PERGOLA PERFORMANCE · LEARNING MAP

400
Things to Master

A field guide to the skills behind masterpiece-grade websites — from raw markup to the taste that makes a page feel authored, not assembled. Every single resource is **free**.

400	16	334	$0
skills to learn	domains	free resources	total cost

Prepared for: Eduardo 2026-06-17 Internal reference

HOW TO USE THIS

- It's a **MAP, NOT A TO-DO LIST** — you don't grind 1→400. Pick the domain you're weakest in and work it top-down (items are ordered foundational → advanced).
- Every link is a **FREE** video, course, doc, interactive tutorial, or tool. Learn by **BUILDING**, not just watching — rebuild a real page after each cluster.
- The last domain, **DESIGN PROCESS & CRAFT**, is the one that turns "correct" sites into "masterpiece" sites. Don't skip it.

The 16 domains

01	HTML & Semantic Markup	001–025
02	CSS Fundamentals	026–050
03	CSS Layout (Flexbox & Grid)	051–075
04	Responsive & Adaptive Design	076–100
05	Typography for the Web	101–125
06	Color & Theming	126–150
07	Visual Design Principles	151–175
08	Layout & Composition	176–200
09	UX & Interaction Design	201–225
10	Accessibility (a11y)	226–250
11	Animation & Motion	251–275
12	JavaScript Essentials	276–300
13	Modern Frontend Engineering	301–325
14	Performance & Core Web Vitals	326–350
15	Tooling, Git & Deployment	351–375
16	Design Process & Craft	376–400

01 HTML & Semantic Markup

001–025

001 **HTML document skeleton: <!DOCTYPE html>, <html lang>, <head>, <body>**

A correct boilerplate triggers standards mode and sets the language so the whole document renders and reads predictably.

DOCS **MDN: Getting started with HTML**

https://developer.mozilla.org/en-US/docs/Learn/HTML/Introduction_to_HTML/Getting_started

002 **Content model & nesting rules: block vs inline vs flow content, valid parent/child pairs**

Knowing what may legally nest inside what prevents broken layouts and invalid markup that browsers silently mangle.

DOCS **MDN: HTML elements reference**

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

003 **Document outline & heading hierarchy: one logical h1, never skipping levels**

A clean heading tree is the backbone of scannability, SEO, and screen-reader navigation that makes a site feel authored, not assembled.

COURSE **web.dev: Headings and landmarks**

https://web.dev/learn/accessibility/structure

004 **Sectioning landmarks: header, nav, main, article, section, aside, footer**

Real landmark elements give every page an explicit structure that assistive tech and search engines can map instantly.

DOCS **MDN: HTML sectioning elements / Document structure**

https://developer.mozilla.org/en-US/docs/Learn/HTML/Introduction_to_HTML/Document_and_website_structure

005 **section vs article vs div: when each is semantically correct**

Choosing the meaning-bearing element over a generic div is what separates professional markup from div-soup.

SITE **HTML5 Doctor: The section element**

https://html5doctor.com/the-section-element/

006 **Text-level semantics: strong, em, mark, small, abbr, time, cite, q, blockquote**

Using the element that carries the right meaning (not just bold/italic styling) makes content machine-readable and richer for everyone.

DOCS **MDN: Emphasis and importance / Advanced text formatting**

https://developer.mozilla.org/en-US/docs/Learn/HTML/Introduction_to_HTML/Advanced_text_formatting

007 **Lists done right: ul, ol (start/reversed/type), dl/dt/dd for definitions**

Proper list semantics announce item counts to screen readers and structure nav, steps, and key-value content cleanly.

DOCS **MDN: HTML lists**

https://developer.mozilla.org/en-US/docs/Learn/HTML/Introduction_to_HTML/HTML_text_fundamentals

008 **Links & anchors: href types, in-page #fragments, target/rel, mailto/tel**

Well-formed links power navigation, deep-linking, and click-to-call CTAs that convert on local-business sites.

DOCS **MDN: Creating hyperlinks**

https://developer.mozilla.org/en-US/docs/Learn/HTML/Introduction_to_HTML/Creating_hyperlinks

009 **rel security & SEO: rel=noopener noreferrer, nofollow, sponsored, ugc**

Correct rel values close the tabnabbing security hole and signal link intent to search engines.

DOCS **MDN: rel attribute**

https://developer.mozilla.org/en-US/docs/Web/HTML/Attributes/rel

010 **Responsive images: img with width/height, alt, srcset & sizes, the picture element**

Serving right-sized art-directed images keeps masterpiece visuals crisp while staying fast on every device.

DOCS **MDN: Responsive images**

https://developer.mozilla.org/en-US/docs/Learn/HTML/Multimedia_and_embedding/Responsive_images

011 **Image performance: loading=lazy, decoding=async, fetchpriority, modern formats**

These attributes prevent layout shift and slow loads that instantly cheapen an otherwise premium design.

SITE **web.dev: Optimize Largest Contentful Paint**

https://web.dev/articles/optimize-lcp

012 **Writing meaningful alt text (and when alt="" for decorative images)**

Good alt text is both an accessibility requirement and an SEO signal that shows craftsmanship in the details.

DOCS **W3C WAI: An alt Decision Tree**

https://www.w3.org/WAI/tutorials/images/decision-tree/

013 **figure & figcaption for images, diagrams, and quotes**

Associating a caption with its media semantically groups them so the relationship survives for assistive tech.

DOCS **MDN: figure element**

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

014 **Audio & video embedding: video/audio with controls, poster, source, captions/track**

Native media elements with captions give you accessible, brand-controlled players instead of clunky third-party embeds.

DOCS **MDN: Video and audio content**

https://developer.mozilla.org/en-US/docs/Learn/HTML/Multimedia_and_embedding/Video_and_audio_content

015 **Embedding third-party content safely with iframe (maps, videos) + sandbox/loading**

Knowing iframe attributes lets you drop in maps and YouTube without tanking performance or security.

DOCS **MDN: From object to iframe — embedding technologies**

https://developer.mozilla.org/en-US/docs/Learn/HTML/Multimedia_and_embedding/Other_embedding_technologies

016 **Forms structure: form, fieldset, legend, and label-to-input association**

Properly labeled, grouped forms are the conversion engine of a business site and the #1 accessibility win.

COURSE **web.dev: Learn Forms**

https://web.dev/learn/forms

017 **Input types & semantics: email, tel, url, number, date, search, range, color**

The right input type triggers the correct mobile keyboard and native validation, making forms feel effortless.

DOCS **MDN: The Input element**

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

018 **Form UX attributes: placeholder, required, pattern, autocomplete, inputmode**

These attributes deliver autofill and instant validation that dramatically reduce abandonment without any JavaScript.

DOCS **MDN: Client-side form validation**

https://developer.mozilla.org/en-US/docs/Learn/Forms/Form_validation

019 **Other controls: textarea, select/optgroup, radio/checkbox groups, button types**

Mastering each control and its grouping lets you build any form interaction with clean, native semantics.

DOCS **MDN: Other form controls**

https://developer.mozilla.org/en-US/docs/Learn/Forms/Other_form_controls

020 **Accessible data tables: table, thead/tbody/tfoot, th with scope, caption**

Structured tables with scoped headers turn raw data into something screen readers and search engines can actually parse.

COURSE **web.dev: Tables (Learn Accessibility)**

https://web.dev/learn/accessibility/structure

021 **The <head> essentials: charset UTF-8, viewport meta, title, canonical, favicon**

A correct head is the silent foundation that makes a site render right, scale on phones, and avoid duplicate-content penalties.

DOCS **MDN: What's in the head? Metadata in HTML**

https://developer.mozilla.org/en-US/docs/Learn/HTML/Introduction_to_HTML/The_head_metadata_in_HTML

022 **SEO meta: meta description, robots, and per-page titling strategy**

These tags control how every page appears in search results, directly driving the clicks a local business depends on.

DOCS **Google Search Central: SEO Starter Guide**

https://developers.google.com/search/docs/fundamentals/seo-starter-guide

023 **Social sharing metadata: Open Graph and Twitter Card tags**

OG tags make shared links render as rich, branded previews instead of bare URLs — a polish cue that builds trust.

DOCS **The Open Graph protocol**

https://ogp.me/

024 **Structured data with JSON-LD Schema.org (LocalBusiness, Organization, FAQ)**

Schema markup unlocks rich results like ratings, hours, and FAQs that make a local site stand out in search.

DOCS **Google Search Central: Intro to structured data**

https://developers.google.com/search/docs/appearance/structured-data/intro-structured-data

025 **Validation & native interactivity: W3C validator, details/summary, dialog**

Validating your markup and using built-in interactive elements yields robust, accessible UI with near-zero JavaScript.

TOOL **W3C Markup Validation Service**

https://validator.w3.org/

02 CSS Fundamentals

026–050

026 **The box model: content/padding/border/margin and box-sizing: border-box**

Every element is a box, and predictable sizing is the bedrock of layouts that don't break under content changes.

DOCS **MDN: The box model**

https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/The_box_model

027 **Selectors: type, class, ID, attribute, and combinators (descendant, child, sibling)**

Targeting exactly the right elements with intent is what keeps stylesheets surgical instead of a tangle of overrides.

INTERACTIVE **CSS Diner**

https://flukeout.github.io

028 **The cascade: origin, importance, and source order resolution**

Knowing why one rule wins lets you author confident styles instead of fighting your own CSS with !important.

DOCS **MDN: Handling conflicts (the cascade)**

https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/Cascade_and_inheritance

029 **Specificity: how (id, class, type) weights are calculated and compared**

Mastering specificity prevents the override wars that make a polished design impossible to maintain.

SITE **CSS-Tricks: Specifics on CSS Specificity**

https://css-tricks.com/specifics-on-css-specificity/

030 **Inheritance and the inherit / initial / unset / revert keywords**

Leveraging inheritance for typography and color cuts repetition and keeps a site visually coherent by default.

DOCS **MDN: Inheritance**

https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_cascade/Inheritance

031 **Absolute vs relative units and when to use px**

Choosing the right unit per property is the difference between a rigid mockup and a layout that flexes gracefully.

DOCS **MDN: CSS values and units**

https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/Values_and_units

032 **rem vs em: building a scalable, accessible sizing system**

Using rem for layout and em for component-relative spacing keeps designs honoring user font preferences and zoom.

SITE **Use rem for Global Sizing; Use em for Local Sizing**

https://css-tricks.com/rem-global-em-local/

033 **Viewport units: vw, vh, and the new dvh/svh/lvh for mobile**

Viewport units power full-bleed hero sections and fluid type while dvh fixes the notorious mobile address-bar jump.

DOCS **web.dev: The large, small, and dynamic viewport units**

https://web.dev/blog/viewport-units

034 **Content-relative units: ch and ex for typography control**

Setting line length in ch enforces readable measure (45-75 chars) that instantly reads as professional typesetting.

DOCS **MDN: length data type (ch, ex)**

https://developer.mozilla.org/en-US/docs/Web/CSS/length

035 **Percentages: what each property resolves them against**

Knowing that percentage padding resolves off width (not height) avoids baffling layout bugs and enables aspect-ratio tricks.

DOCS **MDN: percentage data type**

https://developer.mozilla.org/en-US/docs/Web/CSS/percentage

036 **Custom properties (CSS variables): declaration, scope, and var() fallbacks**

Variables turn a stylesheet into a design system where one token change cascades a coherent rebrand everywhere.

DOCS **MDN: Using CSS custom properties**

https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_cascading_variables/Using_CSS_custom_properties

037 **Theming with custom properties: light/dark mode and component variants**

Scoped variable overrides give you instant, DRY theming that makes a site feel intentionally designed, not patched.

DOCS **web.dev: Building a color scheme**

https://web.dev/articles/building/a-color-scheme

038 **The :root pseudo-class and a global design-token layer**

Centralizing spacing, color, and type tokens at :root is the single highest-leverage habit for masterpiece consistency.

DOCS **MDN: :root**

https://developer.mozilla.org/en-US/docs/Web/CSS/:root

039 **Pseudo-classes for state: :hover, :focus-visible, :active, :checked**

Crisp, intentional interaction states are what separate a living, premium interface from a flat static page.

DOCS **MDN: Pseudo-classes**

https://developer.mozilla.org/en-US/docs/Web/CSS/Pseudo-classes

040 **Pseudo-elements: ::before, ::after, ::selection, ::marker**

Generated content lets you add decorative accents and custom details without polluting your HTML markup.

DOCS **MDN: Pseudo-elements**

https://developer.mozilla.org/en-US/docs/Web/CSS/Pseudo-elements

041 **Structural and relational selectors: :nth-child, :not(), :is(), :where()**

These collapse repetitive rules into elegant one-liners and :where() keeps resets at zero specificity for easy overrides.

DOCS **MDN: :is()**

https://developer.mozilla.org/en-US/docs/Web/CSS/:is

042 **The :has() relational selector (parent/previous-sibling targeting)**

Styling a parent based on its children unlocks layouts that previously demanded JavaScript, keeping things lean and modern.

DOCS **MDN: :has()**

https://developer.mozilla.org/en-US/docs/Web/CSS/:has

043 **Logical properties: margin-inline, padding-block, inset, border-inline**

Writing flow-relative CSS future-proofs layouts for RTL languages and vertical writing modes with zero rework.

DOCS **MDN: CSS logical properties and values**

https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_logical_properties_and_values

---

044 **Margin collapsing: when adjacent and nested margins merge**

Understanding collapse explains mysterious vertical gaps and lets you control rhythm instead of guessing.

DOCS **MDN: Mastering margin collapsing**

https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_box_model/Mastering_margin_collapsing

---

045 **Stacking contexts and z-index: what creates a new context**

Knowing that transform/opacity/filter spawn stacking contexts is the only way to debug z-index that 'won't work'.

DOCS **MDN: The stacking context**

https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_positioned_layout/Stacking_context

---

046 **Positioning: static, relative, absolute, fixed, sticky and containing blocks**

Precise positioning powers sticky nav, overlay badges, and modals that feel deliberately engineered rather than hacked.

