CSS
From Beginner to Advanced

A Complete Property-by-Property Guide with Hands-On Exercises

Prepared by
ZolaXTech

zolaxtech.com.et

Written and practiced entirely in Visual Studio Code
August 2026

How to Use This Guide

This guide continues directly from the zolaxtech.com.et HTML5 guide and takes you from a bare, unstyled page to a fully responsive, animated, professionally organized stylesheet, using Visual Studio Code throughout. Every property or concept is presented the same way, so you always know what to expect:

- **The property or concept:** shown in a monospace font, e.g. justify-content.
- **What it does:** a plain-language explanation of its purpose and behavior.
- **Values / Details:** a table of the property's most important values and exactly what each one does.
- **Example:** a real, working code snippet you can type directly into VS Code's styles.css.
- **Exercise:** a short hands-on task, using zolaxtech.com.et as the running example, so you practice each concept immediately instead of just reading about it.

Each chapter ends with a larger "Chapter Project" exercise that combines everything covered in that chapter into one realistic piece of the Zolax Tech website. By the final chapter, you will have styled a complete, responsive, accessible, professionally organized multi-page business website built and previewed entirely inside Visual Studio Code.

Table of Contents

Chapter 1: Getting Started with CSS in Visual Studio Code

CSS (Cascading Style Sheets) controls how your HTML looks: colors, fonts, spacing, layout, and animation. This chapter sets up your workspace so you can write, save, and preview CSS instantly, continuing the zolaxtech.com.et project.

Step 1: Reuse your project folder

Open the same zolaxtech-site folder from the HTML course in VS Code (File > Open Folder). You should already have index.html and an empty styles.css inside it.

Step 2: Confirm Live Server is installed

If you skipped the HTML course, install the "Live Server" extension from the Extensions panel (Ctrl+Shift+X) so saved CSS changes refresh the browser automatically.

Step 3: Install a CSS-focused extension

Search for and install "CSS Peek" (jumps from a class in your HTML straight to its CSS rule) and confirm the built-in "CSS Language Features" are enabled (they are on by default in VS Code).

Step 4: Link your stylesheet

Open index.html and confirm the <head> contains <link rel="stylesheet" href="styles.css">. Every page that should share the same look must link to this same file.

Step 5: Write your first rule

In styles.css, type body {and press Enter VS Code will auto-suggest properties as you type. Add background-color: honeydew; and save with Ctrl+S. With Live Server running, the browser background should change instantly.

Step 6: Use the color picker

Click the small colored square VS Code shows next to any color value (like honeydew or #ff0000) to open a visual color picker a fast way to experiment with colors without memorizing codes.

Step 7: Format on save

Open Settings (Ctrl+,), search for "format on save", and enable it. VS Code will now auto-indent and clean up your CSS every time you save, keeping your stylesheet readable as it grows.

Chapter Summary Project

🏁 Chapter Project

Confirm styles.css is linked correctly from index.html by adding body { background-color: honeydew; } and watching Live Server refresh the browser. Then add a second temporary rule, h1 { color: crimson; }, save, and confirm both changes appear together.

Chapter 2: CSS Syntax & Ways to Add CSS

Before diving into individual properties, you need to understand exactly how a CSS rule is built, and the three different places CSS can live.

Anatomy of a CSS Rule

Every CSS rule has the same shape: a selector (what to style), followed by a declaration block in curly braces containing one or more declarations. Each declaration is a property and a value, separated by a colon, and ended with a semicolon.

Values / Details

Property / Value	What it does
Selector	Chooses which HTML element(s) the rule applies to, e.g. h1, .card, or #logo.
Property	The aspect of the element you want to change, e.g. color, margin, font-size.
Value	The setting applied to that property, e.g. red, 20px, bold.
Declaration block	The full { property: value; property: value; } group attached to one selector.

Example

```
h1 {
  color: navy;
  font-size: 32px;
}
```

> **➖ Exercise**
> In styles.css, write a rule for the h1 selector that sets both color and font-size, then identify (in a comment above it) which part is the selector, which is the property, and which is the value.

Inline CSS (style attribute)

CSS written directly on a single HTML element using the style attribute. It applies only to that one element and overrides almost everything else, which makes it convenient for one-off tests but bad for real maintainable projects.

Example

```
<p style="color: red; font-weight: bold;">This paragraph is styled inline.</p>
```

> **➖ Exercise**
> Add an inline style to one paragraph on your homepage that temporarily changes its color, purely to see the effect then remove it, since real projects should avoid inline styles.

Internal CSS (<style> in <head>)

CSS written inside a <style> tag in the HTML document's <head>. It applies to the whole page it's written in, but has to be repeated on every page not reusable across a multi-page site like zolaxtech.com.et.

Example

```
<head>
  <style>
    body { font-family: Arial, sans-serif; }
    h1 { color: darkblue; }
  </style>
</head>
```

> **➖ Exercise**
>
> Add a temporary <style> block to about.html (a different page than index.html) and confirm the rules only affect that page, not index.html.

External CSS (linked file)

CSS written in a separate .css file and linked from the HTML with <link rel="stylesheet">. This is the professional standard: one file can style every page on a site, browsers can cache it separately from the HTML, and it keeps content and presentation cleanly separated.

Example

```
<!-- in every page's <head> -->
<link rel="stylesheet" href="styles.css">

/* in styles.css */
body {
  font-family: Arial, sans-serif;
  margin: 0;
}
```

> **➖ Exercise**
>
> Make sure every page of your zolaxtech.com.et project (index.html, about.html, services.html, contact.html) links to the same styles.css, and confirm a single rule change (like a new background color) applies to all of them at once.

Comments

Notes in a CSS file that the browser ignores completely, written between /* and */. Comments can span multiple lines and are extremely useful for labeling sections of a growing stylesheet.

Example

```
/* ==========================
   Header Styles
   ========================== */
header {
```

```
 background-color: navy;
}
```

> **➖ Exercise**
> Organize your growing styles.css into labeled sections using comments: /* Reset */, /* Header */, /* Main Content */, and /* Footer */, even if some sections are still empty.

The Cascade & Specificity (overview)

When multiple rules target the same element, the browser decides which one wins using three factors, in order: importance (! important beats everything, but should rarely be used), specificity (how precise the selector is IDs beat classes, classes beat tags), and source order (when specificity ties, the rule written LAST in the file wins).

Values / Details

Property / Value	What it does
!important	Forces a declaration to override any other, regardless of specificity an escape hatch to avoid, since it makes debugging much harder.
Specificity order (low to high)	Type selectors (p, div) < class/attribute/pseudo-class selectors (.card, [type], :hover) < ID selectors (#logo) < inline style attribute.
Source order	If two rules have identical specificity, whichever rule appears later in the CSS file (or later <link>/<style>) wins.

Example

```
p { color: blue; }        /* specificity: 0-0-1 */
. intro {color: green;}  /* specificity: 0-1-0  wins over the tag rule */
#main-intro { color: red; } /* specificity: 1-0-0  wins over the class rule */
```

> **➖ Exercise**
> Create a paragraph with class="intro" and id="main-intro". Write three conflicting color rules for p, .intro, and #main-intro. Predict which color will actually display, then check the browser to confirm your understanding of specificity.

Chapter Summary Project

> **🏁 Chapter Project**
> Refactor your zolaxtech.com.et project so that ALL styling lives in one external styles.css linked from every page, with zero inline style attributes remaining and clearly commented sections. Deliberately create one specificity conflict (a tag rule and a class rule targeting the same element with different colors) and add a code comment explaining which one wins and why.

Chapter 3: Selectors

Selectors are how you tell CSS exactly which elements to target. This chapter covers the full range, from the simplest to the most precise.

Type (element) selector

Targets every instance of a given HTML tag on the page.

Example

```
p {
  line-height: 1.6;
}
```

> **➖ Exercise**
> Write a type selector rule that sets line-height on every <p> across your site.

Class selector (.classname)

Targets every element carrying a given class attribute. The most commonly used selector in real projects because it's reusable across many elements.

Example

```
.card {
  border: 1px solid #ddd;
  padding: 16px;
}
```

> **➖ Exercise**
> Give three of your service /<article> elements a shared class="card" and write one .card rule that styles all of them identically.

ID selector (#idname)

Targets the single element with a matching id attribute. Since ids must be unique per page, this selector should be reserved for one-off elements like a page's main header or a specific anchor target not for anything repeated.

