LECTURE 7

CSS Positioning and Stacking

Taking elements out of normal flow — moving, pinning, and layering them — plus floats, an older tool you'll still see.

CSC336 Web Technologies

OVERVIEW

In This Lecture

1. Normal document flow and static positioning
2. The four positioning schemes: relative, absolute, fixed, and sticky — with a worked example of each
3. Offset properties, the containing block, and a common mistake
4. z-index and stacking contexts
5. Floats — left, right, side-by-side columns, clearing, and common gotchas

THE DEFAULT

Normal Flow and Static Positioning

1. Normal flow: elements are placed one after another, in source order
2. Every element's position defaults to static — "follow normal flow, nothing special"
3. Offset properties (top/right/bottom/left) have NO effect on a static element

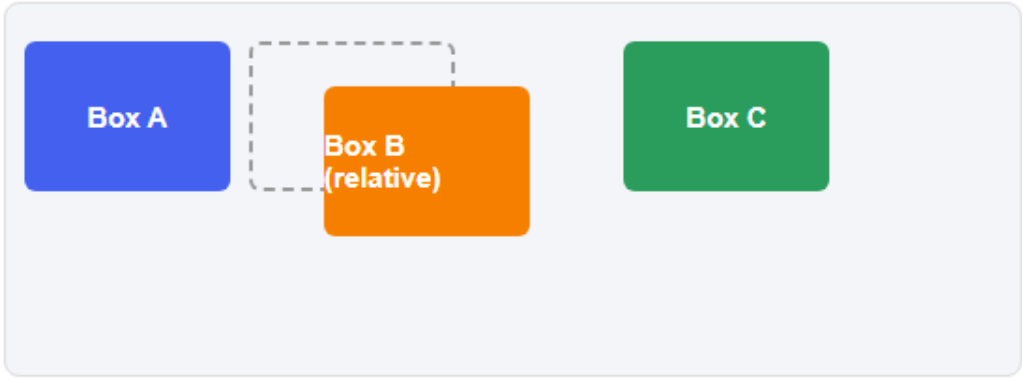
Five Ways to Lay Out an Element

Static flow, relative (shifted but reserved), absolute (pinned to an ancestor), fixed (pinned to the viewport), sticky (relative, then fixed).

```
.static   { position: static; }
.relative { position: relative;
            top: 10px; left: 20px; }
.absolute { position: absolute;
            top: 0; right: 0; }
.fixed    { position: fixed;
            bottom: 20px; right: 20px; }
.sticky   { position: sticky; top: 0; }
```

relative

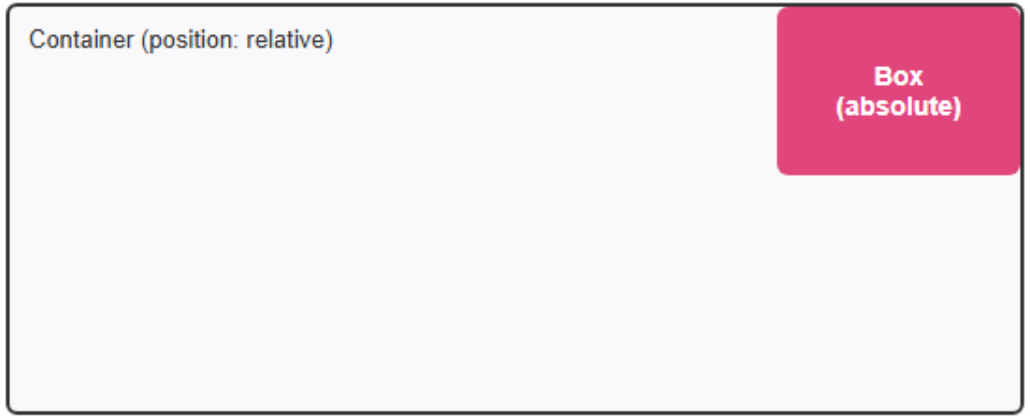
```
.box {
  position: relative;
  top: 10px;
  left: 20px;
}
```



