Positioning

Previous | Overview: CSS layout | Next

Positioning allows you to take elements out of the normal document layout flow, and make them behave differently; for example sitting on top of one another, or always remaining in the same place inside the browser viewport. This article explains the different `position` values, and how to use them.

Prerequisites:	HTML basics (study Introduction to HTML), and an idea of How CSS works (study Introduction to CSS.)
Objective:	To learn how CSS positioning works.

We'd like you to follow along with the exercises on your local computer, if possible — grab a copy of `0_basic-flow.html` from our Github repo (source code here) and use that as a starting point.

Introducing positioning

The whole idea of positioning is to allow us to override the basic document flow behavior described above, to produce interesting effects. What if you want to slightly alter the position of some boxes inside a layout from their default layout flow position, to give a slightly quirky, distressed feel? Positioning is your tool. Or if you want to create a UI element that floats over the top of other parts of the page, and/or always sits in the same place inside the browser window no matter how much the page is scrolled? Positioning makes such layout work possible.

There are a number of different types of positioning that you can put into effect on HTML elements. To make a specific type of positioning active on an element, we use the `position` property.

Static positioning

Static positioning is the default that every element gets — it just means "put the element into its normal position in the document layout flow — nothing special to see here."

To demonstrate this, and get your example set up for future sections, first add a `class` of `positioned` to the second `<p>` in the HTML:

```
<p class="positioned"> ... </p>
```

Now add the following rule to the bottom of your CSS:

```
.positioned {
  position: static;
  background: yellow;
}
```

If you now save and refresh, you'll see no difference at all, except for the updated background color of the 2nd paragraph. This is fine — as we said before, static positioning is the default behavior!

> **Note**: You can see the example at this point live at `1_static-positioning.html` (see source code).

Relative positioning

Relative positioning is the first position type we'll take a look at. This is very similar to static positioning, except that once the positioned element has taken its place in the normal layout flow, you can then modify its final position, including making it overlap other elements on the page. Go ahead and update the `position` declaration in your code:

```
position: relative;
```

If you save and refresh at this stage, you won't see a change in the result at all. So how do you modify the element's position? You need to use the `top`, `bottom`, `left`, and `right` properties, which we'll explain in the next section.