Example

```
#site-header {
  background-color: navy;
  color: white;
}
```

> **➖ Exercise**
> Give your <header> an id="site-header" and write a matching #site-header rule.

Universal selector (*)

Targets every single element on the page. Most commonly used in a CSS reset to zero out default browser margins/padding before applying your own spacing.

Example

```
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}
```

> **➖ Exercise**
> Add a universal-selector reset rule to the very top of styles.css that zeroes margin/padding and sets box-sizing to border-box for every element.

Grouping selector (comma)

Applies the same declaration block to several different selectors at once by separating them with commas, avoiding repeated code.

Example

```
h1, h2, h3 {
  font-family: Georgia, serif;
  color: #1f3864;
}
```

> **➖ Exercise**
> Write one grouped rule that gives all your heading levels (h1, h2, h3) the same font-family in a single declaration block.

Descendant combinator (space)

Targets an element that is nested ANYWHERE inside another element, at any depth.

Example

```
nav a {
  text-decoration: none;
}
```

> **➖ Exercise**
> Write a descendant selector that removes the underline from every <a> that lives inside your <nav>, without affecting links elsewhere on the page.

Child combinator (>)

Targets an element that is a DIRECT child of another one level deep only, not grandchildren.

Example

```
nav > ul > li {
  display: inline-block;
}
```

> **➖ Exercise**
> Write a child-combinator selector that targets only the direct <li> children of your nav's <ul>, and set their display to inline-block to lay the menu out horizontally.

Adjacent sibling combinator (+)

Targets an element that comes immediately after another specific element, sharing the same parent.

Example

```
h2 + p {
  margin-top: 0;
}
```

> **➖ Exercise**
> Write a rule using + that removes the top margin from any paragraph that comes immediately after an h2, tightening the space between a heading and its first paragraph.

General sibling combinator (~)

Targets any element that comes after another specific element and shares the same parent, not just the immediately adjacent one.

Example

```
h2 ~ p {
  color: #444;
}
```

> **➖ Exercise**
> Write a rule using ~ that gives every paragraph following an h2 (anywhere after it, not just the first) a slightly muted text color.

Attribute selectors

Targets elements based on the presence or value of an HTML attribute extremely useful for styling form inputs by type without adding extra classes.

Values / Details

Property / Value	What it does
[attr]	Matches any element that has the attribute at all, regardless of value, e.g. [required].
[attr="value"]	Matches only when the attribute equals an exact value, e.g. [type="email"].
[attr^="value"]	Matches when the attribute's value STARTS WITH the given text, e.g. [href^="https"].
[attr$="value"]	Matches when the attribute's value ENDS WITH the given text, e.g. [href$=".pdf"].
[attr*="value"]	Matches when the attribute's value CONTAINS the given text anywhere, e.g. [class*="card"].

Example

```
input[type="email"] {
  border-color: steelblue;
}
a[href$=".pdf"] {
  color: darkred;
}
```

> **➖ Exercise**
> Write an attribute selector that gives every input[type="email"] on your quote form a distinct border color, and another that colors links ending in .pdf differently from regular links.

Pseudo-class selectors

Target an element based on a special STATE it's currently in, rather than its type or attributes.

Values / Details

Property / Value	What it does
:hover	Matches while the mouse pointer is over the element.
:focus	Matches while the element (usually a form field) has keyboard focus.
:active	Matches while the element is being clicked/pressed.
:visited	Matches links the user has already visited (limited styling allowed for privacy reasons).
:first-child / :last-child	Matches an element only if it is the first/last child of its parent.
:nth-child(n)	Matches elements by position using a formula, e.g. :nth-child(2n) for every even item, or :nth-child(3) for exactly the third.

Property / Value	What it does
:not(selector)	Matches any element that does NOT match the selector inside the parentheses.
:checked	Matches a checkbox or radio input that is currently checked.
:disabled	Matches a form element that is currently disabled.

Example

```
a:hover {
  color: crimson;
}
input:focus {
  outline: 2px solid steelblue;
}
li:nth-child(2n) {
  background-color: #f5f5f5;
}
button:not(.primary) {
  background: #eee;
}
```

> **Exercise**
> Add a :hover rule that changes link color across your site, a :focus rule that gives form inputs a visible outline, and an :nth-child(2n) rule that stripes every other row of your pricing table for readability.

Pseudo-element selectors

Target a specific PART of an element, or generate new content, rather than the whole element.

Values / Details

Property / Value	What it does
::before	Inserts generated content immediately before an element's actual content (requires a content property, even if empty).
::after	Inserts generated content immediately after an element's actual content.
::first-line	Targets only the first rendered line of a block of text.
::first-letter	Targets only the first letter of a block of text, useful for decorative drop caps.
::selection	Styles the portion of text currently highlighted/selected by the user.

Example

```
blockquote::before {
  content: "\201C";
  font-size: 2em;
  color: #ccc;
}
p::first-letter {
  font-size: 1.6em;
  font-weight: bold;
}
```

> **✏ Exercise**
> Use ::before to add a decorative opening quotation mark to your client testimonial <blockquote>, and use ::first-letter to create a subtle drop-cap effect on your homepage's introductory paragraph.

Chapter Summary Project

> **🏁 Chapter Project**
> Build a 'Selectors Showcase' section in styles.css for zolaxtech.com.et that uses at least ten different selector types from this chapter together: a class selector for repeated cards, an ID selector for the header, a child combinator for your nav menu, an attribute selector for a specific input type, at least two pseudo-classes (:hover and :nth-child), and at least one pseudo-element (::before or ::after).

Chapter 4: The Box Model

Every single element on a page is a rectangular box. Understanding exactly how that box's size is calculated is the single most important concept in CSS layout.

content, padding, border, margin (the four layers)

From the inside out, every box is made of four layers: the content itself (text/images), padding (space between content and border), the border (a visible or invisible line around the padding), and margin (space outside the border, separating this box from others). Picture concentric rectangles, each layer wrapping the one before it.

Example

```
/* Conceptually, from inside out: */
.box {
  /* content: whatever is inside the element */
  padding: 20px; /* space around content, inside the border */
  border: 2px solid #333; /* the visible edge */
  margin: 16px;   /* space outside the border, pushing other boxes away */
}
```

> **➼ Exercise**
> Create a test with some text inside it, then apply the example rule and use your browser's DevTools (right-click > Inspect) to hover over the element and see the box model diagram it shows for padding/border/margin.

width & height

Sets the size of an element's content box (by default) in a chosen unit. Percentages are relative to the parent element's size.

Values / Details

Property / Value	What it does
px	A fixed number of pixels predictable but not flexible.
%	A percentage of the parent element's width/height.
auto	Let’s the browser calculate the size automatically (the default for block elements' width).
min-width / max-width	Set a floor or ceiling the element's width can't cross, even if width or content would otherwise push past it essential for responsive images and cards.

Example

```
.hero-image {
  width: 100%;
  max-width: 800px;
  height: auto;
}
```

> **➼ Exercise**
> Set your homepage's hero image to width: 100% with a max-width of 800px and height: auto, then resize your browser window narrower and wider to see it scale responsively without ever exceeding 800px.

padding

Space between an element's content and its border. Can be set for all four sides at once or individually.

Values / Details

Property / Value	What it does
padding: 10px;	Applies 10px to all four sides.
padding: 10px 20px;	10px top & bottom, 20px left & right.
padding: 10px 20px 15px 5px;	Top, right, bottom, left, in that clockwise order.

Property / Value	What it does
padding-top / -right / -bottom / -left	Sets a single side individually.

Example

```
.card {
  padding: 20px;
}
.card-header {
  padding: 10px 20px 15px 20px;
}
```

> **➖ Exercise**
> Give your service cards 20px of padding on all sides, then give just the card's header area extra padding-bottom to separate it visually from the body text below it.

margin

Space outside an element's border, pushing away neighboring elements. Uses the same shorthand patterns as padding. Vertical margins between two block elements can 'collapse' into a single margin equal to the larger of the two a common source of confusion for beginners.

Values / Details

Property / Value	What it does
margin: 0 auto;	The classic centering trick: zero top/bottom margin, and auto left/right margin, which centers a block element horizontally (the element must have a set width).
negative margins	A negative value pulls an element closer to (or overlapping) its neighbor used carefully for fine-tuned overlaps.

