What is position property in CSS and its type?

The `position` property specifies the type of positioning method used for an element (static, relative, fixed, absolute or sticky).

The position Property

The `position` property specifies the type of positioning method used for an element.

There are five different position values:

- `static`
- `relative`
- `fixed`
- `absolute`
- `sticky`

Elements are then positioned using the top, bottom, left, and right properties. However, these properties will not work unless the `position` property is set first. They also work differently depending on the position value.

All CSS Positioning Properties

Property	Description
bottom	Sets the bottom margin edge for a positioned box
clip	Clips an absolutely positioned element
left	Sets the left margin edge for a positioned box
position	Specifies the type of positioning for an element
right	Sets the right margin edge for a positioned box
top	Sets the top margin edge for a positioned box

How many types of positioning are there in CSS?

1. Static

> `position: static` is the default value that an element will have. This means if you don't declare `position` for an element in CSS, it will automatically be set to `static`.

It's important to note that having a static position is the same as not setting the `position` property at all. (This will come into play a bit later on with absolute positioning.)

> Elements that are statically positioned will appear on the page in what we call the normal flow. For example, if you have multiple `<div>` elements one after the other, they will appear on the page directly below one another.
>
> Here's a quick demo to illustrate static position. We are using the following HTML markup:

```
<div class="first purple"> position: static </div>
```

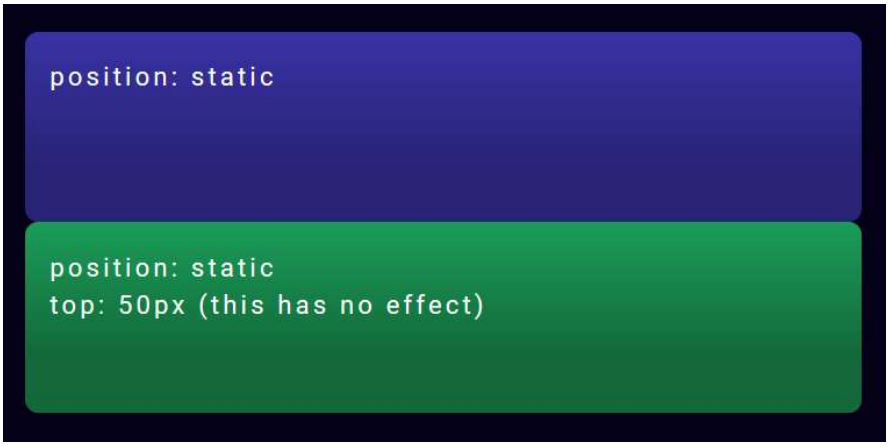
```
<div class="another green"> position: static<br> top: 50px (this
has no effect) </div>
```

And here's the CSS we're using:

```
.first {

   // No position set, so it's static

}

.another {

 // No position set, so it's static top: 50px;

}
```

The second element has a `top` property set to `50px`. You would think that this would move it down 50px, right?

However, here is how it will look on a webpage:



Since both elements have a static position, none of the layout CSS properties will do anything. This makes that `top` property have no effect on how the second element is displayed.

So that second element ends up being directly below the first element, with no space between.

How can we fix this? Let's move on to the next position:

2. Relative

`position: relative` is similar to `static` in that relatively positioned elements will follow the normal flow of the webpage. But the main difference is that using `relative` will now unlock the other CSS layout properties.

Think about it this way: you are setting the element to be positioned relative to other elements on the page.

Let's see what this looks like, and adjust our CSS like this:

```
.first {

 position: static;

}

.another {

 position: relative;
```

```
  top: 50px;


}
```

All the CSS is exactly the same, except that we changed the second element to use `position: relative`. Doing this makes that `top: 50px` work!

You can see that the second element is now 50px below the first one, adding that space between them.

Relatively positioned parent and child elements

Let's try another example, using a parent element with a child element nested inside it. Both have `position: relative` set.

