

link: <https://developer.mozilla.org/en-US/docs/Web/CSS/position>

position

The `position` [CSS](#) property sets how an element is positioned in a document. The [top](#), [right](#), [bottom](#), and [left](#) properties determine the final location of positioned elements.

Try it

CSS Demo: position

RESET

position: static;

position: relative;
top: 40px; left: 40px;

position: absolute;
top: 40px; left: 40px;

position: sticky;
top: 20px;

In this demo you can control the `position` property for the yellow box.

To see the effect of sticky positioning, select the `position: sticky` option and scroll this container.

The element will scroll along with its container, until it is at the top of the container (or reaches the offset specified in `top`), and will then stop scrolling,

Syntax

```
position: static;
position: relative;
position: absolute;
position: fixed;
position: sticky;

/* Global values */
position: inherit;
position: initial;
position: revert;
position: revert-layer;
position: unset;
```

Values

static

The element is positioned according to the normal flow of the document. The [top](#), [right](#), [bottom](#), [left](#), and [z-index](#) properties have *no effect*. This is the default value.

relative

The element is positioned according to the normal flow of the document, and then offset *relative to itself* based on the values of [top](#), [right](#), [bottom](#), and [left](#). The offset does not affect the position of any other elements; thus, the space given for the element in the page layout is the same as if position were `static`.

This value creates a new [stacking context](#) when the value of `z-index` is not `auto`. Its effect on `table-*`-group, `table-row`, `table-column`, `table-cell`, and `table-caption` elements is undefined.

absolute

The element is removed from the normal document flow, and no space is created for the element in the page layout. It is positioned relative to its closest [positioned](#) ancestor, if any; otherwise, it is placed relative to the initial [containing block](#). Its final position is determined by the values of [top](#), [right](#), [bottom](#), and [left](#).

This value creates a new [stacking context](#) when the value of `z-index` is not `auto`. The margins of absolutely positioned boxes do not [collapse](#) with other margins.

fixed

The element is removed from the normal document flow, and no space is created for the element in the page layout. It is positioned relative to the initial [containing block](#) established by the [viewport](#), except when one of its ancestors has a `transform`, `perspective`, or `filter` property set to something other than `none` (see the [CSS Transforms Spec](#)), or the `will-change` property is set to `transform`, in which case that ancestor behaves as the containing block. (Note that there are browser inconsistencies with `perspective` and `filter` contributing to containing block formation.) Its final position is determined by the values of [top](#), [right](#), [bottom](#), and [left](#).

This value always creates a new [stacking context](#). In printed documents, the element is placed in the same position on *every page*.

sticky

The element is positioned according to the normal flow of the document, and then offset relative to its *nearest scrolling ancestor* and [containing block](#) (nearest block-level ancestor), including table-related elements, based on the values of [top](#), [right](#), [bottom](#), and [left](#). The offset does not affect the position of any other elements.

This value always creates a new [stacking context](#). Note that a sticky element "sticks" to its nearest ancestor that has a "scrolling mechanism" (created when `overflow` is `hidden`, `scroll`, `auto`, or `overlay`), even if that ancestor isn't the nearest actually scrolling ancestor.

Description

Types of positioning

- A **positioned element** is an element whose [computed](#) position value is either `relative`, `absolute`, `fixed`, or `sticky`. (In other words, it's anything except `static`.)
- A **relatively positioned element** is an element whose [computed](#) position value is `relative`. The [top](#) and [bottom](#) properties specify the vertical offset from its normal position; the [left](#) and [right](#) properties specify the horizontal offset.
- An **absolutely positioned element** is an element whose [computed](#) position value is `absolute` or `fixed`. The [top](#), [right](#), [bottom](#), and [left](#) properties specify offsets from the edges of the element's [containing block](#). (The containing block is the ancestor relative to which the element is positioned.) If the element has margins, they are added to the offset. The element establishes a new [block formatting context](#) (BFC) for its contents.