Example

```
.container {
  width: 1000px;
  max-width: 90%;
  margin: 0 auto;
}
```

> **➖ Exercise**
> Create a .container class with a fixed width and margin: 0 auto, wrap your page's main content in it, and confirm it stays centered as you resize the browser window.

border

Draws a line around an element's padding. Shorthand combines width, style, and color into one declaration.

Values / Details

Property / Value	What it does
border-width	Thickness of the line, e.g. 1px, 2px, thick.
border-style	solid, dashed, dotted, double, or none (the default without a style, width and color have no visible effect).
border-color	The color of the line.
border-radius	Rounds the corners of the box; a large enough value relative to the box's size produces a circle or pill shape.

Example

```
.card {
  border: 1px solid #cccccc;
  border-radius: 8px;
}
.avatar {
  border-radius: 50%;
}
```

> **✏ Exercise**
> Give your service cards a light 1px border with 8px rounded corners, and give a profile/team photo a border-radius of 50% to turn it into a perfect circle (the image must be square first).

box-sizing

Controls HOW the width/height you set is calculated. This single property resolves most beginner confusion about the box model.

Values / Details

Property / Value	What it does
content-box (default)	width/height apply ONLY to the content padding and border are added ON TOP, making the element's actual rendered size larger than what you typed.
border-box	width/height include content, padding, AND border all together the element's rendered size matches exactly what you typed, which is far more predictable. Almost all modern CSS resets set this globally.

Example

```
* {
  box-sizing: border-box;
}
.box {
  width: 300px;
  padding: 20px;
```

```
  border: 5px solid black;
  /* with border-box, this box is STILL exactly 300px wide total */
}
```

> **➖ Exercise**
> Create two identical 300px-wide boxes with 20px padding and a 5px border side by side one with box-sizing: content-box and one with border-box. Inspect both in DevTools and compare their actual rendered widths to see the difference firsthand.

overflow

Controls what happens when content is too large to fit inside its box.

Values / Details

Property / Value	What it does
visible (default)	Content spills outside the box, uncontained.
hidden	Anything beyond the box's edge is simply clipped and invisible.
scroll	Always shows scrollbars, even if content fits.
auto	Shows scrollbars only when content actually overflows the most commonly used value.

Example

```
.scrollable-panel {
  height: 200px;
  overflow-y: auto;
}
```

> **➖ Exercise**
> Create a small fixed-height with a long block of placeholder text inside it, and set overflow-y: auto so it becomes scrollable instead of spilling out of its box.

Chapter Summary Project

> **🏁 Chapter Project**
> Redesign your service cards on zolaxtech.com.et applying full box-model control: set box-sizing: border-box globally, give each card a fixed width with padding, a border with rounded corners, and consistent margin between cards using a shared .card class. Use DevTools to inspect one card and confirm its total rendered width matches what you intended.

Chapter 5: Colors, Backgrounds & Shadows

This chapter covers every way to specify color in CSS and the properties that use color to bring your zolaxtech.com.et pages to life.

Color value formats

CSS accepts several different ways of writing the same color pick whichever is clearest for the situation.

Values / Details

Property / Value	What it does
Named colors	Plain English keywords like red, navy, or steelblue over 140 exist, readable but limited.
Hexadecimal	#RRGGBB (or shorthand #RGB), e.g. #1f3864 the most common format, compact and precise.
rgb() / rgba()	rgb(31, 56, 100) sets red/green/blue values 0-255; rgba() adds a fourth alpha value from 0 (transparent) to 1 (opaque).
hsl() / hsla()	hsl(210, 53%, 26%) sets hue (0-360 on the color wheel), saturation %, and lightness % often more intuitive for tweaking a color's shade.
currentColor	A special keyword that reuses whatever the element's current text color is useful for making borders or icons automatically match text color.

Example

```
.alert {
  color: #b30000;
  background-color: rgba(255, 0, 0, 0.08);
  border: 1px solid hsl(0, 100%, 35%);
}
```

> **➖ Exercise**
> Pick a primary brand color for Zolax Tech and write it in all three formats (hex, rgb, hsl) as CSS comments above a rule that uses it, to practice converting between them.

color

Sets the color of an element's text (and, via currentColor, potentially its borders/icons too).

Example

```
h1 {
  color: #1f3864;
}
```

> ➼ **Exercise**
> Set a consistent text color for all your headings that matches your chosen Zolax Tech brand color.

background-color

Sets a solid background color behind an element's content and padding.

Example

```
header {
  background-color: #1f3864;
  color: white;
}
```

> ➼ **Exercise**
> Give your <header> a dark background-color and set its text color to white so it remains readable.

background-image

Sets an image as an element's background, often combined with several other background-* properties to control how it displays.

Values / Details

Property / Value	What it does
url('path')	The path to the image file.
linear-gradient()	Generates a smooth color gradient instead of a photo, e.g. linear-gradient(to right, #1f3864, #2e74b5).

Example

```
.hero {
  background-image: linear-gradient(to right, #1f3864, #2e74b5);
  color: white;
}
```

> ➼ **Exercise**
> Give your homepage's hero/banner section a linear-gradient background using two shades of your brand color.

background-size, -position, -repeat

Fine-tune how a background image fills its box.

Values / Details

Property / Value	What it does
background-size: cover	Scales the image to completely fill the box, cropping if needed, preserving aspect ratio the most common choice for hero banners.
background-size: contain	Scales the image to fit entirely within the box without cropping, possibly leaving empty space.
background-position	Positions the image within its box, e.g. center center, top right.
background-repeat: no-repeat	Prevents a smaller image from tiling repeatedly (the default behavior).

Example

```
.hero {
  background-image: url('hero.jpg');
  background-size: cover;
  background-position: center;
  background-repeat: no-repeat;
}
```

> **Exercise**
> Add a full-width hero banner image to your homepage using background-image, with size: cover, position: center, and repeat: no-repeat so it fills the space cleanly at any screen width.

box-shadow

Adds a shadow effect around an element's box, commonly used to lift cards or buttons visually off the page.

Values / Details

Property / Value	What it does
offset-x offset-y	How far the shadow is shifted horizontally and vertically.
blur-radius	How soft/spread-out the shadow's edge is (0 = a hard edge).
spread-radius (optional)	Expands or shrinks the shadow's size beyond the element's box.
color	The shadow's color, usually a semi-transparent black via rgba().
inset	An optional keyword that flips the shadow to appear INSIDE the box instead of outside.

Example

```
.card {
  box-shadow: 0 4px 12px rgba(0, 0, 0, 0.1);
}
.card:hover {
```

```
  box-shadow: 0 8px 20px rgba(0, 0, 0, 0.18);
}
```

> **➖ Exercise**
> Add a subtle box-shadow to your service cards, and use a :hover rule to make the shadow grow larger/darker when the visitor's mouse is over the card, giving it a 'lifting' effect.

opacity

Controls the transparency of an ENTIRE element, including its text and children 0 is fully invisible, 1 is fully opaque. Different from a transparent background-color, which only affects the background layer.

Example

```
.watermark {
  opacity: 0.3;
}
```

> **➖ Exercise**
> Create a small decorative element (like a background logo mark) and set its opacity to 0.3 so it appears faded/watermarked behind other content.

Chapter Summary Project

> **🏁 Chapter Project**
> Design a complete color scheme for zolaxtech.com.et: define a primary and accent brand color (in hex), apply them consistently across headers/buttons/links, add a gradient hero banner, give every card a box-shadow that intensifies on hover, and confirm your text/background color combinations have enough contrast to remain easily readable.

Chapter 6: Typography

Text is most of what visitors actually read on zolaxtech.com.et this chapter covers every property that controls how it looks.

font-family

Sets the typeface. Always provide a comma-separated 'font stack' ending in a generic family, so the browser has a fallback if the first choice isn't installed/loaded.

Values / Details

Property / Value	What it does
Generic families	serif (has small foot strokes, formal), sans-serif (clean, modern), monospace (fixed-width, for code), cursive, fantasy.

Example

```
body {
  font-family: 'Segoe UI', Arial, sans-serif;
}
code, pre {
  font-family: 'Consolas', 'Courier New', monospace;
}
```

> **➖ Exercise**
> Set a font stack for your body text ending in sans-serif, and a separate monospace stack for any <code> or <pre> elements on your site.