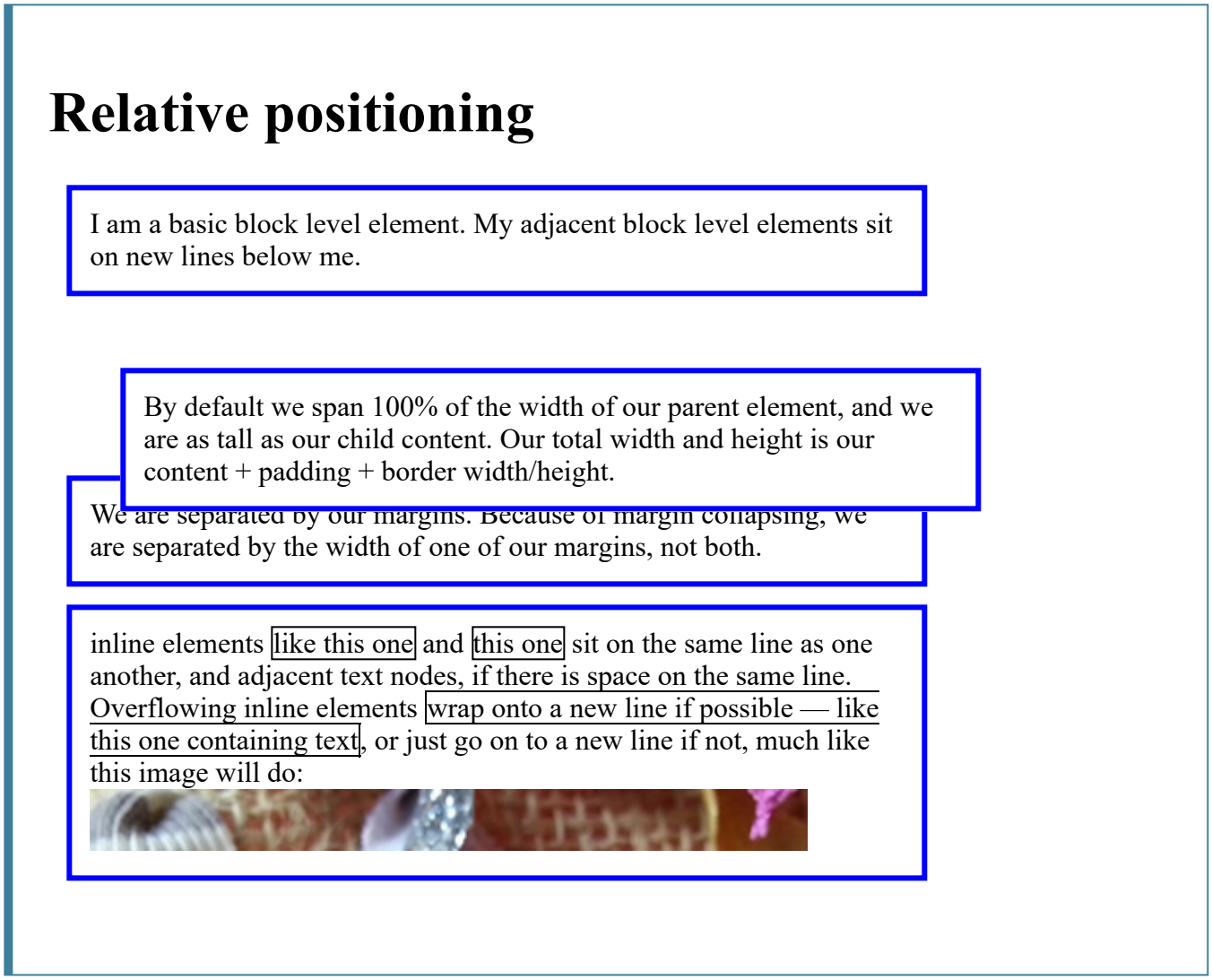
Introducing top, bottom, left, and right

`top`, `bottom`, `left`, and `right` are used alongside `position` to specify exactly where to move the positioned element to. To try this out, add the following declarations to the `.positioned` rule in your CSS:

```
top: 30px;
left: 30px;
```

> **Note**: The values of these properties can take any units you'd logically expect — pixels, mm, rems, %, etc.

If you now save and refresh, you'll get a result something like this:



Cool, huh? Ok, so this probably wasn't what you were expecting — why has it moved to the bottom and right if we specified top and left? Illogical as it may initially sound, this is just the way that relative positioning works — you need to think of an invisible force that pushes the specified side of the positioned box, moving it in the opposite direction. So for example, if you specify `top: 30px;`, a force pushes the top of the box, causing it to move downwards by 30px.

