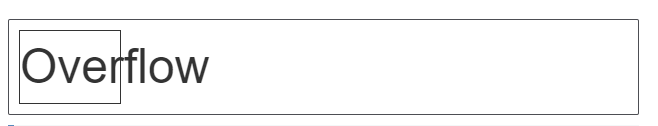
height: 100px;

}

<div class="box">This box has a height and a width. This means that if there is too much content to be displayed within the assigned height, there will be an overflow situation. If overflow is set to hidden then any overflow will not be visible.</div>

<p>This content is outside of the box.</p>

The second example is a word in a box. The box has been made too small for the word and so it breaks out of the box.



.word {

border: 1px solid #333333;

width: 100px;

font-size: 250%;

}

<div class="word">Overflow</div>

The  [overflow](https://developer.mozilla.org/en-US/docs/Web/CSS/overflow) property is how you take control of an element's overflow. It is the way you instruct the browser how it should behave. The default value of overflow is visible. With this default, we can see content when it overflows.

// overflow: hidden;

// overflow: scroll

In the example above, we only need to scroll on the y axis, however we get scrollbars in both axes. To just scroll on the y axis, you could use the [overflow-y](https://developer.mozilla.org/en-US/docs/Web/CSS/overflow-y) property, setting overflow-y: scroll.

If you only want scrollbars to appear when there is more content than can fit in the box, use overflow: auto. This allows the browser to determine if it should display scrollbars.

## Sizing Items in css

An empty [<div>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/div) however, has no size of its own. If you add a [<div>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/div) to your HTML with no content, then give it a border , you will see a line on the page. This is the collapsed border on the element — there is no content to hold it open

Take our <div> from the example above — we can give it specific [width](https://developer.mozilla.org/en-US/docs/Web/CSS/width) and [height](https://developer.mozilla.org/en-US/docs/Web/CSS/height)

.box {

border: 5px solid darkblue;

height: 150px;

width: 200px;

}

Using percentages

When using a percentage you need to be aware what it is a percentage *of*. In the case of a box inside another container, if you give the child box a percentage width it will be a percentage of the width of the parent container.

.box {

border: 5px solid darkblue;

width: 50%;

}

<div class="box">

I have a percentage width.

</div>

Percentage margins and padding

Min and max sizes

In addition to giving things a fixed size, we can ask CSS to give an element a minimum or a maximum size. If you have a box that might contain a variable amount of content, and you always want it to be *at least* a certain height, you could set the [min-height](https://developer.mozilla.org/en-US/docs/Web/CSS/min-height) property on it. The box will always be at least this height, but will then grow taller if there is more content than the box has space for at its minimum height.

.box {

border: 5px solid darkblue;

min-height: 150px;

width: 200px;

}

<div class="wrapper">

<div class="box"></div>

<div class="box">These boxes both have a min-height set, this box has content in it which will need more space than the assigned height, and so it grows from the minimum.</div>

</div>

## Sizing images

everything in CSS generates a box. If you place an image inside a box that is smaller or larger than the intrinsic dimensions of the image file in either direction, it will either appear smaller than the box, or overflow the box. You need to make a decision about what happens with the overflow

In the example below we have two boxes, both 200 pixels in size:

* One contains an image which is smaller than 200 pixels — it is smaller than the box and doesn't stretch to fill it.
* The other is larger than 200 pixels and overflows the box.

.box {

width: 200px;

}

img {

}

<div class="wrapper">

<div class="box"><img src="star.png" alt="star"></div>

<div class="box"><img src="balloons.jpg" alt="balloons"></div>

</div>

**Try adding max-width: 100% to the <img> element in the example above. You will see that the smaller image remains unchanged, but the larger one becomes smaller to fit into the box.**

The [object-fit](https://developer.mozilla.org/en-US/docs/Web/CSS/object-fit) property can help you here. When using object-fit the replaced element can be sized to fit a box in a variety of ways.

Below we have used the value cover, which sizes the image down, maintaining the aspect ratio so that it neatly fills the box. As the aspect ratio is maintained some parts of the image will be cropped by the box.

.box {

width: 200px;

height: 200px;

}

img {

height: 100%;

width: 100%;

}

.cover {

object-fit: cover;

}

<div class="wrapper">

<div class="box"><img src="balloons.jpg" alt="balloons" class="cover"></div>

<div class="box"><img src="balloons.jpg" alt="balloons" class="contain"></div>

</div>

If we use contain as a value the image will be scaled down until it is small enough to fit inside the box. This will result in "letterboxing" if it is not the same aspect ratio as the box.

## Form elements

Many form controls are added to your page by way of the [<input>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input) element — this defines simple form fields such as text inputs, through to more complex fields added in HTML5 such as color and date pickers. There are some additional elements, such as [<textarea>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/textarea) for multiline text input, and also elements used to contain and label parts of forms such as [<fieldset>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/fieldset) and [<legend>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/legend).

Styling text input elements

Elements that allow for text input, such as <input type="text">, specific types such as <input type="email">, and the <textarea> element are quite easy to style and tend to behave just like other boxes on your page. The default styling of these elements will differ however based on the operating system and browser that your user visits the site with.

input[type="text"],

input[type="email"] {

border: 2px solid #000;

margin: 0 0 1em 0;

padding: 10px;

width: 100%;

}

input[type="submit"] {

border: 3px solid #333;

background-color: #999;

border-radius: 5px;

padding: 10px 2em;

font-weight: bold;

color: #fff;

}

input[type="submit"]:hover {

background-color: #333;

}

<form>

<div><label for="name">Name</label>

<input type="text" id="name"></div>

<div><label for="email">Email</label>

<input type="email" id="email"></div>

<div class="buttons"><input type="submit" value="Submit"></div>

</form>

## A typical HTML table

Let's start by looking at a typical HTML table

<table>

<caption>A summary of the UK's most famous punk bands</caption>

<thead>

<tr>

<th scope="col">Band</th>

<th scope="col">Year formed</th>

<th scope="col">No. of Albums</th>

<th scope="col">Most famous song</th>

</tr>

</thead>

<tbody>

<tr>

<th scope="row">Buzzcocks</th>

<td>1976</td>

<td>9</td>

<td>Ever fallen in love (with someone you shouldn't've)</td>

</tr>

<tr>

<th scope="row">The Clash</th>

<td>1976</td>

<td>6</td>

<td>London Calling</td>

</tr>

<tr>

<th scope="row">The Stranglers</th>

<td>1974</td>

<td>17</td>

<td>No More Heroes</td>

</tr>

</tbody>

<tfoot>

<tr>

<th scope="row" colspan="2">Total albums</th>

<td colspan="2">77</td>

</tr>

</tfoot>

</table>

If we draw this, it wont look good so let’s apply some styling

Spacing and layout

table {

table-layout: fixed;

width: 100%;

border-collapse: collapse;

border: 3px solid purple;

}

thead th:nth-child(1) {

width: 30%;

}

thead th:nth-child(2) {

width: 20%;

}

thead th:nth-child(3) {

width: 15%;

}

thead th:nth-child(4) {

width: 35%;

}

th, td {

padding: 20px;

}

**Q-> How to extend Width of parent div to fit child div based on child div’s width**

    <div id="content">

      parent

      <div class="container-fluid">

          <div>

              <div>

                  <h1> Child: Some title here</h1>

              </div>

          </div>

      </div>

    </div>

#content {

    background: #ff0000;

    min-height: 200px;

}

.container-fluid {

    min-width: 800px;

    background: #ff0;

}

To achieve this behavior we can use different css property on parent div as below

width: fit-content;

or

display:inline-block;