@font-face & web fonts

Let’s you load a custom font file (or, more commonly, a font served by a provider like Google Fonts) instead of relying only on fonts already installed on the visitor's device.

Values / Details

Property / Value	What it does
@font-face { }	Defines a custom font by pointing to font file(s) with the src property, then giving it a font-family name to use elsewhere.
<link> method (Google Fonts)	Simpler in practice: paste a provided <link> tag in your HTML <head>, then reference the font's name in font-family as normal.

Example

```
<!-- in <head> -->
<link href="https://fonts.googleapis.com/css2?family=Poppins:wght@400;700&display=swap" rel="stylesheet">

/* in styles.css */
h1, h2, h3 {
  font-family: 'Poppins', sans-serif;
}
```

> **➖ Exercise**
> Pick a free Google Font that fits a tech company's branding, link it in your HTML <head>, and apply it to your headings via font-family, keeping sans-serif as the fallback.

font-size

Sets the size of text. Relative units are strongly preferred over fixed pixels for accessibility, since they respect a visitor's browser zoom/font-size preferences.

Values / Details

Property / Value	What it does
px	A fixed pixel size simple but doesn't scale with user preferences.
em	Relative to the font-size of the PARENT element compounds when nested, which can cause surprises.
rem	Relative to the font-size of the ROOT (<html>) element only predictable and the generally recommended unit for font sizing.
%	A percentage of the parent's font-size, similar behavior to em.

Example

```
html {
  font-size: 16px; /* the 1rem baseline */
}
h1 {
  font-size: 2.5rem; /* 40px */
}
p {
  font-size: 1rem; /* 16px */
}
```

> **➖ Exercise**
> Set your <html> element's font-size as your rem baseline, then define all your heading and paragraph font sizes using rem instead of px, and confirm they still scale correctly.

font-weight

Sets how bold or light text appears.

Values / Details

Property / Value	What it does
normal / bold	The two most common keyword values.
100–900	Numeric scale for fonts that support multiple weights (100 = thinnest, 900 = boldest); the font file must actually include that weight to render correctly.

Example

```
h1 {
  font-weight: 700;
}
.fine-print {
  font-weight: 300;
}
```

> ➼ **Exercise**
> If you loaded a Google Font with multiple weights in the previous exercise, set your h1 to weight 700 and a fine-print note to a lighter weight like 300.

font-style

Toggles italic text.

Values / Details

Property / Value	What it does
normal / italic / oblique	italic uses the font's true italic design if available; oblique artificially slants the normal style.

Example

```
blockquote {
  font-style: italic;
}
```

> ➼ **Exercise**
> Set your testimonial <blockquote> text to italic.

text-align

Horizontally aligns inline content (text, inline images) within its block container.

Values / Details

Property / Value	What it does
left / right / center / justify	justify stretches each line to fill the full width by adjusting word spacing, similar to a newspaper column.

Example

```
.hero h1 {
  text-align: center;
}
```

> ➼ **Exercise**
> Center-align the heading text inside your hero banner section.

text-decoration

Adds or removes lines on text most commonly used to remove the default underline from links.

Values / Details

Property / Value	What it does
underline / overline / line-through / none	The type of line, or none to remove any default decoration entirely.

Example

```
nav a {
  text-decoration: none;
}
nav a:hover {
  text-decoration: underline;
}
```

> **➖ Exercise**
> Remove the default underline from your navigation links, then add it back only on :hover so visitors still get a clear interactive cue.

text-transform

Changes the capitalization of text visually, without altering the actual underlying HTML content.

Values / Details

Property / Value	What it does
uppercase / lowercase / capitalize / none	capitalize uppercases just the first letter of each word.

Example

```
nav a {
  text-transform: uppercase;
  letter-spacing: 1px;
}
```

> **➖ Exercise**
> Make your navigation menu links display in uppercase with a small amount of extra letter-spacing for a polished look.

line-height

Sets the vertical space a line of text occupies, directly affecting readability of paragraphs. A unitless number (like 1.5) is recommended, since it scales proportionally with font-size.

Example

```
p {
  line-height: 1.6;
}
```

➖ Exercise

Set line-height: 1.6 on your body paragraphs and compare readability to the browser's cramped default of roughly 1.2.

letter-spacing & word-spacing

Adjust the horizontal space between individual letters or whole words, respectively useful for stylistic headings or uppercase labels, but should be used sparingly on body text.

Example

```
h1 {
  letter-spacing: 0.5px;
}
```

➖ Exercise

Add a small positive letter-spacing to one of your headings and observe the effect; try a negative value too and note how it can hurt readability if overused.

text-shadow

Adds a shadow behind text, using the same offset-x/offset-y/blur/color pattern as box-shadow often used to keep text legible over a busy background image.

Example

```
.hero h1 {
  color: white;
  text-shadow: 0 2px 6px rgba(0, 0, 0, 0.5);
}
```

➖ Exercise

Add a text-shadow to your white hero heading so it stays legible over your gradient or image background from Chapter 5.

Chapter Summary Project

🏁 Chapter Project

Establish a complete typographic system for zolaxtech.com.et: load one Google Font for headings and choose a system sans-serif stack for body text, define a rem-based sizing scale for h1 through p, set a comfortable line-height on paragraphs, style your nav links with text-transform and a hover text-decoration, and add a text-shadow to your hero heading so it stays readable over its background.

Chapter 7: Display, Positioning & Stacking

This chapter explains how elements are placed on the page: their default flow, how to take them out of that flow, and how overlapping elements are stacked.

display

The single most powerful property for controlling an element's layout behavior.

Values / Details

Property / Value	What it does
block	Takes up the full width available, always starts on a new line the default for elements like , <p>, <h1>.
inline	Takes up only as much width as its content needs, sits in the flow of surrounding text, and IGNORES width/height/vertical margin the default for , <a>, .
inline-block	Behaves like inline (sits alongside other content) but DOES respect width, height, and margin like a block a useful middle ground, often used before Flexbox existed.
none	Removes the element entirely from the page and the layout, as if it wasn't in the HTML at all (different from visibility: hidden, which keeps its space reserved).
flex / grid	Turns the element into a flex or grid CONTAINER, changing how its direct children are laid out covered fully in Chapters 8 and 9.

Example

```
nav ul {
  display: flex;
}
.hidden-on-mobile {
  display: none;
}
```

> **➖ Exercise**
> Find one element you'd like to completely hide on your site under certain conditions (like a decorative sidebar) and toggle it with display: none, then confirm the space it occupied collapses completely.

visibility

Hides an element visually, but unlike display: none the space it would have occupied REMAINS reserved in the layout, just empty.

Values / Details

Property / Value	What it does
visible / hidden	The two main values.

Example

```
.placeholder-icon {
  visibility: hidden;
}
```

> **➖ Exercise**
> Compare display: none and visibility: hidden side by side on two identical test elements to see the layout difference.

position

Controls how an element is positioned relative to its normal place in the flow, its parent, or the whole page.

Values / Details

Property / Value	What it does
static (default)	Normal document flow; top/right/bottom/left have no effect.
relative	Stays in normal flow but can be nudged from its original spot using top/right/bottom/left, and importantly becomes the reference point for any absolutely positioned children.
absolute	Removed from normal flow entirely and positioned relative to its nearest ancestor that has position set to anything other than static (or the page itself if none exists).
fixed	Removed from normal flow and positioned relative to the BROWSER WINDOW, staying in place even when the page scrolls commonly used for a sticky header or a 'back to top' button.
sticky	Behaves like relative until the page scrolls to a certain point, then 'sticks' like fixed within its parent commonly used for section headers that stay visible while their section scrolls past.

Example

```
.site-header {
  position: sticky;
  top: 0;
}
.badge {
  position: absolute;
  top: 8px;
  right: 8px;
}
```

> **➖ Exercise**
> Make your site's <header> position: sticky with top: 0 so it stays visible as the visitor scrolls down the page. Then add a small 'New' badge to one service card using position: absolute inside a position: relative card.

top, right, bottom, left

Used together with any position value other than static to specify exactly how far to offset the element from each edge of its positioning context.

Example

```
.back-to-top {
  position: fixed;
  bottom: 24px;
  right: 24px;
}
```

> **➖ Exercise**
> Add a 'Back to Top' link or button fixed to the bottom-right corner of the viewport using position: fixed with bottom and right offsets.

z-index

Controls the STACKING ORDER of overlapping positioned elements higher numbers appear on top of lower numbers. Only has an effect on elements whose position is NOT static.