DOCS **MDN: Positioning**

https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Positioning

---

047 **The CSS reset / normalize and a modern default base**

A deliberate reset erases inconsistent browser defaults so your design starts from a clean, predictable canvas.

SITE **Josh W. Comeau: A Modern CSS Reset**

https://www.joshwcomeau.com/css/custom-css-reset/

---

048 **calc(), min(), max(), and clamp() for fluid, responsive values**

clamp() gives buttery fluid typography and spacing between breakpoints — a hallmark of high-craft modern sites.

DOCS **web.dev: min(), max(), and clamp()**

https://web.dev/articles/min-max-clamp

---

049 **The overall CSS foundations course path (cascade through layout)**

A structured, project-based run-through cements the fundamentals into muscle memory rather than scattered facts.

COURSE **The Odin Project: Intermediate HTML and CSS**

https://www.theodinproject.com/paths/full-stack-javascript/courses/intermediate-html-and-css

---

050 **Browser support strategy: checking caniuse and @supports feature queries**

Knowing what's safe and progressively enhancing lets you ship cutting-edge polish without breaking on older browsers.

TOOL **Can I Use**

https://caniuse.com

---

03 CSS Layout (Flexbox & Grid)

051–075

051 **The CSS box model & box-sizing: border-box**

Every layout sits on the box model; controlling content/padding/border math prevents the off-by-a-pixel breakage that makes work look amateur.

DOCS **MDN: The box model**

https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/The_box_model

052 **display values & normal document flow (block vs inline vs inline-block)**

Knowing how elements flow by default lets you reach for layout tools deliberately instead of fighting the browser's defaults.

DOCS **MDN: Normal Flow**

https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Normal_Flow

053 **Float & clear (legacy layout + clearfix) and why we left it behind**

Understanding the old float system explains why Flexbox/Grid exist and lets you read and rescue legacy or CMS markup.

DOCS **MDN: Floats**

https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Floats

054 **Flexbox fundamentals: main vs cross axis, flex-direction**

Almost every component row, nav, and card uses Flexbox, so internalizing axis thinking is the daily workhorse of layout.

COURSE **Flexbox (Wes Bos free course)**

https://flexbox.io

055 **Flexbox alignment: justify-content, align-items, align-content, gap**

Precise centering and even spacing without margin hacks is the difference between tidy, intentional UI and lopsided spacing.

DOCS **CSS-Tricks: A Complete Guide to Flexbox**

https://css-tricks.com/snippets/css/a-guide-to-flexbox/

056 **flex shorthand: flex-grow, flex-shrink, flex-basis**

Mastering the grow/shrink/basis trio is how you build flexible components that fill space gracefully instead of overflowing or collapsing.

INTERACTIVE **Flexbox Froggy**

https://flexboxfroggy.com

057 **align-self, order, and flex-wrap for responsive rows**

Per-item alignment and wrapping let one flex container reflow elegantly across breakpoints without media-query gymnastics.

VIDEO **Kevin Powell — Flexbox playlist**

https://www.youtube.com/@KevinPowell

058 **CSS Grid fundamentals: grid-template-columns/rows, the fr unit**

Grid is the only tool that controls two axes at once, making it the backbone of distinctive, non-template page structures.

COURSE **CSS Grid (Wes Bos free course)**

https://cssgrid.io

059 **Grid placement: grid-line numbers, grid-column/grid-row span**

Explicit line placement lets you build asymmetric, editorial layouts that break the predictable 3-column template look.

INTERACTIVE **Grid Garden**

https://cssgridgarden.com

---

060 **Named grid template areas**

Naming regions makes complex layouts readable and lets you rearrange entire page sections per breakpoint with one rule.

DOCS **CSS-Tricks: A Complete Guide to Grid**

https://css-tricks.com/snippets/css/complete-guide-grid/

---

061 **repeat(), minmax(), auto-fill vs auto-fit responsive grids**

This pattern builds card galleries that reflow column counts automatically — fluid, professional, and zero media queries.

COURSE **web.dev: Learn CSS — Grid**

https://web.dev/learn/css/grid

---

062 **Grid gap, alignment (justify/align-items & -content, *-self)**

Grid's two-axis alignment gives pixel-perfect gutters and item placement that signal a deliberately designed page.

DOCS **MDN: Box alignment in Grid layout**

https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_grid_layout/Box_alignment_in_grid_layout

---

063 **Implicit grids: grid-auto-rows, grid-auto-flow, dense packing**

Controlling auto-generated tracks lets dynamic content (CMS posts, products) flow neatly into masonry-like layouts.

DOCS **MDN: Auto-placement in CSS Grid**

https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_grid_layout/Auto-placement_in_CSS_grid_layout

---

064 **Choosing Flexbox vs Grid (content-out vs layout-in)**

Picking the right tool per situation keeps markup clean and avoids the hacky nesting that makes layouts fragile.

DOCS **CSS-Tricks: Quick! What's the Difference Between Flexbox and Grid?**

https://css-tricks.com/quick-whats-the-difference-between-flexbox-and-grid/

---

065 **Positioning: relative, absolute, fixed, sticky & containing blocks**

Position is how you layer badges, overlays, and sticky headers — core to polished, interactive marketing UI.

SITE **Josh W. Comeau: A Friendly Introduction to Positioning**

https://www.joshwcomeau.com/css/understanding-layout-algorithms/

---

066 **position: sticky for sticky headers, sidebars, and section nav**

Sticky elements add a premium, app-like feel to long marketing pages when used with intention and correct scroll containers.

COURSE **web.dev: position: sticky**

https://web.dev/learn/css/layout

---

067 **z-index & stacking contexts (and why z-index sometimes fails)**

Understanding stacking contexts ends the random z-index:9999 war and keeps overlays, modals, and dropdowns predictable.

SITE **Josh W. Comeau: Stacking Contexts**

https://www.joshwcomeau.com/css/stacking-contexts/

---

068 **A z-index management strategy (scales/tokens, isolation: isolate)**

A deliberate layering scale prevents the spaghetti z-index that plagues large sites and makes overlays maintainable.

SITE **Managing Z-Index In A Component-Based Web Application**

https://www.smashingmagazine.com/2019/04/z-index-component-based-web-application/

069 **Overflow control: visible/hidden/scroll/auto, clip, scroll containers**

Handling overflow correctly prevents the horizontal-scroll and clipped-shadow bugs that instantly read as unpolished.

DOCS **MDN: overflow**

https://developer.mozilla.org/en-US/docs/Web/CSS/overflow

070 **Intrinsic sizing: min-content, max-content, fit-content**

Intrinsic keywords size elements to their content perfectly, killing awkward fixed widths that break with real copy.

DOCS **Intrinsic size (MDN Glossary): min-content, max-content, fit-content**

https://developer.mozilla.org/en-US/docs/Glossary/Intrinsic_Size

071 **min(), max(), and clamp() for fluid responsive sizing**

clamp() gives buttery fluid type and spacing between breakpoints — a hallmark of high-end, custom-feeling sites.

DOCS **web.dev: min(), max(), and clamp()**

https://web.dev/articles/min-max-clamp

072 **Centering anything (the modern Grid place-items trick) & the holy grail layout**

Reliable centering and full-page layouts are the constant low-level moves that, done cleanly, make everything look right.

INTERACTIVE **Josh W. Comeau: The Center of CSS**

https://www.joshwcomeau.com/css/center-a-div/

073 **Container queries (@container) for truly modular components**

Components that respond to their own container, not the viewport, let you reuse cards anywhere — the modern masterpiece-layout edge.

DOCS **MDN: CSS container queries**

https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_containment/Container_queries

074 **Subgrid for aligning nested content to a parent grid**

Subgrid aligns card titles, bodies, and buttons across a row perfectly — eliminating ragged, mismatched component layouts.

DOCS **MDN: Subgrid**

https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_grid_layout/Subgrid

075 **Verifying layout cross-browser & at scale (Can I Use, DevTools grid/flex inspectors)**

Inspecting overlays and checking support before you ship guarantees layouts hold up everywhere, which separates pro work from 'works on my machine'.

TOOL **Can I Use**

https://caniuse.com

04 Responsive & Adaptive Design

076–100

076 **The viewport meta tag: width=device-width, initial-scale=1 and why it's non-negotiable**

Without it mobile browsers render at a fake 980px desktop width and zoom out, making every responsive rule below silently fail.

DOCS **MDN: Viewport meta tag**

https://developer.mozilla.org/en-US/docs/Web/HTML/Viewport_meta_tag

077 **Mobile-first methodology: write base styles for small screens, layer up with min-width**

Designing up from the constraint forces content priority and produces leaner, more maintainable CSS than stripping a desktop layout down.

COURSE **web.dev Learn: Responsive Design**

https://web.dev/learn/design

078 **Media query anatomy: @media, min-width vs max-width, and the and/or logic**

Media queries are the core switch that lets one layout serve every device, and knowing min vs max prevents overlapping, conflicting rules.

DOCS **MDN: Using media queries**

https://developer.mozilla.org/en-US/docs/Web/CSS/Media_Queries/Using_media_queries

079 **Choosing breakpoints by content, not device models (let the design break, then add a query)**

Content-driven breakpoints age gracefully and avoid the brittle, generic feel of copying iPhone/iPad pixel widths.

DOCS **web.dev: How to pick breakpoints**

https://web.dev/learn/design/media-queries

080 **Relative units done right: em, rem, %, and why rem for spacing/typography**

Relative units make a layout scale with user font settings, the foundation of both responsiveness and accessibility.

SITE **Josh W. Comeau: The Surprising Truth About Pixels and Accessibility**

https://www.joshwcomeau.com/css/surprising-truth-about-pixels-and-accessibility/

081 **Viewport units: vw, vh, vmin, vmax for screen-relative sizing**

Viewport units let hero sections, type, and spacing respond directly to screen size for full-bleed, intentional layouts.

SITE **CSS-Tricks: Fun with Viewport Units**

https://css-tricks.com/fun-viewport-units/

082 **The mobile 100vh bug and the new dvh/svh/lvh dynamic viewport units**

Mobile address bars make 100vh overflow and clip content; dvh fixes the single most common 'why is my hero broken on iPhone' bug.

DOCS **web.dev: The large, small, and dynamic viewport units**

https://web.dev/blog/viewport-units

083 **Fluid typography with clamp(): clamp(min, preferred-vw, max)**

clamp() gives type that scales smoothly between breakpoints with no jarring jumps, a hallmark of polished, modern sites.

SITE **CSS-Tricks: Linearly Scale font-size with clamp()**

https://css-tricks.com/linearly-scale-font-size-with-css-clamp-based-on-the-viewport/

084 **Generating a full fluid type scale (and fluid spacing) with a calculator**

A consistent fluid scale ties heading sizes and whitespace together across all viewports for a designed, harmonious rhythm.

TOOL **Utopia Fluid Type Scale Calculator**

https://utopia.fyi/

085 **min(), max(), and calc() for adaptive sizing without media queries**

These functions let widths, padding, and gaps respond fluidly so you write fewer breakpoints and get smoother behavior.

DOCS **MDN: min()**

https://developer.mozilla.org/en-US/docs/Web/CSS/min

086 **Container queries: @container and the container-type/container-name setup**

Components style themselves by their own width, so a card looks right in a sidebar or a hero without knowing the page layout.

DOCS **MDN: CSS container queries**

https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_containment/Container_queries

087 **Container query units (cqw, cqi) and component-driven responsive thinking**

Truly reusable, masterpiece-grade components adapt to their container, not the viewport, which is the future of responsive design.

DOCS **web.dev: Container queries**

https://web.dev/blog/cq-stable

088 **Intrinsic responsive layout: min-content, max-content, fit-content, and the RAM pattern (repeat(auto-fit, minmax()))**

Grids that wrap and resize themselves with no media queries are leaner and respond to any content gracefully.

COURSE **web.dev: Learn CSS Grid**

https://web.dev/learn/css/grid

089 **Flexbox for responsive components: flex-wrap, flex-basis, flex-grow/shrink shorthand**

Flex-wrap with a flex-basis is the simplest way to make navbars, button rows, and card strips reflow without breakpoints.

INTERACTIVE **Flexbox Froggy**

https://flexboxfroggy.com/

090 **Responsive images with srcset (width descriptors) and the sizes attribute**

Serving each device a right-sized image is the biggest single win for mobile load speed and sharpness on retina screens.

DOCS **MDN: Responsive images**

https://developer.mozilla.org/en-US/docs/Web/HTML/Responsive_images

091 **Art direction with <picture> and <source> media for different crops per breakpoint**

A wide hero crop on desktop and a tight vertical crop on mobile keeps the subject impactful instead of letting it shrink to nothing.

COURSE **web.dev: Responsive images**

https://web.dev/learn/images

092 **Modern image formats and loading: AVIF/WebP via <picture>, loading=lazy, decoding=async, width/height to prevent CLS**

Right format plus reserved dimensions cuts bytes and stops layout shift, both core to a fast, stable responsive experience.

DOCS **web.dev: Use responsive images and modern formats**

https://web.dev/learn/images/performance-issues

---

093 **Touch target sizing: minimum 24x24 (WCAG) / 44x44 (Apple) and adequate spacing**

Tap-friendly targets are the difference between a frustrating and a premium mobile feel, and now a WCAG requirement.

DOCS **web.dev: Accessible tap targets**

https://web.dev/articles/accessible-tap-targets

---

094 **Pointer and hover media features: @media (hover: hover) and (pointer: fine/coarse)**

Detecting touch vs mouse lets you avoid hover-only menus and dead 'sticky hover' states that trap mobile users.

DOCS **MDN: hover media feature**

https://developer.mozilla.org/en-US/docs/Web/CSS/@media/hover

---

095 **prefers-reduced-motion and prefers-color-scheme as adaptive design, not just a11y**

Adapting motion and theme to user/system preferences signals craft and respect, elevating perceived quality.

