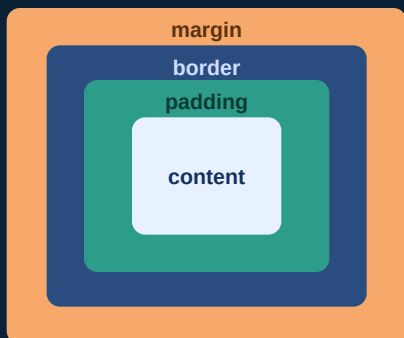




BioGlytch Studios LLC

Learn | Build | Grow



```
.card {  
  color: teal;  
  padding: 1rem;  
  display: grid;  
  gap: 0.5rem;  
}
```



Selectors, the box model, color and motion

INTRO TO CSS

COMPUTER CODING: LEARN TO PROGRAM

WRITTEN BY

Clarence Scott & BioGlytch Studios

BioGlytch Studios LLC | Est. 2026

In Partnership with Gloria Coles Flint Public Library

bioglytchstudios.tech | 2026 Edition

Intro to CSS

Computer Coding: Learn to Program

BioGlytch Studios LLC

COPYRIGHT 2026 © BioGlytch Studios LLC © Genesee County, MI EST 2026

</HELLO CODER>

Welcome to the engineering world of BioGlytch Studios LLC! We are thrilled to present this book to you. Published by BioGlytch Studios LLC and offered as a front-facing resource of Gloria Coles Flint Public Library, it is dedicated to making computer science and technology education more accessible to our community. Designed for aspiring technology enthusiasts and their supportive parents, this book offers an opportunity to explore the exciting world of computing together. If you are a parent who has chosen this book to introduce your child to technology, we thank you for your trust and support.

Through this book, and our partnership with Gloria Coles Flint Public Library, BioGlytch Studios LLC aims to ignite curiosity and inspire a passion for coding, programming, and computing technologies in individuals of all ages. We recognize that access to quality educational resources can be limited by financial barriers, time constraints, and the rising costs of traditional university courses. Our goal goes beyond teaching the fundamentals—we want to empower you to think critically, solve problems creatively, and begin your journey toward becoming a skilled and innovative engineer.

Throughout this book, you will develop essential skills such as critical thinking, logical outlooks, problem solving, discipline and career-ready structure. Together, we are committed to building a stronger, more technologically empowered community, one coder at a time.

Each successful assessment will grant a Certificate from BioGlytch Studios LLC, along with a PDF of your course completion to show your next employer.

Thank you for joining us on this journey. We look forward to supporting your growth and celebrating your progress as you put your best code forward!

Sincerely,

Clarence Scott

CEO & Founder, BioGlytch Studios LLC

In Partnership with Gloria Coles Flint Public Library

Publishing Information

Book Title

Intro to CSS

Author

Clarence Scott

Publisher

BioGlytch Studios LLC

2026 Edition

Copyright Notice

© 2026 BioGlytch Studios LLC. All rights reserved.

BioGlytch Studios LLC is the sole owner of all content in this book. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means without the prior written permission of the publisher, except in the case of brief quotations embodied in critical articles or reviews.

Library Partnership

Gloria Coles Flint Public Library offers this book as a front-facing resource for its organization. The partnership does not transfer ownership of any content, which remains solely with BioGlytch Studios LLC.

Edition

First Edition

Disclaimer

The information contained in this book is provided for educational purposes only. While every effort has been made to ensure accuracy, the author and publisher are not responsible for errors, omissions, or damages resulting from the use of this material.

Printing Information

Printed in Genesee County, Michigan

Printed by BioGlytch Studios LLC

Contact Information

For inquiries, feedback, or bulk orders, please contact:

Email: software@bioglytchstudios.tech

Website: <https://bioglytchstudios.tech>

Phone: (989) 900-6747

Table of Contents

UNVEILING CSS

- RULES & SELECTORS
- CLASS & ID SELECTORS
- TEXT STYLING
- BOX MODEL
- BACKGROUND
- LIST STYLING
- LINK STYLING
- TABLE STYLING
- FORM STYLING
- LAYOUT STYLING
- POSITIONING STYLING
- TRANSFORMS STYLING
- TRANSITIONS STYLING
- KEYFRAMES & ANIMATIONS STYLING

Modern CSS for 2026

- Custom Properties (CSS Variables)
- Responsive Design with Media Queries
- Light and Dark Mode
- Nesting, :has() and Container Queries
- Logical Properties and Modern Colors
- Organizing Larger Stylesheets with @layer

2026 CSS Best Practices

Thank You for Choosing BioGlytch Studios LLC!