The dashed outline is Box B's original slot — still reserved in the flow. Box B is visually shifted 10px down and 20px right from that slot, but Box C does not move to fill the gap.

absolute

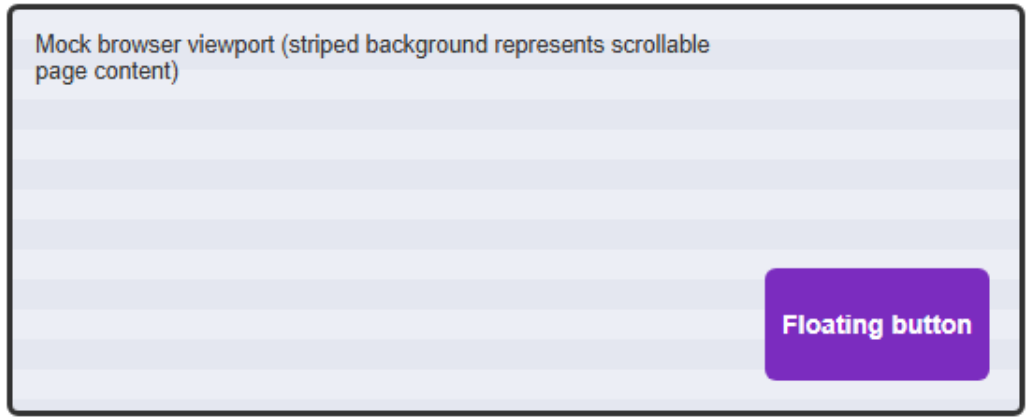
```
.box {
  position: absolute;
  top: 0;
  right: 0;
}
```



The absolutely positioned box is removed from normal flow entirely and pinned to the top-right corner of its containing block — it does not push or reserve space for anything else.

fixed

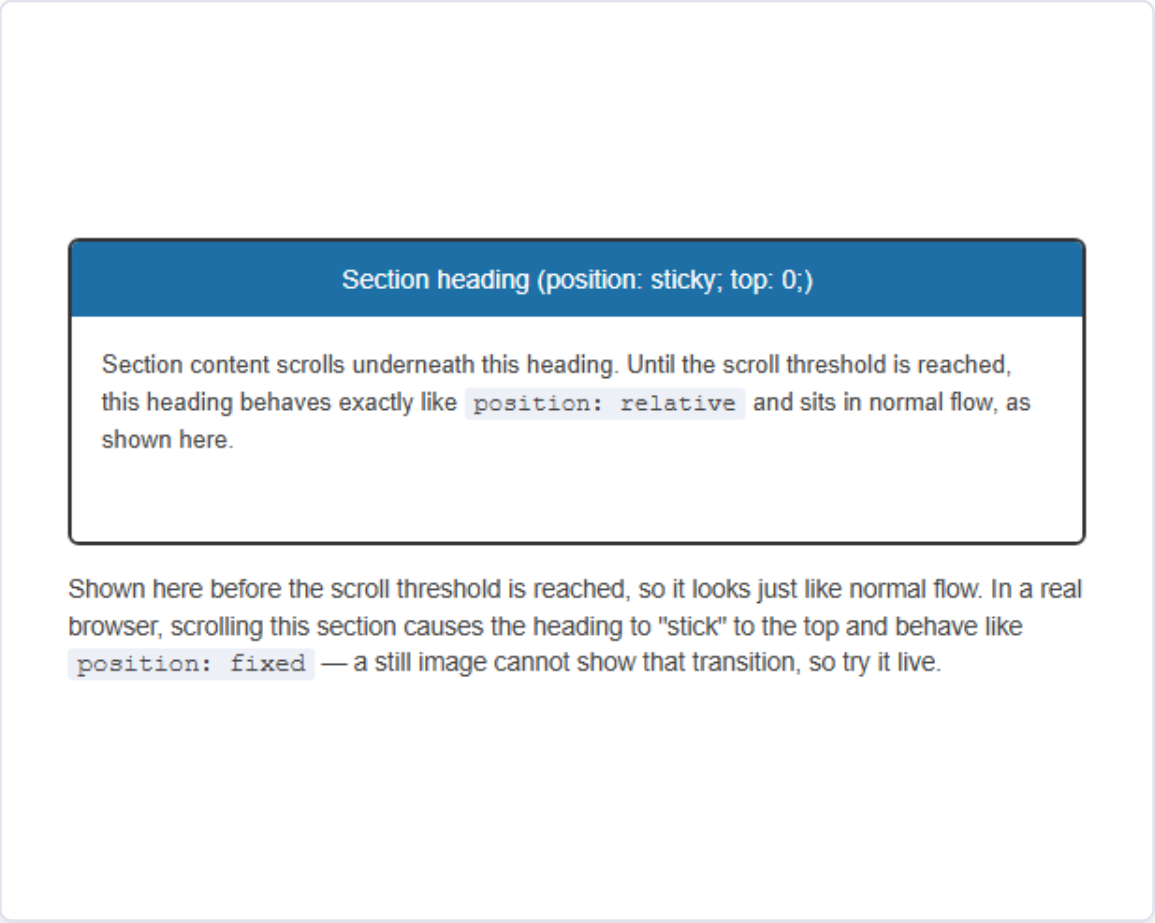
```
.back-to-top {
  position: fixed;
  bottom: 20px;
  right: 20px;
}
```