DOCS **web.dev: prefers-reduced-motion**

https://web.dev/articles/prefers-reduced-motion

---

096 **Responsive typography readability: ch units, max line-length (45-75ch), and fluid line-height**

Capping measure with ch keeps body copy readable on huge monitors, a subtle detail that separates pro work from amateur.

COURSE **web.dev Learn CSS: Typography**

https://web.dev/learn/css/typography

---

097 **Safe-area insets and env() for notched devices (env(safe-area-inset-*)) with viewport-fit=cover**

Respecting the notch and home indicator makes full-bleed mobile layouts feel native instead of clipped or cramped.

DOCS **env() CSS function: safe-area-inset for notched devices**

https://developer.mozilla.org/en-US/docs/Web/CSS/env

---

098 **Responsive tables: overflow-x scroll containers, stacked-card pattern, and CSS Grid table reflow**

Data tables are the most common thing that breaks on mobile; knowing the patterns keeps content usable on any screen.

SITE **CSS-Tricks: Responsive Data Tables**

https://css-tricks.com/responsive-data-tables/

---

099 **Testing across viewports: Chrome DevTools device mode, real-device testing, and the 320px-to-2560px sweep**

Catching breakpoints between the obvious ones is how you ship a layout that never visibly breaks, the masterpiece bar.

DOCS **Chrome DevTools: Simulate mobile devices with Device Mode**

https://developer.chrome.com/docs/devtools/device-mode

---

100 **Responsive performance: Core Web Vitals (LCP, CLS, INP) on mobile and Lighthouse mobile audits**

A responsive site that's slow on a real phone isn't done; mobile vitals are what users and Google actually judge.

DOCS **web.dev: Web Vitals**

https://web.dev/articles/vitals

---

05 Typography for the Web

101–125

101 **Anatomy of type: x-height, cap-height, ascenders/descenders, counters**

Knowing how letterforms are built lets you judge fonts and spacing instead of guessing, which separates intentional type from generic defaults.

DOCS **MDN: Fundamental text and font styling**

https://developer.mozilla.org/en-US/docs/Learn/CSS/Styling_text/Fundamentals

102 **font-family stacks and system font stacks for instant, dependable rendering**

A robust stack guarantees text never falls back to an ugly default and gives you a fast, native-feeling baseline.

SITE **CSS-Tricks: System Font Stack**

https://css-tricks.com/snippets/css/system-font-stack/

103 **Choosing typefaces with purpose: serif vs sans vs display, voice and context**

The typeface sets a brand's entire tone before a word is read, so deliberate selection is the first lever toward a distinctive site.

SITE **Butterick's Practical Typography**

https://practicaltypography.com/

104 **Font pairing: pairing by contrast and superfamily, limiting to two faces**

Confident, contrasting pairings create hierarchy and personality while avoiding the muddy, template-y look of mismatched fonts.

TOOL **Fontpair**

https://www.fontpair.co/

105 **Google Fonts: selecting, subsetting, and self-hosting families**

Mastering the largest free library and self-hosting it gives you control over both aesthetics and load performance.

TOOL **Google Fonts**

https://fonts.google.com/

106 **Type scale and modular scale (ratios like 1.25, 1.333, golden)**

A mathematical scale makes every heading size feel harmonious and intentional rather than arbitrary.

TOOL **Type Scale**

https://typescale.com/

107 **Establishing visual hierarchy with size, weight, color, and spacing**

Clear hierarchy guides the eye through a page in seconds and is the backbone of professional, scannable design.

SITE **Refactoring UI: Hierarchy is Everything**

https://www.refactoringui.com/

108 **Measure (line length): targeting 45-75 characters with the ch unit and max-width**

Optimal line length is the single biggest driver of comfortable reading, instantly making body copy feel considered.

COURSE **web.dev: Learn CSS — Sizing units (ch)**

https://web.dev/learn/css/sizing/

---

109 **Leading (line-height): unitless values and tuning by font size and measure**

Correct line spacing makes paragraphs breathe and dramatically improves readability and perceived polish.

DOCS **CSS-Tricks: line-height**

https://css-tricks.com/almanac/properties/l/line-height/

---

110 **Letter-spacing (tracking): tightening large headings, opening uppercase**

Subtle tracking adjustments are what make headlines and labels look designed rather than dropped-in.

DOCS **MDN: letter-spacing**

https://developer.mozilla.org/en-US/docs/Web/CSS/letter-spacing

---

111 **Vertical rhythm and baseline spacing using a consistent spacing unit**

Aligning text and gaps to a rhythmic grid gives a page an underlying order that reads as quality.

SITE **Smashing Magazine: Setting Web Type to a Baseline Grid**

https://www.smashingmagazine.com/2012/12/css-baseline-the-good-the-bad-and-the-ugly/

---

112 **Fluid typography with clamp() for responsive scale without breakpoints**

clamp() lets headlines scale smoothly across every screen, keeping type elegant on phones and large monitors alike.

SITE **Josh W. Comeau: The Sandbox (CSS articles)**

https://www.joshwcomeau.com/css/

---

113 **rem/em strategy and respecting the user's root font size**

Building type in relative units honors user settings and keeps a system scalable and accessible.

COURSE **web.dev: Learn CSS — Sizing units**

https://web.dev/learn/css/sizing/

---

114 **@font-face and the font-display property (swap, optional, fallback)**

Controlling how custom fonts load eliminates invisible-text flashes that make a polished site look broken on load.

DOCS **MDN: @font-face**

https://developer.mozilla.org/en-US/docs/Web/CSS/@font-face

---

115 **Web font performance: WOFF2, preload, subsetting, and reducing requests**

Fast fonts protect Core Web Vitals so beautiful type never costs you load speed or rankings.

SITE **web.dev: Best practices for fonts**

https://web.dev/articles/font-best-practices

---

116 **Eliminating layout shift (CLS) with size-adjust and font fallback metrics**

Matching fallback metrics to the web font keeps text from jumping during load, a hallmark of refined engineering.

SITE **web.dev: Improved font fallbacks**

https://web.dev/articles/css-size-adjust

---

117 **Variable fonts: weight, optical size, and width axes via font-variation-settings**

One variable font delivers a full weight range with smaller payloads, unlocking expressive hierarchy cheaply.

DOCS **MDN: Variable fonts guide**

https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_fonts/Variable_fonts_guide

---

118 **OpenType features: ligatures, oldstyle vs lining numerals, small caps (font-feature-settings)**

Activating real OpenType features gives type the craftsmanship and detail found on award-winning sites.

DOCS **CSS-Tricks: font-feature-settings**

https://css-tricks.com/almanac/properties/f/font-feature-settings/

---

119 **tabular-nums and font-variant-numeric for aligned figures in tables and pricing**

Tabular figures keep prices and stats neatly aligned, signaling the precision that builds client trust.

DOCS **MDN: font-variant-numeric**

https://developer.mozilla.org/en-US/docs/Web/CSS/font-variant-numeric

---

120 **Microtypography: smart quotes, em/en dashes, non-breaking spaces, and hanging punctuation**

Correct punctuation and detail are invisible when right but scream amateur when wrong, defining typographic craft.

SITE **Practical Typography: Punctuation**

https://practicaltypography.com/straight-and-curly-quotes.html

---

121 **Text wrapping control: text-wrap balance and pretty for clean headlines and paragraphs**

Balanced headlines and orphan-free paragraphs remove ragged awkwardness that cheapens otherwise good layouts.

DOCS **text-wrap (balance and pretty) - MDN**

https://developer.mozilla.org/en-US/docs/Web/CSS/text-wrap

---

122 **Readability and contrast: color, font size minimums, and WCAG text contrast**

Legible, accessible text ensures your design works for every visitor and passes the audits clients expect.

DOCS **W3C WAI: Text and Readability**

https://www.w3.org/WAI/tips/designing/

---

123 **Responsive type systems: scaling measure, leading, and hierarchy across breakpoints**

A type system that adapts holistically keeps reading comfortable and hierarchy intact on every device.

SITE **Smashing Magazine: Responsive Typography**

https://www.smashingmagazine.com/category/typography/

---

124 **Defining typographic design tokens (CSS custom properties) for a reusable type system**

Codifying scale, weights, and spacing into tokens makes type consistent across a site and fast to theme per client.

DOCS **MDN: Using CSS custom properties**

https://developer.mozilla.org/en-US/docs/Web/CSS/Using_CSS_custom_properties

---

125 **Expressive display typography: oversized type, mixed weights, and editorial layouts**

Bold, art-directed type is what pushes a marketing site from clean to memorable masterpiece territory.

SITE **Awwwards**

https://www.awwwards.com/

---

06 Color & Theming

126–150

126 **Color models: how RGB, HEX, and HSL actually map to a screen**

You can't reason about palettes or fix a 'muddy' color until you understand what each channel does and how the same color is expressed three ways.

DOCS **MDN: <color> CSS data type**

https://developer.mozilla.org/en-US/docs/Web/CSS/color_value

127 **HSL thinking: tuning hue/saturation/lightness instead of guessing HEX codes**

HSL lets you build and adjust palettes intuitively (lighten, mute, shift hue) which is the foundation of every consistent theme.

DOCS **CSS-Tricks: A Guide to Working With HSL Colors**

https://css-tricks.com/almanac/functions/h/hsl/

128 **Color theory basics: hue wheel, warm vs cool, value, and saturation**

Distinctive sites use deliberate hue relationships and value structure, not random swatches, so the theory underpins every choice.

SITE **Interaction Design Foundation: Color Theory**

https://www.interaction-design.org/literature/topics/color-theory

129 **Color harmonies: complementary, analogous, triadic, split-complementary**

Choosing a harmony scheme gives a palette internal logic so it feels intentional and harmonious rather than clashing.

TOOL **Adobe Color Wheel (interactive harmony tool)**

https://color.adobe.com/create/color-wheel

130 **Picking one brand/primary color that fits the business and its emotion**

A masterpiece site usually radiates from one confident, well-chosen anchor color that signals the brand's personality.

TOOL **Realtime Colors (test a palette live on a real layout)**

https://www.realtimecolors.com

131 **Building a full palette: primary, secondary, accent, neutrals from one seed**

Real interfaces need a structured set (not one color) so backgrounds, text, and CTAs all relate without looking flat.

TOOL **Coolors palette generator**

https://coolors.co

132 **Tonal/tint+shade scales: generating 50–900 steps for each color**

Polished UIs use a numbered scale per hue so hovers, borders, and surfaces stay consistent across the whole product.

DOCS **Building a color scheme (tints/shades tonal scales)**

https://web.dev/articles/building/a-color-scheme

133 **The 60-30-10 rule for distributing color across a layout**

Controlling how much of each color appears is what separates a calm, premium look from a noisy, amateur one.

SITE **Refactoring UI articles (Steve Schoger / Adam Wathan)**

https://www.refactoringui.com/

134 **Neutrals done right: warm vs cool grays and tinted neutrals**

Tinting your grays toward the brand hue is a subtle pro move that makes the whole page feel cohesive and custom.

SITE **Refactoring UI: Building Your Color Palette (neutrals/grays)**

https://refactoringui.com/previews/building-your-color-palette/

135 **WCAG contrast ratios: 4.5:1 body text, 3:1 large text/UI**

Legible, accessible color contrast is both a legal/usability requirement and a hallmark of professional, trustworthy work.

DOCS **W3C WAI: Contrast (Minimum) Understanding**

https://www.w3.org/WAI/WCAG21/Understanding/contrast-minimum.html

136 **Measuring contrast with a checker and fixing failing pairs**

You need a fast loop to verify text/background and button pairs actually pass before shipping.

TOOL **WebAIM Contrast Checker**

https://webaim.org/resources/contrastchecker/

137 **Why luminance != lightness, and why HSL contrast is deceptive**

Two colors with the same HSL lightness can have wildly different perceived brightness, which is why naive palettes fail contrast.

DOCS **web.dev: Color and contrast accessibility**

https://web.dev/articles/color-and-contrast-accessibility

138 **Don't rely on color alone: redundant cues for state and meaning**

Conveying errors/success/links with more than hue keeps your UI usable for colorblind users and clearer for everyone.

SITE **Understanding color blindness: don't rely on color alone**

https://www.a11yproject.com/posts/understanding-colourblindness/

139 **Designing for color blindness and simulating it**

Testing your palette under deuteranopia/protanopia ensures your accents and charts still read for ~8% of male users.

TOOL **Coblis — Color Blindness Simulator**

https://www.color-blindness.com/coblis-color-blindness-simulator/

140 **Semantic color tokens: naming colors by role, not by hue**

Tokens like --color-surface and --color-danger decouple meaning from value, the core of a maintainable, themeable system.

DOCS **Material Design 3: Color roles & system**

https://m3.material.io/styles/color/system/overview

141 **Implementing color tokens with CSS custom properties**

A single source of truth in CSS variables lets you restyle an entire site (or rebrand for a client) by editing a few lines.

DOCS **MDN: Using CSS custom properties (variables)**

https://developer.mozilla.org/en-US/docs/Web/CSS/Using_CSS_custom_properties

142 **Dark mode: building a second token set, not just inverting colors**

Great dark themes desaturate, re-balance elevation, and remap tokens — a true second palette that signals craftsmanship.

COURSE **web.dev: Building a color scheme (dark mode section)**

https://web.dev/articles/building/a-color-scheme

143 **prefers-color-scheme and a no-flash theme toggle**

Respecting the user's OS preference and persisting a toggle is the expected, polished behavior on any modern site.

DOCS **MDN: prefers-color-scheme**

https://developer.mozilla.org/en-US/docs/Web/CSS/@media/prefers-color-scheme

144 **Elevation & surface color in dark UI (lighter = closer)**

Communicating depth with subtly lighter surfaces instead of shadows is what makes dark interfaces feel premium.

DOCS **Material Design: Dark theme guidance**

https://m2.material.io/design/color/dark-theme.html

145 **OKLCH and perceptually-uniform color spaces**

OKLCH gives evenly-bright tonal scales and predictable lightness, the modern way to build palettes that contrast correctly.

TOOL **OKLCH Color Picker & Converter (oklch.com)**

https://oklch.com

146 **Why OKLCH beats HSL/HEX for scales, and using it in CSS**

Understanding perceptual uniformity lets you generate accent and neutral ramps that look even and stay accessible.