- A **stickily positioned element** is an element whose [computed](#) `position` value is `sticky`. It's treated as relatively positioned until its [containing block](#) crosses a specified threshold (such as setting `top` to value other than `auto`) within its flow root (or the container it scrolls within), at which point it is treated as "stuck" until meeting the opposite edge of its [containing block](#).

Most of the time, absolutely positioned elements that have `height` and `width` set to `auto` are sized so as to fit their contents. However, non-replaced, absolutely positioned elements can be made to fill the available vertical space by specifying both `top` and `bottom` and leaving `height` unspecified (that is, `auto`). They can likewise be made to fill the available horizontal space by specifying both `left` and `right` and leaving `width` as `auto`.

Except for the case just described (of absolutely positioned elements filling the available space):

- If both `top` and `bottom` are specified (technically, not `auto`), `top` wins.
- If both `left` and `right` are specified, `left` wins when [direction](#) is `ltr` (English, horizontal Japanese, etc.) and `right` wins when [direction](#) is `rtl` (Persian, Arabic, Hebrew, etc.).

Accessibility concerns

Ensure that elements positioned with an `absolute` or `fixed` value do not obscure other content when the page is zoomed to increase text size.

- [MDN Understanding WCAG, Guideline 1.4 explanations](#)
- [Visual Presentation: Understanding SC 1.4.8 | Understanding WCAG 2.0](#)

Performance & Accessibility

Scrolling elements containing `fixed` or `sticky` content can cause performance and accessibility issues. As a user scrolls, the browser must repaint the sticky or fixed content in a new location. Depending on the content needing to be repainted, the browser performance, and the device's processing speed, the browser may not be able to manage repaints at 60 fps, causing accessibility concerns for people with sensitivities and jank for everyone. One solution is to add [will-change: transform](#) to the positioned elements to render the element in its own layer, improving repaint speed and therefore improving performance and accessibility.

Formal definition

Initial value	static
Applies to	all elements
Inherited	no
Computed value	as specified
Animation type	discrete
Creates stacking context	yes

Formal syntax

```
position =
  static      .|.
  relative    .|.
  absolute    .|.
  sticky      .|.
  fixed       .|.
  running( )
```

Examples

Relative positioning

Relatively positioned elements are offset a given amount from their normal position within the document, but without the offset affecting other elements. In the example below, note how the other elements are placed as if "Two" were taking up the space of its normal location.

HTML

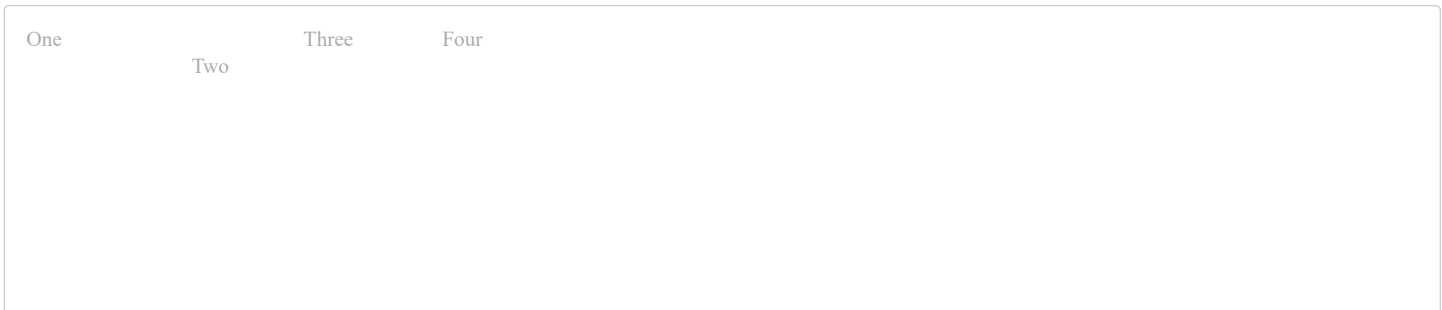
```
<div class="box" id="one">One</div>
<div class="box" id="two">Two</div>
<div class="box" id="three">Three</div>
<div class="box" id="four">Four</div>
```

