



# HTML CSS Reference

By: [supersimple.dev](https://supersimple.dev)

Video: <https://youtu.be/G3e-cpL7ofc>

## HTML Basics

`<button>Hello</button>`

Creates a button with the text "Hello" inside.

`<p>paragraph of text</p>`

Creates a paragraph of text.

## HTML Syntax

Syntax = rules for writing HTML code (like grammar in English).

1. Elements should have an opening tag and a matching closing tag.

`<button>Hello</button>`



**Opening Tag**

(start of the button)



**Closing Tag**

(end of the button)

2. In HTML, extra spaces and newlines are combined into 1 space.

`<p>paragraph of text</p>`

`<p>paragraph of text</p>`

`<p>`

paragraph of text

`</p>`

All 3 examples above will show the same result on the web page.

## Attributes

Attributes modify how an HTML element behaves.

`<a href="https://youtube.com">`

Link to YouTube

`</a>`

`<a>` = link to another website.

`href` = modifies which website is opened when clicking this link.

```
<a href="https://youtube.com" target="_blank">
  Link to YouTube
</a>
```

`target="_blank"` causes the link to be opened in a new tab.

## CSS Basics

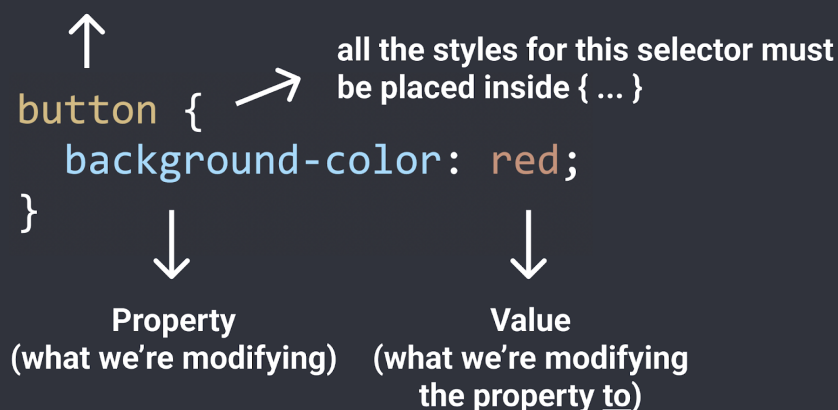
One way of writing CSS code is using the `<style>` HTML element.

```
<style>
  button {
    background-color: red;
    color: white;
  }
</style>
```

Modifies all `<button>`s on the page.  
Change background color to red.  
Change text color to white.

## CSS Syntax

CSS Selector = which element(s) will be modified



## CSS Properties

Here are some common CSS properties we can use:

```
button {
```

```
  background-color: red;
```

Sets the background color. Common values:

- Color name: `red`, `white`, `black`
- rgb value: `rgb(0, 150, 255);`
- Hex value: `#0096FF`

```
  color: white;
```

Sets the text color. Takes the same values as background-color (color name, rgb, hex).

```
  height: 36px;
```

Sets the height. Common values:

- Pixel value: `36px`
- Percentage: `50%`

```
  width: 105px;
```

Sets the width. Takes the same values as height.

<code>border: none;</code>	Removes the border.
<code>border-radius: 2px;</code>	Creates rounded corners.
<code>cursor: pointer;</code>	Changes the mouse/cursor when hovering over the element.
<code>border-color: red;</code>	Sets the border color.
<code>border-style: solid;</code>	Sets the border style. Common values: <ul style="list-style-type: none"> <li>• <code>solid</code></li> <li>• <code>dotted</code></li> <li>• <code>dashed</code></li> </ul>
<code>border-width: 1px;</code>	Sets the border width.
<code>}</code>	

## How To Google CSS Properties

We regularly use [Google](#) to search for CSS properties that we don't know or don't remember.

When using Google, search what you're trying to accomplish. Examples:

"css rounded corners"

"css text italic"

"css adjust space between lines"

## CSS Values

Each CSS property has a set of values that are allowed (`background-color` allows color values, `cursor` allows `solid`, `dotted`, `dashed`, etc.)