SITE **Falling For OKLCH: Color Spaces, Gamuts, And CSS**

https://www.smashingmagazine.com/2023/08/oklch-color-spaces-gamuts-css/

147 **Wide-gamut color, color() and display-p3 for vivid accents**

Targeting P3 lets accents and gradients pop on modern displays while you provide an sRGB fallback for the rest.

DOCS **High definition CSS color guide: color() and display-p3**

https://developer.chrome.com/docs/css-ui/high-definition-css-color-guide

148 **Gradients done well: multi-stop, hue interpolation, avoiding gray midpoints**

Tasteful gradients (right interpolation space, added midpoint stops) read as designed, not as a default linear-gradient.

SITE **Josh W. Comeau: The Joy of Gradients (Beautiful Gradients)**

https://www.joshwcomeau.com/css/make-beautiful-gradients/

149 **Modern color functions: color-mix(), relative color syntax, transparency**

Deriving hovers, tints, and overlays from one token with color-mix() keeps palettes DRY and effortlessly consistent.

DOCS **MDN: color-mix()**

https://developer.mozilla.org/en-US/docs/Web/CSS/color_value/color-mix

150 **Pairing color with type, imagery, and overlays for brand mood**

A masterpiece comes from color working with photography and text (legible overlays, on-brand tints), not color in isolation.

SITE **NN/g: The Role of Color in UX (Nielsen Norman Group)**

https://www.nngroup.com/articles/color-enhance-design/

07 Visual Design Principles

151–175

151 **The squint test: blur your design to check that hierarchy, grouping, and focal points survive without legible text**

It instantly reveals whether structure reads at a glance, which is how every visitor first perceives a page.

SITE **Refactoring UI (free articles & samples)**

https://www.refactoringui.com

152 **Visual hierarchy: ordering elements by size, weight, color, and position so the eye lands on the most important thing first**

A clear hierarchy is the single biggest difference between a masterpiece and a flat, confusing template.

SITE **NN/g: Visual Hierarchy in UX**

https://www.nngroup.com/articles/visual-hierarchy-ux-definition/

153 **Contrast as a design tool: using value, size, weight, and color difference to separate primary from secondary content**

Without deliberate contrast everything competes equally and nothing feels intentional or premium.

SITE **Refactoring UI: Hierarchy is everything (article)**

https://www.refactoringui.com

154 **Alignment: establishing strong invisible edges so every element relates to a shared grid line**

Crisp alignment is what subconsciously signals professionalism versus a thrown-together amateur layout.

SITE **NN/g: Principles of Visual Design**

https://www.nngroup.com/articles/principles-visual-design/

155 **Proximity: grouping related items close and pushing unrelated items apart to communicate relationships**

Correct proximity lets users parse a page without reading, making complex layouts feel effortless.

SITE **Laws of UX: Law of Proximity**

https://lawsofux.com/law-of-proximity/

156 **Repetition and consistency: reusing colors, spacing, shapes, and type styles to unify a design**

Consistent repetition creates rhythm and brand cohesion that makes a site feel designed, not assembled.

SITE **Laws of UX: Law of Similarity**

https://lawsofux.com/law-of-similarity/

157 **Whitespace and negative space: using generous, intentional emptiness to give content room to breathe**

Whitespace is the fastest way to make any layout look expensive, calm, and high-end.

SITE **Design Principles: Space And The Figure-Ground Relationship**

https://www.smashingmagazine.com/2014/05/design-principles-space-figure-ground-relationship/

158 **Focal points: deliberately creating one dominant element per view to anchor attention**

A single clear focal point gives each section a purpose and prevents visual chaos.

SITE **Interaction Design Foundation: Emphasis & Focal Points**

https://www.interaction-design.org/literature/topics/visual-hierarchy

159 **Gestalt: figure/ground relationship — controlling what the eye reads as object versus background**

Mastering figure/ground lets you build depth and layered compositions that feel sophisticated.

SITE **Smashing Magazine: Design Principles — Gestalt**

https://www.smashingmagazine.com/2014/03/design-principles-visual-perception-and-the-principles-of-gestalt/

160 **Gestalt: closure — letting the mind complete shapes the design only implies**

Closure enables elegant, minimal compositions that feel clever rather than literal.

SITE **Principle of Closure in Visual Design (NN/g)**

https://www.nngroup.com/articles/principle-closure/

161 **Gestalt: continuation — arranging elements along a line or curve to guide the eye smoothly**

Continuation directs the viewer's gaze through your layout in the exact order you intend.

SITE **Laws of UX: Law of Common Region & Continuation**

https://lawsofux.com/articles/

162 **Gestalt: common region — using shared containers/backgrounds to group elements**

Common region is a powerful, modern grouping tool behind card-based and bento layouts.

SITE **Laws of UX: Law of Common Region**

https://lawsofux.com/law-of-common-region/

163 **Symmetrical vs asymmetrical balance: distributing visual weight evenly or with intentional tension**

Choosing the right balance type sets the entire mood — stable and trustworthy vs dynamic and modern.

SITE **IxDF: Balance in Visual Design**

https://www.interaction-design.org/literature/topics/visual-design

164 **Visual weight: estimating how much attention each element draws via size, color, density, and isolation**

Knowing weight lets you balance and rebalance a composition deliberately instead of by guesswork.

SITE **NN/g: Visual Weight & Emphasis**

https://www.nngroup.com/articles/

165 **Establishing dominance, subordination, and accent: a clear primary, supporting secondary, and small accent**

This three-tier relationship is the backbone of every composition that reads as polished and intentional.

SITE **Refactoring UI: Establishing a hierarchy**

https://www.refactoringui.com

166 **Scale and proportion: using dramatic size contrast and ratios to create impact and elegance**

Bold scale contrast is what gives hero sections and editorial layouts their wow factor.

SITE **Smashing Magazine: Scale in Design**

https://www.smashingmagazine.com/category/design/

167 **Directing the eye / eye flow: building deliberate scan paths (Z-pattern, F-pattern, gaze direction)**

Controlling reading order ensures the headline, value prop, and CTA are seen in the right sequence.

SITE **NN/g: F-Shaped Pattern of Reading**

https://www.nngroup.com/articles/f-shaped-pattern-reading-web-content/

168 **Visual rhythm and pattern: repeating spacing/elements at intervals to create a satisfying cadence**

Rhythm makes long pages feel cohesive and guides the eye comfortably from section to section.

SITE **IxDF: Repetition, Pattern, and Rhythm**

https://www.interaction-design.org/literature/topics/visual-design

169 **Tension and white-space activation: using off-center placement and breathing room to energize layouts**

Active negative space turns static designs into compositions that feel deliberate and alive.

SITE **Design Principles: Space And The Figure-Ground Relationship**

https://www.smashingmagazine.com/2014/05/design-principles-space-figure-ground-relationship/

170 **Contrast of texture, depth, and dimensionality: combining flat, soft-shadow, and layered elements**
Subtle depth contrast adds richness that separates premium work from flat generic templates.

SITE **Refactoring UI: Creating depth**
https://www.refactoringui.com

171 **Unity and variety: balancing a cohesive system with enough variation to stay interesting**
Too much unity is boring, too much variety is chaos — the sweet spot is what feels masterful.

SITE **IxDF: Unity & Variety in Visual Design**
https://www.interaction-design.org/literature/topics/visual-design

172 **The 5-second / first-impression test: validating that the main message and brand land instantly**
Local-business visitors decide in seconds, so the design must communicate before they read a word.

SITE **NN/g: First Impressions & The 5-Second Test**
https://www.nngroup.com/articles/first-impressions-human-automaticity/

173 **Composition with grids and the rule of thirds: placing focal points on intersections and grid lines**
Grid-driven placement gives layouts an underlying order that feels intentional and balanced.

SITE **Smashing Magazine: Grids in Web Design**
https://www.smashingmagazine.com/category/design/

174 **Critiquing and refining: systematically auditing a design against each principle to find what feels off**
A repeatable critique loop is how good designers reliably elevate work to masterpiece quality.

SITE **NN/g: Design Critique articles**
https://www.nngroup.com/articles/design-critiques/

175 **Synthesizing all principles into a deliberate composition system for a full marketing page**
The mark of mastery is combining every principle into one coherent, distinctive page rather than applying them in isolation.

SITE **IxDF: Visual Design Principles (topic hub)**
https://www.interaction-design.org/literature/topics/visual-design

08 Layout & Composition

176–200

176 **The box model & document flow: how block/inline elements stack and push each other**
Every layout decision rests on understanding how the browser places boxes before you override it.

DOCS **MDN: The box model**
https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/The_box_model

177 **Spacing scale: choosing a consistent step set (4/8px base) instead of arbitrary margins**
A single repeating spacing scale is what separates calm professional layouts from random amateur ones.

SITE **Refactoring UI: Spacing and sizing systems**
https://www.refactoringui.com/

178 **Vertical rhythm: aligning line-height, margins, and headings to one baseline unit**
Consistent vertical rhythm makes a page feel typeset and intentional rather than thrown together.

SITE **24 Ways: Compose to a Vertical Rhythm**
https://24ways.org/2006/compose-to-a-vertical-rhythm/

179 **Flexbox: main/cross axis, justify vs align, and flex-grow/shrink/basis**

Flexbox is the everyday tool for component-level alignment and distributing space precisely.

INTERACTIVE **Flexbox Froggy**

https://flexboxfroggy.com

---

180 **CSS Grid fundamentals: rows, columns, tracks, gaps and the fr unit**

Grid is the foundation of any real page-level layout system and true two-dimensional control.

INTERACTIVE **Grid Garden**

https://cssgridgarden.com

---

181 **CSS Grid: named template areas for readable, rearrangeable page structures**

Named areas let you draw a whole layout in plain English and recompose it per breakpoint.

COURSE **Wes Bos: CSS Grid (free course)**

https://cssgrid.io

---

182 **Establishing a column grid system (12-col, gutters, margins) and designing to it**

A shared column grid gives every section underlying alignment that reads as professional polish.

DOCS **Grids & Spacing - Material Design 3**

https://m3.material.io/foundations/layout/grids-spacing/grids

---

183 **Alignment as the core principle: edges and baselines that line up across sections**

Crisp shared alignment lines are the single biggest signal of a designed-not-defaulted page.

SITE **5 Principles of Visual Design in UX (NN/g)**

https://www.nngroup.com/articles/principles-visual-design/

---

184 **Proximity & grouping: using whitespace to define relationships, not borders**

Correct grouping through space makes complex pages instantly scannable and uncluttered.

SITE **Proximity Principle in Visual Design (NN/g)**

https://www.nngroup.com/articles/gestalt-proximity/

---

185 **Whitespace & negative space: macro and micro spacing as a design element**

Generous, deliberate whitespace is what makes a site feel premium rather than crammed.

SITE **The Power of White Space in Design (Interaction Design Foundation)**

https://ixdf.org/literature/article/the-power-of-white-space

---

186 **Visual hierarchy: size, weight, color, and position to control reading order**

A clear hierarchy guides the eye to the offer and CTA, which is the whole point of a marketing site.

COURSE **web.dev Learn: Layout**

https://web.dev/learn/css/layout

---

187 **Reading patterns: F-pattern and Z-pattern layouts for scannable pages**

Placing key content along natural scan paths means visitors absorb the message without effort.

SITE **NN/g: F-Shaped Pattern of Reading**

https://www.nngroup.com/articles/f-shaped-pattern-reading-web-content/

---

188 **Rule of thirds & focal points for composing hero sections and imagery**

Off-center focal placement creates dynamic, photographic heroes instead of dead-centered blandness.

SITE **Interaction Design Foundation: The Rule of Thirds**

https://www.interaction-design.org/literature/topics/rule-of-thirds

---

189 **Golden ratio & modular proportion for sizing sections and content widths**

Proportional sizing relationships give layouts an underlying harmony viewers feel but cannot name.

SITE **Applying Divine Proportion To Your Web Designs**

https://www.smashingmagazine.com/2008/05/applying-divine-proportion-to-web-design/

190 **Measure & content width: optimal line length and max-width for readability**

Constraining text width to ~60-75 characters keeps long-form sections readable and elegant.

DOCS **web.dev: Centering elements & content width patterns**

https://web.dev/learn/css/layout

---

191 **Density & information density: tuning how much content lives in a given space**

Matching density to audience and content prevents both sparse-empty and overwhelming-busy pages.

SITE **Content Density: The Negative Impact of Mobile-First on Desktop**

https://www.nngroup.com/articles/content-dispersion/

---

192 **Page flow & rhythm: alternating section heights, backgrounds, and pacing**

Varied section pacing keeps a scroll engaging instead of a monotonous stack of identical bands.

SITE **Land-book — landing page gallery**

https://land-book.com

---

193 **Editorial layout: magazine-style composition, pull quotes, and column mixing**

Editorial techniques borrowed from print make a marketing site feel curated and distinctive.

SITE **Smashing Magazine: Design Principles Behind Editorial Layouts**

https://www.smashingmagazine.com/category/typography/

---

194 **Asymmetric & broken-grid layouts that still respect an underlying structure**

Intentional asymmetry is the hallmark of award-winning sites that escape the centered-template look.

SITE **Awwwards — collections & inspiration**

https://www.awwwards.com

---

195 **Breaking the grid on purpose: overlapping elements with negative margins, grid spans, and z-index**

Controlled rule-breaking creates the bold, layered compositions that make a site memorable.

VIDEO **Kevin Powell (CSS layout, YouTube)**

https://www.youtube.com/@KevinPowell

---

196 **Layering & overlap with CSS Grid (stacking items in the same cell)**

Grid layering lets you build sophisticated overlapping art-directed sections without absolute-position hacks.

DOCS **CSS-Tricks: A Complete Guide to CSS Grid**

https://css-tricks.com/snippets/css/complete-guide-grid/

---

197 **Intrinsic & responsive layouts: minmax, auto-fit/auto-fill, clamp() for fluid composition**

Intrinsic techniques let layouts adapt gracefully across viewports with almost no media queries.