- Unveiling CSS
- Your Journey, Your Achievement

UNVEILING CSS

Actual name for CSS is, Cascading Style Sheets. CSS is used to change colors , font size, etc., as well as created layouts. Layouts are useful for navigation menus, columns, sidebars and many more things.

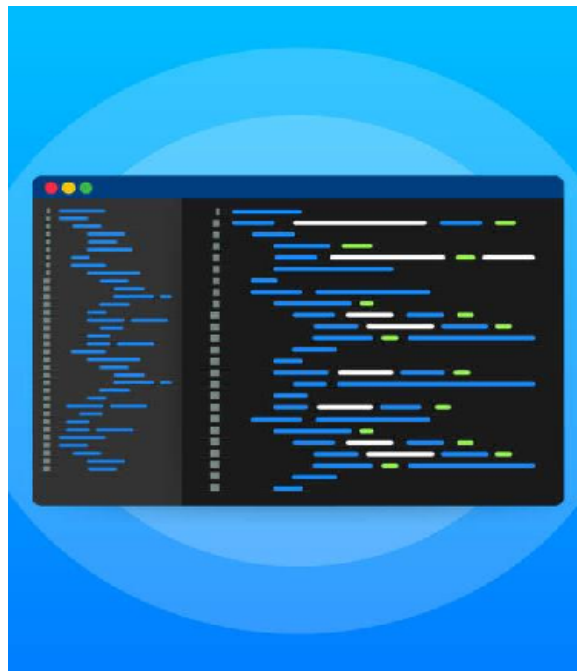
If you remember from our HTML tutorials, we used elements to create layouts on our HTML page. CSS uses **rules** to style those elements. An rule can cause all div containers to have a blue background with white text. A rule is a group of style **declarations**.

There are many ways to apply cascading styling sheet, inline, in HTML using **<style>** **</style>**, or by adding the CSS document from an external sheet. Using the **style** attribute on HTML elements applies the style to that element only. We would have to copy the same style attribute and define it on all the elements needed. To avoid redundancy, we want to avoid doing this. So what can we do??

It is common practice to define CSS rules in a different document and link it to our HTML document. You can target elements **selectors**, this allows you to target specific elements and apply the same styling.

To add an external CSS document to your HTML page, put a link element inside the <head>. Create a style.css file in your project folder first:

```
<link rel="stylesheet" href="style.css">
```





RULES & SELECTORS

It's best to learn about Rules & Selectors prior to using CSS sheets. The style rules are defined in a CSS file using **selectors**, these are your elements from your HTML documents.

```
header {color: green;}
```

Here we are targeting all of the header elements of our HTML document and changing the text color to green.

selector { property: value; }. Declarations are placed inside of curly braces. We also can add multiple styles for our selector, example:

```
header { background-color: green; color:red; }
```

Multiple selectors can be styled at once by using a comma to separate them. This will make all of the selectors have the same **property** and **value**.

```
selector1, selector2 { property: value; }
```

CLASS & ID SELECTORS

The **class selector** is very powerful when we are styling our HTML documents. Often there may be a single element or just a set of elements you need to style, without changing the others. This is where we will use the **class selector**, we will need to add a **class** attribute to the elements we will be targeting.

```
<h1 class="heading">CSS is awesome</h1>
<p id="firstP">CSS Intro</p>
```

After you've done this, we can make the class selector in CSS. All class selectors needs to with a **dot**, followed by the name we gave to our element.

```
.heading { color: green; background-color:gray; }
```

Anything we decide to add this class selector too will have the same styling, simply by adding the attribute.

To create an **id** selector, you simply need to use the hash symbol, **or hashtag #** , before the value rather than the **dot** . Only one element can have the same **id** on the page.

```
<h1 class="heading">CSS is awesome</h1>
<p id="first-paragraph">CSS Intro</p>
```

```
#first-paragraph { color: green; font-weight: bold; }
```

2026 Best Practice: Style with classes whenever you can. IDs are very strong in the cascade and are hard to override, and a page can have only one of each. Use short, lowercase, hyphenated names such as `.site-header`.

We can also target location based elements. For example if you have an element **** inside of a list **** , you can target that element using:

```
li b { color:orange; }
```

Styles that are given to parent elements are inherited by the child elements. If you decide to change the **color** or **font** on an element every element inside of it will also be styled.