CSS

```
* {
  box-sizing: border-box;
}

.box {
  display: inline-block;
  width: 100px;
  height: 100px;
  background: red;
  color: white;
}

#two {
  position: relative;
  top: 20px;
  left: 20px;
  background: blue;
}
```



Absolute positioning

Elements that are relatively positioned remain in the normal flow of the document. In contrast, an element that is absolutely positioned is taken out of the flow; thus, other elements are positioned as if it did not exist. The absolutely positioned element is positioned relative to its *nearest positioned ancestor* (i.e., the nearest ancestor that is not `static`). If a positioned ancestor doesn't exist, it is positioned relative to the ICB (initial containing block — see also the [W3C definition](#)), which is the containing block of the document's root element.

HTML

```
<h1>Absolute positioning</h1>
```

```
<p>
  I am a basic block level element. My adjacent block level elements sit on new
  lines below me.
</p>

<p class="positioned">
  By default we span 100% of the width of our parent element, and we are as tall
  as our child content. Our total width and height is our content + padding +
  border width/height.
</p>

<p>
  We are separated by our margins. Because of margin collapsing, we are
  separated by the width of one of our margins, not both.
</p>

<p>
  inline elements <span>like this one</span> and <span>this one</span> sit on
  the same line as one another, and adjacent text nodes, if there is space on
  the same line. Overflowing inline elements
  <span>wrap onto a new line if possible – like this one containing text</span>,
  or just go on to a new line if not, much like this image will do:
  
</p>
```

CSS

```
* {
  box-sizing: border-box;
}

body {
  width: 500px;
  margin: 0 auto;
}

p {
  background: aqua;
  border: 3px solid blue;
  padding: 10px;
  margin: 10px;
}

span {
  background: red;
  border: 1px solid black;
}

.positioned {
  position: absolute;
  background: yellow;
  top: 30px;
  left: 30px;
}
```

Result

Absolute positioning

By default we span 100% of the width of our parent element, and we are as tall as our child content. Our total width and height is our content + padding + border width/height.

I am a basic block level element. My adjacent block level elements sit on new lines below me.

We are separated by our margins. Because of margin collapsing, we are separated by the width of one of our margins, not both.

inline elements like this one and this one sit on the same line as one another, and adjacent text nodes, if there is space on the same line. Overflowing inline elements wrap onto a new line if possible — like this one containing text, or just go on to a new line if not, much like this image will do:



Fixed positioning

Fixed positioning is similar to absolute positioning, with the exception that the element's [containing block](#) is the initial containing block established by the *viewport*, unless any ancestor has `transform`, `perspective`, or `filter` property set to something other than `none` (see [CSS Transforms Spec](#)), which then causes that ancestor to take the place of the elements [containing block](#). This can be used to create a "floating" element that stays in the same position regardless of scrolling. In the example below, box "One" is fixed at 80 pixels from the top of the page and 10 pixels from the left. Even after scrolling, it remains in the same place relative to the viewport. Also, when the [will-change](#) property is set to `transform`, a new containing block is established.

