

# Absolute and relative position

## (OPTIONAL)

**Note:** This section is optional material included for the curious. It will not appear on any graded question.

### POSITIONED ELEMENTS

We read about the positioned elements in an earlier section. There are five positioning properties (`left`, `top`, `right`, `bottom`, and `z-index`) that can be used to influence the position of an element. But these properties, by default, will have no effect on any element.

This is because all elements are `position: static` by default, and static elements ignore those positioning properties. Only positioned elements, which are elements where the position property is set to something besides static, respect the properties.

We saw `position: fixed`, which is fairly simple - the positioning properties cause the fixed elements to be positioned relative the browser window - they don't even scroll with the content. Besides `fixed` and `static` the `position` property can also be set to `relative` and `absolute`. For these values, the positioning properties have different interpretations.

### RELATIVE

```
position: relative;
```

The relative value is exactly like static in that the "flowing text" model of layout is setting the initial position for the element (including margins and display). But, unlike static, elements with relative position respect the positioning properties (`left`, `top`, `right` and `bottom`). These properties will move the named edge of the element **from** its initial position. So a value of `top: 20px;` will move the top edge of the element 20 pixels further down the page. And similarly, a value of `left: 20px;` will move an element 20 pixels from its original left edge, which means move it 20 pixels to the right.

The `relative` position property has three primary gotchas of which you should be aware

- items are moved independently of siblings
- opposite positioning properties (like `left` and `right`) cannot both be used
- no automatic size adjustments

## independence - margin-top vs top

**IMPORTANT:** The positioning properties (`left`, `top`, `right` and `bottom`) adjust the placement the element ***independent of its siblings***. What does this mean? Let's imagine we have a list and we want to move one of the items a little further down the page. Should we use `margin-top` to move it? Or `position: relative` in conjunction with the `top` property? The answer to that question is contingent upon whether you want any of the other list items to move as well? If you want the siblings to move down as well, then use `margin-top`.

Here is an example. Here are two lists. We want the third item in the list to have a background color and to be moved down by 30 pixels. Compare what happens when we use `margin-top` to move it, versus a positioning property (`top`). When we use `top` to adjust the "Third" item appears overlapping the Fourth and Fifth items, they did not move at all.

margin-top		top	
CSS	Result	CSS	Result
<pre><code>.third {   margin-top: 30px;   background-color: lightblue; }</code></pre>	<ul style="list-style-type: none"> <li>• First</li> <li>• Second</li> <li>• Third</li> <li>• Fourth</li> <li>• Fifth</li> </ul>	<pre><code>.third {   position: relative;   top: 30px;   background-color: lightblue; }</code></pre>	<ul style="list-style-type: none"> <li>• First</li> <li>• Second</li> <li>• <del>Fourth</del> Third</li> <li>• Fifth</li> </ul>

This is why, in our introduction to CSS, we said that `margin` should be your "go to" property when you want to adjust position.

## cannot use opposite properties

When using `position: relative` if you use the `left` property you cannot also use

the `right` property. And, if you use the `top` property you cannot also use the `bottom` property. If both properties are applied, then the CSS precedence rules will determine which "wins", which is usually just the last one applied.

Again, this is unlike margins where both `margin-right` and `margin-left` can be meaningfully used.

## no automatic size adjustments

This is a corollary to the previous limitation. You may recall that block level elements take the width of their parent (when no width is specified). And when using left or right margins on a block level element that does not have an explicit width the browser will smartly size the element down for you to make it fit. But this size adjustment does not happen when using `position: relative` and the `left` or `right` positional properties. This is easily illustrated with an example. Below is a block level paragraph with a border applied to it. When a `margin-left` is applied to it, the paragraph is made smaller and no part of it goes outside its parent. But when it is `position: relative` and moved with the `left` property, it can leave the bounds of its parent, or go offscreen.

margin-left		left	
CSS	Result	CSS	Result
<pre>p {   margin-left:   40px; }</pre>	<div>Dorothea was altogether captivated by the wide embrace of this conception.</div>	<pre>p {   position:   relative;   left:   40px; }</pre>	<div>Dorothea was altogether captivated by the wide embrace of this conception.</div>

Admittedly, this is not necessarily a "limitation", for many layout situations preserving the size is exactly what is wanted.