TEXT STYLING

Styling text is a way to capture the eye of your web page visitors. In the previous page we have already seen how to change text color with the property color. As we learned in the HTML portion of this book we can use: color names, hex & rgb to change color values in our styling. Along with this we have many ways to change our **font-size**

This property accepts many units. Pixels (px) are easy to start with, but in 2026 the best practice is to size text in **rem** units (1rem is the browser's default text size, usually 16px) so people who increase their browser text size are respected. Load up a generic page for this exercise and type up a single paragraph, we will change the styling of the paragraph in our document to a **font-size of 48px** . This is definitely over and beyond for what we are trying to accomplish but I want you to have something that sticks out to you.

```
html { font-size: 100%; }
p    { font-size: 1.125rem; line-height: 1.6; }
h1   { font-size: clamp(2rem, 1.2rem + 3vw, 3.5rem); }
```

The `clamp(min, preferred, max)` function makes text grow smoothly with the screen but never smaller or larger than the limits you set.

Sometimes it is best to change the font-family of the text throughout your webpage. Always list several fonts so that if one is not available the browser goes to the next one, and finish with a generic family such as `sans-serif`, `serif` or `monospace`. Modern "system font" stacks load instantly because the fonts are already on the visitor's device:

```
body {  
  font-family: system-ui, -apple-system, "Segoe UI", Roboto, sans-serif;  
}
```

To use your own font, host a modern `.woff2` file and load it with `@font-face`. Add `font-display: swap` so text is visible while the font downloads:

```
@font-face {  
  font-family: "Brand Sans";  
  src: url("fonts/brand-sans.woff2") format("woff2");  
  font-weight: 400 700;  
  font-display: swap;  
}
```

We can also use the `font-style` property to make text italic, and `font-weight` for the boldness of your text, from 100 (thin) to 900 (heavy). 400 is normal text while 700 is bold.

The best thing to do when styling your font-family is to choose multiple fonts so that if one is not available the browser will go to the next available font. We can also use the `font-style` property to make text italic, `font-weight` is used for the boldness of your text. 100 (thin) to 900 (thick). 400 is normal text thickness while 700 is the equivalent to bold.

Using the `text-transform` allows you to change text in regards to: uppercase, lowercase, capitalize:

- **uppercase:** Transforms the text to capitals.
- **lowercase:** Transforms the text to lower case.
- **capitalize:** Transforms all words to have the first letter capitalized.

`Text-decoration` allows you to: underline, overline or line-through text. `text-shadow:` (x-offset) (y-offset) (blur radius) (color);

Example: `.heading { text-shadow: 2px 3px 5px green; }`

Multiple shadows can be developed by comma separation.

Example: `.heading { text-shadow: 2px 3px 5px green, 6px 8px 2px red; }`

Aligning your text gives you control over the container text layout. Values can either be aligned by the `text-align` with: left, right, center, justify. Sometimes you may need additional line separation. This can be achieved by using `line-height` property. Simply inputting `line-height: 1.4;` will cause the line to become 1.4 times the height of the current font. Letter

and word spacing can cause desired spacing for clarity using certain fonts. Simply use letter-spacing or word-spacing and chose the pixels that you want apart from your letters or words. Example: header{ word-spacing: 8px; or letter-spacing: 3px; }

BOX MODEL

The box model is something you should keep a mental visualization of. This helps you maintain organization on your web page when styling different elements.

The **content** box is the area where the content is displayed. The **padding** box is the teal space around the content. The **border** box comes after the padding. The **margin** box is the last one and defines the space between this box and other elements. (Space outside the box).

In the standard box model, the height and width attributes define the size of the content box. Any padding and border is then added to that width and height to get the total size of the box.

By default the width you set covers only the content, which makes layouts hard to predict. Nearly every modern stylesheet starts with this reset so that padding and border are included in the width:

```
*, *::before, *::after {  
  box-sizing: border-box;  
}
```

2026 Best Practice: Put `box-sizing: border-box` at the top of every stylesheet. It is the single most useful layout habit in modern CSS.

For example, the total width of the box with paddings will be the sum of width plus padding left and padding right. Margin is not counted in this model due to it not affecting the box space itself, rather the space outside of the box.