DOCS **web.dev: Every Layout / Intrinsic responsive design**

https://web.dev/learn/css/grid

---

198 **Container queries: composing components that respond to their container, not the viewport**

Container queries make reusable sections truly modular, adapting wherever you drop them.

DOCS **MDN: CSS container queries**

https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_containment/Container_queries

---

199 **Consistent layout primitives: stack, cluster, sidebar, switcher, cover, grid (composable patterns)**

A small set of reusable layout primitives makes every page systematic, fast to build, and coherent.

SITE **Every Layout (free pattern explanations)**

https://every-layout.dev/

---

200 **Composition critique: spacing audits, alignment checks, and the squint test on your own work**

A repeatable critique loop is how you catch misalignment and uneven rhythm before a client ever sees it.

SITE **Refactoring UI (free articles & book site)**

https://www.refactoringui.com/

09 UX & Interaction Design

201–225

201 **Nielsen's 10 usability heuristics: scan any UI against the full list**

These ten principles are the fastest checklist for spotting why an interface feels confusing before users ever do.

SITE **10 Usability Heuristics for User Interface Design**

https://www.nngroup.com/articles/ten-usability-heuristics/

202 **Visibility of system status: every action gets immediate, legible feedback**

Users trust a site that always tells them what just happened and what is happening now, which kills hesitation on a sales page.

SITE **Visibility of System Status (NN/g)**

https://www.nngroup.com/articles/visibility-system-status/

203 **Affordances and signifiers: making clickable things look clickable**

A masterpiece looks effortless because every control silently signals exactly how it should be used.

SITE **Beyond Blue Links: Making Clickable Elements Recognizable**

https://www.nngroup.com/articles/clickable-elements/

204 **Mental models and Jakob's Law: matching user expectations from other sites**

Honoring conventions for nav, cart, and forms lets visitors focus on your message instead of relearning the interface.

INTERACTIVE **Jakob's Law (Laws of UX)**

https://lawsofux.com/jakobs-law/

205 **Information architecture: grouping and labeling content so it's findable**

Clear IA is the invisible skeleton that makes a complex local-business site feel simple and premium.

SITE **Information Architecture Basics**

https://www.usability.gov/what-and-why/information-architecture.html

206 **Card sorting to derive navigation from real user grouping**

Letting structure emerge from how users actually categorize content prevents the org-chart navigation that confuses everyone.

SITE **Card Sorting (NN/g)**

https://www.nngroup.com/articles/card-sorting-definition/

207 **Visual hierarchy for scanning: directing the eye to what matters first**

Controlling the order people perceive elements is how you make a hero section sell in three seconds.

SITE **Visual Hierarchy (NN/g)**

https://www.nngroup.com/articles/visual-hierarchy-ux-definition/

208 **F-pattern and Z-pattern scanning: laying out content the way people read**

Placing key copy and CTAs along natural scan paths means your message lands even when users skim.

SITE **F-Shaped Pattern for Reading Web Content**

https://www.nngroup.com/articles/f-shaped-pattern-reading-web-content/

209 **Navigation patterns: choosing the right nav for the site's depth**

The right nav model keeps a multi-service business site feeling uncluttered and instantly navigable.

SITE **Navigation Patterns (Smashing Magazine)**

https://www.smashingmagazine.com/category/navigation/

210 **User flows and task analysis: mapping the path to conversion**

Designing the full journey, not just screens, removes friction at every step from landing to lead.

SITE **User Journey Mapping (NN/g)**

https://www.nngroup.com/articles/journey-mapping-101/

211 **Interaction states: hover, focus, active, disabled, loading, empty, error**

Designing every state, not just the happy path, is what separates polished products from prototypes.

SITE **Designing Better Error Messages UX**

https://www.smashingmagazine.com/2022/08/error-messages-ux-design/

212 **Loading and skeleton states: keeping perceived performance high**

Smart loading feedback makes a site feel fast and intentional instead of broken during waits.

SITE **Progress Indicators (NN/g)**

https://www.nngroup.com/articles/progress-indicators/

213 **Empty states as onboarding and guidance moments**

A well-designed empty state turns a dead screen into a helpful nudge that builds trust.

SITE **The Role Of Empty States In User Onboarding**

https://www.smashingmagazine.com/2017/02/user-onboarding-empty-states-mobile-apps/

214 **Forms UX: layout, single-column, logical grouping, and field order**

Forms are where leads are won or lost, so reducing their friction directly raises conversions.

SITE **Web Form Design Guidelines (NN/g)**

https://www.nngroup.com/articles/web-form-design/

215 **Inline validation and helpful error recovery**

Catching errors gently in context keeps users from abandoning the one form that becomes a sale.

SITE **Designing Better Form Validation (Smashing)**

https://www.smashingmagazine.com/2022/09/inline-validation-web-forms-ux/

216 **Input types, labels, placeholders, and autofill that work**

Correct field semantics make forms faster on mobile and accessible, which lifts completion rates.

DOCS **MDN: HTML Forms Guide**

https://developer.mozilla.org/en-US/docs/Learn/Forms

217 **Microcopy: writing buttons, labels, and helper text that guide and reassure**

Precise tiny words remove doubt at decision points and make the whole product feel considered.

SITE **UX Writing / Microcopy (NN/g)**

https://www.nngroup.com/articles/microcontent-how-to-write-headlines-page-titles-and-subject-lines/

218 **Error message writing: clear, blameless, actionable**

Good error copy turns a frustrating dead-end into a recoverable moment that protects the conversion.

SITE **Error-Message Guidelines (NN/g)**

https://www.nngroup.com/articles/error-message-guidelines/

219 **Call-to-action design: clarity, specificity, and single primary action**

One unmistakable CTA per view is the core mechanic of a conversion-focused marketing site.

SITE **Better CTA Buttons (NN/g)**

https://www.nngroup.com/articles/clickable-elements/

220 **Conversion-focused design: above-the-fold value prop and trust signals**

Communicating value and credibility instantly is what makes a local-business page actually generate leads.

SITE **Homepage Design: 5 Fundamental Principles**

https://www.nngroup.com/articles/homepage-design-principles/

221 **Persuasion and behavioral principles: reciprocity, social proof, scarcity (ethical use)**

Understanding decision psychology lets you design honest nudges that move visitors to act.

SITE **The Reciprocity Principle: Give Before You Take in Web Design**

https://www.nngroup.com/articles/reciprocity-principle/

222 **Reducing cognitive load: Hick's Law, chunking, and progressive disclosure**

Fewer, well-staged choices keep users moving toward the goal instead of stalling.

SITE **Progressive Disclosure (NN/g)**

https://www.nngroup.com/articles/progressive-disclosure/

223 **Mobile interaction design: touch targets, thumb zones, and tap-friendly nav**

Most local-business traffic is mobile, so thumb-first design decides whether the lead converts.

DOCS **Touch Target Sizes (Material Design)**

https://m3.material.io/foundations/designing/structure

224 **Designing modals, overlays, and notifications without trapping users**

Interruptions done right inform and confirm; done wrong they break flow and erode trust.

SITE **Modal & Nonmodal Dialogs (NN/g)**

https://www.nngroup.com/articles/modal-nonmodal-dialog/

225 **Usability testing: running a quick 5-user test to validate flows**

Watching real people use your site is the only reliable way to confirm a design truly works before launch.

SITE **Usability Testing 101 (NN/g)**

https://www.nngroup.com/articles/usability-testing-101/

10 Accessibility (a11y)

226–250

226 **Semantic HTML: use native elements (button, nav, main, header, footer, article) instead of div soup**

Native semantics give screen readers, keyboard, and SEO meaning for free, so a masterpiece is accessible by default before any ARIA is added.

DOCS **MDN: HTML elements reference**

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

227 **The four WCAG principles (POUR): Perceivable, Operable, Understandable, Robust, and conformance levels A/AA/AAA**

POUR is the mental model that turns scattered a11y rules into a coherent standard you can design and audit against.

DOCS **W3C WAI: WCAG 2 Overview**

https://www.w3.org/WAI/standards-guidelines/wcag/

228 **Document structure: one logical h1, nested heading hierarchy (h1>h2>h3) with no skipped levels**

Screen-reader users navigate by headings, so a clean outline is the table of contents that makes a polished page actually usable.

COURSE **web.dev: Headings and landmarks**

https://web.dev/learn/accessibility/structure

229 **Landmark regions: header, nav, main, aside, footer for screen-reader page navigation**

Landmarks let assistive-tech users jump between regions instantly, the structural polish that separates pro sites from div piles.

SITE **ARIA Landmark Roles and HTML5 Implicit Mapping**

https://www.a11yproject.com/posts/aria-landmark-roles/

230 **Color contrast: meet WCAG AA ratios (4.5:1 body text, 3:1 large text and UI components)**

Sufficient contrast keeps your beautiful palette legible for low-vision users and in sunlight, which is non-negotiable for credible marketing sites.

TOOL **WebAIM Contrast Checker**

https://webaim.org/resources/contrastchecker/

231 **Never rely on color alone: pair color cues with icons, text, or patterns**

Color-blind users miss meaning conveyed only by hue, so distinguishable states keep your design clear without dumbing it down.

DOCS **WCAG: Use of Color (SC 1.4.1)**

https://www.w3.org/WAI/WCAG21/Understanding/use-of-color.html

232 **Alt text: write meaningful descriptions for informative images and empty alt='' for decorative ones**

Good alt text conveys the image's purpose to screen readers and crawlers, the invisible craft that makes visuals work for everyone.

DOCS **W3C WAI: Images Tutorial (alt decision tree)**

https://www.w3.org/WAI/tutorials/images/

233 **Accessible link text: descriptive, unique labels (no 'click here' / 'read more' alone)**

Screen-reader users browse links out of context, so self-describing links make navigation effortless and boost SEO.

SITE **Nielsen Norman Group: Better Link Labels**

https://www.nngroup.com/articles/better-link-labels/

234 **Keyboard navigation: every interactive element reachable and operable with Tab/Shift-Tab/Enter/Space/Esc**

Keyboard-only and motor-impaired users must reach everything without a mouse, the baseline test a masterpiece always passes.

COURSE **web.dev: Keyboard access**

https://web.dev/learn/accessibility/focus

235 **Logical focus order: DOM order matches visual reading order, avoid positive tabindex**

A sensible tab sequence keeps interaction predictable so clever CSS layouts never trap or scramble keyboard users.

DOCS **WCAG: Focus Order (SC 2.4.3)**

https://www.w3.org/WAI/WCAG21/Understanding/focus-order.html

236 **Visible focus indicators: design custom :focus-visible styles instead of removing outlines**

A beautiful, high-contrast focus ring tells keyboard users where they are, turning an a11y requirement into a design signature.

SITE **Sara Soueidan: A guide to designing accessible focus indicators**

https://www.sarasoueidan.com/blog/focus-indicators/

237 **Skip links: a 'Skip to main content' link as the first focusable element**

Skip links spare keyboard users from tabbing through the whole nav on every page, a tiny detail that signals real craftsmanship.

SITE **WebAIM: Skip Navigation Links**

https://webaim.org/techniques/skipnav/

238 **Accessible forms: every input has an associated <label>, grouped with fieldset/legend**

Properly labeled fields are the difference between a lead form anyone can submit and one that silently loses disabled customers.

COURSE **web.dev: Forms (accessibility)**

https://web.dev/learn/forms/

239 **Form errors & validation: inline messages tied to inputs via aria-describedby and aria-invalid**

Clear, programmatically linked errors let everyone recover from mistakes, protecting conversions on the forms that earn your clients money.

DOCS **W3C WAI: Form Validation & Error Messages**

https://www.w3.org/WAI/tutorials/forms/notifications/

240 **Input affordances: correct type, autocomplete tokens, and inputmode for mobile/assistive tech**

Right input types trigger the proper keyboard and autofill, smoothing the journey so polished forms feel effortless.

DOCS **MDN: HTML autocomplete attribute**

https://developer.mozilla.org/en-US/docs/Web/HTML/Attributes/autocomplete

241 **ARIA first rule: prefer native HTML; only add ARIA when no native element exists**

Misused ARIA breaks more than it fixes, so knowing when NOT to use it keeps your custom components genuinely robust.

DOCS **W3C: Using ARIA (Rules of ARIA use)**

https://www.w3.org/TR/using-aria/

242 **ARIA roles, states & properties: aria-label, aria-expanded, aria-current, aria-hidden used correctly**

Precise ARIA on custom UI (menus, tabs, accordions) makes distinctive components speak clearly to screen readers.

DOCS **MDN: ARIA reference**

https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA

243 **ARIA Authoring Practices: build accessible disclosures, tabs, dialogs, menus, and accordions to spec**

The APG gives battle-tested keyboard and ARIA patterns so your fancy interactive widgets behave correctly everywhere.

DOCS **ARIA Authoring Practices Guide (APG) Patterns**

https://www.w3.org/WAI/ARIA/apg/patterns/

244 **Focus management for modals & menus: trap focus inside, return focus on close, manage initial focus**

Correct focus handling makes overlays feel native rather than broken, the hallmark of a high-end interactive site.

COURSE **web.dev: Focus management**

https://web.dev/learn/accessibility/focus

245 **Live regions: announce dynamic updates (toasts, async results, cart changes) with aria-live politely/assertively**

Live regions tell screen-reader users what changed on the page, so your slick AJAX interactions aren't silent to them.

DOCS **MDN: ARIA live regions**

https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA/ARIA_Live_Regions

246 **Reduced motion: honor prefers-reduced-motion to tone down parallax, autoplay, and large animations**

Respecting motion sensitivity keeps your animations delightful for most while preventing nausea for vestibular users.

DOCS **web.dev: prefers-reduced-motion**

https://web.dev/articles/prefers-reduced-motion

247 **Accessible names & accessible name computation: how labels, aria-label, alt, and title resolve**

Understanding how the accessible name is computed lets you guarantee every control announces the right thing, every time.