Here's the HTML for that:

```
<div class="parent purple">

 <div class="child magenta"></div>

</div>
```

And our CSS styles:

```
.parent {

 position: relative;

}
```

```
.child {

 position: relative;

 top: 0px;

 left: 0px;

}
```

And here's what that code will look like in real life:

You can see that the pink child element is nicely nested inside the purple parent element. The child is also positioned as close to the top and left inside the parent as is possible. (It will go as far up as the parent text)

Position relative is relatively straight-forward, right? Well hold on to your hats, because things are about to get crazy with `position absolute`.

3. Absolute

`position: absolute` will cause an element to be taken out of that normal flow of the webpage.

Wait, what does that mean?

So before, using static or relative positioning, elements would be nicely displayed one below the other, depending on their order in the

HTML markup. But with absolute positioning, the element is completely taken out of that entire flow.

To help explain, let's do a comparison to illustrate the difference between relative and absolute positioning.

In the previous example, we had a parent element with a child element, both positioned relatively. And the child was nested inside the parent element.

Let's change that child to be positioned absolutely in the parent!

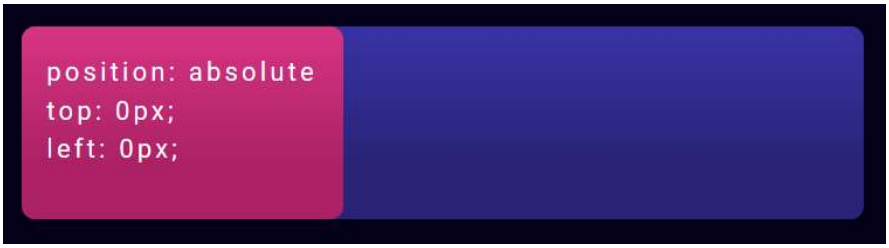
Our CSS will now look like this:

```
.parent {

 position: relative;

}
```

```
.child {

 position: absolute;

 top: 0px;

 left: 0px;

}
```

The pink child element now looks very different from our last

example:



While it is still within the confines of the parent element, it is positioned at the very top and very left of the parent. It's even covering up the parent text content!

This is due to the `top: 0px` and `left: 0px` styles of the child, combined with the child being absolutely positioned. In the normal flow of things, elements wouldn't be on top of (and covering up) other elements.

But since the child is absolute, it's essentially on a different layer than the normal elements. So it can be positioned on top of what else is on the webpage.

But it will stay within the boundaries of the parent element- as long as the parent has its position set. Which leads us to our next point.

There is one other tricky aspect to child elements with absolute positioning...

An absolutely positioned element needs to position itself in relation to a positioned ancestor.

When you take an element out of the normal flow by using `position: absolute`, it will look for an ancestor element that has its own position set. This is so the child knows what element it should position itself in relation to.

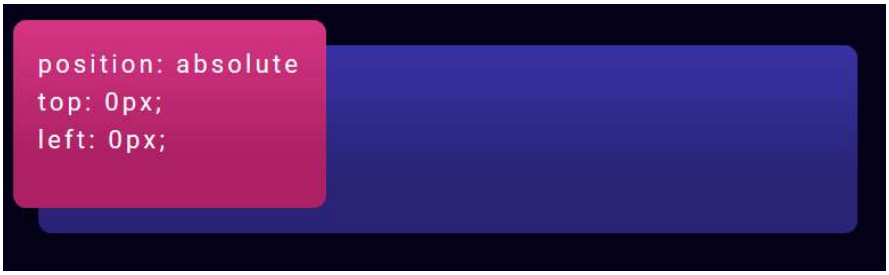
So what happens if a child element is absolutely positioned, but the parent element doesn't have a position set?

Here's our CSS for this illustration:

```
.parent {

 // No position set, so it's static

}

.child {
```

```
 position: absolute;

 top: 0px;

 left: 0px;

}
```

And here's what the resulting webpage would look like:



The child has now escaped the confines of the parent element, since the parent has no position set. And the child has gone up to the next (grand)parent element, in this case the `<body>` element, which is as far as it can go.

(A somewhat sad metaphor would be that this “orphaned” child is looking up the ancestry tree for someone that will be its “parent.”)

This is a huge cause of unexpected behavior in CSS for many developers.