Margin and **padding** can both be styled with a shorthand that goes clockwise: **top, right, bottom, left**. Example:

```
margin: 5px 10px 5px 10px; /* top right bottom left */  
margin: 5px 10px;          /* top and bottom 5px, left and right 10px */  
margin-inline: auto;        /* center a block with a set width */  
padding-block: 1rem;         /* top and bottom padding (logical property) */
```

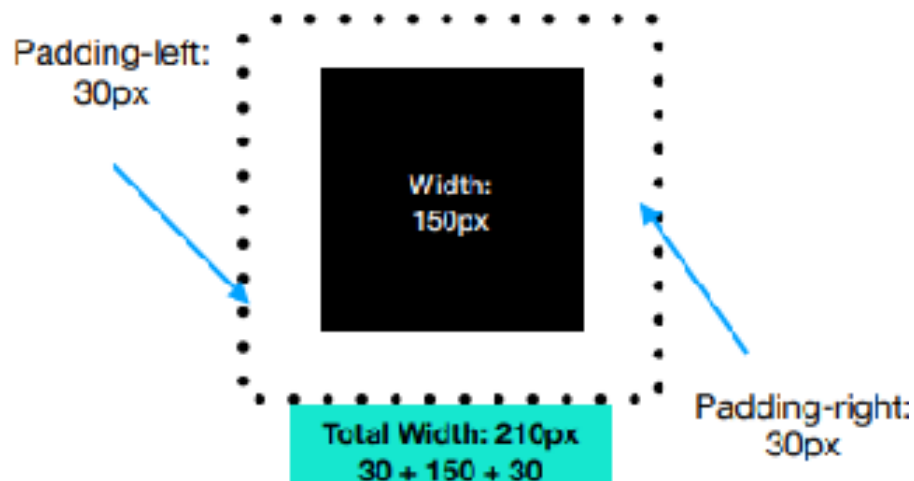
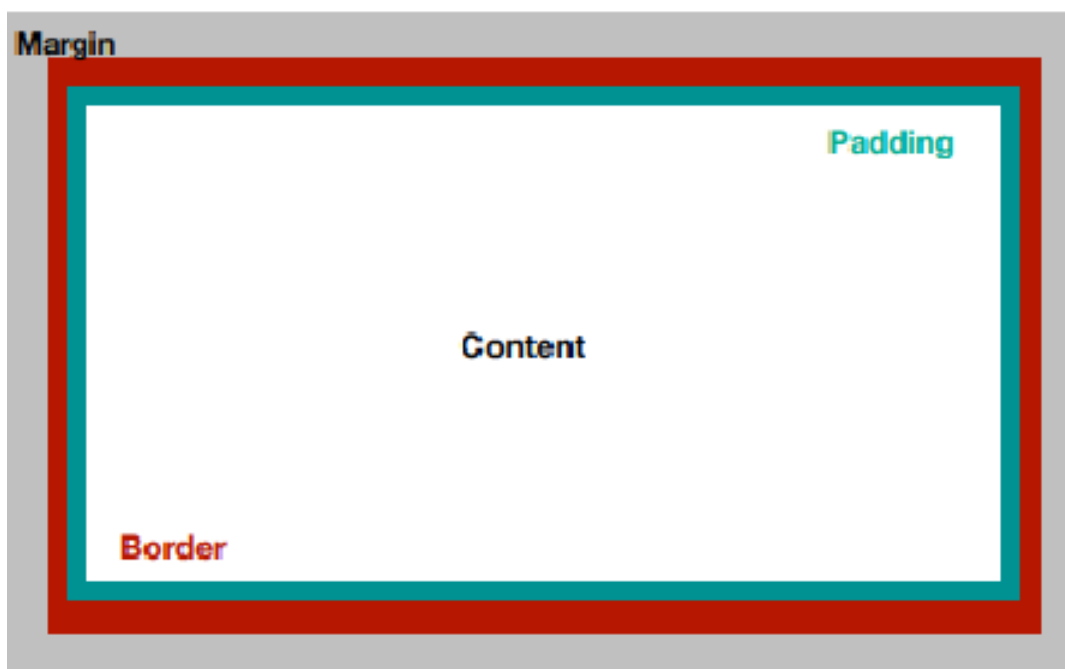
Elements can get border styling, using the border property, this allows you to display the container outline for elements. The border is always there BUT it is not visible until the border is defined.