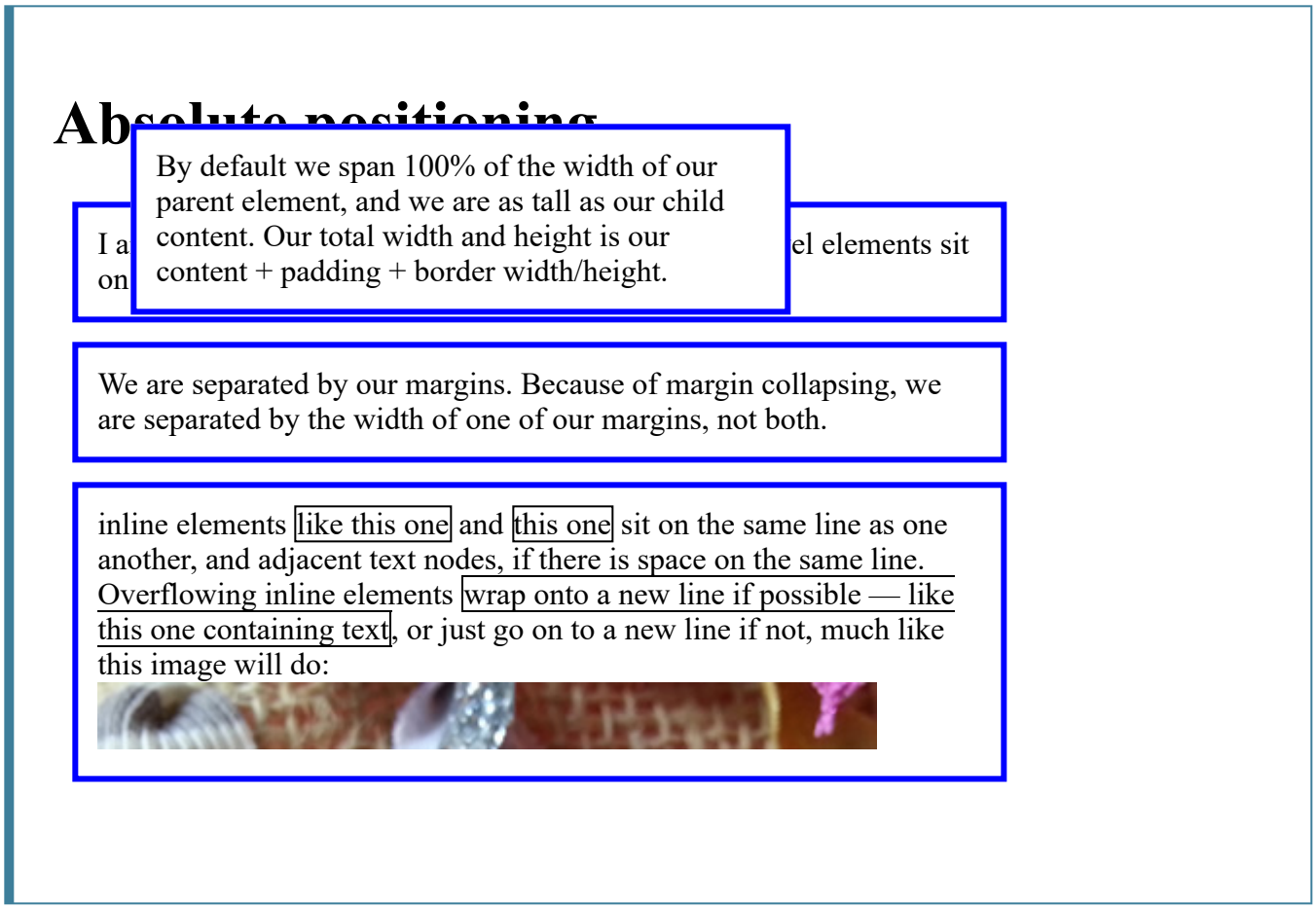
> **Note**: You can see the example at this point live at `2_relative-positioning.html` (see source code).

Absolute positioning

Absolute positioning brings very different results. Let's try changing the position declaration in your code as follows:

```
position: absolute;
```

If you now save and refresh, you should see something like so:



First of all, note that the gap where the positioned element should be in the document flow is no longer there — the first and third elements have closed together like it no longer exists! Well, in a way, this is true. An absolutely positioned element no longer exists in the normal document layout flow. Instead, it sits on its own layer separate from everything else. This is very useful: it means that we can create isolated UI features that don't interfere with the position of other elements on the page. For example, popup information boxes and control menus; rollover panels; UI features that can be dragged and dropped anywhere on the page; and so on...

Second, notice that the position of the element has changed — this is because `top`, `bottom`, `left`, and `right` behave in a different way with absolute positioning. Instead of specifying the direction the element should move in, they specify the distance the element should be from each containing element's sides. So in this case, we are saying that the absolutely positioned element should sit 30px from the top of the "containing element", and 30px from the left.

> **Note**: You can use `top`, `bottom`, `left`, and `right` to resize elements if you need to. Try setting `top: 0; bottom: 0; left: 0; right: 0;` and `margin: 0;` on your positioned elements and see what happens! Put it back again afterwards...