DOCS **MDN: Accessible name**

https://developer.mozilla.org/en-US/docs/Glossary/Accessible_name

248 **Screen reader testing: drive VoiceOver (Mac/iOS) and NVDA (Windows) to experience your site as users do**

Hearing your own page read aloud surfaces problems no checklist catches, turning guesswork into confident craft.

SITE **WebAIM: Screen Reader testing guides**

https://webaim.org/articles/screenreader_testing/

249 **Automated + manual auditing: run axe DevTools / Lighthouse, then manually keyboard- and SR-test the gaps**

Tools catch ~40% of issues, so combining automated scans with manual checks is how you actually ship accessible work.

TOOL **Deque axe DevTools (browser extension)**

https://www.deque.com/axe/devtools/

250 **Responsive a11y: text resize to 200%, reflow at 320px, 44px+ touch targets, and zoom without loss**

Accessible responsiveness ensures your masterpiece stays usable on real devices for low-vision and motor-impaired visitors.

DOCS **WCAG: Reflow (SC 1.4.10) & Target Size**

https://www.w3.org/WAI/WCAG21/Understanding/reflow.html

11 Animation & Motion

251–275

251 **CSS transitions: property, duration, timing-function, delay shorthand**

Transitions are the cheapest way to make state changes feel intentional instead of jarring, the baseline of every polished UI.

DOCS **MDN: Using CSS transitions**

https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_transitions/Using_CSS_transitions

252 **Which properties are safe to animate (transform/opacity) vs layout-triggering (width/top/margin)**

Animating only compositor-friendly properties keeps motion at 60fps and prevents the janky reflows that scream 'amateur'.

DOCS **web.dev: Animations and performance**

https://web.dev/articles/animations-guide

253 **@keyframes fundamentals: percentage steps, from/to, animation shorthand**

Keyframes unlock multi-step motion (loaders, attention nudges, hero reveals) that transitions alone can't express.

DOCS **MDN: Using CSS animations**

https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_animations/Using_CSS_animations

254 **Easing curves: ease-in vs ease-out vs ease-in-out and why ease-out feels best for entrances**

The right curve is the difference between motion that feels mechanical and motion that feels physical and premium.

TOOL **Easings.net**

https://easings.net

255 **Custom cubic-bezier() curves and building a signature easing for a brand**

A consistent custom curve gives a site a recognizable 'feel' that generic ease-defaults never achieve.

TOOL **cubic-bezier.com**

https://cubic-bezier.com

256 **The 12 principles of animation applied to UI (timing, spacing, anticipation, follow-through)**

Disney's principles are the deep grammar behind why elite motion reads as alive rather than arbitrary.

VIDEO **The Illusion of Life (12 Principles) — Cento Lodigiani**

https://www.youtube.com/watch?v=uDqjIdI4bF4

257 **Animation duration scales: 150–300ms for UI, when slow vs fast reads right**

Durations tuned to human perception make interactions feel instant yet smooth instead of sluggish or twitchy.

DOCS **Material Design: Motion — Speed**

https://m2.material.io/design/motion/speed.html

258 **Micro-interactions: hover, focus, active, and toggle feedback states**

Tiny tactile responses on every interactive element are what separate a 'designed' product from a static page.

SITE **NN/g: Microinteractions in UX**

https://www.nngroup.com/articles/microinteractions/

259 **transform-origin and chaining transforms (translate, scale, rotate) for richer motion**

Controlling the pivot point and combining transforms produces the nuanced movement that flat scale/fade can't.

DOCS **MDN: transform-origin**

https://developer.mozilla.org/en-US/docs/Web/CSS/transform-origin

260 **will-change, GPU layers, and avoiding over-promotion / layer explosions**

Knowing when to hint the compositor (and when not to) keeps animations smooth without trashing memory.

DOCS **CSS-Tricks: Almanac — will-change**

https://css-tricks.com/almanac/properties/w/will-change/

261 **prefers-reduced-motion: building accessible, opt-out motion**

Respecting reduced-motion is both an accessibility duty and a mark of professional craft that template sites ignore.

DOCS **web.dev: prefers-reduced-motion**

https://web.dev/articles/prefers-reduced-motion

262 **Staggered animations: sequencing list/grid items with incremental delays**

Stagger turns a flat block reveal into a choreographed entrance that feels expensive and considered.

SITE **Josh W. Comeau: An Interactive Guide to CSS Transitions**

https://www.joshwcomeau.com/animation/css-transitions/

263 **FLIP technique for animating layout changes performantly**

FLIP lets you animate position/size shifts (the forbidden properties) smoothly via transforms, enabling slick reordering.

SITE **CSS-Tricks: Animating Layouts with the FLIP Technique**

https://css-tricks.com/animating-layouts-with-the-flip-technique/

264 **Spring physics vs duration-based easing and when springs feel more natural**

Spring-based motion mirrors real-world inertia, giving interactive UI a responsive, tactile quality fixed durations lack.

SITE **Emil Kowalski: Great Animations**

https://emilkowal.ski/ui/great-animations

265 **Scroll-driven animations with native CSS: animation-timeline, scroll() and view()**

Native scroll-linked motion delivers buttery reveal/parallax effects with zero JS, the modern way to do scroll storytelling.

INTERACTIVE **Chrome Developers: Scroll-driven animations**

https://scroll-driven-animations.style/

266 **IntersectionObserver for scroll-triggered reveal animations (JS fallback)**

Observer-based reveals are the reliable, performant pattern for 'animate in when visible' across all browsers.

DOCS **MDN: Intersection Observer API**

https://developer.mozilla.org/en-US/docs/Web/API/Intersection_Observer_API

267 **Parallax and scroll-progress effects done tastefully (depth without nausea)**

Subtle layered depth adds richness and 'wow' to hero sections while overdone parallax instantly looks cheap.

VIDEO **Kevin Powell (CSS) — YouTube**

https://www.youtube.com/@KevinPowell

268 **The Web Animations API (element.animate) for dynamic, JS-controlled motion**

WAAPI gives keyframe power with runtime control (play, pause, reverse, sequence) without a heavy library.

DOCS **MDN: Web Animations API**

https://developer.mozilla.org/en-US/docs/Web/API/Web_Animations_API

269 **requestAnimationFrame and the render loop fundamentals**

Understanding the frame budget (~16ms) and rAF is essential for writing custom JS motion that never stutters.

DOCS **MDN: requestAnimationFrame**

https://developer.mozilla.org/en-US/docs/Web/API/Window/requestAnimationFrame

270 **GSAP: timelines, tweens, and ScrollTrigger for complex choreography**

GSAP is the industry standard for award-winning sites, enabling precise multi-element timelines beyond what CSS can do.

COURSE **GSAP Learning Resources**

https://gsap.com/resources/

271 **Framer Motion (Motion): declarative animations, variants, layout & exit animations in React**

Motion makes production-grade entrance/exit and shared-layout transitions trivial for React-based client sites.

DOCS **Motion (Framer Motion) Documentation**

https://motion.dev/docs

272 **Lottie: shipping designer-made vector animations (After Effects → web) at tiny file size**

Lottie lets you deliver rich, illustrative motion (icons, mascots, empty states) that hand-coded CSS can't match.

TOOL **LottieFiles**

https://lottiefiles.com/

273 **SVG animation: animating paths, stroke-dashoffset line draws, and morphing**

Animated SVG (logo draws, icon morphs) creates the crisp, scalable brand motion that defines distinctive sites.

SITE **CSS-Tricks: A Guide to SVG Animations (SMIL)**

https://css-tricks.com/guide-svg-animations-smil/

274 **Page & view transitions: the View Transitions API for seamless route/state changes**

Smooth cross-page morphs make a multi-page site feel like a cohesive app, a top signal of premium craft.

DOCS **Chrome Developers: View Transitions API**

https://developer.chrome.com/docs/web-platform/view-transitions

275 **Motion choreography & restraint: a consistent system, purposeful motion, knowing when NOT to animate**

A disciplined, unified motion language is what makes a site feel like a masterpiece rather than a demo reel of effects.

DOCS **Material Design 3: Motion**

https://m3.material.io/styles/motion/overview

12 JavaScript Essentials

276–300

276 **Variables & scope: let, const, and block scoping vs var**

Choosing const by default and understanding scope prevents the subtle state bugs that make interactive sites flaky.

DOCS **javascript.info: Variables**

https://javascript.info/variables

277 **Data types & type coercion: ===, falsy values, typeof**

Knowing how JS coerces types stops the mysterious truthy/equality bugs that silently break conditional UI logic.

DOCS **MDN: Equality comparisons and sameness**

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Equality_comparisons_and_sameness

278 **Functions, arrow functions, and default/rest parameters**

Clean, reusable functions are the unit of organized code that keeps a polished site maintainable as it grows.

DOCS **javascript.info: Functions**

https://javascript.info/function-basics

279 **Template literals & string methods for dynamic text**

Backtick interpolation lets you build dynamic, formatted content without messy concatenation, keeping rendered copy clean.

DOCS **MDN: Template literals**

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Template_literals

280 **Objects: literals, dot/bracket access, shorthand, optional chaining (?.)**

Objects model the structured data behind every component, and optional chaining safely renders nested data without crashes.

DOCS **javascript.info: Objects**

https://javascript.info/object

281 **Destructuring & the spread/rest operators**

Destructuring and spread make data-handling concise and immutable, the patterns expected in modern professional codebases.

DOCS **MDN: Destructuring assignment**

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring

282 **Array essentials: map, filter, reduce, find, some/every**

These methods transform data into UI declaratively, the core skill for rendering lists and cards from a dataset.

DOCS **javascript.info: Array methods**

https://javascript.info/array-methods

283 **Selecting the DOM: querySelector / querySelectorAll**

Precisely grabbing elements is the gateway to making any static design interactive.

DOCS **MDN: Locating DOM elements using selectors**

https://developer.mozilla.org/en-US/docs/Web/API/Document_object_model/Locating_DOM_elements_using_selectors

284 **Manipulating the DOM: classList, textContent, dataset, setAttribute**

Toggling classes and data attributes drives polished state changes (active, open, loading) without rewriting markup.

DOCS **MDN: Manipulating documents**

https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Scripting/DOM_scripting

285 **Creating & inserting nodes: createElement, append, insertAdjacentHTML**

Generating elements in code is how you render dynamic content like testimonials or product grids from data.

DOCS **MDN: Element.insertAdjacentHTML**

https://developer.mozilla.org/en-US/docs/Web/API/Element/insertAdjacentHTML

286 **Events: addEventListener, the event object, preventDefault**

Event handling turns clicks, scrolls, and form submits into the interactions that make a site feel alive and intentional.

DOCS **javascript.info: Introduction to browser events**

https://javascript.info/introduction-browser-events

287 **Event delegation & bubbling**

Delegating events to a parent keeps handlers efficient and works for dynamically added elements, a hallmark of clean code.

DOCS **javascript.info: Event delegation**

https://javascript.info/event-delegation

288 **Forms: capturing input, validation, and the submit event**

Handling forms gracefully (real validation, no page reload) is essential for the lead-capture forms every client site needs.

COURSE **web.dev: Learn Forms**

https://web.dev/learn/forms

289 **JavaScript30 — 30 vanilla JS projects, no frameworks**

Building real interactive widgets by hand cements DOM and event skills far better than reading ever could.

COURSE **JavaScript30 by Wes Bos**

https://javascript30.com

290 **The event loop, callbacks, setTimeout & asynchronicity basics**

Understanding async execution explains why timing-dependent animations and data loads behave the way they do.

DOCS **javascript.info: Introduction: callbacks**

https://javascript.info/callbacks

291 **Promises: then/catch, chaining, Promise.all**

Promises are the foundation of all modern data fetching and let you coordinate multiple loads without callback chaos.

DOCS **javascript.info: Promise**

https://javascript.info/promise-basics

292 **async/await and try/catch error handling**

async/await makes asynchronous data code read like clean synchronous logic, with proper error handling for resilient UIs.

DOCS **javascript.info: Async/await**

https://javascript.info/async-await

293 **Fetch API: GET/POST, headers, and parsing JSON responses**

Fetch is how you pull live data (reviews, listings, pricing) into a site to make it feel dynamic and current.

DOCS **MDN: Using the Fetch API**

https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch

294 **Fetching & rendering data: loading, empty, and error states**

Designing for loading and failure states is what separates a masterpiece-quality data UI from one that flashes broken.

COURSE **web.dev: Learn JavaScript**

https://web.dev/learn/javascript

295 **ES Modules: import/export and organizing code into files**

Modules keep code modular and reusable so a growing site stays clean instead of becoming one tangled script.

DOCS **javascript.info: Modules, introduction**

https://javascript.info/modules-intro

296 **Debugging with DevTools: breakpoints, console methods, the Network tab**

Systematic debugging turns mysterious failures into solvable problems, saving hours and keeping output professional.

DOCS **Chrome DevTools: JavaScript debugging**

https://developer.chrome.com/docs/devtools/javascript

297 **Throttle & debounce for scroll, resize, and input handlers**

Rate-limiting expensive handlers keeps scroll animations and live search smooth instead of janky, a key polish detail.

SITE **CSS-Tricks: Debouncing and Throttling Explained**

https://css-tricks.com/debouncing-throttling-explained-examples/

298 **IntersectionObserver for scroll-triggered reveals & lazy loading**

This is the efficient, modern API behind tasteful scroll-in animations and performant lazy-loaded media.

DOCS **MDN: Intersection Observer API**

https://developer.mozilla.org/en-US/docs/Web/API/Intersection_Observer_API

299 **Common patterns: guard clauses, immutability, and pure helper functions**

Clean idiomatic patterns make code predictable and easy to extend, the discipline behind genuinely maintainable sites.

COURSE **The Odin Project: JavaScript course**

https://www.theodinproject.com/paths/full-stack-javascript/courses/javascript

300 **localStorage & JSON for persisting simple client state**

Saving small bits of state (theme, dismissed banners, cart) lets a site remember users and feel thoughtfully built.