Elements can get border styling, using the border property, which displays the outline of an element. The border is always there but it is not visible until you define it. Common border styles are solid, dashed, dotted, double, groove, ridge, inset and outset. The format is: border: thickness style color. Try this on an element from your project:

```
.card {
  border: 4px dotted blue;
  border-radius: 12px; /* round all corners */
}
```

Each side of a border can have its own style (border-top, border-right and so on). The border-radius property rounds the corners using pixels, rem or percentages, and it applies even if the border is not visible.

Border styling is similar to margin and padding styling as far as styling top, bottom, left, right. Each side of an border an have its own style. By setting the border radius, you can round the corners of elements with pixels or percentage. You can style each corner separately or using one value for all corners. Radius applies even if the border is not visible.



BACKGROUND

The background-color property sets the background color for an element. In your CSS document, if you have one open still, lets style the body element. Simply type in CSS body { background-color: yellow; }. Background colors can be set using hex, rgb or color names.

Using the background-image property allows you to set the background of an container or page to an specific image. The image address is placed in quotes using the url() attribute.

```
body {  
  background-image: url("images/test.png");  
  background-repeat: no-repeat;  
  background-size: cover;  
}
```

Naturally your background image will be tiled, the following rules can be applied in the styling to work with that :**no-repeat**, **repeat-x** , **repeat-y** , **repeat** (x=horizontal, y=vertical).

Applying the **background-position** property sets the position in which the background image appears. The position property uses coordinate systems to place the image, its start point is always (0,0). **Background-position** can take the values: pixel/percentage, units and keywords. The **background-size** allows the styling for how to fit the image in the box. 2 values are used for this **cover** and **contain** . Cover, covers the entire box while keeping the images aspect ration. Meanwhile contain, fits the image inside the container completely.

LIST STYLING

By default list items are styled with bullet points and ordered list are marked with numbers. Using **list-style-type** you can change the list markings from circles, squares images or nothing at all. Using **list-style-image** you can have an image as an list marker.

```
ul { list-style-type: square; }  
ul.custom { list-style-image: url("images/check.svg"); }  
nav ul { list-style: none; margin: 0; padding: 0; } /* menu with no bullets */
```

LINK STYLING

Default styling for links are blue in color and underlined. In CSS you can change these properties in many ways using the **a** selector. Link styling can be specific to the “state” of the link, a:link - defines the style for normal unvisited links. a:visited - defines the style for visited links. a:active - a link becomes active once you click on it. a:hover - a link is hovered when the mouse is over it. These are known as **pseudo selectors** . Using **text-decoration** we can change the styling of an link or remove the underline, by using **text-decoration:none**; . You can add background color to images when you hover with the a:hover { background-color: value } **selector** and **property** . Common practice is to use links as buttons, so normally you would place your links in side of your list items and style each link using li a {**property:value**};

```

a:link { color: #0b57d0; }
a:visited { color: #6b3fa0; }
a:hover { text-decoration: underline; }
a:focus-visible { outline: 3px solid #ff8c00; outline-offset: 2px; }
a:active { color: #b3261e; }

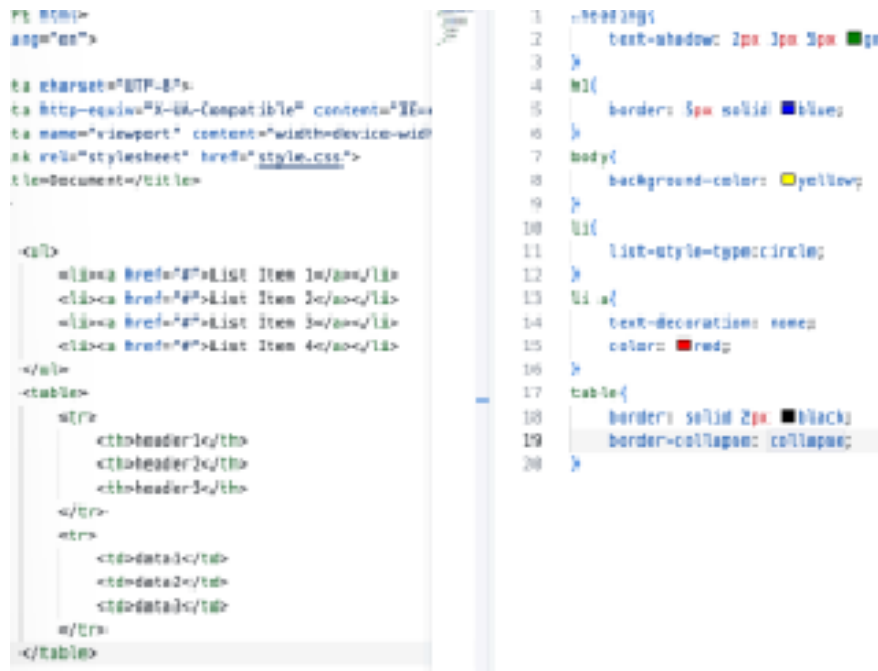
```

2026 Best Practice: Always give links and buttons a visible focus style. Use `:focus-visible` so keyboard users see the outline without distracting mouse users. If you use several link states, write them in the order link, visited, focus, hover, active.