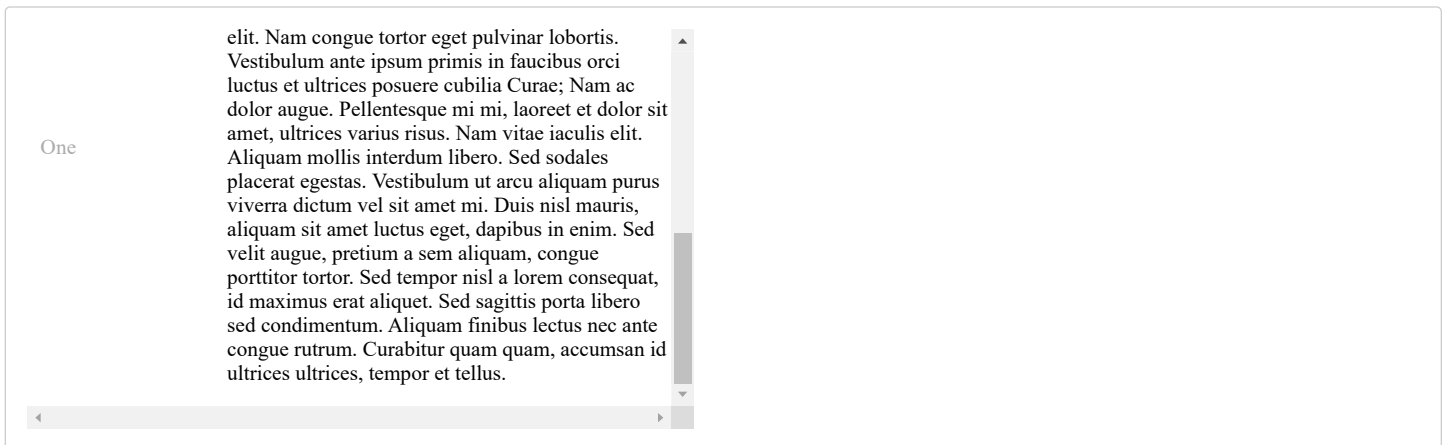
HTML

```
<div class="outer">
  <p>
    Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam congue tortor
    eget pulvinar lobortis. Vestibulum ante ipsum primis in faucibus orci luctus
    et ultrices posuere cubilia Curae; Nam ac dolor augue. Pellentesque mi mi,
    laoreet et dolor sit amet, ultrices varius risus. Nam vitae iaculis elit.
    Aliquam mollis interdum libero. Sed sodales placerat egestas. Vestibulum ut
    arcu aliquam purus viverra dictum vel sit amet mi. Duis nisl mauris, aliquam
    sit amet luctus eget, dapibus in enim. Sed velit augue, pretium a sem
    aliquam, congue porttitor tortor. Sed tempor nisl a lorem consequat, id
    maximus erat aliquet. Sed sagittis porta libero sed condimentum. Aliquam
    finibus lectus nec ante congue rutrum. Curabitur quam quam, accumsan id
    ultrices ultrices, tempor et tellus.
  </p>
  <p>
    Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam congue tortor
    eget pulvinar lobortis. Vestibulum ante ipsum primis in faucibus orci luctus
    et ultrices posuere cubilia Curae; Nam ac dolor augue. Pellentesque mi mi,
    laoreet et dolor sit amet, ultrices varius risus. Nam vitae iaculis elit.
    Aliquam mollis interdum libero. Sed sodales placerat egestas. Vestibulum ut
    arcu aliquam purus viverra dictum vel sit amet mi. Duis nisl mauris, aliquam
    sit amet luctus eget, dapibus in enim. Sed velit augue, pretium a sem
    aliquam, congue porttitor tortor. Sed tempor nisl a lorem consequat, id
    maximus erat aliquet. Sed sagittis porta libero sed condimentum. Aliquam
    finibus lectus nec ante congue rutrum. Curabitur quam quam, accumsan id
    ultrices ultrices, tempor et tellus.
  </p>
  <div class="box" id="one">One</div>
</div>
```

CSS

```
* {  
  box-sizing: border-box;  
}  
  
.box {  
  width: 100px;  
  height: 100px;  
  background: red;  
  color: white;  
}  
  
#one {  
  position: fixed;  
  top: 80px;  
  left: 10px;  
  background: blue;  
}  
  
.outer {  
  width: 500px;  
  height: 300px;  
  overflow: scroll;  
  padding-left: 150px;  
}
```

Result



Sticky positioning

Sticky positioning can be thought of as a hybrid of relative and fixed positioning when its nearest scrolling ancestor is the viewport. A stickily positioned element is treated as relatively positioned until it crosses a specified threshold, at which point it is treated as fixed until it reaches the boundary of its parent. For example:

```
#one {  
  position: sticky;  
  top: 10px;  
}
```

The above CSS rule would position the element with id *one* relatively until the viewport was scrolled such that the element would be less than 10 pixels from the top. Beyond that threshold, the element would be fixed to 10 pixels from the top.