DOCS **MDN: Using the Web Storage API**

https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API/Using_the_Web_Storage_API

13 Modern Frontend Engineering

301–325

301 **Component thinking: decomposing a UI into a tree of small, single-responsibility components**
Breaking a marketing page into reusable Hero/Card/Button units is the mental model that keeps masterpiece sites consistent and maintainable as they grow.

DOCS **Thinking in React**
https://react.dev/learn/thinking-in-react

302 **npm fundamentals: package.json, dependencies vs devDependencies, scripts, semver, lockfiles**
Every modern build pipeline runs on npm, so understanding installs, scripts, and version pinning prevents the broken-dependency chaos that wrecks polished projects.

DOCS **An introduction to the npm package manager (Node.js docs)**
https://nodejs.org/en/learn/getting-started/an-introduction-to-the-npm-package-manager

303 **ES Modules: import/export, named vs default exports, module scope**
Modules are the unit of code organization in every framework and bundler, so clean imports/exports keep a large component codebase navigable.

DOCS **JavaScript Modules (MDN)**
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules

304 **Modern JS essentials for frameworks: destructuring, spread/rest, arrow fns, ternaries, map/filter, optional chaining**
Framework code is written almost entirely in these idioms, so fluency here is the difference between fighting the syntax and expressing UI cleanly.

COURSE **JavaScript30 (Wes Bos, free)**
https://javascript30.com

305 **Vite: scaffolding a project, dev server with HMR, and production build**
Vite gives you instant hot-reload and an optimized build with near-zero config, which is the fastest path from idea to a deployable polished app.

DOCS **Vite Guide — Getting Started**
https://vite.dev/guide/

306 **JSX: expressions in markup, attributes, conditional rendering, list rendering with keys**
JSX is how you express dynamic, data-driven layouts in React, and getting keys/conditionals right avoids the subtle render bugs that break visual polish.

DOCS **Writing Markup with JSX (React docs)**
https://react.dev/learn/writing-markup-with-jsx

307 **Props: passing data down, prop types/shape, default props, children prop**
Props are the contract that lets one well-designed component (e.g. a Button) be reused everywhere with controlled variation, the backbone of a consistent design system.

DOCS **Passing Props to a Component (React docs)**
https://react.dev/learn/passing-props-to-a-component

308 **Composition over configuration: children, slots, and compound components**
Composing components (Card with Card.Header/Card.Body) instead of piling on props yields flexible, elegant APIs that scale to complex layouts without bloat.

DOCS **Passing JSX as children (React docs)**
https://react.dev/learn/passing-props-to-a-component#passing-jsx-as-children

309 **useState: local component state and the immutable update pattern**

State is what makes a site interactive (menus, tabs, toggles), and updating it immutably is the rule that keeps re-renders correct and predictable.

DOCS **State: A Component's Memory (React docs)**

https://react.dev/learn/state-a-components-memory

310 **Rendering & re-rendering: how React decides what to update, and why pure render matters**

Understanding the render cycle lets you reason about performance and avoid the flicker/lag that instantly makes a site feel cheap.

DOCS **Render and Commit (React docs)**

https://react.dev/learn/render-and-commit

311 **Handling events and forms: controlled inputs, onChange/onSubmit, lifting state up**

Real lead-capture forms (the conversion engine of a marketing site) require controlled inputs and shared state done correctly.

DOCS **Sharing State Between Components (React docs)**

https://react.dev/learn/sharing-state-between-components

312 **useEffect: synchronizing with external systems, dependency arrays, and cleanup**

Effects power scroll listeners, analytics, and data fetches, and knowing when NOT to reach for them prevents the infinite-loop and stale-data bugs that plague beginners.

DOCS **Synchronizing with Effects (React docs)**

https://react.dev/learn/synchronizing-with-effects

313 **useRef: accessing DOM nodes and persisting mutable values without re-render**

Refs are essential for focus management, measuring elements, and driving animation libraries that need a real DOM handle.

DOCS **Manipulating the DOM with Refs (React docs)**

https://react.dev/learn/manipulating-the-dom-with-refs

314 **Custom hooks: extracting reusable stateful logic (useToggle, useMediaQuery, useScroll)**

Custom hooks let you share interactive behavior across components without duplication, keeping a polished codebase DRY and testable.

DOCS **Reusing Logic with Custom Hooks (React docs)**

https://react.dev/learn/reusing-logic-with-custom-hooks

315 **Lists, keys, and conditional UI patterns at scale (empty states, loading, error)**

Designing every state (not just the happy path) is what separates a finished, professional feel from a demo that breaks on real data.

DOCS **Rendering Lists (React docs)**

https://react.dev/learn/rendering-lists

316 **Component-scoped styling approaches: CSS Modules vs Tailwind vs CSS-in-JS tradeoffs**

Choosing a styling strategy that scopes styles per component prevents the global-CSS collisions that erode visual consistency on bigger sites.

DOCS **Styling and CSS (React docs)**

https://react.dev/learn/scaling-up-with-reducer-and-context

317 **Context API: avoiding prop-drilling for theme, locale, and shared UI state**

Context cleanly distributes design tokens (theme/dark-mode) and global state so deeply nested components stay consistent without messy prop chains.

DOCS **Passing Data Deeply with Context (React docs)**

https://react.dev/learn/passing-data-deeply-with-context

318 **useReducer + Context: structuring non-trivial state (carts, multi-step forms, wizards)**

As interactions grow, a reducer keeps state transitions explicit and bug-free where scattered useState calls would become unmanageable.

DOCS **Scaling Up with Reducer and Context (React docs)**

https://react.dev/learn/scaling-up-with-reducer-and-context

319 **Headless/unstyled component libraries: Radix UI / Headless UI for accessible primitives**

Headless libraries give you accessible, keyboard-correct modals, menus, and tabs that you style yourself, delivering bespoke looks without reinventing a11y.

DOCS **Radix Primitives — Introduction**

https://www.radix-ui.com/primitives/docs/overview/introduction

320 **shadcn/ui: copy-in, ownable components built on Radix + Tailwind**

shadcn gives you fully customizable, design-system-grade components you own in your repo, the fastest route to a distinctive (not template-y) UI.

DOCS **shadcn/ui — Documentation**

https://ui.shadcn.com/docs

321 **TypeScript basics for components: typing props, useState generics, event types**

Types catch broken props and mismatched data at edit time, eliminating an entire class of runtime bugs that would otherwise ship to clients.

DOCS **TypeScript for JavaScript Programmers (Handbook)**

https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html

322 **TypeScript + React patterns: typed children, generics, discriminated unions for variants**

Properly typed component variants (size='sm'|'lg') make a design system self-documenting and impossible to misuse, raising overall craft.

DOCS **React TypeScript Cheatsheet**

https://react-typescript-cheatsheet.netlify.app/

323 **Performance: code-splitting with lazy/Suspense, memoization (memo/useMemo/useCallback)**

Splitting bundles and avoiding needless re-renders keeps Core Web Vitals green, which directly affects how fast and premium a site feels.

DOCS **web.dev — Code splitting with React.lazy and Suspense**

https://web.dev/articles/code-splitting-suspense

324 **When NOT to use a framework: static HTML/CSS for brochure sites, islands, and progressive enhancement**

Knowing that many local-business sites are faster and simpler as static HTML keeps you from over-engineering and shipping bloated JS for a five-section page.

SITE **The market for lemons (Read the Tea Leaves / Alex Russell on JS overuse) — web.dev: When do you not need a framework?**

https://web.dev/learn/

325 **Choosing a framework: React vs Vue vs Svelte vs Astro and matching the tool to the project**

Picking Astro for content sites or Svelte for tiny bundles instead of defaulting to React lets each project ship the leanest, best-fit architecture.

DOCS **Astro Documentation — Why Astro?**

https://docs.astro.build/en/concepts/why-astro/

14 Performance & Core Web Vitals

326–350

326 **What Core Web Vitals are: LCP, CLS, INP thresholds (good/needs-work/poor)**

You can't optimize what you can't name — knowing the exact pass thresholds turns 'feels slow' into a measurable target.

DOCS **web.dev — Web Vitals**

https://web.dev/articles/vitals

327 **Reading a Lighthouse report: scores, opportunities, diagnostics, lab vs field data**

Lighthouse is your primary verification gate, so understanding which numbers are real and which are estimates prevents chasing the wrong fixes.

DOCS **Chrome DevTools — Lighthouse**

https://developer.chrome.com/docs/lighthouse/overview

328 **Running PageSpeed Insights on field (CrUX) data vs your local lab run**

Real users on real Houston phones see different numbers than your fast laptop, and only field data tells you if a site truly passes.

TOOL **PageSpeed Insights**

https://pagespeed.web.dev

329 **Reading the DevTools Network waterfall: request order, blocking, timing bars**

The waterfall exposes exactly which resource stalls first paint, the single most actionable performance view you have.

DOCS **Chrome DevTools — Network features reference**

https://developer.chrome.com/docs/devtools/network

330 **Modern image formats: serving AVIF/WebP with the <picture> fallback pattern**

Images are usually the heaviest bytes on a marketing site, and AVIF/WebP cut them 30-70% with no visible quality loss.

DOCS **MDN — Responsive images**

https://developer.mozilla.org/en-US/docs/Web/HTML/Guides/Responsive_images

331 **Responsive images with srcset and sizes to ship the right resolution per viewport**

Sending a 2000px hero to a phone wastes bandwidth and tanks LCP; srcset hands each device exactly what it needs.

COURSE **web.dev — Serve responsive images**

https://web.dev/learn/design/responsive-images

332 **Always setting width/height (or aspect-ratio) on media to reserve space**

Unsized images are the number-one cause of layout shift, and a reserved box keeps CLS at zero.

DOCS **web.dev — Optimize Cumulative Layout Shift**

https://web.dev/articles/optimize-cls

333 **Native lazy-loading (loading="lazy") for below-the-fold images and iframes**

Deferring offscreen media frees the network for the content the user actually sees first, speeding up LCP for free.

DOCS **Browser-level image lazy loading for the web**

https://web.dev/articles/browser-level-image-lazy-loading

334 **Prioritizing the LCP image: fetchpriority="high" and NOT lazy-loading the hero**

The single biggest LCP win is telling the browser your hero image is the most important byte on the page.

DOCS **web.dev — Optimize Largest Contentful Paint**

https://web.dev/articles/optimize-lcp

335 **Web font optimization: font-display: swap, subsetting, WOFF2, self-hosting**

Fonts block text rendering and cause flashes; controlling load behavior keeps your distinctive type from hurting LCP and CLS.

DOCS **web.dev — Best practices for fonts**

https://web.dev/articles/font-best-practices

336 **Preloading critical fonts and resources with <link rel="preload">**

Preloading the font used in your headline removes a hidden round-trip that otherwise delays your most visible text.

DOCS **web.dev — Preload critical assets**

https://web.dev/articles/preload-critical-assets

337 **Eliminating render-blocking CSS and JS in the <head>**

Every blocking resource in the head pushes back first paint, so identifying and deferring them is core to a fast feel.

DOCS **Render-blocking CSS**

https://web.dev/articles/critical-rendering-path/render-blocking-css

338 **Critical CSS: inlining above-the-fold styles, deferring the rest**

Inlining the styles for what's visible lets the page paint before the full stylesheet downloads — a major perceived-speed win.

DOCS **web.dev — Extract critical CSS**

https://web.dev/articles/extract-critical-css

339 **defer vs async vs module scripts and where to place <script> tags**

Mis-loaded scripts block parsing and paint; knowing the right attribute keeps JS from sabotaging an otherwise fast page.

DOCS **MDN — <script>: The Script element**

https://developer.mozilla.org/en-US/docs/Web/HTML/Reference/Elements/script

340 **Minification and compression: minified CSS/JS plus Brotli/gzip on the server**

Smaller text payloads transfer faster on slow connections, and Brotli often beats gzip by another 15-20%.

DOCS **web.dev — Reduce network payloads using text compression**

https://web.dev/articles/reduce-network-payloads-using-text-compression

341 **HTTP caching: Cache-Control, immutable, fingerprinted filenames**

Proper cache headers make repeat visits near-instant and slash your origin load — a polish most template sites skip.

DOCS **web.dev — HTTP caching**

https://web.dev/articles/http-cache

342 **CDN delivery and edge caching (Cloudflare Pages) for global low latency**

Serving assets from an edge near the user shrinks every round-trip, which directly improves LCP for distant visitors.

DOCS **web.dev — Content delivery networks (CDNs)**

https://web.dev/articles/content-delivery-networks

343 **Resource hints: preconnect and dns-prefetch for third-party origins**

Warming up connections to fonts/analytics domains early removes DNS+TLS delays that otherwise stall critical requests.

DOCS **web.dev — Establish network connections early with rel=preconnect**

https://web.dev/articles/preconnect-and-dns-prefetch

344 **Optimizing INP: minimizing main-thread work and long tasks on interaction**

INP replaced FID as a Core Web Vital, and snappy click/tap response is what makes a site feel genuinely premium.

DOCS **web.dev — Optimize Interaction to Next Paint**

https://web.dev/articles/optimize-inp

345 **Profiling with the DevTools Performance panel: long tasks, main-thread flame chart**

The Performance panel is where you see exactly which script blocks interaction, turning vague jank into a named function.

DOCS **Chrome DevTools — Performance panel**

https://developer.chrome.com/docs/devtools/performance

346 **Auditing and deferring third-party scripts (analytics, chat, embeds, maps)**

A single chat widget or map embed can dwreck INP and LCP, so loading them lazily protects your hard-won speed.

DOCS **web.dev — Loading third-party JavaScript**

https://web.dev/articles/efficiently-load-third-party-javascript

347 **Animating only transform and opacity to stay on the GPU compositor**

Animating layout properties causes reflow jank; sticking to transform/opacity keeps your tasteful motion buttery at 60fps.

DOCS **web.dev — Animations and performance**

https://web.dev/articles/animations-guide

348 **Perceived performance: skeletons, optimistic UI, instant feedback, blur-up placeholders**

A site that feels fast beats one that merely measures fast — perception is the real masterpiece-level differentiator.