TABLE STYLING

When styling tables in HTML/CSS, due to the unattractive presentation of tables without styling, it's good to make sure your table is as sleek in design as possible. We can use the **border-collapse** property to make all touching borders collapse into 1 border. This is going to sharpen the look of your border of the table. Just like everything else in HTML the cells, `td` & `th`, can receive padding styling for more comfortability in the table. Background color is also available for the cells or table styling. Another pseudo selector that comes in handy with rows and columns in your table, is the **nth-child()** selector. This selector allows you to chose specific rows by **odd** or **even** , and if you wanted by specific **numbers** !

Example: **th:nth-child(even)** { property:value } (Selects all even columns in the table.)



FORM STYLING

Forms can be styled by styling the: form, input, or label. Add some styling to the input and label. Notice that the button will also receive the same styling as it is an input element as well.

```
label { display: block; font-weight: 600; margin-block-end: 0.25rem; }
input, textarea { width: 100%; padding: 0.5rem; border: 2px solid #5f6b7a; border-radius: 6px; }
input:focus-visible { outline: 3px solid #1a73e8; outline-offset: 2px; }
input:user-invalid { border-color: #b3261e; }
```

The `:user-invalid` pseudo-class only shows the error style after the visitor has interacted with the field, so a blank form does not look broken.

Example: `input { border: 2px solid red; width:100%; padding: 4px; } label { font-weight: bold; color:blue; }`

We can target specific elements inside of our inputs. For example if I want to only target the "name" element, I would use the following code: `input[id="name"] { property:value; }`

These elements can also be styled just as we would a paragraph or heading etc. Applying text to our button is the same exact way, it is definitely best practice to style your button in a way that is attractive and stands out. Pseudo selectors work with these elements as well.

`<form>`

```
<label for="name">Name: </label>
```

```
<input type="text" id="name" placeholder="Jon Doe">
```

```
<label for="email">Email: </label>
```

```
<input type="email" id="email" placeholder="name@domain.com">
```

```
<input type="button" value="send">
```

`</form>`

Form example in HTML.

LAYOUT STYLING

In HTML we have **block** and **inline** elements. Block elements extend 100% of the width of their parent, starting on a new line. Inline elements sit inside block elements, on the same line as their content. The `display` property changes whether an element behaves as block or inline. Modern layout uses two more values: `flex` and `grid`.

Flexbox (`display: flex`) arranges items in one direction, a row or a column. Use it for navigation bars, button groups and centering:

```
.nav {
  display: flex;
  gap: 1rem;           /* space between items */
  justify-content: space-between;
  align-items: center;
  flex-wrap: wrap;
}
.nav .grow { flex: 1; }
```

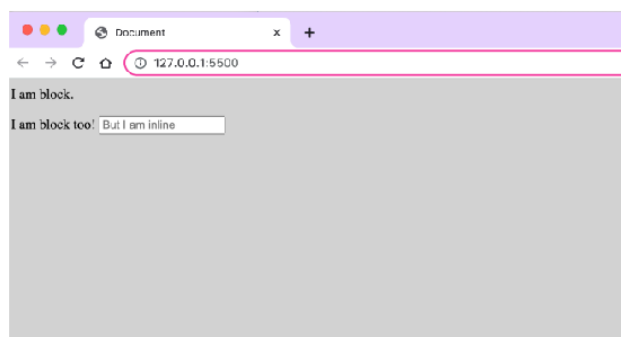
Grid (`display: grid`) arranges items in rows and columns at the same time. Use it for page layouts and card galleries:

```
.cards {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(16rem, 1fr));
  gap: 1.5rem;
}
```

This one rule makes a gallery that automatically fits as many columns as there is room for, with no media queries. A good rule of thumb: Flexbox for one-dimensional rows, Grid for two-dimensional layouts.



Code example for block and inline elements.



