What You May Not Know About the Z-Index Property

By **Steven Bradley**, 9 Dec 2013

The `z-index` property in CSS seems simple enough, but there's a lot to discover beneath the surface if you really want to understand how it works. In this tutorial we'll clarify the inner workings of z-index, by looking at **stacking contexts** and a few practical examples.

CSS provides three different positioning schemes for the layout of boxes:

- normal document flow
- floats
- absolute positioning

The last completely removes an element from the normal flow and relies on the developer to tell the element where to display.

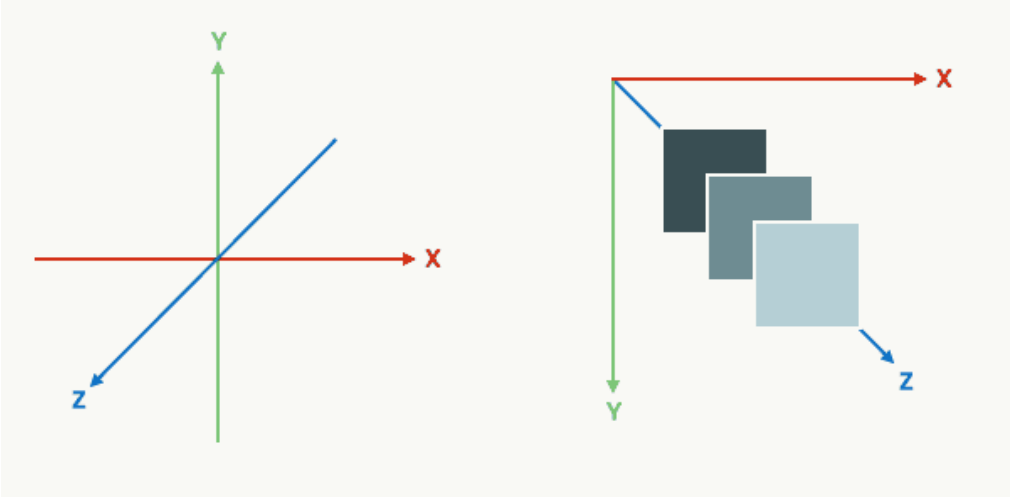
You give it top, left, bottom, and right values to position the element in two-dimensional space, but CSS also allows you to place it in the third dimension using the z-index property.

On the surface, z-index seems like an easy property to use. You set values to determine where on this third axis the element will be located, and you're done. Below the surface there's plenty more to dive into, including a number of rules for which types of elements sit on top of others.

Let's start with the basics to make sure we're all on the same page and then we'll talk about stacking to understand more of what's going on with z-index behind the scenes.

Z-Index Basics

I'm sure you're familiar with three-dimensional coordinate space. We have an x-axis which is typically used to represent the horizontal, a y-axis to represent the vertical, and a z-axis used to represent what happens into and out of the page, or the screen in our case.



3-Dimensional Coordinate Space

We don't literally *see* the z-axis, as the screen is a two-dimensional plane. We see it in the form of perspective and of some elements appearing in front of or behind other elements when they share the same two-dimensional space.

To determine where along this third axis an element is located, CSS allows us to set three values on the z-index property.

- auto (the default)
- (integer)
- inherit

For the moment let's focus on the integer value. This can be positive, negative, or 0. The greater the value, the closer to the viewer the element appears. The lower the value, the further away it appears.

If two elements are positioned so they both occupy a shared area of two-dimensional space, the element with the greater z-index will obscure or occlude the element with the lower z-index in the areas where they share the space.

I'm assuming the above is easy enough to understand and most likely

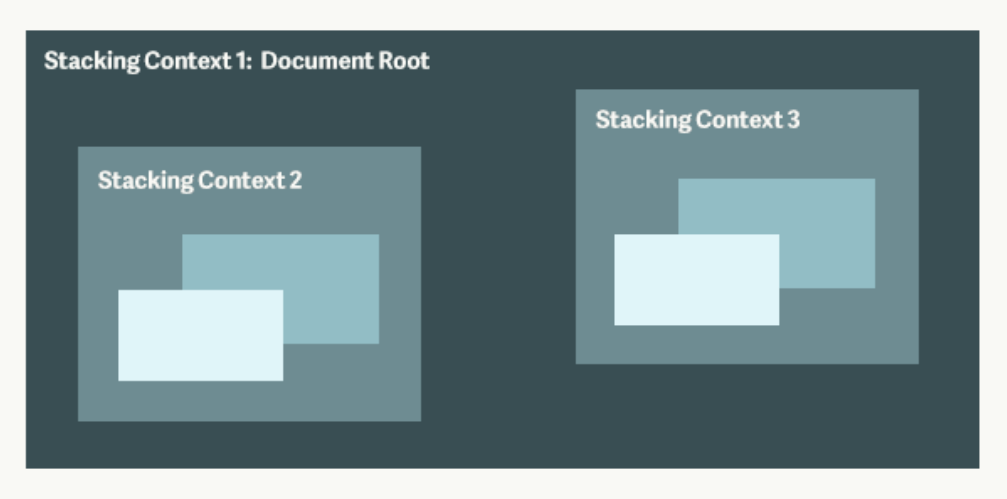
behaves exactly as you expect. However, there are some unanswered questions hanging around.