Shown here in its resting position. In a real browser this element stays glued to that same spot on screen even as the page scrolls — a still image cannot show that motion, so try it live.

sticky

```
.section-heading {
  position: sticky;
  top: 0;
}
```



QUICK RECAP

relative, absolute, fixed, sticky

Value	Behavior
relative	Stays in normal flow (space still reserved), then shifts visually via offsets
absolute	Removed from flow; positioned against its nearest positioned ancestor
fixed	Removed from flow; positioned against the browser viewport, stays put when scrolled
sticky	Behaves like relative until a scroll threshold, then like fixed within its parent

A COMMON PATTERN

Relative Parent, Absolute Child

```
.card { position: relative; }

.card .badge {
  position: absolute;
  top: 8px; right: 8px;
}
```

.card becomes the containing block, so the badge is pinned to the card's corner, not the whole page.

The Badge, Rendered

A COMMON MISTAKE

Absolute Without a Positioned Ancestor

```
<div class="card">          <!-- position NOT set -->
  <span class="badge">New</span>  <!-- position: absolute; top:8px; right:8px; -->
</div>
```

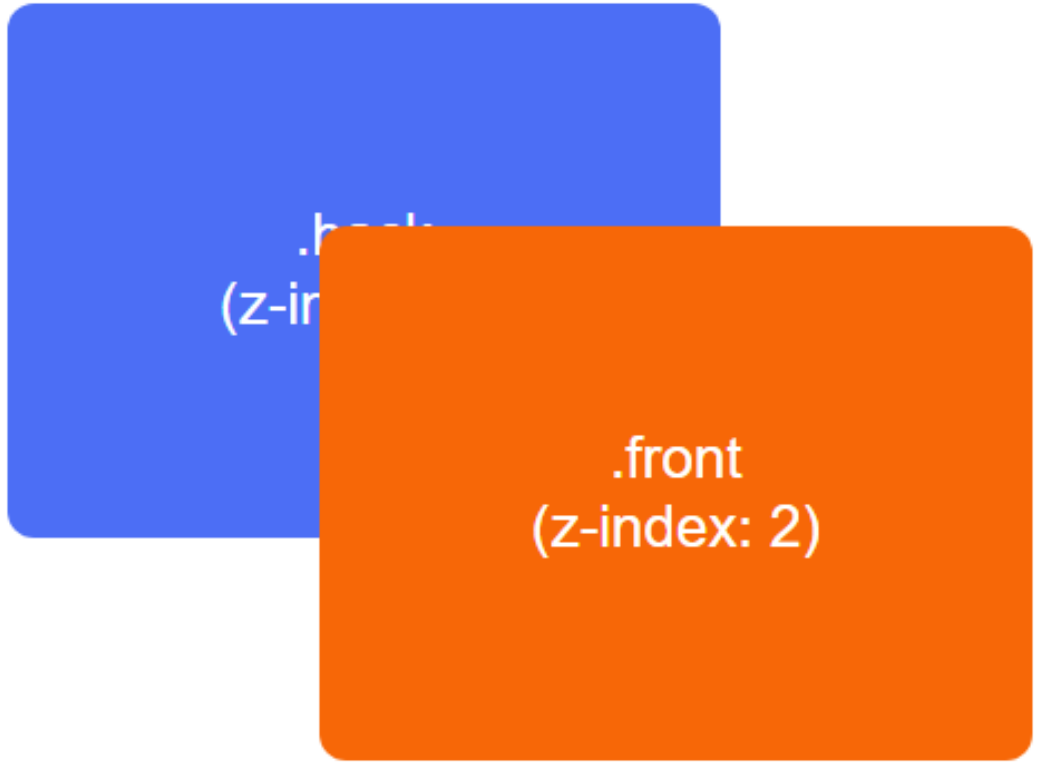
Without position: relative on .card, .badge's containing block becomes the whole page — it jumps to the page's top-right corner, not the card's.

