DISPLAY

Display:block -

Good work! If you didn't notice much of a difference, don't worry. Our<div>s were block elements by default; as we specify different display values, they'll start to move around.

Display :inline -

Did you see that? Your <div>s all moved onto the same line! You can already start to see how this type of positioning can be useful for navigation bars like the one at the top of the main Codecademy page (where you can click "Learn," "Teach," and so on).

The inline-block value allows you to put several block elements on the same line. The inline value places all your elements next to one another, but not as blocks: they don't keep their dimensions.

Display:none -

The good news is, inline places all your elements on a single line. The bad news is that it doesn't maintain their "box"ness: as you saw, all your<div>s got squished to the smallest possible width!

The inline display value is better suited for HTML elements that are blocks by default, such as headers and paragraphs.

Finally, we'll try out the display valuenone. As you might expect, this prevents the page from displaying the selected element. As you might *not*expect, this removes the selected element from the page *entirely*, including any children and any content. Poof! Gone! (But not gone forever—changing the display value away fromnone will bring everything back.)

FLOATS

When you float an element on the page, you're telling the webpage: "I'm about to tell you where to put this element, but you have to put it into the **flow** of other elements." This means that if you have several elements all floating, they all know the others are there and don't land on top of each other.

You can think of the HTML page as sort of like a sea, and floating elements as boats on it: all the boats have positions on the sea, and they all see and steer clear of each other.

1. float right
2. float left
3. add text. Text will float left, box will float right.
4. Clearing floats. If you tell an element to clear: left, it will immediately move below any floating elements on the left side of the page; it can also clear elements on theright. If you tell it to clear: both, it will get out of the way of elements floating on the left *and* right!

POSITION

If you don't specify an element's positioning type, it defaults to static. This just means "where the element would normally go." If you don't tell an element how to position itself, it just plunks itself down in the document.

1. Static: Note that static is the default position value of an element, should you fail to apply any other value. If you have three statically positioned elements in your code, they will stack one on top of the next, as you might expect.
2. Relative: Position: relative - places an element in relevance to where it would normally occur. The space where the element would have normally occupied is left blank. Another way to think of it is, with relative positioning, all elements are first laid out in its default position. Then the element with the relative positioning is moved to its new position, while the rest of the elements remain in their fixed places.
   1. For starters, we can adjust a relatively positioned element with offset properties: top,right, bottom, and left. Using the markup from our previous example, let’s add an offset position to #box\_2
   2. Our three blocks are stacked up nicely, but this time the blue block (#box\_2) is pushed out 200 pixels from the left. This is where we start to bend the law of gravity to our will. The blue block is still in the flow of the document—elements are stacking one on top of the other—but notice the green block (#box\_3) on the bottom. It’s sitting underneath the blue block, even though the blue block isn’t directly above it. When you use the offset property to shift a relatively positioned element, it doesn’t affect the element(s) that follow. The green box is still positioned as if the blue box were in its non-offset position.
3. Absolute: The first type of positioning is **absolute** positioning. When an element is set to position: absolute, it's then positioned in relation to the first parent element it has that doesn't have position: static. If there's no such element, the element with position:absolute gets positioned relative to <html>.
   1. Unlike the static and relative values, an absolutely positioned element is removed from the normal flow. This means you can put it anywhere, and it won’t *affect or be affected* by any other element in the flow. Think of it as an element with a giant strip of velcro on its back. Just tell it where to stick and it sticks. Exactly like the relative value, absolutely positioned elements respond to offset properties for positioning. You can set an element to top: 100px and left: 200px; and that element will sit exactly 100px from the top and 200px from the left of the document. Let’s look at an example
   2. Since we set each box’s position value to absolute, we’ve essentially velcroed a box to each corner of our browser window. As you resize the browser, those boxes will stay in their respective corners. If you shrink the browser window so that the boxes overlap, you’ll notice that there is no interaction at all—that’s because they’re out of the document’s normal flow.
4. Fixed: An element with position: fixed shares all the rules of an absolutely positioned element, except that the viewport (browser/device window) positions the fixedelement, as opposed to any parent element. Additionally, a fixed element does not scroll with the document. It stays, well…fixed. Let’s look at an example
   1. As you scroll, notice that it doesn’t move. Notice that the left and right offset properties are set to zero. Since the fixed value behaves similar to the absolute value, we can stretch the width of the element to fit the viewport while fixing the element to the bottom using bottom: 0;. Use caution with the fixed value: Support in older browsers is spotty at best. For example, older versions of Internet Explorer renderfixed elements as static elements. And, you now know that static elements don’t behave like fixed elements, right?