> **Note**: Yes, margins still affect positioned elements. Margin collapsing doesn't, however.

> **Note**: You can see the example at this point live at `3_absolute-positioning.html` (see source code).

Positioning contexts

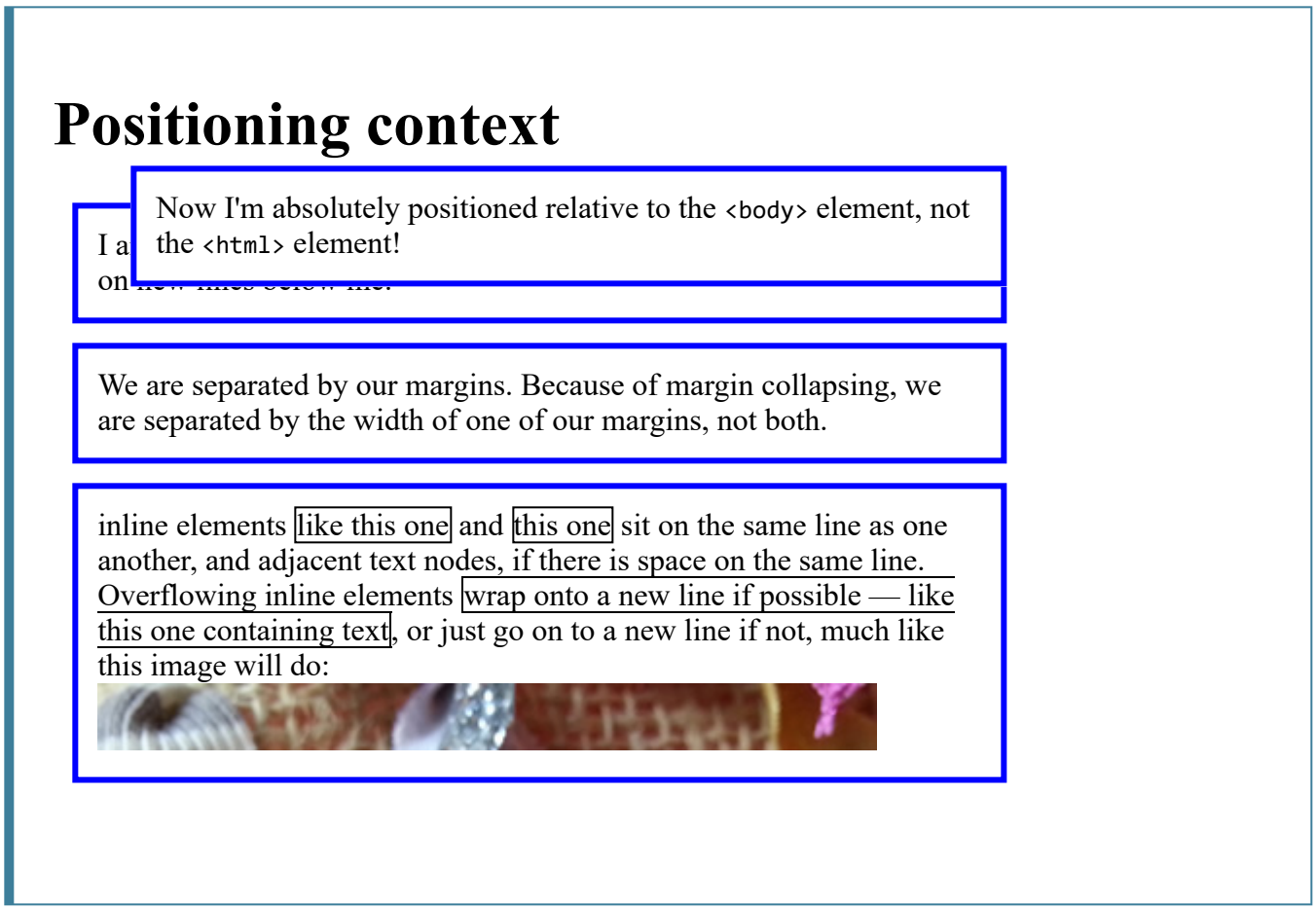
Which element is the "containing element" of an absolutely positioned element? This is very much dependent on the position property of the ancestors of the positioned element (See Identifying the containing block).