A common use for sticky positioning is for the headings in an alphabetized list. The "B" heading will appear just below the items that begin with "A" until they are scrolled offscreen. Rather than sliding offscreen with the rest of the content, the "B" heading will then remain fixed to the top of the viewport until all the "B" items have scrolled offscreen, at which point it will be covered up by the "C" heading, and so on.

You must specify a threshold with at least one of `top`, `right`, `bottom`, or `left` for sticky positioning to behave as expected. Otherwise, it will be indistinguishable from relative positioning.

HTML

```
<dl>
  <div>
    <dt>A</dt>
    <dd>Andrew W.K.</dd>
    <dd>Apparat</dd>
    <dd>Arcade Fire</dd>
    <dd>At The Drive-In</dd>
    <dd>Aziz Ansari</dd>
  </div>
  <div>
    <dt>C</dt>
    <dd>Chromeo</dd>
    <dd>Common</dd>
    <dd>Converge</dd>
    <dd>Crystal Castles</dd>
    <dd>Cursive</dd>
  </div>
  <div>
    <dt>E</dt>
    <dd>Explosions In The Sky</dd>
  </div>
  <div>
    <dt>T</dt>
    <dd>Ted Leo & The Pharmacists</dd>
    <dd>T-Pain</dd>
    <dd>Thrice</dd>
    <dd>TV On The Radio</dd>
    <dd>Two Gallants</dd>
  </div>
</dl>
```

CSS

```
* {
  box-sizing: border-box;
}

dl > div {
  background: #fff;
  padding: 24px 0 0 0;
}

dt {
  background: #b8c1c8;
  border-bottom: 1px solid #989ea4;
  border-top: 1px solid #717d85;
  color: #fff;
  font: bold 18px/21px Helvetica, Arial, sans-serif;
  margin: 0;
  padding: 2px 0 0 12px;
  position: -webkit-sticky;
  position: sticky;
```



```
top: -1px;
}

dd {
  font: bold 20px/45px Helvetica, Arial, sans-serif;
  margin: 0;
  padding: 0 0 0 12px;
  white-space: nowrap;
}

dd + dd {
  border-top: 1px solid #ccc;
}
```

Result



Specifications

Specification
CSS Positioned Layout Module Level 3 # position-property

Browser compatibility

[Report problems with this compatibility data on GitHub](#)

	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android	Opera Android
position	Chrome 1	Edge 12	Firefox 1	Opera 4	Safari 1	Chrome 18 Android	Firefox 4 for Android	Opera Android

	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android	Opera Android
Absolutely-positioned flex children	Chrome 52	Edge 12	Firefox 52	Opera 39	Safari 11	Chrome 52 Android	Firefox for 52 Android	Opera 40 Android
fixed	Chrome 1	Edge 12	Firefox 1	Opera 4	Safari 1	Chrome 18 Android	Firefox 4 for Android	Opera 40 Android
Table elements as sticky positioning containers	Chrome 56	Edge 16	Firefox 59	Opera 43	Safari 8	Chrome 56 Android	Firefox 59 for Android	Opera 40 Android
sticky	Chrome 56	Edge 16	Firefox 32	Opera 43	Safari 13	Chrome 56 Android	Firefox for 32 Android	Opera 40 Android

Tip: you can click/tap on a cell for more information.

- Full support
- See implementation notes.
- Requires a vendor prefix or different name for use.
- Has more compatibility info.

See also

- [Learn CSS: Positioning](#)

Last modified: Nov 30, 2022, [by MDN contributors](#)