Here are some categories of values that are useful to know:

### Color Values

1. A color name: `red`, `white`, `black`

2. RGB value: `rgb(0, 150, 255);`

RGB is a more precise way of measuring color. Every color can be created using a combination of red, green, and blue (RGB). In CSS, this is represented by `rgb( . . . );`

`rgb(0, 150, 255);`



red



green



blue

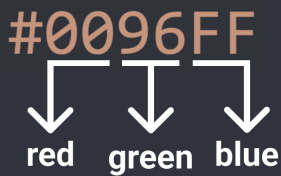
Each color has a min value = 0 and a max value = 255.

`rgb(0, 0, 0);` = black

`rgb(255, 255, 255);` = white

### 3. Hex value

Hex is another way to write RGB.



- Each character in Hex is base 16, which means it can have a value of 0, 1, 2, ... 8, 9, A, B, C, D, E, F (16 possible values).
  - Using the first 2 characters, we can have  $16 * 16 = 256$  possible values from 0 - 255:  
`00` = 0  
`01` = 1  
...  
`0F` = 15  
`10` = 16  
`11` =  $(1 * 16) + 1 = 17$   
...  
`FF` =  $(15 * 16) + 15 = 255$
  - This is the same range as RGB (0 to 255), so the first 2 characters in Hex are used to represent red, the second 2 characters represent green, and the third 2 characters represent blue. Usually, it's easier to use a Hex to RGB calculator to convert.
4. RGBA value: `rgba(0, 150, 255, 0.5);`  
Same as RGB, except with an additional a-value (alpha value). The a-value determines how see-through the color is. 0 = complete see-through, 1 = solid color and not see-through, 0.5 = 50% see-through.

## Measurement Values

### 1. Pixels: `50px`, `100px`

Pixels (px) are a common unit of measurement in the digital world.

For example: a 4K screen is 3840px by 2160px.

### 2. Percent: `50%`, `100%`

A relative measurement. For example, `width: 50%;` means 50% of the width of the page (or if the element is inside another element, 50% of the width of the container element).

### 3. em / rem: `1em`, `1rem`

Relative measurements that are useful for accessibility.

em = relative to the font-size of the element (`2em` means 2 times the font size).

rem = relative to the font-size of the page, **which is 16px by default** (`2rem` means 2 times the font size of the page =  $2 * 16px = 32px$  by default).

## Class Attribute

Class attribute = lets us target specific elements with CSS.

```
<button class="subscribe-button">
  SUBSCRIBE
</button>
.subscribe-button {
  ...
}
```

Add a class to an element. The class name (the text between the "...") can be anything you want, but no spaces.

Target all elements on the page with `class="subscribe-button"`

---

```
<button class="youtube-button">
  SUBSCRIBE
</button>
<button class="youtube-button">
  JOIN
</button>
```

Multiple elements can have the same class

---

```
<button class="youtube-button subscribe-button">
  SUBSCRIBE
</button>
```

An element can have multiple classes, separated by space

```
button {
  ...
}
```

Elements can be targeted by multiple CSS selectors. Here, all 3 CSS selectors will target the button above.

```
.youtube-button {
  ...
}
```

```
.subscribe-button {
  ...
}
```

## CSS Pseudo-Classes

```
.subscribe-button:hover {
  ...
}
```

These styles only apply when hovering over an element with `class="subscribe-button"`

```
.subscribe-button:active {
  ...
}
```