If no ancestor elements have their position property explicitly defined, then by default all ancestor elements will have a static position. The result of this is, the absolutely positioned element will be contained in the **initial containing block**. The initial containing block has the dimensions of the viewport, and is also the block that contains the `<html>` element. Simply put, the absolutely positioned element will be contained outside of the `<html>` element, and be positioned relative to the initial viewport.

The positioned element is nested inside the `<body>` in the HTML source, but in the final layout, it is 30px away from the top and left of the edge of the page. We can change the **positioning context** — which element the absolutely positioned element is positioned relative to. This is done by setting positioning on one of the element's ancestors — to one of the elements it is nested inside (you can't position it relative to an element it is not nested inside). To demonstrate this, add the following declaration to your `body` rule:

```
position: relative;
```

This should give the following result:



The positioned element now sits relative to the `<body>` element.

> **Note**: You can see the example at this point live at `4_positioning-context.html` (see source code).

Introducing z-index

All this absolute positioning is good fun, but there is another thing we haven't considered yet — when elements start to overlap, what determines which elements appear on top of which other elements? In the example we've seen so far, we only have one positioned element in the positioning context, and it appears on the top, since positioned elements win over non-positioned elements. What about when we have more than one?

Try adding the following to your CSS, to make the first paragraph absolutely positioned too:

```
p:nth-of-type(1) {
  position: absolute;
  background: lime;
```

```
    top: 10px;
    right: 30px;
}
```

At this point you'll see the first paragraph colored green, moved out of the document flow, and positioned a bit above from where it originally was. It is also stacked below the original `.positioned` paragraph, where the two overlap. This is because the `.positioned` paragraph is the second paragraph in the source order, and positioned elements later in the source order win over positioned elements earlier in the source order.