z-index

```
.back  { position: absolute; z-index: 1; }
.front { position: absolute; z-index: 2; }  /* drawn on top */
```

z-index only works on positioned elements (not static) — a higher value draws on top among overlapping elements.

z-index, Rendered



Stacking Contexts

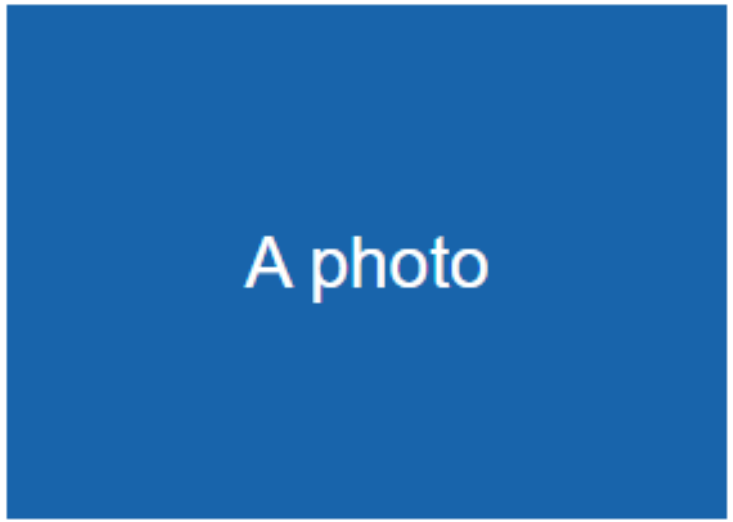
- A stacking context is a self-contained layer for z-index comparisons
- position + z-index, opacity < 1, or transform can create a new stacking context
- z-index values only compare WITHIN the same stacking context — a huge z-index can still lose if it's trapped inside an ancestor's context

float: left

```
.image { float: left; margin-right: 15px; }
```

Originally for text wrapping around an image. A floated element shifts to one edge; other content flows around it.

Text Wrapping a Float, Rendered



This paragraph text will wrap around the floated image, flowing along its right side instead of starting below it. Notice how the text hugs the right and bottom edges of the image once it clears past the image's height, continuing normally afterward as if the image were never there.

float: right — the Mirror Image

```
.image {
  float: right;
  margin-left: 15px;
}
```

Exactly the same idea, flipped: the image sits at the right edge, and text wraps along its left side instead.

A Classic Two-Column Layout

```
.sidebar { float: left;  width: 25%; }
.main    { float: left;  width: 75%; }

<div class="sidebar">...</div>
<div class="main">...</div>
```

Before flexbox and grid existed, this float + matching-width pattern was the standard way to place two columns side by side.