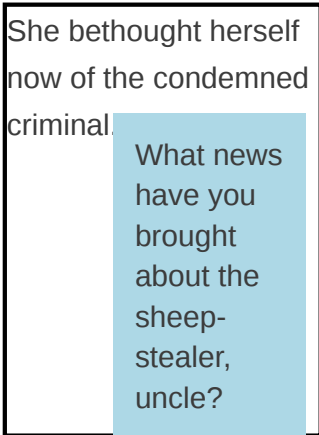
## ABSOLUTE

`position: absolute;`

Absolute positioning, as it is realized in the browsers via CSS, can be very powerful and that ease and power is very seductive to many CSS newbies. But there are some big trade-offs incurred that are often not appreciated. Many experience professional CSS developers completely forswear absolute positioning - they will not use it under any circumstances.

An element that is positioned absolutely is taken out of the normal text "flow" that governs elements positioned statically or relatively. Instead, an absolutely positioned element is positioned by the `left`, `top`, `right`, and/or `bottom` properties. The size or position of siblings have no effect on an absolutely positioned element that has some positioning properties set (`left`, `top`, etc.).

Let's see this with a simple example. Here we have a paragraph that contains some text and an inner `<q>`. To help illustrate, the paragraph has its height set and a border applied. The `<q>` is positioned absolutely.

HTML	CSS	Result
<pre>&lt;p&gt;She bethought herself now of the condemned criminal. &lt;q&gt;What news have you brought about the sheep-stealer, uncle? &lt;/q&gt;&lt;/p&gt;</pre>	<pre>p {   height: 200px;   border: 2px solid black; } q {   position: absolute;   left: 50px;   top: 50px;   background- color: lightblue;   padding: 10px;   display: block;   width: 70px; }</pre>	

So that seems fairly straightforward and useful. But there are some subtle caveats and trade-offs of which you must be wary:

- interpretation of positioning depends upon parent/grandparent elements being positioned elements
- best practice: use both a horizontal and a vertical positioning property on every absolute element
- absolutely positioned elements do not contribute to size of parent
- absolute positioned block level elements do not get the width of their parent
- margins do not work the same
- opposite properties can be used to size element


interpretation of positioning properties (top, left, etc) depends up parent/grandparent being positioned elements (or not).

*IMPORTANT:* for an absolutely positioned element **where** the left, top (etc.) are calculated **from** depends upon the **position** property of the parent and grandparents of the element in question. If the parent of the element is a positioned element (meaning its position is set to anything except **position:static**), then an absolutely positioned child is positioned relative to that parents rectangle (or grandparent, or great-grandparent, etc). But if none of the parents are positioned elements, the child is positioned relative to the bounds of the document.

This means that how the position properties of an absolutely positioned element are interpreted depends upon the position property of its parent (and grandparents). For many developers, this is spooky action at a distance. Changing the position property on an element may not affect that element at all, but can cause it children (or great great grandchildren) to suddenly jump to another part of the page.

In the example below, someone who did not read the section in Week 3 about how to style list items has decided to put their own numbers on. There are four list items each containing child spans which are absolutely positioned. Two of the list items are **position:relative**, so the spans are positioned starting from their rectangle. But two of the list items are **position:static** (the default), so the spans are moved up to the `<ul>` (which is also **position:relative**) where they overlap each other. (The red 1 is hidden behind the red 2). Borders have been added in the result below, so you can easily see the rectangle for the `<li>` versus the `<ul>`