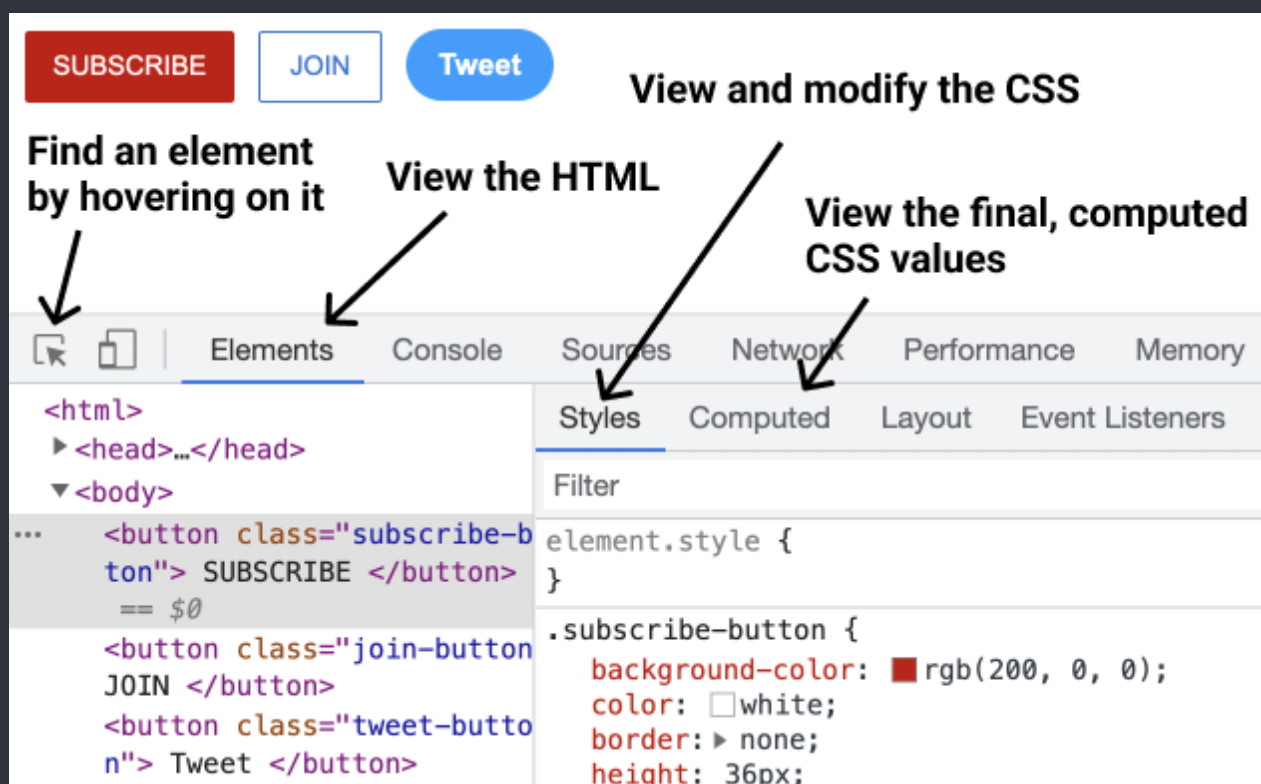
These styles only apply when clicking on an element with `class="subscribe-button"`

## Intermediate CSS Properties

<code>.subscribe-button {   opacity: 0.5;   opacity: 0;   opacity: 1;</code>	Sets how see-through an element is: 0.5 = 50% see-through. 0 = complete see-through (invisible). 1 = not see-through (this is the default value).
<code>transition: &lt;property&gt; &lt;duration&gt;;  transition: background-color 1s; transition: color 0.15s; transition: &lt;property1&gt; &lt;duration1&gt;,   &lt;property2&gt; &lt;duration2&gt;,   ...; transition: background-color 0.15s,   color 0.15s;</code>	Transition smoothly when changing styles (often used when hovering). Transition background color over 1 second. Transition text color over 0.15 seconds. Transition multiple properties by separating them with a comma.  Transition both background color and text color over 0.15 seconds.
<code>box-shadow: &lt;h-position&gt; &lt;v-position&gt; &lt;blur&gt; &lt;color&gt;; box-shadow: 3px 4px 5px black;</code>	Creates a shadow that's 3px to the right of the element, 4px to the bottom, with 5px of blur, and color of black.
<code>box-shadow: 3px 4px 0 rgba(0, 0, 0, 0.15); }</code>	Creates a shadow that's 3px to the right, 4px to the bottom, with no blur, and a very faint black color.

## Chrome DevTools

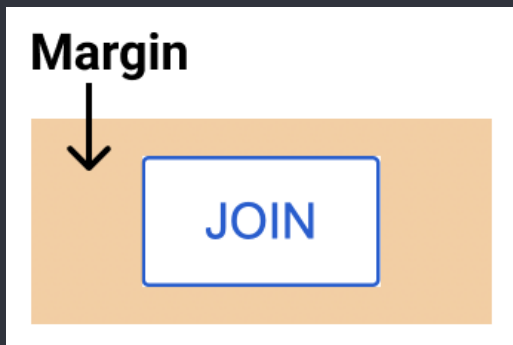
Lets us view (and modify) the HTML and CSS of a website directly in the browser.  
To open the DevTools: right-click > Inspect.



# CSS Box Model

- Determines how much space an element takes up.
- Determines how far away elements are from each other.