POSITIONING STYLING

The `float` property pushes an element to the left or right so text wraps around it. In 2026, use `float` only for its original purpose, such as wrapping text around an image. Do not use it for page layout; use `Flexbox` or `Grid` instead.

```
img.portrait { float: left; margin: 0 1rem 1rem 0; }
```

The `position` property is also useful. The default is `static`. The properties `top`, `right`, `bottom` and `left` tell the browser how to shift a positioned element. Position values are:

fixed- causes the element to stick to a specific part of the visible container, which means it is always in the same spot even on scroll

absolute- this removes the element from the normal flow of the page

relative- positioned relatively to its normal position

z-index- allows elements to overlap other elements, this is a stack order of flow or overlay (if positioned outside of the normal flow).

```
# style.css
# style.css > ...
1
2 body{
3   background-color: lightgrey;
4 }
5 header, p{
6   float:right;
7 }
8
```



TRANSFORMS STYLING

Transformations in CSS opens the ability to **translate** , **rotate** , **scale** and **skew** our HTML elements. These change position, shape and size. Using the **transform- origin** property allows us to change the position of elements as well. This is simply done by:

transform-origin: 20% 50%; (You can place any value you want in here).

Translate() moves an element from its current position on the **x-axis** and **y-axis**, from its default position in the parent element. **Skew()** method skews an element along the x and y-axis. Skew is another word for **tilt** or **angle**. Essentially we are tilting the element at an angle. **Scale()** increases and decreases element size, with height and width.

```
.card:hover {  
  transform: translateY(-4px) scale(1.05) rotate(2deg);  
}
```

You can combine several transform functions in one declaration. `scale(1.3)` makes the element 30% bigger in both directions; `scale(1.3, 0.8)` sets width then height.

TRANSITIONS STYLING

Transitioning in CSS allows to change values from one to another over a specified duration. The code in the illustration below shows how we can use a div container to practice on an width expanding **transition** when the element is hovered over with our cursor.

The **transition-timing-function** we can use to change the curve of the transition. The default timing value is **ease** if nothing is selected. **ease - the animation starts slowly, then accelerates quickly. ease-in - starts slowly, then accelerates, and stops abruptly. ease-out - starts quickly, but decelerates to a stop. ease-in-out - similar to ease, but with more subtle acceleration and deceleration. linear - constant speed throughout the animation; often best for color or opacity changes.**

KEYFRAMES & ANIMATIONS STYLING

We use animations for a series of movements, and we can change as many properties and values as we want. We define an animation with the `@keyframes` rule and then attach it to an element. This example name is pulse:

```
@keyframes pulse {  
  from { transform: scale(1); }  
  to   { transform: scale(1.1); }  
}  
  
.badge {  
  animation: pulse 1.5s ease-in-out 0s infinite alternate;  
}
```

from and to are the same as 0% and 100%. You can add steps in between, such as 50% { opacity: 0.5; }. The animation shorthand order is: name, duration, timing-function, delay, iteration-count, direction. The long forms are animation-name, animation-duration, animation-timing-function, animation-delay, animation-iteration-count (a number or infinite) and animation-direction (normal, reverse, alternate, alternate-reverse). Whenever you add motion, respect visitors who have asked their device to reduce it:

```
@media (prefers-reduced-motion: reduce) {  
  *, *::before, *::after {  
    animation-duration: 0.01ms !important;  
    animation-iteration-count: 1 !important;  
    transition-duration: 0.01ms !important;  
  }  
}
```

2026 Best Practice: Animate transform and opacity where you can. They are smooth because the browser can run them on the graphics processor, while animating width, height or margin can make a page stutter.

Modern CSS for 2026

The features below are well supported in current versions of Chrome, Edge, Firefox and Safari (they are part of the "Baseline widely available" set), so you can use them today.

Custom Properties (CSS Variables)

Store a value once and reuse it everywhere. Change the variable and the whole site updates:

```
:root {
  --brand: #2ecc40;
  --space: 1rem;
  --radius: 0.5rem;
}
.button {
  background: var(--brand);
  padding: var(--space) calc(var(--space) * 2);
  border-radius: var(--radius);
}
```

Responsive Design with Media Queries

Write your base styles for small screens first, then add rules for larger screens with min-width media queries:

```
.layout { display: grid; gap: 1rem; }

@media (min-width: 48rem) {
  .layout { grid-template-columns: 1fr 2fr; }
}
```

Do not forget the viewport tag in your HTML head: `<meta name="viewport" content="width=device-width, initial-scale=1">`.

