CSS Position Ultimate Guide

January 17, 2022

CSS

The position property in CSS only has a few valid values, but those values can lead to endless design possibilities which makes learning position quite difficult. In this article I will be breaking down each position value and explaining everything you need to know about them. This includes even the more obscure concepts related to position that most articles/videos ignore.

If you prefer to learn visually, check out the video version of this article.

Static Position

To start we are going to cover the easiest position value which you probably have never heard of but use all the time and that is the `static` position. `static` is the default value for the position property and essentially just means that an element will follow the normal document flow and will position itself based on the standard positioning rules.

Any element that you do not apply a position property to will be `static` which means most elements you work with are statically positioned. `static` positioned elements cannot have the `z-index`, `top`, `left`, `right`, or `bottom` properties applied to them.

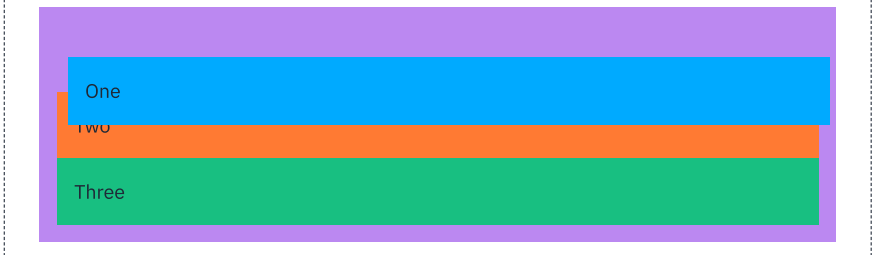
```
.one {
  position: static;
}
```

Relative Position

The next simplest position type is `relative` position. A `relative` position element works exactly the same as `static` position, but you can now add `z-index`, `top`, `left`, `right`, and `bottom` properties to it. If you make an element `relative` positioned without setting any of these extra properties you will notice it looks exactly the same as a `static` positioned element. This is because `relative` positioned elements also follow the normal document flow, but you can offset them using the `top`, `left`, `right`, and `bottom` properties.

```
.one {
  position: relative;
}
```


```
.one {
  position: relative;
  top: 30px;
  left: 10px;
}
```



From the above example you can see that the one element works just like `static` when there is no extra properties defined. Once you add a property like `left`, or `top`, though, you can see that the element offsets itself relative to its normal position by these `top`, `left`, `right`, and `bottom` properties. You will notice, however, that the other elements do not move to account for the offset position of the `relative` positioned element. This is because all the other elements assume that the `relative` positioned element has no offsets and they determine their position based on where the `relative` positioned element would have been if it was `static`.

Now on its own `relative` position is not that useful as you do not usually want to offset an element without also moving all the elements around it. The main use cases for position `relative` are to either set the `z-index` of the element, or to be used as a container for `absolute` positioned elements which we will talk about next.

Absolute Position

The `absolute` position is the first position that completely removes the element from the normal document flow. If you give an element position `absolute` all other elements will act as if the `absolute` positioned element doesn't even exist.

```
.one {
  position: absolute;
}
```

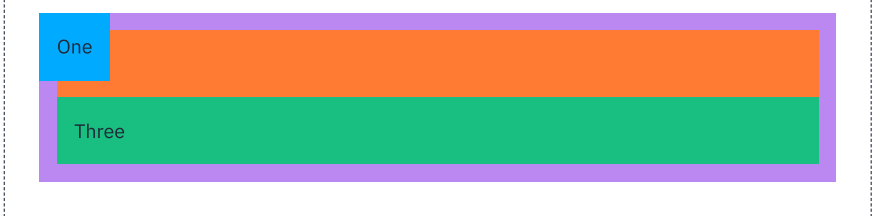
As you can see elements two and three are laid out as if element one never even existed. You will also notice that element one no longer fills the full width. This is because `absolute` positioned elements have their width defaulted to `auto` instead of being full width like a div. Also, by default an `absolute` positioned element will place itself in the document where it normally would have rendered if it was a static element, but we can change that with the `top`, `left`, `right`, and `bottom` properties.

```
.one {
  position: absolute;
  top: 0;
  left: 0;
}
```

Now you can see our element has moved to the top left corner of our dashed border. I am using this dashed border to represent the entire screen since by default a position `absolute` element will position itself relative to the body so a `top` and `left` of 0 means the element will appear in the top left corner of the body. You can change the element that the `absolute` positioned element is positioned off of by setting the position of one of its parent elements to anything other than `static`. This is one of the most common places `relative` position is used.

```
.purple-parent {
  position: relative;
}

.one {
  position: absolute;
  top: 0;
  left: 0;
}
```



By setting the purple parent element to a position of `relative` I am now forcing the `absolute` positioned child one element to be in the top left corner of the parent instead of the body. This combination of `relative` and `absolute` positions is incredibly common.

Fixed Position

Now we come to one of the lesser used positions which is the `fixed` position. `fixed` position is a bit like `absolute` position in that it removes the element from the document flow, but `fixed` position elements are always positioned relative to the screen no matter what position its parent elements are.

```
.one {
  position: fixed;
  top: 0;
  left: 0;
}
```

As you can see our one element is fixed in the top left corner of our page. You will also notice when you scroll the page that it stays stuck in the top left corner of the page no matter where you scroll to. This is the main difference between `fixed` and `absolute`.

Sticky Position

The final position value is `sticky`. This position is a combination of both `fixed` and `static` position and combines the best of them both. An element with position `sticky` will act like a `static` positioned element until the page scrolls to a point where the element hits the `top`, `left`, `right`, or `bottom` value specified. It will then act like a fixed position element and scroll with the page until the element gets to the end of its container.

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

At first you will notice that the one element is in its normal position in the document flow as if it were `static`. Once you scroll the page to the point that the top of the one element is 10px from the top of the page it will then stick to the top of the page as if it were position `fixed`. This will last this way all the way until the element gets to the bottom of the purple parent container in which case it will stop scrolling with the page.

`sticky` position is the perfect position for navbars that scroll with the page, headings in long lists, and many other use cases.

Absolute/Fixed Position Advanced

Now there is one thing about `fixed` and `absolute` position elements that you need to know since it could cause some weird, hard to find bugs. An element that is `absolute` positioned will use the first parent element that has a non-static position as its container to offset itself from and it will default to the body if no parent has a position value other than `static`. This we already know, but this is not the only way to define a parent container. `absolute` positioned elements will also check for a parent that has either the `transform`, `filter`, or `perspective` properties set. If any of those are found on a parent it will use that parent as the container.

```
.purple-parent {
  transform: translateX(0);
}

.one {
  position: absolute;
  top: 0;
  left: 0;
}
```

In the above example our purple parent element has a `static` position, but since it has a `transform` set it acts as the container for our `absolute` positioned child.

The same thing works for `fixed` position elements. This also makes the scrolling behavior of the `fixed` position element not really work anymore.

```
.purple-parent {
  transform: translateX(0);
}

.one {
  position: fixed;
  top: 0;
  left: 0;
}
```

You will notice even as we scroll our page the `fixed` position element is stuck inside the purple parent container because it has a `transform` property set. This is generally not what you want

which is why it is best to always define `fixed` position elements at the top level to avoid this potential bug.

Conclusion

The position property in CSS has only a few values, but each value has lots of nuance around when and how you can use it. This leads to lots of complexity but also lots of potential.