## 1. Margin = space on the outside



## 2. Padding = space on the inside



## 3. Border



```
.join-button {  
  margin-right: 10px;  
  margin-left: 10px;  
  margin-top: 10px;  
  margin-bottom: 10px;  
  margin-right: -20px;  
}
```

Add **10px** of space on the outside of the element.

Normal margin pushes things away from an element.  
Negative margin pulls things towards an element like this:



```
margin: 10px;  
margin: 10px 20px;  
margin: <top> <left & right> <bottom>;  
margin: <top> <right> <bottom> <left>;
```

Shorthand for adding **10px** of margin on all sides.

Add **10px** of margin top & bottom and **20px** left & right

```
padding-right: 10px;  
padding-left: 10px;  
padding-top: 10px;  
padding-bottom: 10px;  
padding-right: -20px;
```

Add **10px** of space on the inside of the element.

Negative padding has no effect.

```
padding: 10px;  
padding: 10px 20px;  
padding: <top> <left & right> <bottom>;  
padding: <top> <right> <bottom> <left>;
```

Shorthand for adding **10px** of padding on all sides.

Add **10px** of padding top & bottom and **20px** left & right

```
border-width: 1px;  
border-style: solid;  
border-color: red;
```

Sets the border width.

Sets the border style (to a solid color).

Sets the border color.

```
border: <width> <style> <color>;  
border: 1px solid red;
```

Shorthand for the 3 properties above.

}

## Text Styles

<pre>.title {   font-family: Arial;   font-family: Roboto, Verdana, Arial;    font-size: 30px;   font-weight: bold;   font-weight: 700;    font-style: italic;   text-align: center;   line-height: 24px;   text-decoration: underline;   text-decoration: none; }</pre>	<p>Change the font.</p> <p>A font-stack: if Roboto is not available, it will fall back to Verdana. If Verdana is not available it will fall back to Arial.</p> <p>Change text size.</p> <p>Change text thickness.</p> <p>Another way to specify font-weight. We can use: 100, 200, 300, ..., 900. bold = 700, regular = 400, semibold = 500</p> <p>Other values we can use: left, right, justified</p> <p>Adjust space between lines of text.</p> <p>Underlines the text.</p> <p>Removes underline.</p>
--	---

<p> by default have margin-top and margin-bottom. A common practice is to:

1. Reset the default margins.

```
p {  
  margin-top: 0;  
  margin-bottom: 0;  
}
```

2. Then apply more precise margins.

```
.title {  
  margin-bottom: 16px;  
}
```

## Text Elements (also called Inline Elements)

- Text elements (<strong>, <u>, <span>, <a>) appear within a line of text.

```
<p>
```

```
  This is a <strong>text element</strong>
```

```
</p>
```

Useful if we want to style only a part of the text.

- <span> is the most generic text element (it doesn't have any default styles).
- We can style text elements using a class:

```
<p>
```

```
  This is a <span class="shop-link">text element</span>
```

```
</p>
```

```
.shop-link {  
  text-decoration: underline;  
}
```



# The HTML Structure

<code>&lt;!DOCTYPE html&gt;</code>	Tells the browser to use a modern version of HTML.
<code>&lt;html&gt;</code>	
<code>&lt;head&gt;</code>	<code>&lt;head&gt;</code> contains everything that's <u>not visible</u> like the title and description (a.k.a. metadata) as well as links to fonts and CSS stylesheets.
<code>...</code>	
<code>&lt;/head&gt;</code>	
<code>&lt;body&gt;</code>	<code>&lt;body&gt;</code> contains everything that's <u>visible</u> like buttons, text, images, etc.
<code>...</code>	
<code>&lt;/body&gt;</code>	
<code>&lt;/html&gt;</code>	

## Elements in the Head Section

<code>&lt;head&gt;</code>	
<code>&lt;title&gt;Title in the tab&lt;/title&gt;</code>	Sets the title in the tabs
<code>&lt;link rel="preconnect" href="https://fonts.googleapis.com"&gt;</code>	
<code>&lt;link rel="preconnect" href="https://fonts.gstatic.com" crossorigin&gt;</code>	
<code>&lt;link rel="stylesheet"</code>	
<code>href="https://fonts.googleapis.com/css2?family=Roboto:wght@400;500&amp;display=swap"&gt;</code>	
<code>^</code>	
Loads a font from Google onto the page. 1) Search "google fonts" in Google. 2) Pick the fonts and styles that you like. 3) Copy the code that Google provides into <code>&lt;head&gt;</code>	
<code>&lt;link rel="stylesheet" href="styles.css"&gt;</code>	Loads a CSS file to the page
<code>&lt;/head&gt;</code>	