Float Gotchas Worth Knowing

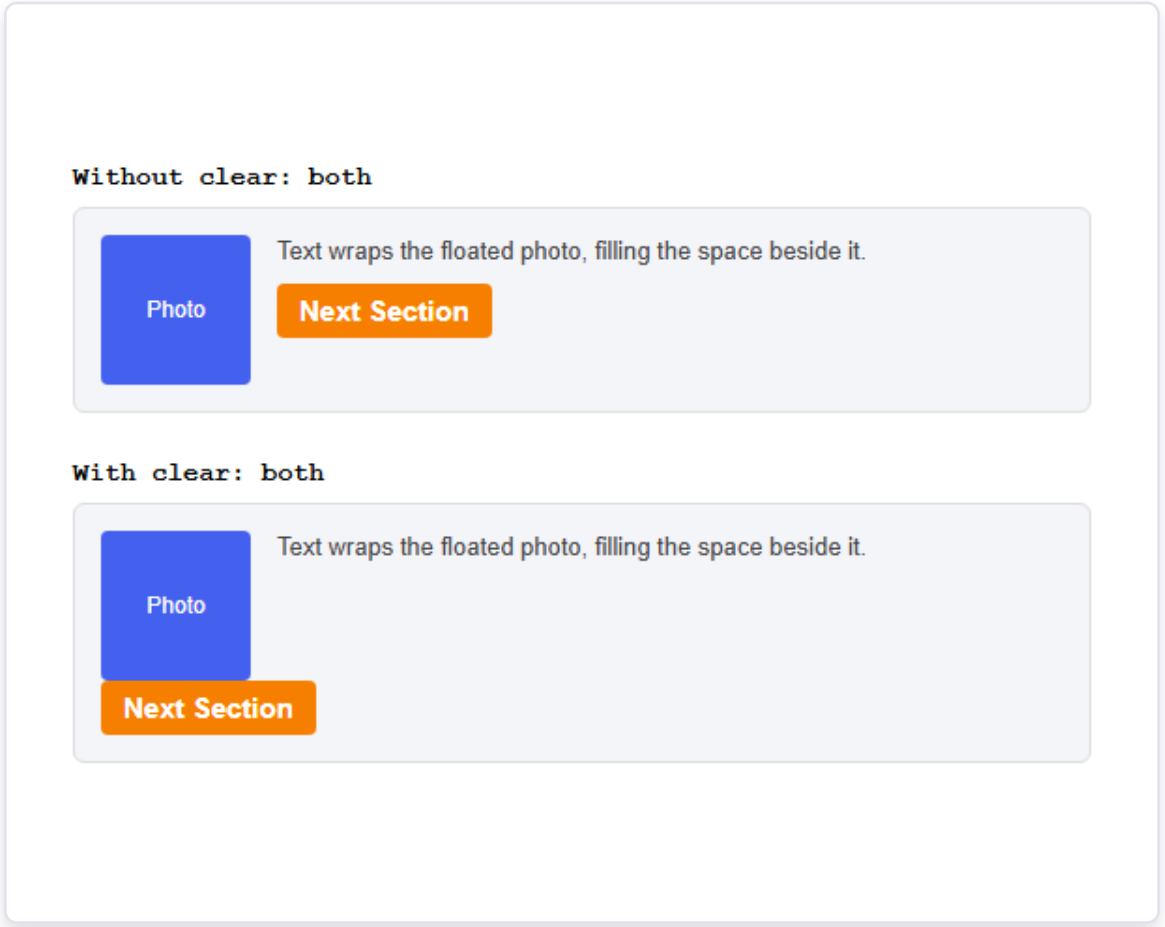
- A floated element needs an explicit width — without one, it can shrink to fit its content unpredictably
- Floating an inline element (like a) makes it behave like a block for sizing purposes
- A float only affects the elements AFTER it in the HTML — it never moves content that comes before it

The clear Property

Value	Behavior
left / right	Moves below any preceding left- / right-floated elements
both	Moves below floats on either side — by far the most common value
none (default)	No clearing — free to sit beside a float

clear, on Its Own

```
<img class="photo" style="float:left;">
<p>Text wraps the photo.</p>
<h2 style="clear: both;">Next Section</h2>
```