- Which appears on top when a positioned element with a z-index overlaps an element in the normal document flow?
- Which appears on top when one element is positioned and one is floated?
- What happens when positioned elements are nested inside other positioned elements?

To answer these questions we need to understand more about how z-index works, specifically the idea of stacking contexts, stacking levels, and stacking orders.

Stacking Contexts and Stacking Levels

Stacking contexts and stacking levels can be a little difficult to conceptualize, so for a moment, picture a table with a bunch of items sitting on top of it. The table represents a stacking context. If there's a second table next to the first, that represents another stacking context.



Stacking context 1 is formed by the root of the document. Both stacking contexts 2 and 3 are stacking levels on stacking context 1. Each also forms a new stacking context with new stacking levels inside.

Now imagine on the first table there are four small blocks, all directly sitting on the table. On top of these four blocks is a plate of glass and on top of the plate of glass is a bowl of fruit. The blocks, glass plate, and fruit bowl each represent a different stacking level within the stacking context that is the table.

A single default stacking context is created for every web page. The root of this context (the table) is the `html` element. Everything inside the html tag sits on a stacking level within this default stacking context (objects sitting on the table).

When you assign a z-index value other than auto to an element, you create a new stacking context with new stacking levels that are independent of other stacking contexts and stacking levels on the page. You bring another table into the room for objects to sit on.

Stacking Order

The easiest way to understand stacking order is with a simple example, so simple we won't even consider positioned elements for a moment.

Think of a very simple web page. Other than the default `<html>`, `<head>`, `<body>`, etc. you'll find on every web page there's a single `<div>`. In your CSS file you set the background color of the html element to blue. On the div, you set values for width and height and a background color of red.

What do you expect to see when you load the page?

Hopefully it didn't take you long to picture a mostly blue screen with the exception of a block of red color that has the dimensions you set for the width and height of the div. The red block is probably in the upper left of the page unless you let your imagination run wild a bit and gave it more css to display it somewhere else.

- HTML
- CSS
-
- Result

Edit on

```
<html lang="en">
<head>
        <meta charset="UTF-8" />
        <title>z-index demo</title>
</head>
```

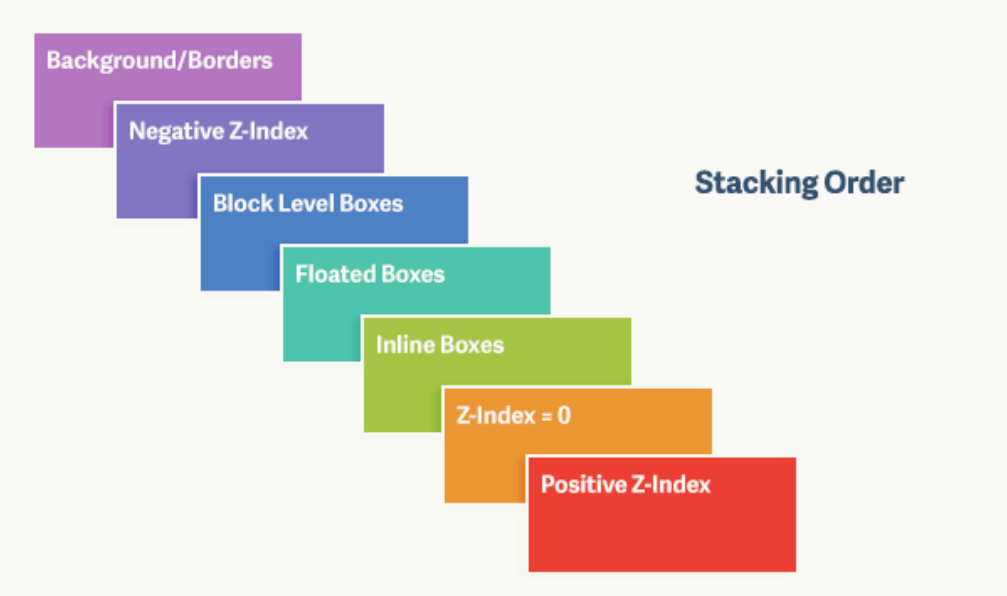
You might be thinking "so what, that was pretty obvious", but what might not be so obvious is why you see a red block on top of a blue background.