If you can remember to always set the parent element of a child element to have `position` set to `relative` or `absolute` (in most cases), you will avoid having your child elements flying up the page to who knows where 😊

So, to summarize relative and absolute positioning:

The main difference between relative and absolute positioning is that `position: absolute` will take a child element completely out of the normal flow of the document. And that child will be positioned in relation to the first parent element that has its own position set.

The last two `position` values, `fixed` and `sticky`, are similar in some ways to `position: absolute`. But they also are related to your scroll position on the page.

Let's take a look:

4. Fixed

`position: fixed` will take the element out of the normal flow, and also position it in the same place in the viewport (what's visible on screen). This means that scrolling will not affect its position at all.

Let's see what this looks like in the code. Here's our HTML:

```
<div class="first purple">

 Lorem ipsum dolor sit amet, consectetur adipiscing elit....

</div>
```

```
<div class="another green"></div>
```

And in our CSS, we've set the second element to be `position: fixed`:

```
.first {

 position: relative;

}

.another {

 position: fixed;

 top: 0px;

 left: 0px;

}
```

Here's a Codepen to illustrate that:

The green fixed element will stay positioned to the top and left corner of the viewport. And if you scroll, the purple element will scroll up normally, but the green element will remain stuck to where we positioned it.

1. Static

`position: static` is the default value that an element will have. This means if you don't declare `position` for an element in CSS, it will automatically be set to `static`.

It's important to note that having a static position is the same as not setting the `position` property at all. (This will come into play a bit later on with absolute positioning.)

Elements that are statically positioned will appear on the page in what we call the normal flow. For example, if you have multiple `<div>`

elements one after the other, they will appear on the page directly below one another.

Here's a quick demo to illustrate static position. We are using the following HTML markup:

```
<div class="first purple"> position: static </div>
```

```
<div class="another green"> position: static<br> top: 50px (this has no effect) </div>
```

And here's the CSS we're using:

```
.first {
```

```
   // No position set, so it's static
```

```
}
```

```
.another {
```

```
 // No position set, so it's static top: 50px;


}
```

The second element has a `top` property set to `50px`. You would think that this would move it down 50px, right?

However, here is how it will look on a webpage:

Since both elements have a static position, none of the layout CSS properties will do anything. This makes that `top` property have no effect on how the second element is displayed.

So that second element ends up being directly below the first element, with no space between.

How can we fix this? Let's move on to the next position:

2. Relative

`position: relative` is similar to `static` in that relatively positioned elements will follow the normal flow of the webpage. But the main difference is that using `relative` will now unlock the other CSS layout properties.

Think about it this way: you are setting the element to be positioned relative to other elements on the page.

Let’s see what this looks like, and adjust our CSS like this:

```
.first {

 position: static;

}

.another {

 position: relative;

 top: 50px;

}
```

All the CSS is exactly the same, except that we changed the second element to use `position: relative`. Doing this makes that `top: 50px` work!

You can see that the second element is now 50px below the first one, adding that space between them.

Relatively positioned parent and child elements

Let's try another example, using a parent element with a child element nested inside it. Both have `position: relative` set.

Here's the HTML for that:

```
<div class="parent purple">

 <div class="child magenta"></div>

</div>
```

And our CSS styles:

```
.parent {

 position: relative;

}

.child {

 position: relative;

 top: 0px;

 left: 0px;
```

```
}
```

And here's what that code will look like in real life:

You can see that the pink child element is nicely nested inside the purple parent element. The child is also positioned as close to the top and left inside the parent as is possible. (It will go as far up as the parent text)

Position relative is relatively straight-forward, right? Well hold on to your hats, because things are about to get crazy with `position absolute`.

3. Absolute

`position: absolute` will cause an element to be taken out of that normal flow of the webpage.

Wait, what does that mean?

So before, using static or relative positioning, elements would be nicely displayed one below the other, depending on their order in the HTML markup. But with absolute positioning, the element is completely taken out of that entire flow.

To help explain, let's do a comparison to illustrate the difference between relative and absolute positioning.

In the previous example, we had a parent element with a child element, both positioned relatively. And the child was nested inside the parent element.

Let's change that child to be positioned absolutely in the parent!