Example

```
.modal-overlay {
  position: fixed;
  inset: 0;
  z-index: 1000;
}
```

> **➖ Exercise**
> Give your 'Back to Top' button from the previous exercise a high z-index (like 999) to guarantee it always appears above every other element on the page, even ones that might otherwise overlap it.

float & clear (legacy layout)

The original way to create side-by-side layouts before Flexbox and Grid existed. Still occasionally used for wrapping text around an image, but no longer recommended for overall page layout.

Values / Details

Property / Value	What it does
float: left / right	Pulls an element to one side and lets inline content (like text) wrap around it.

Property / Value	What it does
clear: left / right / both	Prevents an element from wrapping next to a floated element, pushing it below instead.

Example

```
img.inline-photo {
  float: left;
  margin-right: 16px;
}
.section-end {
  clear: both;
}
```

> **➖ Exercise**
>
> Try floating a small image to the left inside a paragraph of text on an about-us style page, and observe how the text wraps around it this is float's one remaining common real-world use.

Chapter Summary Project

> **🏁 Chapter Project**
>
> Apply positioning to zolaxtech.com.et: make your header sticky at the top of the viewport, add a fixed 'Back to Top' button in the bottom-right corner with an appropriate high z-index, and add one absolutely-positioned badge or label nested inside a relatively-positioned card. Test scrolling behavior in the browser to confirm each element behaves as intended.

Chapter 8: Flexbox One-Dimensional Layout

Flexbox arranges items along a single row or column, automatically handling spacing, alignment, and sizing. It is the modern default for navigation bars, card rows, and centering content.

display: flex

Turns an element into a flex container; every DIRECT child automatically becomes a flex item, laid out in a row by default.

Example

```
.card-row {
  display: flex;
}
```

> **➖ Exercise**
>
> Turn the container holding your three service cards into a flex container and observe them line up in a row automatically, without any float or inline-block needed.

flex-direction

Sets the main axis the flex items are laid out along.

Values / Details

Property / Value	What it does
row (default)	Items placed left-to-right in a horizontal line.
row-reverse	Items placed right-to-left.
column	Items placed top-to-bottom in a vertical line.
column-reverse	Items placed bottom-to-top.

Example

```
.sidebar-menu {
  display: flex;
  flex-direction: column;
}
```

> **➖ Exercise**
> Create a small vertical menu (like a sidebar list of links) using flex-direction: column.

justify-content

Aligns flex items along the MAIN axis (horizontally, if flex-direction is row).

Values / Details

Property / Value	What it does
flex-start (default)	Items packed toward the start.
flex-end	Items packed toward the end.
center	Items packed toward the center.
space-between	Equal space distributed BETWEEN items, none at the outer edges.
space-around	Equal space around EACH item, so edge items have half-space at the container's outer edges.
space-evenly	Perfectly equal space between every gap, including the outer edges.

Example

```
nav ul {
  display: flex;
  justify-content: space-between;
}
```

> **➼ Exercise**
> Set your nav menu's justify-content to space-between so your logo sits at one end and the menu links sit at the other, with the space evenly distributed.

align-items

Aligns flex items along the CROSS axis (vertically, if flex-direction is row) this single property solves the classic 'how do I vertically center this' problem that used to require workarounds.

Values / Details

Property / Value	What it does
stretch (default)	Items stretch to fill the container's cross-axis size.
flex-start / flex-end	Items align to the start/end of the cross axis.
center	Items are vertically centered (when direction is row).
baseline	Items align along their text baselines.

Example

```
.site-header {
  display: flex;
  align-items: center;
  justify-content: space-between;
}
```

> **➼ Exercise**
> Use align-items: center together with justify-content: space-between on your header so your logo and nav links are both perfectly vertically centered relative to each other, regardless of their individual heights.

flex-wrap

Controls whether flex items are forced onto a single line (potentially shrinking to fit) or allowed to wrap onto multiple lines when they run out of room.

Values / Details

Property / Value	What it does
nowrap (default)	All items forced onto one line, shrinking if necessary.
wrap	Items wrap onto additional lines as needed, like text wrapping.

Example

```
.card-row {
  display: flex;
  flex-wrap: wrap;
  gap: 20px;
}
```

> **➖ Exercise**
> Add flex-wrap: wrap to your card row so that on a narrow browser window, cards drop to a new line instead of squeezing/overflowing.

gap

Sets consistent spacing BETWEEN flex items, without adding unwanted margin on the outer edges of the first/last item the way manual margins would.

Values / Details

Property / Value	What it does
gap: 20px;	Equal gap in both directions.
row-gap / column-gap	Set the horizontal and vertical gap independently.

Example

```
.card-row {
  display: flex;
  gap: 24px;
}
```

> **➖ Exercise**
> Replace any manual margins between your cards with a single gap: 24px on the flex container instead.

flex-grow, flex-shrink, flex-basis (and flex shorthand)

Control how an INDIVIDUAL flex item grows or shrinks relative to its siblings to fill available space.

Values / Details

Property / Value	What it does
flex-grow	A unitless number representing how much of the extra available space this item should consume relative to siblings (0 = don't grow, the default).
flex-shrink	How much this item should shrink relative to siblings when there isn't enough room (1 = shrink normally, the default).
flex-basis	The item's starting size before growing/shrinking is calculated, similar to width.
flex: 1;	Common shorthand meaning 'grow to fill available space equally with siblings' (equivalent to flex-grow:1; flex-shrink:1; flex-basis:0).

Example

```
.card {
  flex: 1;
```

```
}
.sidebar {
  flex: 0 0 250px; /* never grow, never shrink, always 250px */
}
```

> **➖ Exercise**
> Set your three service cards to flex: 1 each so they always share the available row width equally, and create a sidebar element set to flex: 0 0 250px that always stays exactly 250px wide regardless of window size.

align-self

Overrides align-items for one specific flex item, letting a single item break away from how the rest of its siblings are aligned.

Example

```
.featured-card {
  align-self: flex-start;
}
```

> **➖ Exercise**
> Inside a flex row where align-items: center is set on the container, use align-self on ONE card to override it and align that single card to flex-start instead.

order

Changes the VISUAL order flex items appear in, independent of their order in the actual HTML source useful for reordering content on smaller screens without duplicating markup.

Example

```
@media (max-width: 600px) {
  .sidebar {
    order: -1; /* moves sidebar above main content on mobile */
  }
}
```

> **➖ Exercise**
> Using a media query (previewed here, covered fully in Chapter 10), set a sidebar element's order to -1 only on small screens, so it visually moves above your main content on mobile despite coming after it in the HTML.

Chapter Summary Project

🏁 Chapter Project

Rebuild your site's <header>, <nav>, and card-row layouts on zolaxtech.com.et entirely with Flexbox: a header using justify-content and align-items to position the logo and menu, a nav menu using flex-direction and gap, and a card row using flex-wrap and flex: 1 so cards resize evenly and wrap gracefully on narrow screens.

Chapter 9: CSS Grid Two-Dimensional Layout

While Flexbox excels at one dimension (a row OR a column), CSS Grid is built for laying out both rows AND columns at once ideal for full page layouts, image galleries, and pricing tables.

display: grid

Turns an element into a grid container; direct children become grid items placed into the grid you define.

Example

```
.gallery {
  display: grid;
}
```

➖ Exercise

Turn your portfolio/gallery section's container into a grid container as the first step before defining columns.

grid-template-columns

Defines the number and size of the grid's columns.

Values / Details

Property / Value	What it does
Fixed sizes	e.g. grid-template-columns: 200px 200px 200px; creates three 200px columns.
fr unit	A 'fraction' unit representing a share of the available space, e.g. 1fr 2fr splits space into one-third/two-thirds.
repeat()	A shorthand to avoid repetition, e.g. repeat(3, 1fr) is identical to 1fr 1fr 1fr.
auto-fit / auto-fill with minmax()	repeat(auto-fit, minmax(220px, 1fr)) automatically fits as many columns as will comfortably fit at a minimum width of 220px, wrapping responsively without any media queries at all.

Example

```
.gallery {
  display: grid;
```

```
  grid-template-columns: repeat(auto-fit, minmax(220px, 1fr));
  gap: 20px;
}
```

> **➖ Exercise**
> Build a responsive image gallery grid using repeat(auto-fit, minmax(220px, 1fr)) and a 20px gap, then resize your browser window and watch the number of columns adjust automatically with zero media queries.

grid-template-rows

Defines the number and size of the grid's rows, using the same size units as columns (fixed, fr, repeat, minmax).

