WEB ENGINEERING FUNDAMENTALS

CSS Display & Positioning

A Complete Beginner's Guide to Controlling Web Page Layouts

Learn how CSS controls the layout of web pages using display properties, positioning, document flow, and stacking order. Understand how browsers place elements before moving on to Flexbox and CSS Grid.

haas.dev • dev-roast-app.vercel.app

Table of Contents

1. Introduction

When you write HTML, the browser automatically places elements on the page.

You don't tell the browser where every paragraph, heading, or button should go.

Instead, the browser follows a set of layout rules known as the normal document flow.

CSS allows you to change these default behaviors.

You can decide:

- whether an element starts on a new line
- whether it sits beside another element
- whether it stays fixed while scrolling
- whether it overlaps other elements
- whether it should appear at all

Understanding these concepts is essential before learning Flexbox or CSS Grid.

2. What Is Document Flow?

Document flow is the browser's default method of arranging HTML elements.

For example:

```
Heading
↓
Paragraph
↓
Image
↓
Button
↓
Footer
```

Each block element naturally appears below the previous one.

This predictable order is called the normal flow.

CSS properties such as display and position allow developers to modify this behavior.

3. Why Display & Positioning Matter

Without these properties, you would struggle to build layouts like:

- navigation bars
- sidebars
- sticky headers
- floating buttons
- image overlays
- modal dialogs
- notification badges

Modern web interfaces rely heavily on display and positioning.

4. Understanding the display Property

The display property tells the browser how an element should behave within the layout.

Common values include:

- block
- inline
- inline-block
- none
- flex
- grid

In this PDF, we'll focus on the first four. Flexbox and Grid will be covered separately.

5. Block vs Inline vs Inline-Block

Block Elements

Block elements:

- start on a new line
- occupy the full available width by default
- stack vertically

Examples include:

-
- <section>
- <article>
- <p>
- <h1> to <h6>

Example:

```
display: block;
```

Block elements are commonly used for page structure.

Inline Elements

Inline elements:

- remain on the same line
- only take up the space they need
- do not start on a new line

Examples:

-
-
-
- <a>

Example:

```
display: inline;
```

Inline elements are useful for styling small pieces of text.

Inline-Block Elements

Inline-block combines features of both block and inline elements.

An inline-block element:

- stays on the same line
- allows custom width and height
- supports margins and padding more predictably

Example:

```
display: inline-block;
```

This was commonly used for layouts before Flexbox became popular.

6. The display: none Property

Sometimes you want an element to disappear completely.

Example:

```
display: none;
```

The browser removes the element from the layout.

Other elements behave as though it never existed.

Common uses:

- mobile menus
- dropdowns
- loading indicators
- conditional content

JavaScript often toggles this property to show or hide elements.

7. Understanding CSS Positioning

The position property changes how an element is placed on the page.

Available values include:

- static
- relative
- absolute
- fixed
- sticky

Each serves a different purpose.

Static Positioning

static is the default positioning mode.

Elements follow the normal document flow.

Example:

```
position: static;
```

Most elements are static unless changed.

8. Relative Positioning

A relatively positioned element stays in the normal flow but can be moved slightly.

Example:

```
position: relative;
top: 10px;
left: 20px;
```

The original space is still reserved.

Relative positioning is also commonly used as a reference point for absolutely positioned child elements.

9. Absolute Positioning

Absolutely positioned elements are removed from the normal flow.

They are positioned relative to the nearest positioned ancestor.

Example:

```
position: absolute;
top: 0;
right: 0;
```

Common uses:

- notification badges
- tooltips
- overlays
- icons inside buttons

10. Fixed Positioning

A fixed element stays attached to the browser window.

It does not move when the user scrolls.

Example:

```
position: fixed;
bottom: 20px;
right: 20px;
```

Common examples:

- "Back to Top" buttons
- chat buttons
- floating action buttons

11. Sticky Positioning

Sticky positioning combines relative and fixed behavior.