The Collapsing Parent Problem

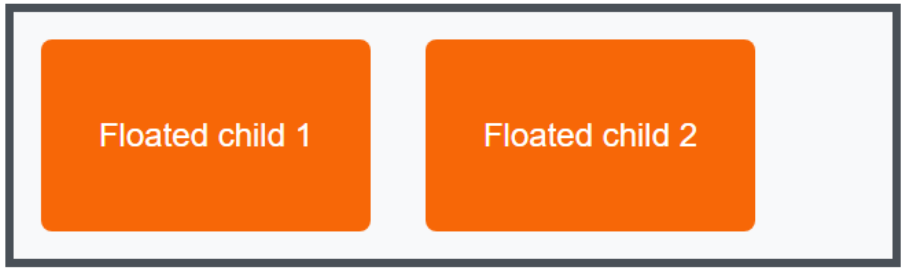
```
.clearfix::after {
  content: "";
  display: block;
  clear: both;
}
```

If every child is floated, a container collapses to zero height. .clearfix adds an invisible element with clear: both to force the parent to contain its floated children again.

Clearfix, Before and After

Without clearfix — the container collapses to (almost) zero height

With `.clearfix` applied — the container expands to contain its floated children

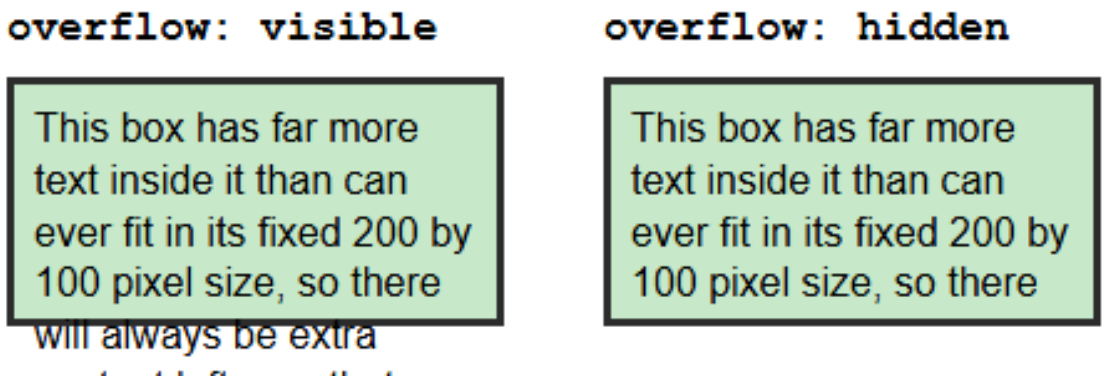


Containing Floats — a Simpler Alternative

Value	Behavior
visible (default)	Content spills outside the box, still visible
hidden	Content that doesn't fit is clipped and hidden
scroll / auto	Adds scrollbars always / only when content actually overflows

Visible vs. Hidden, Rendered

```
.box {
  width: 200px;
  height: 100px;
  overflow: visible; /* default: extra content spills out */
  /* overflow: hidden;  clips content that doesn't fit  */
}
```



A One-Line Clearfix Alternative

TIP

overflow: hidden (or auto) on a float's parent fixes the collapsing-parent problem too, in one line. Trade-off: it also clips any OTHER content that overflows that container, like a dropdown or tooltip — the .clearfix trick is the safer, general-purpose tool.

Common Layout Pitfalls

- position: absolute without a relative ancestor jumps relative to the whole page
- z-index "not working" is almost always a stacking-context issue on an ancestor
- A position: fixed element is fixed to the viewport, not to a scrolling div — you likely want sticky instead
- A floated element with no explicit width, or an un-cleared float, are the two most common float bugs

KEY TAKEAWAYS

Lecture 7 in Six Points

1. Normal flow places elements in source order; position: static means "stay in flow," and offsets do nothing on it.
2. relative shifts an element while reserving its space; absolute/fixed remove it from flow entirely; sticky is a hybrid.
3. The containing block for absolute is the nearest positioned ancestor — forgetting it sends the element to the whole page.
4. z-index only compares within the same stacking context.
5. float: left/right pulls an element aside and wraps content around it — give it a width, and clear or contain its parent.
6. Prefer flexbox and grid for modern layout; understand floats mainly to read older code.