## Filepaths

<code>href="styles.css"</code>	Looks for a file called <code>styles.css</code> beside the HTML file.
<code>href="fold1/styles.css"</code>	Looks for a folder called <code>fold1</code> beside the HTML file, then goes into the folder and looks for <code>styles.css</code> .
<code>href="fold1/fold2/styles.css"</code>	Go into <code>fold1</code> , go into <code>fold2</code> , look for <code>styles.css</code>

## Images

<code>&lt;img src="image.png"&gt;</code>	Loads an image <code>image.png</code> beside the HTML file.
<code>&lt;img src="pics/image.png"&gt;</code>	Loads <code>image.png</code> in the <code>pics</code> folder.
<code>&lt;img class="image" src="pics/image.png"&gt;</code>	
<code>.image {</code>	
<code>width: 300px;</code>	Resizes the image to a width of <code>300px</code> . Height will also resize to keep the image's dimensions.
<code>height: 300px;</code>	If both <code>width</code> and <code>height</code> are set, the image may stretch.
<code>object-fit: cover;</code>	Enlarges the image to cover the entire <code>width * height</code> area without stretching or distorting.
<code>object-fit: contain;</code>	Shrinks the image so that it's contained in the <code>width * height</code> area.

object-position: left;	Determines where the image is positioned in the width * height area.
object-position: right;	
object-position: top;	
object-position: bottom;	

## Inputs

<input type="text">	Creates a text box.
<input type="text" placeholder="Search">	Add a placeholder (a label) to the text box.
<input type="checkbox">	Creates a checkbox.
<input class="search-bar" type="text">	

.search-bar { font-size: 30px; }	Changes the font-size when typing into the text box.
.search-bar::placeholder { font-size: 30px; }	Changes the font-size of the placeholder.

## CSS Display Property

.element { display: block; display: inline-block; vertical-align: middle; display: inline; }	<p>Element will take up the <u>entire line</u> in its container.</p> <p>Element will only take up as much space as needed.</p> <p>Determines vertical alignment of inline-block elements.</p> <p>Element will appear within a line of text (a text element).</p>
---	--

## <div> Element

<div> is a container. We generally put other elements (including other <div>s) inside (nesting).

```
<div class="container">
  <p>Name</p>
  <input type="text">
</div>
<div class="container">
  <p>Quantity</p>
  <div>
    <button>1</button>
    <button>2</button>
  </div>
  <button>Submit</button>
</div>
```

```
.container {
  display: inline-block;
  width: 200px;
}
```

<div>s allow us to group elements together and create more complex layouts.

Name

Quantity

1

2

Submit

# Nested Layouts Technique

There are 2 types of layouts:

## 1. Vertical Layout

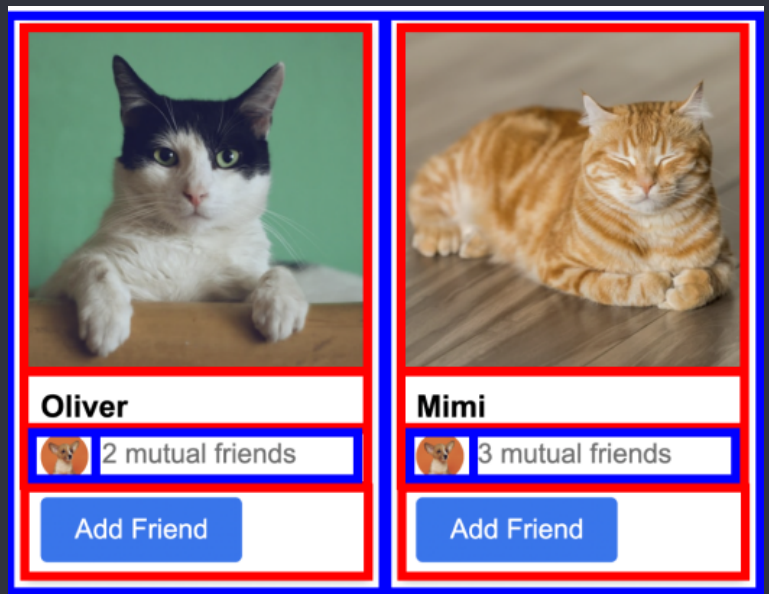
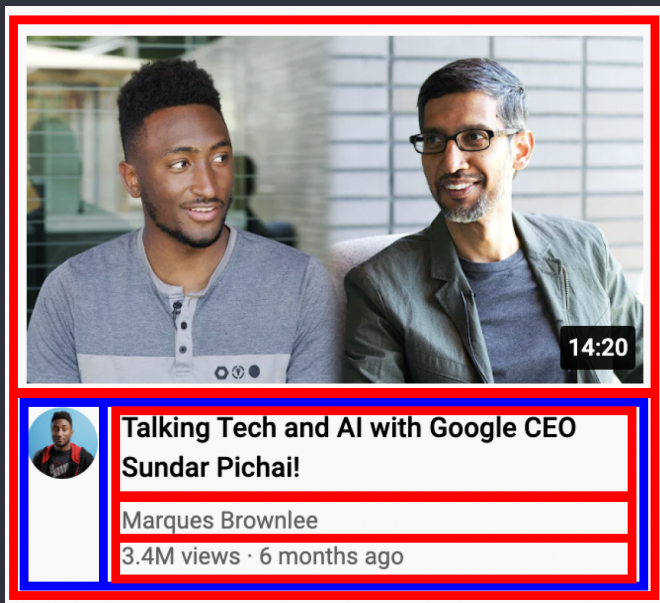


## 2. Horizontal Layout



Most designs can be created using:

- Vertical layout inside horizontal layout inside vertical layout ...  
OR
- Horizontal layout inside vertical layout inside horizontal layout ...



## To Create the Vertical Layouts

- Use `<div>`s with `display: block` (most common)
- Use flexbox (explained later) with `flex-direction: column`
- Use CSS grid (explained later) with 1 column

## To Create the Horizontal Layouts

- Use `<div>`s with `display: inline-block` (not recommended)
- Use flexbox with `flex-direction: row`
- Use CSS grid with multiple columns

## Inline CSS Styles

Another way of writing css, using the `style="..."` attribute:

```
<div style="
```

```
  background-color: red;
```

```
  color: white;
```

```
">
```

```
  ...
```

```
</div>
```

- Inline style = CSS is written within a line of HTML.
- Inline styles only affect the element with the `style="..."` attribute (no selectors are needed).

## CSS Grid

```
.grid {  
  display: grid;  
  grid-template-columns: 100px 100px;  
  
  column-gap: 20px;  
  row-gap: 40px;  
}
```

Turns an element into a grid container.  
Sets how many columns are in the grid and how wide the columns are.  
Sets space between the columns.  
Sets space between the rows.

```
.grid {  
  display: grid;  
  grid-template-columns: 100px 1fr;  
  
  grid-template-columns: 1fr 1fr;  
  
  grid-template-columns: 1fr 2fr;
```

`1fr` = the column will take up the remaining amount of space in the grid container.  
The columns will take up an equal amount of the remaining space (since they're both `1fr`).  
The number in front of `fr` = relatively how much space the column gets. Here, the 2nd column gets twice the amount of space as the 1st.

```
  justify-content: center;  
  justify-content: space-between;  
  align-items: center;  
}
```

Aligns the columns horizontally in the center.  
Spread out the columns evenly horizontally.  
Aligns the columns vertically in the center.

For more examples, see [grid.html](#).

## Flexbox

```
.flexbox {  
  display: flex;  
  flex-direction: row;
```

Turns an element into a flexbox container.  
Lays out elements horizontally inside the flexbox. Usually we don't need to specify `flex-direction: row`; because it is the default value.

```
justify-content: center;
justify-content: space-between;
```

Centers the elements in the flexbox horizontally.  
Spreads out the elements in the flexbox evenly across the horizontal space.

```
align-items: center;
align-items: space-between;
```

Centers the elements in the flexbox vertically.  
Spreads out elements evenly in the vertical space.

```
}
```

```
.element-inside-flexbox {
  width: 100px;
  flex: 1;
```

Sets the width of the flexbox element to 100px.  
Take up the remaining amount of space. The value 1 determines relatively how much space.

```
flex-shrink: 0;
width: 0;
```

Don't shrink the element when resizing.  
Allow the element to shrink down when resizing.

```
}
```

```
<div style="
  display: flex;
  flex-direction: row;
">
  <p style="width: 100px;">
    Flexbox element 1
  </p>
  <p style="flex: 1;">
    Flexbox element 2
  </p>
  <p style="flex: 2;">
    Flexbox element 2
  </p>
</div>
```

Creates a flexbox where elements are placed horizontally (`flex-direction: row`; is the default so it's not mandatory to have that in the CSS)

This element has a width of 100px.