Why do you see the div on top of the html element? The reason is because both are following stacking order rules.

In the case of this simple example, the rules say that descendent blocks in the normal flow (the div) sit on a higher level than the backgrounds and borders of the root element (the html element). You see the div on top because it's on a higher stacking level.

While the example above only contains a two level stack, there are **seven possible levels** on each stacking context, which are listed below.

1. **Background and borders** — of the element forming the stacking context. The lowest level in the stack.
2. **Negative Z-Index** — the stacking contexts of descendants elements with negative z-index.
3. **Block Level Boxes** — in-flow non-inline-level non-positioned descendants.
4. **Floated Boxes** — non-positioned floats
5. **Inline Boxes** — in-flow inline-level non-positioned descendants.
6. **Z-index: 0** — positioned elements. These form new stacking contexts.
7. **Positive Z-index** — positioned elements. The highest level in the stack.



The seven levels on a stacking context

These seven levels form the stacking order rules. An element on level seven will display above (nearer the viewer) than elements on levels one to

six. An element on level five displays above an element on level two. An element on…well you get the idea.

The first time I encountered the stacking order rules above, a few things stood out to me. If you only look at levels two, six, and seven (the ones where z-index is mentioned), it fits what you likely assume about z-index. Positive z-index is at a higher level than 0 z-index, which is at a higher level than negative z-index. That's probably where most of us stop thinking about all these stacking layers, though.

Prior to these rules I would have assumed everything else was equivalent to a z-index of 0. Clearly that isn't so. In fact most everything sits at a level below the z-index = 0 level.

What's also interesting is that non-positioned elements are located at four different stacking levels. It makes sense when you think about it. If all non-positioned elements were at the same stacking level, we wouldn't see text (inline box) on top of a div (block level box).

Putting it all Together

A couple of times I've mentioned creating new stacking contexts. When you assign a z-index value other than `auto` to an element you create a new stacking context, independent of other stacking contexts.

Let's think again of tables as stacking contexts. Before, we had a table and on top of the table were four blocks, a plate of glass, and a fruit bowl. Imagine the second table we introduced also has four blocks of the same size and a plate of glass on top, but no fruit bowl.

You'd expect the fruit bowl from table one to be the highest item in the room. It sits at the highest level (it has the greatest z-index). What would happen though if we moved table one and everything on top of it to the basement? That fruit bowl would now be lower than everything on table two, because table one itself has been moved to a level below table two.

The same thing happens with positioned elements on a web page. Consider the markup and styling below. Will `div.two` display above or below `div.four`?

HTML:

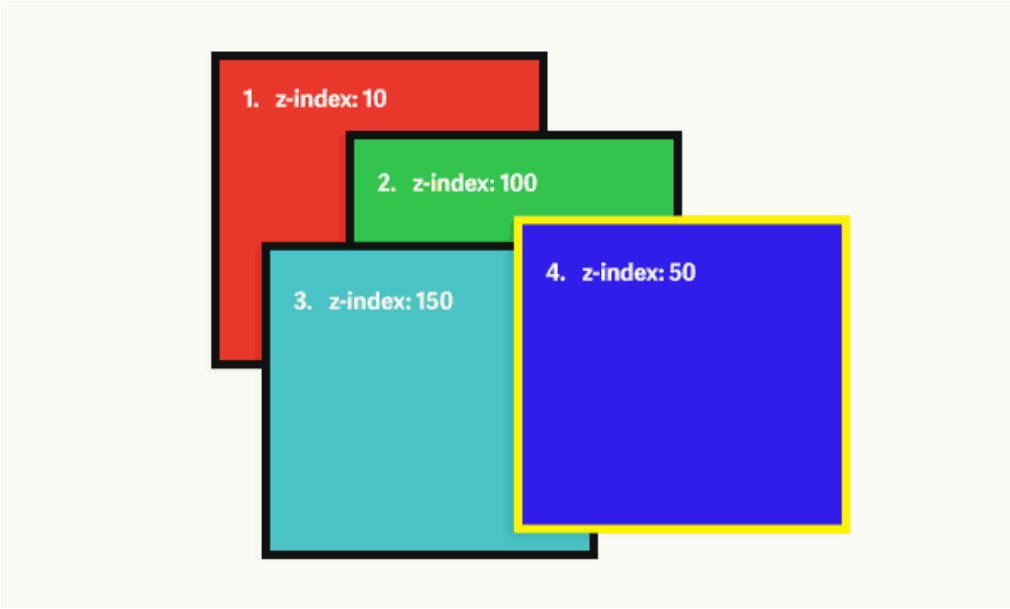
```
<div class="one">
  <div class="two"></div>
  <div class="three"></div>
</div>
<div class="four"></div>
```