Example

```
.page-layout {
  display: grid;
  grid-template-rows: auto 1fr auto;
  min-height: 100vh;
}
```

> **➖ Exercise**
> Build a full-page layout skeleton using grid-template-rows: auto 1fr auto for header/main/footer, with min-height: 100vh, so your footer sits at the bottom of the viewport even on short pages.

gap (row-gap, column-gap)

Sets spacing between grid rows and columns, just like it does for Flexbox.

Example

```
.gallery {
  gap: 24px 16px; /* row-gap column-gap */
}
```

> **➖ Exercise**
> Set different row-gap and column-gap values on your gallery grid to see them controlled independently.

grid-column & grid-row (item placement)

Placed on an individual grid ITEM (not the container) to make it span multiple columns or rows, or to position it at a specific spot in the grid.

Values / Details

Property / Value	What it does
grid-column: 1 / 3;	Spans from grid line 1 to grid line 3 (covering two columns).
grid-column: span 2;	A simpler way to say 'span 2 columns from wherever this item naturally falls'.
grid-row: 1 / 3;	Same concept applied vertically, spanning two rows.

Example

```
.gallery-item.featured {
  grid-column: span 2;
  grid-row: span 2;
}
```

> **➖ Exercise**
> In your image gallery grid, give one 'featured' item grid-column: span 2 and grid-row: span 2 so it appears twice as large as the surrounding items, like a magazine layout.

grid-template-areas

A highly readable way to lay out a whole page by giving each region a name and literally drawing the layout as a grid of names in the CSS itself.

Values / Details

Property / Value	What it does
grid-template-areas	Defined on the container as quoted rows of area names, e.g. "header header" "sidebar main" "footer footer".
grid-area (on each item)	Assigns a child element to one of the named areas defined above.

Example

```
.page {
  display: grid;
  grid-template-columns: 250px 1fr;
  grid-template-areas:
    "header header"
    "sidebar main"
    "footer footer";
}
header { grid-area: header; }
.sidebar { grid-area: sidebar; }
main { grid-area: main; }
footer { grid-area: footer; }
```

➖ Exercise

Lay out a full page template for zolaxtech.com.et using grid-template-areas with named header, sidebar, main, and footer regions, then assign each real element to its area with grid-area.

justify-items & align-items (Grid)

Similar to their Flexbox counterparts, but control how EVERY item aligns within its own grid cell horizontally (justify-items) and vertically (align-items).

Example

```
.gallery {
  justify-items: center;
  align-items: center;
}
```

➖ Exercise

Set justify-items and align-items to center on your gallery grid so every item is centered within its own cell, even if items are different sizes.

justify-self & align-self (Grid)

Override the containers justify-items/align-items for a single individual grid item.

Example

```
.gallery-item.cta {
  justify-self: end;
}
```

➖ Exercise

Pick one item in your grid and override its horizontal alignment within its cell using justify-self, independent of the rest of the grid.

Chapter Summary Project

🏁 Chapter Project

Design your zolaxtech.com.et portfolio/gallery page using CSS Grid: a responsive auto-fit grid of project cards with one deliberately larger 'featured' card using grid-column/grid-row span, and rebuild your overall page skeleton (header/sidebar/main/footer, if you have a sidebar) using named grid-template-areas for maximum readability.

Chapter 10: Responsive Design & Media Queries

Most visitors to zolaxtech.com.et will be on a phone. This chapter covers building layouts that adapt gracefully from small screens to large desktop monitors.

The viewport meta tag (recap)

Responsive CSS only works correctly if the HTML <head> includes the viewport meta tag, telling mobile browsers to render at the device's actual width instead of a zoomed-out desktop-sized layout.

Example

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

> **➖ Exercise**
> Confirm this exact meta tag is present in every page of your project responsive CSS silently fails on mobile without it.

Relative & viewport units

Units that scale relative to something else, rather than being a fixed pixel size, are the foundation of flexible layouts.

Values / Details

Property / Value	What it does
%	Relative to the parent element's corresponding dimension.
vw / vh	1vw = 1% of the viewport's width; 1vh = 1% of the viewport's height great for full-screen hero sections.
vmin / vmax	1% of whichever is smaller/larger, the viewport's width or height useful for elements that must fit regardless of orientation.
em / rem	Relative to font-size, as covered in Chapter 6 useful for spacing that should scale with text size.

Example

```
.hero {
  min-height: 60vh;
}
.container {
  width: 90%;
  max-width: 1200px;
}
```

> **➖ Exercise**
> Set your hero section's min-height using vh so it always fills roughly 60% of the visible screen height, on any device.

@media (media queries)

Applies a block of CSS rules ONLY when a certain condition about the browser/device is true most commonly, the viewport's width.

Values / Details

Property / Value	What it does
max-width	Applies the rules when the viewport is AT OR BELOW the given width the basis of a 'desktop-first' approach.
min-width	Applies the rules when the viewport is AT OR ABOVE the given width the basis of a 'mobile-first' approach (write base styles for mobile, then add complexity for larger screens).
orientation	Matches portrait or landscape device orientation.
prefers-color-scheme	Matches the visitor's OS-level light or dark mode preference, e.g. (prefers-color-scheme: dark).

Example

```
/* Mobile-first base styles above, then: */
@media (min-width: 768px) {
  .card-row {
    flex-direction: row;
  }
}
@media (max-width: 600px) {
  nav ul {
    flex-direction: column;
  }
}
```

> **➖ Exercise**
>
> Convert your nav menu to stack vertically (flex-direction: column) only on screens narrower than 600px, using a max-width media query, while it stays horizontal by default on larger screens.

Mobile-first vs desktop-first strategy

Mobile-first means writing your simplest, single-column base styles OUTSIDE any media query, then progressively adding multi-column complexity inside min-width queries as the screen grows. This is the industry-recommended approach, since it forces you to prioritize content and generally results in less CSS overall.

Example

```
/* Base (mobile) styles: no media query needed */
.card-row {
  display: flex;
  flex-direction: column;
}
```

```
/* Enhance for tablets and up */
@media (min-width: 768px) {
  .card-row {
    flex-direction: row;
  }
}
```

> **➖ Exercise**
> Rewrite your card-row and page layout rules mobile-first: write the single-column mobile version with no media query at all, then add a min-width: 768px query that switches to a multi-column row layout for larger screens.

Common breakpoints

There's no official standard, but most real-world projects use similar approximate breakpoints based on common device sizes.

Values / Details

Property / Value	What it does
~480px	Small phones.
~768px	Tablets / large phones (landscape).
~1024px	Small laptops / tablets (landscape).
~1280px+	Desktops and larger.

Example

```
@media (min-width: 480px) { /* larger phones */ }
@media (min-width: 768px) { /* tablets */ }
@media (min-width: 1024px) { /* small desktops */ }
@media (min-width: 1280px) { /* large desktops */ }
```

> **➖ Exercise**
> Add comments in styles.css marking out these four breakpoint tiers, even before filling in every rule, to plan your responsive strategy for the rest of the site.

Responsive images

Beyond max-width: 100% (Chapter 4), CSS and HTML together ensure images never break a layout on any screen size.

Example

```
img {
  max-width: 100%;
  height: auto;
  display: block;
```

```
}
```

> **Exercise**
> Add this rule globally to every <img> on your site as a safety net so no image can ever overflow its container on a small screen, even ones you forget to size individually.

Chapter Summary Project

> **Chapter Project**
> Make your entire zolaxtech.com.et site fully responsive using a mobile-first approach: write single-column base styles with no media query, then add min-width breakpoints at roughly 768px and 1024px that progressively introduce your Flexbox/Grid multi-column layouts from Chapters 8-9. Test by resizing your browser window (or using DevTools' device toolbar) from a small phone width up to a large desktop width, confirming no content overflows or overlaps at any size.

Chapter 11: Transitions, Transforms & Animations

This chapter covers motion smoothly animating changes to make zolaxtech.com.et feel polished and interactive rather than static.

transition

Smoothly animates a property's change from one value to another over time, instead of the change happening instantly used constantly with: hover, focus, and other state changes.