DOCS **Perceived performance**

https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Performance/Perceived_performance

349 **Measuring real-user CWV in the field with the web-vitals JS library**

Lab scores lie about real users; field measurement is the only way to know your actual visitors are passing.

TOOL **GitHub — GoogleChrome/web-vitals**

https://github.com/GoogleChrome/web-vitals

350 **Setting a performance budget and full Learn Performance course mastery**

A budget stops feature creep from silently bloating your sites, turning speed into a maintained standard rather than a one-time win.

COURSE **web.dev — Learn Performance**

https://web.dev/learn/performance

15 Tooling, Git & Deployment

351–375

351 **Command line basics: cd/ls/mkdir/mv/rm, pipes, and absolute vs relative paths**

Every modern build, deploy, and Git workflow runs through a terminal, so fluency here removes friction from all other tooling.

COURSE **The Odin Project: Command Line Basics**

https://www.theodinproject.com/lessons/foundations-command-line-basics

352 **Shell power-ups: globbing, environment variables, chaining with && and ||, and reading exit codes**

Composing commands reliably is what turns one-off typing into repeatable, scriptable deploy steps you can trust.

DOCS **MDN: Command line crash course**

https://developer.mozilla.org/en-US/docs/Learn_web_development/Getting_started/Environment_setup/Command_line

353 **Git mental model: working tree, staging area (index), commits, and HEAD**

Understanding what each Git command actually moves prevents the panic-driven mistakes that lose work on a client site.

DOCS **Pro Git (free book) — Git Basics**

https://git-scm.com/book/en/v2

354 **Core Git commands: init, status, add, commit, log, diff, show**

A clean, frequent commit habit gives you a reversible history so no design experiment can ever permanently break a site.

COURSE **The Odin Project: Git Basics**

https://www.theodinproject.com/lessons/foundations-git-basics

355 **Writing good commit messages (imperative subject, why-focused body, atomic commits)**

A readable history is documentation; future-you can find when a layout broke and reason about every change.

SITE **How to Write a Git Commit Message (Chris Beams)**

https://cbea.ms/git-commit/

356 **.gitignore patterns for web projects (node_modules, dist, .env, OS junk)**

Keeping build output and secrets out of history keeps repos clean, fast to clone, and safe to share.

TOOL **GitHub gitignore templates**

https://github.com/github/gitignore

357 **Branching and merging: create, switch, fast-forward vs merge commits**

Branches let you build a redesign in isolation while the live client site stays on a stable main branch.

INTERACTIVE **Learn Git Branching (interactive)**

https://learngitbranching.js.org/

358 **Remotes and syncing: clone, remote add, push, pull, fetch, and tracking branches**

Pushing to GitHub is both your offsite backup and the trigger that most static hosts use to deploy.

DOCS **GitHub Docs: About remote repositories**

https://docs.github.com/en/get-started/git-basics/about-remote-repositories

359 **GitHub repositories: README, licenses, and connecting a local repo to GitHub**

A well-structured GitHub repo is the professional hub clients and collaborators expect a real agency to have.

DOCS **GitHub Quickstart**

https://docs.github.com/en/get-started/quickstart

360 **Pull requests and code review flow (even solo: PR, review diff, merge)**

PRs create a deliberate checkpoint to catch mistakes before they hit a live client site.

INTERACTIVE **GitHub Skills: Introduction to GitHub**

https://github.com/skills/introduction-to-github

361 **Undoing things safely: restore, revert, reset (soft/mixed/hard), and reflog recovery**

Knowing how to cleanly back out a bad change—and recover 'lost' commits—turns Git disasters into non-events.

DOCS **Atlassian Git Tutorial: Undoing Changes**

https://www.atlassian.com/git/tutorials/undoing-changes

362 **Resolving merge conflicts confidently**

Conflicts are inevitable once you branch; resolving them calmly keeps momentum instead of derailing a build.

DOCS **Resolving a merge conflict using the command line**

https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/addressing-merge-conflicts/resolving-a-merge-conflict-using-the-command-line

363 **Tags and semantic versioning for releases**

Tagging releases gives you named, reproducible snapshots to roll a client site back to a known-good design.

DOCS **Semantic Versioning 2.0.0**

https://semver.org/

364 **Node.js and npm fundamentals: package.json, install, scripts, semver ranges**

npm scripts are the command center for building, previewing, and shipping nearly every modern front-end project.

DOCS **npm Docs: About npm**

https://docs.npmjs.com/about-npm

365 **Lockfiles and reproducible installs (package-lock.json, npm ci)**

Locked dependencies guarantee the site you tested is byte-for-byte the site that deploys—no surprise breakage.

DOCS **npm Docs: package-lock.json**

https://docs.npmjs.com/cli/v10/configuring-npm/package-lock-json

366 **Choosing and using a package manager (npm vs pnpm) and global vs project deps**

Picking the right manager and scoping dependencies keeps installs fast and projects portable across machines.

DOCS **pnpm Documentation**

https://pnpm.io/motivation

367 **Modern bundlers/dev servers: Vite for instant HMR and optimized production builds**

Vite gives you fast local previews and minified, tree-shaken output that makes masterpiece sites load instantly.

DOCS **Vite Guide**

https://vite.dev/guide/

368 **npx and CLI tooling (running tools without global installs, scaffolding starters)**
npx lets you spin up scaffolds and one-off tools instantly, keeping your global environment clean and current.

DOCS **npm Docs: npx**
https://docs.npmjs.com/cli/v10/commands/npx

369 **Static hosting on Cloudflare Pages: connect repo, build command, output dir, instant rollbacks**
Cloudflare Pages turns a git push into a globally CDN-served site with preview URLs and one-click rollback.

DOCS **Cloudflare Pages: Get started**
https://developers.cloudflare.com/pages/get-started/

370 **Netlify deploys: build settings, deploy previews, redirects and headers files**
Knowing a second host (and its _redirects/_headers) means you can match any client's stack and constraints.

DOCS **Get Started with Netlify**
https://docs.netlify.com/get-started/

371 **Custom domains and DNS: A/AAAA, CNAME, nameservers, apex vs subdomain, propagation**
A polished site on a wrong-configured domain looks broken to the client, so DNS fluency is non-negotiable for delivery.

SITE **Cloudflare Learning: What is DNS?**
https://www.cloudflare.com/learning/dns/what-is-dns/

372 **HTTPS/TLS and HTTP caching headers for production sites**
Correct certs and cache headers make sites trustworthy (padlock) and fast on repeat visits—core to perceived quality.

DOCS **MDN: HTTP caching**
https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/Caching

373 **Environment variables and secrets hygiene (.env, never commit keys, host-side secrets, .env.example)**
Leaking an API key from a marketing site can cost a client real money and your reputation—handle secrets correctly.

DOCS **web.dev / 12-Factor App: Config**
https://12factor.net/config

374 **CI/CD with GitHub Actions: lint, build, and deploy on push; status checks on PRs**
Automated checks catch broken builds before they ship, so every push to a client site stays reliably green.

DOCS **GitHub Docs: Understanding GitHub Actions**
https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions

375 **Deploy-time quality gates: run Lighthouse CI / link & build checks before promoting to production**
Wiring performance and accessibility budgets into deploy ensures every release stays masterpiece-grade, not just the first.

TOOL **Lighthouse CI Getting Started**
https://github.com/GoogleChrome/lighthouse-ci/blob/main/docs/getting-started.md

16 Design Process & Craft

376–400

376 **Building a swipe file: systematically collecting references into a moodboard before designing**

Masterpiece work starts from a deep visual library, not a blank canvas, so you remix proven patterns instead of defaulting to generic ones.

SITE **Land-book — landing page design inspiration gallery**

https://land-book.com

377 **Reading award-winning sites critically: deconstructing why Awwwards/Godly winners feel premium**

Training your eye to name the specific moves behind 'expensive-looking' design is what lets you reproduce that quality on purpose.

SITE **Awwwards — Website Awards & Inspiration**

https://www.awwwards.com

378 **Developing taste: studying the gap between amateur and professional output**

Taste is the ability to perceive the quality gap, and acknowledging that gap is the engine that pushes work from generic to distinctive.

VIDEO **Ira Glass on Taste (the 'gap' talk) — via The Creative Gap**

https://www.youtube.com/results?search_query=ira+glass+on+taste+the+gap

379 **Figma fundamentals: frames, constraints, and the layers/properties model**

Fluency in Figma lets you explore ten directions fast and lock a concept before writing a single line of code.

DOCS **Figma — official Figma for Beginners / Help Center**

https://help.figma.com

380 **Figma Auto Layout: padding, gap, and resizing for responsive components**

Auto Layout makes your mockups behave like real CSS flexbox, so design intent survives the jump to code intact.

VIDEO **Figma YouTube — Auto Layout tutorials**

https://www.youtube.com/@Figma

381 **Figma components, variants, and properties for reusable UI**

Components enforce consistency across a whole site and stop the 'every button is slightly different' look that screams amateur.

DOCS **Figma Help — Components, styles, and libraries**

https://help.figma.com

382 **Figma variables and modes (design tokens for color, spacing, radius)**

Tokenizing decisions in Figma mirrors how you'll structure CSS custom properties, keeping design and code in lockstep.

DOCS **Figma Help — Guide to variables in Figma**

https://help.figma.com

383 **Design tokens theory: naming a token taxonomy (primitive → semantic → component)**

A layered token system is what makes a design system scalable and themeable instead of a pile of one-off hex codes.

DOCS **web.dev — Design tokens / building a design system**

https://web.dev/learn

384 **Establishing a spacing scale and applying it ruthlessly (4/8px system)**

Consistent rhythm is the single biggest tell of professional vs. template-y layout, and a fixed scale removes arbitrary guesses.

SITE **Refactoring UI — Spacing & sizing (Adam Wathan / Steve Schoger articles)**

https://www.refactoringui.com

385 **Visual hierarchy: directing the eye with size, weight, contrast, and position**

Hierarchy is how a page tells the user what matters first, turning a flat wall of content into a guided experience.

SITE **NN/g — Visual Hierarchy articles**

https://www.nngroup.com/articles

386 **Layout & alignment discipline: grids, optical alignment, and edge alignment**

Precise alignment is invisible when right and glaring when wrong, and it separates crafted layouts from sloppy ones.

SITE **Refactoring UI — Layout and spacing fundamentals**

https://www.refactoringui.com

387 **Consistency systems: defining one source of truth for buttons, cards, inputs**

A coherent component vocabulary makes a five-page site feel like one designed product rather than five separate templates.

DOCS **Material Design 3 — component & foundations guidelines**

https://m3.material.io

388 **Borrowing from platform design systems (Apple HIG / Material) for proven patterns**

Mature design systems encode decades of usability research, so leaning on them raises your baseline quality instantly.

DOCS **Apple — Human Interface Guidelines**

https://developer.apple.com/design/human-interface-guidelines

389 **The render-critique-fix loop: screenshot your own work and critique it like a stranger**

You can't fix what you can't see, and viewing a flat render breaks the 'I built it so it looks fine' bias that hides flaws.

SITE **Anthropic — frontend-design skill (render → critique → fix protocol)**

https://www.anthropic.com/engineering

390 **Self-critique frameworks: running heuristic checklists against a design**

Structured critique turns vague 'something feels off' into specific, fixable issues you can resolve before shipping.

SITE **NN/g — 10 Usability Heuristics for User Interface Design**

https://www.nngroup.com/articles/ten-usability-heuristics/

391 **Naming things well: BEM/utility class names and semantic component names**

Clear, consistent naming keeps a growing codebase navigable and is itself an act of design thinking about structure.

SITE **CSS-Tricks — BEM 101 and naming conventions**

https://css-tricks.com

392 **Design-to-code translation: reading a mockup as spacing, type, and color tokens**

Faithfully converting design decisions into code is where most 'good in Figma, mediocre in browser' projects fall apart.

DOCS **Figma — Dev Mode: inspect, measure, and code**

https://help.figma.com

393 **Avoiding the generic AI/template look: spotting the default Inter+purple+centered-hero tells**

Naming the clichés that mark AI-generated and template sites is the first step to deliberately designing around them.

SITE **frontend-design skill — anti-generic design heuristics**

https://www.anthropic.com/engineering

394 **Choosing a distinctive aesthetic direction (style tile) per client before building**

Committing to a clear stylistic point of view up front is what gives a site personality instead of generic blandness.

SITE **Godly — astronomically good web design inspiration**

https://godly.website

395 **Detail polish: shadows, borders, radii, and depth done with restraint**

The difference between okay and masterpiece lives in micro-details like layered shadows and hairline borders done tastefully.

SITE **Josh W. Comeau — Designing Beautiful Shadows in CSS**

https://www.joshwcomeau.com/css/designing-shadows/

396 **Practical design intuition: the small rules that make UI feel right (Emil Kowalski)**

Internalizing battle-tested polish rules lets you make confident micro-decisions instead of nudging pixels randomly.

SITE **Emil Kowalski — Articles on UI design & craft**

https://emilkowal.ski/

397 **Refining with Refactoring UI tactics: 'start with too much white space, then remove'**

A pile of concrete, non-obvious tactics shortcuts years of trial and error toward consistently professional results.

SITE **Refactoring UI — free article archive**

https://www.refactoringui.com

398 **Documenting a mini design system: a one-page spec of tokens, components, and rules**

Writing down your system forces coherence and lets every page and future edit stay on-brand without re-deciding.

SITE **Smashing Magazine — design systems articles**

https://www.smashingmagazine.com

399 **Iteration velocity: generating 3+ visual directions fast, then killing the weakest**

Masterpieces emerge from selection among options, not from polishing the first idea, so volume-then-cull beats single-pass design.

VIDEO **Figma YouTube — exploration & rapid prototyping workflows**

https://www.youtube.com/@Figma

400 **Building a personal process & critique habit: pinning inspiration and reviewing past work**

A deliberate, repeatable craft process is what compounds taste over time and turns one-off wins into a consistent standard.

SITE **Smashing Magazine — design workflow & process articles**

https://www.smashingmagazine.com