HTML	CSS	Result
<pre> &lt;ul&gt; &lt;li&gt;First &lt;span&gt;1&lt;/span&gt; &lt;/li&gt; &lt;li&gt;Second &lt;span&gt;2&lt;/span&gt; &lt;/li&gt; &lt;li class="rel"&gt;Third &lt;span&gt;3&lt;/span&gt; &lt;/li&gt; &lt;li class="rel"&gt;Fourth &lt;span&gt;4&lt;/span&gt; &lt;/li&gt; &lt;/ul&gt; </pre>	<pre> ul { position: relative; }  .rel { position: relative; }  span { position: absolute; left: 0px; top: 0px; } </pre>	

best practice: use both a horizontal and a vertical positioning property on every absolute element

There is a very subtle extension to the previous interpretation problem in that if an element is set to be `position: absolute`, but has no horizontal positioning property (ie, `left` or `right`), then it will be displayed in the flow exactly as it would have been.

Except, later, if `left: 0px` were added (for example), then the element may jump to be positioned at the left edge of the first parent/grandparent that is a positioned element. The same applies vertically. This is a bewildering behavior, as most users do not expect there to be a difference between `left: 0px` and no `left` property at all.

Therefore, the best practice to follow is for any absolutely positioned element, ensure that one of the horizontal positioning properties (ie `left` or `right`) and one of the vertical properties (`top` or `bottom`) are set.

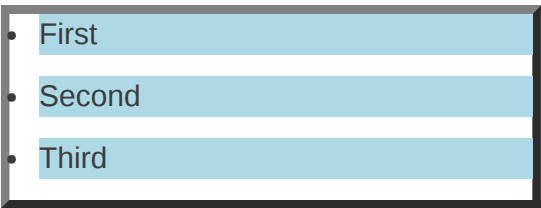

absolutely positioned elements do not contribute to size of parent

Whether you realize it or not, one of the most useful default behaviors is that the height of a parent element is automatically extended to include all its children, its content. Designers

working in CSS unconsciously lean on this fact as they plan out layouts and adjust element positions. But this is **not** true for children that are positioned absolutely. Absolutely positioned children do not contribute to the size of the parent element. A parent element that contains only absolutely positioned children will have a height of 0, it has no "measurable" content and will behave as if it is empty.

A consequence of this is that as a design starts to use absolute positioning, it may also have to start explicitly set the dimensions of containers, which makes the overall design brittle and less adaptable.

In the example below there are two lists (<ul>) each with a fat border. The list on the left is normal - its children contribute to making the <ul> taller and the fat border extends around, enclosing everything correctly. But the list items (<li>) on the right are positioned absolutely. So those list items on the right do not contribute to the height of the parent. As a result it ends up with a height of 0, as if it were empty. The fat border just becomes a fat flat line, and the list items themselves are not enclosed.

default	absolute
	

## absolute positioned block level elements do not get the width of their parent

Earlier we learned that block level element automatically get the width of their parent - ie, they extend to become full width. But this is only true for static and relative positioned elements. Elements that are absolute positioned (or fixed) do not exhibit this behavior. If you look at the table above, from the previous point, the individual list items have a light blue background color. All the list items are block level elements, and the ones on the left which are `position: static`, extend their rectangle rightward to fill the entire line. But the right column of absolutely positioned items do not. Their initial size is simply the size of their content.

## margins do not work the same

For static and relative positioned items, margins can be used to adjust an elements position and keep neighboring siblings "away". We quickly intuit this about margins and assume it. But when an element is absolutely positioned, a given margin *might* be able to move the element, but will not result in moving any siblings. Margins cannot be used to keep siblings "away", to fight crowding.

As a general rule, if a positioning property is being used (like `left`), then the matching margin (`margin-left`) can also be used to additionally adjust the position of the element. But otherwise the margin will likely have no effect.

## opposite properties can be used to size element

This is one of the nicer features. Working with preset dimension properties (`height` and `width`) can make your design brittle and reduce its adaptability. But absolutely positioned items can instead set the opposite positioning properties (like `left` and `right`) and leave the matching dimensional property (`width`) unspecified. The element will grow or shrink based on the size of ancestor it is positioning against. Note that this feature is only available to absolute and fixed position elements.