Light and Dark Mode

Let the visitor's device choose. `color-scheme` tells the browser you support both, and `light-dark()` picks a value for each:

```
:root { color-scheme: light dark; }
body {
  background: light-dark(#ffffff, #121212);
  color: light-dark(#1a1a1a, #f0f0f0);
}
```

Nesting, :has() and Container Queries

CSS now lets you nest rules, select a parent based on its children and style a component based on the size of its container:

```

.card {
  padding: 1rem;
  & h2 { font-size: 1.25rem; }
  &:hover { box-shadow: 0 4px 16px rgb(0 0 0 / 0.2); }
}

/* A form group that has an invalid input inside it */
.field:has(input:user-invalid) { border-color: #b3261e; }

/* Component responds to its container, not the whole screen */
.panel { container-type: inline-size; }
@container (min-width: 30rem) {
  .panel .media { display: grid; grid-template-columns: 8rem 1fr; }
}

```

Logical Properties and Modern Colors

Use margin-inline, padding-block and inset instead of left/right/top/bottom so your layout works in every writing direction. For colors, rgb(0 0 0 / 0.2) supports transparency with a slash, oklch() gives evenly bright colors and color-mix() blends two colors:

```

.tag {
  background: color-mix(in oklch, var(--brand) 20%, white);
  color: oklch(35% 0.1 145);
}

```

Organizing Larger Stylesheets with @layer

Cascade layers let you decide which group of styles wins without fighting specificity:

```

@layer reset, base, components, utilities;

@layer base {
  body { font-family: system-ui, sans-serif; line-height: 1.6; }
}

```

2026 CSS Best Practices

- **Use an external stylesheet** linked from the head, not repeated inline style attributes.
- **Start with a small reset:** `box-sizing: border-box`, and `img { max-width: 100%; height: auto; }`.
- **Prefer classes** over IDs and avoid `!important`. Keep selectors short.
- **Use rem and relative units** for text and spacing, and `clamp()` for fluid sizes.
- **Lay out with Flexbox and Grid.** Do not use floats or tables for layout.
- **Design mobile-first** and test at narrow, medium and wide screen sizes.
- **Keep text readable:** contrast of at least 4.5:1 for body text, a line length near 60 to 75 characters and a line-height around 1.5.
- **Never remove focus outlines** without giving a clear replacement. Use `:focus-visible`.
- **Respect user settings:** `prefers-reduced-motion` and `prefers-color-scheme`.
- **Use variables** for colors, spacing and fonts so changes happen in one place.
- **Keep it fast:** few font files (woff2), compress images and avoid animating layout properties.
- **Check support** at caniuse.com or MDN before using a very new feature, and add a fallback where needed.
- **Format and lint** your CSS with Prettier and Stylelint so it stays tidy.

Thank You for Choosing BioGlytch Studios LLC!

At BioGlytch Studios LLC, we express our gratitude for choosing to embark on this coding journey with us! Whether you're a dedicated learner or a parent nurturing the next software engineer, your trust means the world to us.

Our mission extends beyond teaching coding skills. We aim to cultivate critical thinking, logical outlooks, problem-solving skills, discipline, and career-ready structures. The evolving landscape of technology should be accessible to all, and we're here to make that happen.

As you progress through the activities, each successful assessment earns you a certificate from BioGlytch Studios LLC, a testament to your accomplishments. Your journey is not just about learning to code; it's about becoming the greatest engineer you can be.

Unveiling CSS

Welcome to the world of Cascading Style Sheets (CSS)! As you delve into the intricacies of styling, remember the power of rules and selectors. Whether you're defining styles for headers, mastering class and ID selectors, or exploring text styling, your journey is filled with creativity.

The CSS journey covers everything from the box model to background styling, list styling, link styling, table styling, form styling, layout styling, positioning styling, transforms, transitions, and animations. Each concept adds a layer to your coding expertise, transforming you into a versatile developer.

Remember, the flex layout and positioning properties give you control, and transformations open a realm of possibilities. Dive into transitions and animations to bring life to your creations.

Your Journey, Your Achievement

Completing this book marks a significant milestone in your coding journey. Your commitment to learning and growing is commendable. As you showcase your best code forward, know that BioGlytch Studios LLC is proud to be a part of your success.

Clarence Scott,

CEO & Founder

BioGlytch Studios LLC