CSS:

```
div {
  width: 200px;
  height: 200px;
  padding: 20px;
}

.one, .two, .three, .four {
  position: absolute;
}

.one {
  background: #f00;
  outline: 5px solid #000;
  top: 100px;
  left: 200px;
  z-index: 10;
}

.two {
  background: #0f0;
  outline: 5px solid #000;
  top: 50px;
  left: 75px;
  z-index: 100;
}

.three {
  background: #0ff;
  outline: 5px solid #000;
  top: 125px;
  left: 25px;
  z-index: 150;
}

.four {
  background: #00f;
  outline: 5px solid #ff0;
  top: 200px;
  left: 350px;
  z-index: 50;
}
```

Even though div.two has the greater z-index (100), it actually sits below div.four (z-index = 50) on the page. You can see the result of the code

above in the image below. The black and yellow borders show the different stacking contexts each element is in.



Since div.two is contained within div.one, its z-index is relative to `div.one`'s stack. In effect what we really have is the following:

- .one — z-index = 10
- .two — z-index = 10.100
- .three — z-index = 10.150
- .four — z-index = 50

What we've done is moved `div.one` and everything it contains below `div.four`. No matter what values we set on the z-index property of elements inside `div.one` they will always display below `div.four`.

If you're like me, this has probably tripped you up once or twice when working with z-index. Hopefully these examples help clear up why an element with a greater z-index sometimes ends up displaying behind another element with a lower z-index.

Conclusion

When you first encounter it, the z-index property seems like a very simple property to understand. Values represent a location on an axis into and out of the screen, and that's all.

A deeper look at z-index reveals there's a bit more going on behind the scenes. There are stacking contexts, stacking levels, and rules to determine which element displays higher or lower in the stacking order.

Positioned elements can create new stacking contexts, and entire stacking levels will display above or below all the levels in another stacking context.

Further Reading

- What No One Told You About Z-Index by Philip Walton
- The Z-Index CSS Property: A Comprehensive Look on Smashing Magazine
- How Well Do You Understand CSS Positioning?
- The stacking context on Mozilla Developer Network
- Z-Index And The CSS Stack: Which Element Displays First?

Article Details
Tags: HTML/CSS, Front End

About the Author

Steven Bradley is a web designer, WordPress developer, and author of the book Design Fundamentals. He more…

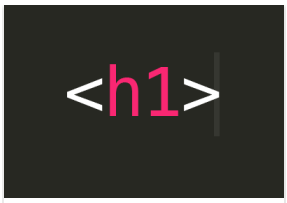
Related Posts

Development
Finishing Off the Merry Christmas Web App Interface

5 days ago

Development
Building the Merry Christmas Web App Interface

19 days ago



Development
The Truth About Multiple H1 Tags in the HTML5 Era

25 days ago

Development
Uber Aesthetics and Responsiveness

24 Oct 2013

25 comments

★ 5

Join the discussion…

Best Community Share Login

Zohaib Hassan • a month ago
Wow.. I never thought about it so deep. :) Thanks for such informative article...
9 • Reply • Share ›

Steven Bradley → Zohaib Hassan • a month ago
Years ago I ran into a problem where IE, maybe version 6, was doing something weird and showing the element with the higher z-index on top of another element with a lower z-index. I started learning a little more about stacking contexts then, though I can't say I really looked into it in-depth until writing an article about all this stuff a couple of years ago.
3 • Reply • Share ›

Omar Zeidan • 20 days ago
Awesome, I enjoyed this article!

Also consider this one, if we have 2 block elements .box1 and .box2 and we moved .box2 on the top of .box1 using the margin property, the .box2 will appear on the top, BUT the text of .box1 will appear also on the top of .box2.

Please find the image below!

• Reply • Share ›

Daniel Furze • 24 days ago
Thanks, I enjoyed this article! :)
• Reply • Share ›

levani01 • 25 days ago
Is there any difference in terms of rendering performance when creating many stacking contexts (e.i. many absolutely positioned elements)?
• Reply • Share ›

Sasha Zinevych • 25 days ago
This is plain awesome! Want to apply it right now! :-)
• Reply • Share ›

alex89x • 25 days ago
Very informative. Even if someone didn't read the whole article (which is very well-written, too!), just knowing that behind the scenes there is much more than the z-index value will save tons of hours trying to understand why the positioning doesn't work as expected!

Development **Back to Top**

Categories

Web Dev	Flash	Mobile Dev	
WordPress	News	Front End	
JavaScript & AJAX	Tutorials	PHP	
iOS SDK	Android SDK	ActionScript	
HTML & CSS	Theme Development	Browse All...	

Software & Tools

PHP
WordPress
JavaScript
iOS SDK
Browse All...

Teaching skills to millions worldwide.

Design & Illustration **Development** **Music & Audio** **Photography**