This element takes up 1/3 of the remaining space.

This element takes up 2/3 of the remaining space.

### flex-direction: column:

```
.flexbox {
  display: flex;
  flex-direction: column;
```

Lays out elements vertically inside the flexbox. Also, `justify-content` and `align-items` are reversed.

```
justify-content: center;
justify-content: space-between;
```

Centers elements vertically inside the flexbox.  
Spreads out elements evenly in the vertical space.

```
align-items: center;
align-items: space-between;
```

Centers elements horizontally.  
Spreads out elements evenly horizontally.

```
}
```

For more examples, see [flexbox.html](#).

## CSS Position

- Create elements that stick to the page while scrolling.
- Create elements that appear on top of other elements.

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

This is the default value that every element starts with.  
`position: static;` causes the element to display normally.

### Position Fixed

```
.fixed {  
  position: fixed;
```

Positions the element in the browser window (sticks to the page while scrolling).

```
  top: 0;  
  bottom: 10px;  
  left: 50px;  
  right: 100px;
```

Places the element `0px` from the top of the browser window.  
`10px` from the bottom of the browser window.  
`50px` from the left of the browser window.  
`100px` from the right of the browser window.

- If you set opposite directions (`top/bottom` or `left/right`), the element will stretch.

```
  top: -5px;
```

Using negative pixels places the element beyond the top edge.

```
  width: 100px;  
  height: 100px;
```

Sets the element's width to `100px`.  
Sets the element's height to `100px`.

- When using `width/height` the element will not resize with the page.
- When using `top/bottom/left/right` the element will resize with the page.

### Position Absolute

```
.absolute {  
  position: absolute;
```

Positions the element on the page (it will scroll with the page and will not stick when scrolling).

```
  top: 0;  
  bottom: 10px;  
  left: 50px;  
  right: 100px;
```

Places the element `0px` from the top of the page.  
`10px` from the bottom of the page.  
`50px` from the left of the page.  
`100px` from the right of the page.

```
  width: 100px;  
  height: 100px;
```

Sets the element's width to `100px`.  
Sets the element's height to `100px`.

### Position Absolute Inside Position Fixed

- When a `position: absolute` element is inside a `position: fixed` element, it will be positioned relative to the `fixed` element.
- This rule also applies to any position value that is not `position: static`.
- This lets us place elements in the corners of other elements. For example, a "Close" button in the top-right corner.

```

<div style="
  position: fixed;
  width: 100px;
">
  <button style="
    position: absolute;
    top: 0;
    right: 0;
  ">
    X
  </button>
</div>

```

The `position: absolute` element will be placed in the top-right of the `position: fixed` element.

## Position Relative

```

.relative {
  position: relative;

  top: 10px;

  bottom: 10px;

  left: 50px;
  right: 100px;

  width: 100px;
  height: 100px;
}

```

The element will appear normally (as if it's `position: static`). We can then push it around with `top/bottom/left/right`.

Places the element `10px` from the top of its original position (pushes it down by `10px`). Unlike `margin`, it won't push the rest of the page down.

Places the element `10px` from the bottom of its original position (pushes it up by `10px`).

Places the element `50px` from the left of its original position.

Places the element `100px` from the right of its original position.

Sets the element's width to `100px`.

Sets the element's height to `100px`.

## Position Absolute Inside Position Relative

- When a `position: absolute` element is inside a `position: relative` element, it will be positioned relative to the `relative` element.
- Useful if we want to display an element normally (using `position: relative`), but still be able to place other elements in the corner (using `position: absolute`).

```

<div style="
  position: relative;
  width: 100px;
">
  <button style="
    position: absolute;
    top: 0;
    right: 0;
  ">
    3
  </button>
</div>

```

The `position: absolute` element will be placed in the top-right of the `position: relative` element.