Values / Details

Property / Value	What it does
transition-property	Which property to animate, e.g. background-color, or all for every animatable property.
transition-duration	How long the animation takes, e.g. 0.3s.
transition-timing-function	The pace of the animation over time: ease (default, starts fast/ends slow), linear (constant speed), ease-in, ease-out, ease-in-out.
transition-delay	How long to wait before the animation starts.
Shorthand	transition: background-color 0.3s ease; combines all four in one line, the most common way to write it.

Example

```
.button {
  background-color: #2e74b5;
  transition: background-color 0.3s ease, transform 0.2s ease;
}
.button:hover {
  background-color: #1f3864;
  transform: translateY(-2px);
```

```
}
```

> **➖ Exercise**
> Add a transition to your primary 'Get a Quote' button so its background-color smoothly fades on :hover instead of changing instantly, over about 0.3 seconds.

transform: translate ()

Moves an element from its normal position along the X and/or Y axis WITHOUT affecting the layout of surrounding elements (unlike changing margin or position offsets).

Values / Details

Property / Value	What it does
translateX() / translateY()	Move along a single axis only.
translate(x, y)	Moves along both axes at once.

Example

```
.card:hover {
  transform: translateY(-6px);
}
```

> **➖ Exercise**
> Add a hover effect to your service cards that lifts them slightly upward using translateY(-6px), combined with the box-shadow growth from Chapter 5, for a convincing 'lift' effect.

transform: scale ()

Resizes an element larger or smaller without affecting the space it reserves in the layout.

Values / Details

Property / Value	What it does
scale(1.1)	110% of original size.
scaleX() / scaleY()	Scale along one axis only, for a stretch effect.

Example

```
img.thumbnail {
  transition: transform 0.3s ease;
}
img.thumbnail:hover {
  transform: scale(1.05);
}
```

> **➖ Exercise**
> Add a subtle zoom effect to your gallery thumbnail images on :hover using transform: scale(1.05) with a transition.

transform: rotate ()

Rotates an element around its center point by a given angle.

Values / Details

Property / Value	What it does
rotate(deg)	Positive values rotate clockwise, negative counter-clockwise.

Example

```
.icon-chevron.open {
  transform: rotate(180deg);
}
```

> **➖ Exercise**
> Simulate an FAQ toggle icon: create a small arrow/chevron element and give it a class like .open that rotates it 180 degrees, with a transition, ready to be toggled by JavaScript later.

@keyframes & animation

While transition only animates between two states (normal and: hover, for example), @keyframes lets you define a MULTI-STEP animation sequence that can run automatically on page load or loop continuously.

Values / Details

Property / Value	What it does
@keyframes name { }	Defines the animation's steps using percentages (0% to 100%) or the from/to keywords.
animation-name	Which @keyframes definition to use.
animation-duration	How long one full cycle takes.
animation-iteration-count	How many times it repeats a number, or infinite.
animation-timing-function	Same pacing options as transitions.
Shorthand	animation: fadeIn 0.6s ease-out forwards; combines the common properties in one line.

Example

```
@keyframes fadeIn {
  from {
```

```
    opacity: 0;
    transform: translateY(10px);
  }
  to {
    opacity: 1;
    transform: translateY(0);
  }
}
.hero h1 {
  animation: fadeIn 0.8s ease-out;
}
```

> **➖ Exercise**
> Create a fadeIn @keyframes animation that moves an element up slightly while fading it in, and apply it to your hero heading so it animates in gently when the page loads.

transform-origin

Changes the pivot point a transform (especially rotate and scale) is calculated from by default the element's exact center.

Example

```
.menu-icon {
  transform-origin: top left;
}
```

> **➖ Exercise**
> Take your rotating chevron icon from earlier and experiment with different transform-origin values (like top left vs center) to see how it changes the rotation's visual pivot point.

Chapter Summary Project

> **🏁 Chapter Project**
> Add polish to zolaxtech.com.et using motion: a smooth color transition on all your buttons and links, a lift-and-shadow hover effect on your service cards using translateY and box-shadow together, a subtle scale zoom on gallery images, and one @keyframes fade-in animation applied to your hero section so it animates gently when the page first loads.

Chapter 12: Custom Properties & Advanced Selectors

This chapter covers CSS variables one of the most useful modern CSS features for maintaining a consistent design system along with a few remaining advanced selectors.

Custom properties (CSS variables)

Let you define a reusable value once and reference it throughout your stylesheet. Changing the variable's definition in one place updates every rule that uses it essential for maintaining a consistent brand color palette or spacing scale across a whole site.

Values / Details

Property / Value	What it does
--variable-name: value;	Defined with a double-hyphen prefix, usually inside a :root selector so it's available globally.
var(--variable-name)	Used to READ the variable's value anywhere a normal CSS value would go.
var(--variable-name, fallback)	An optional second argument provides a fallback value if the variable isn't defined.

Example

```
:root {
  --color-primary: #1f3864;
  --color-accent: #2e74b5;
  --spacing-md: 20px;
  --radius-card: 8px;
}

.card {
  border-radius: var(--radius-card);
  padding: var(--spacing-md);
}
.button {
  background-color: var(--color-primary);
}
.button:hover {
  background-color: var(--color-accent);
}
```

> **➖ Exercise**
> Define a: root block of custom properties for your Zolax Tech brand colors, a standard spacing value, and a standard border-radius. Refactor every hard-coded color and spacing value elsewhere in styles.css to use var() referencing these variables instead.

Variables with local scope

Unlike variables defined on: root (global), a custom property defined inside a specific selector is scoped to that element and its descendants only, letting you create local overrides for example, a 'dark mode' card variant.

Example

```
.card {
  --card-bg: white;
  background-color: var(--card-bg);
```

```
}
.card.dark {
  --card-bg: #222; /* overrides only within .card.dark */
}
```

> **➖ Exercise**
> Create a .card.dark variant class that overrides your card's background-color variable locally, without touching the global :root definition.

:root pseudo-class

Matches the document's root element (effectively <html>), and is the conventional place to declare global custom properties since it has the lowest possible specificity while still being globally accessible.

Example

```
:root {
  --font-body: 'Segoe UI', sans-serif;
}
body {
  font-family: var(--font-body);
}
```

> **➖ Exercise**
> Move your font-family choices from Chapter 6 into :root custom properties as well, referencing them via var() in your body and heading rules.

:is() and :where()

Let you group several selectors together to avoid repeating a complex descendant chain: is() carries the specificity of its most specific argument, while :where() always carries ZERO specificity, useful for easily-overridable base styles.

Example

```
/* Instead of repeating: */
header h1, header h2, footer h1, footer h2 { margin: 0; }

/* You can write: */
:is(header, footer) :is(h1, h2) {
  margin: 0;
}
```

> **➖ Exercise**
> Find a place in your stylesheet where you've written several similar descendant selectors and combine them using :is() to shorten the rule.

:nth-of-type() and :first-of-type / :last-of-type

Similar to: nth-child (), but count only among siblings of the SAME element type, ignoring other kinds of elements in between useful when a container mixes different tag types.

Example

```
article p:first-of-type {
  font-size: 1.2rem;
  font-weight: 500;
}
```

> **➖ Exercise**
> Inside one of your <article> blocks (which might mix headings and paragraphs), use p:first-of-type to give just the first paragraph a slightly larger 'lede' style.

CSS functions: calc(), clamp(), min(), max()

Let you compute values dynamically, even mixing different units, instead of hard-coding a single fixed number.

Values / Details

Property / Value	What it does
calc()	Performs math between values, even mixed units, e.g. calc(100% - 40px).
clamp(min, preferred, max)	Picks a fluid value that scales with something (often a viewport unit) but never goes below min or above max extremely useful for responsive font sizes without any media queries.
min() / max()	Pick the smaller/larger of a list of values.

Example

```
.container {
  width: calc(100% - 40px);
}
h1 {
  font-size: clamp(1.75rem, 4vw + 1rem, 3rem);
}
```

> **➖ Exercise**
> Replace one of your heading's fixed rem font-sizes with a clamp() value that scales fluidly between a sensible minimum and maximum as the viewport changes width, and test it by resizing the browser.

Chapter Summary Project

🏁 Chapter Project

Build a complete design-token system for zolaxtech.com.et in a :root block: colors, spacing scale, border-radius, and font families, all referenced via var() throughout the rest of styles.css. Add one clamp()-based fluid heading size, and refactor at least one repeated selector pattern using :is().