An element scrolls normally until it reaches a specified point, then sticks to that position.

Example:

```
position: sticky;
top: 0;
```

Common uses:

- sticky navigation bars
- table headers
- sidebar menus

12. Understanding z-index

Sometimes elements overlap.

The z-index property determines which element appears on top.

Higher values appear above lower values.

Example:

```
z-index: 100;
```

z-index only works on positioned elements.

13. Real-World Examples

Navigation Bar

Often uses:

- position: sticky
- top: 0

This keeps navigation visible while scrolling.

Notification Badge

Usually uses:

- position: absolute

inside a

- position: relative

parent.

Chat Support Button

Typically uses:

- position: fixed

to remain visible in the bottom corner.

Modal Window

Usually combines:

- position: fixed
- high z-index

to appear above all other content.

> Learn More
> Read CSS Box Model to understand how spacing and sizing work before positioning elements.

14. Common Beginner Mistakes

- Using absolute positioning for every layout.
- Forgetting that absolutely positioned elements leave the normal flow.
- Misusing z-index without understanding stacking contexts.
- Using inline elements where block elements are more appropriate.
- Forgetting that display: none removes the element entirely from the layout.

15. Practical Action Plan

Build a simple webpage that includes:

- a sticky navigation bar
- a fixed "Back to Top" button
- a notification badge on a profile image
- hidden content that appears when a button is clicked

Experiment with different display and position values to observe how the layout changes.

16. Mini Architecture Challenge

Challenge

Design the layout for a blog page.

Decide:

- Which elements should be block?
- Which elements should be inline?
- Which components should use sticky positioning?
- Which UI elements should remain fixed?
- Where is absolute positioning appropriate?

Sketch the layout before writing CSS.

17. Key Takeaways

- Document flow controls the default placement of elements.
- The display property changes how elements behave in the layout.
- Positioning allows precise control over element placement.
- Relative positioning maintains normal flow.
- Absolute positioning removes elements from the flow.
- Fixed positioning attaches elements to the viewport.
- Sticky positioning combines scrolling with fixed behavior.
- z-index controls stacking order.

18. Summary Page

Display & Positioning Cheat Sheet

display: block → New line, full width

display: inline → Same line, content width

display: inline-block → Same line with custom sizing

display: none → Hidden and removed from layout

position: static → Default behavior

position: relative → Offset from original position

position: absolute → Positioned relative to ancestor

position: fixed → Attached to viewport

position: sticky → Scrolls, then sticks

z-index → Controls overlap order

19. Display & Positioning Cheat Sheet

A quick-reference checklist you can keep beside your editor while you build.

- [ ] Know whether an element is block, inline, or inline-block by default
- [ ] Use display: none only for content that should fully disappear
- [ ] Default to position: static unless you need to override it
- [ ] Use relative positioning as an anchor for absolute children
- [ ] Reserve fixed positioning for viewport-attached UI (chat buttons, back-to-top)
- [ ] Use sticky for nav bars and table headers that should scroll then lock
- [ ] Apply z-index only to positioned elements
- [ ] Test overlapping elements at different screen sizes

20. Related Resources

CSS Fundamentals

Why read it: Learn the core styling concepts before controlling layouts.

CSS Box Model

Why read it: Understand spacing and sizing before positioning elements.

HTML Fundamentals

Why read it: Learn the HTML structure that CSS layouts are built upon.

Responsive Web Design

Why read it: Discover how display and positioning adapt across different screen sizes.

21. Recommended Next Learning Path

```
Step 1

HTML Fundamentals
↓
Step 2

Semantic HTML
↓
Step 3

CSS Fundamentals
↓
Step 4

CSS Box Model
↓
Step 5

CSS Display & Positioning (Current PDF)
↓
Step 6

Flexbox
↓
Step 7

CSS Grid
↓
Step 8

Responsive Web Design
↓
Step 9

JavaScript Fundamentals
```

---

haas.dev • dev-roast-app.vercel.app