## z-index

Determines which elements appear in front and behind:

- Elements with a higher **z-index** appear in front of elements with a lower z-index. The default **z-index** is 0.
- Elements with **position: static**; always appear at the back. **z-index** has no effect.
- If the **z-index** is equal or both elements are **position: static**, the element that was written later in the code will appear in front.

```
.fixed {  
  position: fixed;      This element will appear in front of the position: absolute;  
  z-index: 2;           element because it has a higher z-index.  
}  
  
.absolute {  
  position: absolute;  
  z-index: 1;  
}  
  
.static {  
  position: static;     This element will appear at the back since it's position: static.  
}
```

For more examples, see [position.html](#).

## Responsive Design

Responsive design = making the website look good on any screen size.

```
@media (max-width: 750px) {  
  .element {  
    width: 350px;  
  }  
}  
  
@media (min-width: 750.02px) and (max-width: 1000px) {  
  .element {  
    width: 450px;  
  }  
}  
  
@media (min-width: 1000.02px) {  
  .element {  
    width: 600px;  
  }  
}
```

Only apply the CSS code below when screen width is between 0px - 750px.

Only apply this CSS code when screen width is between 750px - 1000px.

Only apply this CSS code when the screen width is over 1000px.

We generally use a gap of **.02px** between the ranges (like above) because the browser can support fractional screen widths like **750.50px**.



# Advanced CSS Selectors

## With Comma

```
.class1, .class2 { ... }  
.class1, p { ... }
```

Target multiple classes at the same time.  
Target a class and all `<p>`s at the same time.

## With Space

```
.class1 img { ... }
```

Target `<img>`s that are inside elements with `class="class1"`

```
.class1 img,  
.class2 .tooltip { ... }
```

Target `<img>`s that are inside elements with `class="class1"` AND `.tooltip` inside elements with `class="class2"`.

```
.class2:hover .tooltip { ... }
```

Target `.tooltip` only when hovering over elements with `class="class2"`.

For a full list of selectors, check out [CSS Selectors](#).

# Inheritance

A text property set on the outer element will be passed down into inner elements:

```
<div style="color: red;">  
  <p>Paragraph</p>  
</div>
```

This paragraph will have red text.

For global text styles (styles we want on the entire page), we can set them on the `<body>`:

```
body {  
  font-family: Roboto, Arial;  
  color: rgb(20, 20, 20);  
}
```

All elements on the page by default will use `font-family: Roboto, Arial` and `color: rgb(20, 20, 20)`. This can be overridden.

# CSS Specificity

If multiple CSS selectors change the same property on the same element (see example below), CSS Specificity determines which selector "wins" (which style gets applied).

```
body { color: black; }  
p { color: red; }  
.title { color: green; }
```

```
<body>  
  <p class="title">  
    Paragraph of text.  
  </p>  
</body>
```

In this example, the `.title` selector has the highest priority (according to CSS Specificity) so the text will be green.

## CSS Specificity Rules

Here's the full set of [CSS Specificity Rules](#) (you don't need to memorise all of these).

Usually, you just need to know a few useful rules and search Google for more if needed:

1. Inline CSS has higher priority than `.class` selectors.
2. `.class` selectors have higher priority than element name selectors (`p`).
3. Element name selectors (`p`) have higher priority than inheritance (from `body`).
4. If 2 selectors have the same priority, the one that is written later wins.

## General Rule of Thumb

A CSS selector that's more specific (targets a more specific set of elements) has higher priority.

## Semantic Elements

Elements that work the same way as `<div>`. However, they also give the HTML meaning when screen readers, search engines, or other devices read the website.

Common semantic elements include:

`<header>`, `<nav>`, `<main>`, `<section>`, etc.

Here's a list of [Semantic Elements](#). They'll be covered in more detail in the accessibility course.

## Comments

Let us write code that the browser ignores. Useful for documenting how the code works.

```
<!-- This is a comment -->
<p class="title">
  Paragraph of text.
</p>
```

Syntax for a comment in HTML: `<!-- ... -->`

```
/* This is a comment */
.title {
  color: green;
}
```

Syntax for a comment in CSS: `/* ... */`

## Other CSS Properties

Here are some other CSS properties that were covered in the course.

```
.tooltip {
  pointer-events: none;
  white-space: nowrap;
}
```

Disables all interactions with the mouse (clicks, hovers, etc.)  
Prevents the text inside the element from wrapping to multiple lines.