Chapter 13: Advanced Layout, Organization & Best Practices

This final chapter covers what separates a working stylesheet from a professional, maintainable one critical as a real site like zolaxtech.com.et grows past a handful of pages.

CSS Reset / Normalize

Different browsers apply slightly different default styles to the same HTML elements. A reset (or the lighter-touch 'normalize' approach) removes or evens out these inconsistencies at the very top of your stylesheet, before your own styles begin.

Example

```
*, *::before, *::after {
  box-sizing: border-box;
  margin: 0;
  padding: 0;
}
img, picture, video {
  max-width: 100%;
  display: block;
}
```

➖ Exercise

Add a small modern reset block like the example to the very top of styles.css, above all your other rules, and confirm your layout still looks correct afterward (you may need to re-add some spacing that the browser was previously supplying by default).

Logical file organization

As a stylesheet grows past a couple hundred lines, grouping related rules together (and clearly commenting each group) becomes essential for staying sane. A common order: reset, variables, typography, layout/utility helpers, components, page-specific overrides.

Example

```
/* 1. Reset
   2. Custom Properties
   3. Typography
   4. Layout (header, nav, footer, containers)
   5. Components (cards, buttons, forms)
   6. Utilities
   7. Media Queries */
```

> **➼ Exercise**
>
> Reorganize your entire styles.css file into these seven labeled sections, moving existing rules into the right group (media queries can either live near their related component or all together at the end pick one approach and stay consistent).

Naming conventions (BEM overview)

BEM (Block, Element, Modifier) is a popular naming convention that makes class names self-documenting and avoids specificity conflicts by relying almost entirely on single classes instead of nested selectors.

Values / Details

Property / Value	What it does
Block	A standalone component, e.g. .card.
Element (__)	A part of that block, e.g. .card__title, .card__price.
Modifier (--)	A variant of a block or element, e.g. .card--featured, .button--secondary.

Example

```
<div class="card card--featured">
  <h3 class="card__title">Pro Plan</h3>
  <p class="card__price">1500 ETB</p>
</div>

.card { }
.card__title { }
.card__price { }
.card--featured { }
```

> **➼ Exercise**
>
> Rename the classes on your pricing cards using BEM conventions (card, card__title, card__price, card--featured for a highlighted plan) and update the matching CSS selectors.

Utility classes

Small, single-purpose classes that do exactly one job (like adding margin or hiding an element), meant to be combined together on an element instead of writing a new custom class for every combination of styles.

Example

```
.mt-md { margin-top: var(--spacing-md); }
.text-center { text-align: center; }
.hidden { display: none; }
```

> **➖ Exercise**
> Create three small utility classes (.mt-md, .text-center, .hidden) in a dedicated 'Utilities' section of your stylesheet and apply one of them directly in your HTML instead of writing a brand-new one-off rule.

Accessibility & CSS

CSS choices directly affect how usable a site is for people with disabilities a few rules matter on nearly every project.

Values / Details

Property / Value	What it does
Focus styles	Never remove the default focus outline (outline: none) without providing a clearly visible custom replacement keyboard users rely on it to know where they are on the page.
Color contrast	Text should maintain sufficient contrast against its background (a widely used guideline is a 4.5:1 ratio for normal body text) check with a browser DevTools contrast checker.
prefers-reduced-motion	A media query that detects when a visitor has requested reduced motion at the OS level, letting you disable large/fast animations for them.

Example

```
a:focus-visible {
  outline: 2px solid var(--color-accent);
  outline-offset: 2px;
}

@media (prefers-reduced-motion: reduce) {
  * {
    animation: none! important;
    transition: none! important;
  }
}
```

> **➖ Exercise**
> Add a clear :focus-visible outline style to your links and buttons using your accent color, and add a prefers-reduced-motion media query that disables your Chapter 11 animations for visitors who have that OS setting enabled.

Print styles

A dedicated set of rules, wrapped in @media print, that only apply when a visitor prints the page commonly used to hide navigation/buttons and ensure text remains readable on paper.

Example

```
@media print {
  nav, footer, .no-print {
    display: none;
  }
  body {
    color: black;
    background: white;
  }
}
```

> **➖ Exercise**
> Add an @media print block that hides your navigation and any decorative background images, ensuring the page prints cleanly in black text on a white background. Preview it using your browser's Print Preview (Ctrl+P).

Performance basics

A few habits keep a growing stylesheet fast to download and render.

Values / Details

Property / Value	What it does
Avoid overly specific/nested selectors	Deeply nested selectors like header nav ul li a span are slower to match and harder to override prefer flatter selectors like .nav-link.
Minimize use of @import	Importing one CSS file from inside another delays rendering because the browser must wait for the imported file before continuing; prefer multiple <link> tags in the HTML instead.
One well-organized file (for small sites)	For a site the size of zolaxtech.com.et, a single well-commented styles.css is usually simpler to maintain than splitting into many small files, unless using a build tool that combines them automatically.

Example

```
/* Prefer this: */
.nav-link { color: var(--color-primary); }

/* Over this: */
header nav ul li a { color: var(--color-primary); }
```

> **➖ Exercise**
> Search your stylesheet for any selector nested more than three levels deep and refactor it into a single, flatter class-based selector instead.

Chapter Summary Project

🏁 Chapter Project

Perform a full 'launch readiness' pass on zolaxtech.com.et's CSS: add a reset block at the top, reorganize the whole file into the seven labeled sections from this chapter, rename your card and button classes to follow BEM, add at least three reusable utility classes, add visible :focus-visible styles plus a prefers-reduced-motion query, and add a basic @media print block. Finish by opening every page with DevTools open and confirming there are no console errors and no elements overflowing their containers at phone, tablet, and desktop widths.

Appendix A: Useful VS Code Shortcuts for CSS

Shortcut	What it does
Ctrl + S	Save the current file (triggers Live Server auto-refresh).
Ctrl + /	Comment or uncomment the selected line(s) with /* */.
Alt + Shift + F	Auto-format/indent the whole stylesheet.
Ctrl + Space	Trigger autocomplete suggestions for properties and values.
Ctrl + Click (on a value)	Open the color picker for a color value under the cursor.
Ctrl + D	Select the next occurrence of the current word (handy for renaming a repeated class).
.card + Tab (Emmet)	Quickly scaffold a class selector block.
Ctrl + `	Open/close the integrated terminal.
Right-click element > Inspect	Opens DevTools to see the live box model, computed styles, and test changes instantly.

Appendix B: Quick Reference Common Values Cheat Sheet

Goal	CSS
Flexbox container	display: flex; justify-content: center; align-items: center; gap: 20px;
Grid container	display: grid; grid-template-columns: repeat(auto-fit, minmax(220px, 1fr)); gap: 20px;
Center a block horizontally	width: fit-content; margin: 0 auto;
Full-viewport section	min-height: 100vh;
Responsive image	max-width: 100%; height: auto; display: block;
Smooth hover transition	transition: all 0.3s ease;
Card shadow	box-shadow: 0 4px 12px rgba(0,0,0,0.1);
Mobile-first breakpoint	@media (min-width: 768px) { ... }
CSS variable	:root { --color-primary: #1f3864; } → color: var(--color-primary);
Fluid font size	font-size: clamp(1.5rem, 4vw, 3rem);

Appendix C: Final Capstone Project

Using everything from this guide, fully style the multi-page zolaxtech.com.et website from the companion HTML5 guide, in Visual Studio Code with Live Server, delivering:

1. A single, well-organized styles.css with a reset, a: root custom-properties design system (colors, spacing, radius, fonts), and clearly commented sections, linked from every page.
2. A sticky header laid out with Flexbox, and a mobile nav menu that stacks vertically under a max-width media query.
3. A responsive card grid (Services and/or Portfolio pages) built with CSS Grid using auto-fit/minmax, including one featured/larger item.
4. A fully responsive layout tested at phone, tablet, and desktop widths using a mobile-first approach with at least two min-width breakpoints.
5. Hover and focus states with smooth transitions on every button and link, plus at least one @keyframes animation on the homepage hero.
6. A pricing table styled with alternating row colors, and a contact/quote form with clear: focus styles on every field.
7. Accessible color contrast throughout, visible: focus-visible outlines, and a prefers-reduced-motion media query.
8. A print stylesheet under @media print that hides navigation and non-essential decoration.