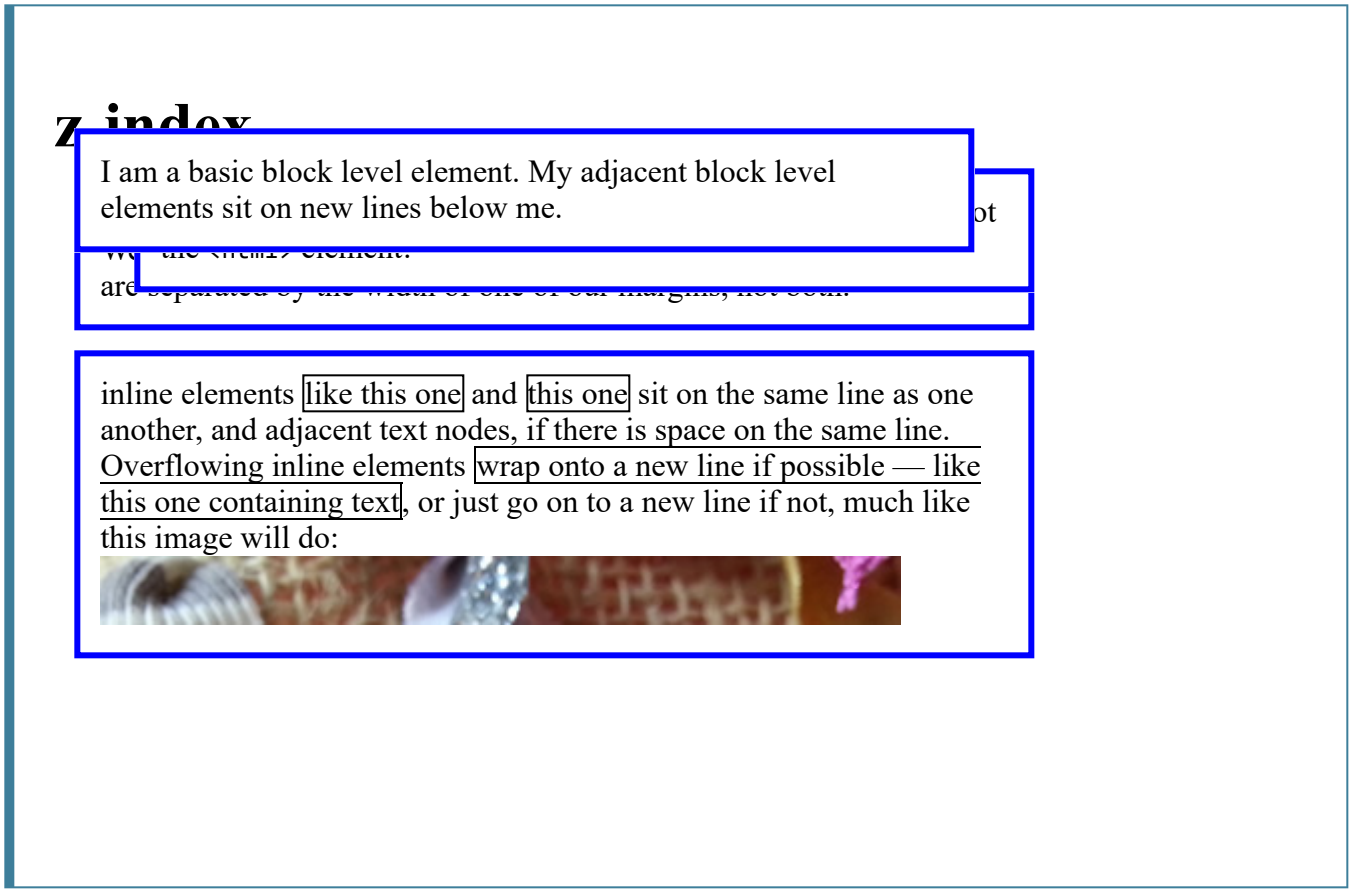
Can you change the stacking order? Yes, you can, by using the `z-index` property. "z-index" is a reference to the z-axis. You may recall from previous points in the course where we discussed web pages using horizontal (x-axis) and vertical (y-axis) coordinates to work out positioning for things like background images and drop shadow offsets. (0,0) is at the top left of the page (or element), and the x- and y-axes run across to the right and down the page (for left to right languages, anyway.)

Web pages also have a z-axis: an imaginary line that runs from the surface of your screen, towards your face (or whatever else you like to have in front of the screen). `z-index` values affect where positioned elements sit on that axis; positive values move them higher up the stack, and negative values move them lower down the stack. By default, positioned elements all have a `z-index` of `auto`, which is effectively 0.

To change the stacking order, try adding the following declaration to your `p:nth-of-type(1)` rule:

```
z-index: 1;
```

You should now see the finished example, with the green paragraph on top:



Note that `z-index` only accepts unitless index values; you can't specify that you want one element to be 23 pixels up the Z-axis — it doesn't work like that. Higher values will go above lower values, and it is up to you what values you use. Using 2 and 3 would give the same effect as 300 and 40000.

> **Note**: You can see the example at this point live at `5_z-index.html` (see source code).

Fixed positioning

Let's now look at fixed positioning. This works in exactly the same way as absolute positioning, with one key difference: whereas absolute positioning fixes an element in place relative to the `<html>` element or its nearest positioned ancestor, fixed positioning fixes an element in place relative to the browser viewport itself. This means that you can create useful UI items that are fixed in place, like persisting navigation menus.

Let's put together a simple example to show what we mean. First of all, delete the existing `p:nth-of-type(1)` and `.positioned` rules from your CSS.

Now, update the `body` rule to remove the `position: relative;` declaration and add a fixed height, like so:

```
body {
  width: 500px;
  height: 1400px;
  margin: 0 auto;
}
```

Now we're going to give the `<h1>` element `position: fixed;`, and get it to sit at the top center of the viewport. Add the following rule to your CSS:

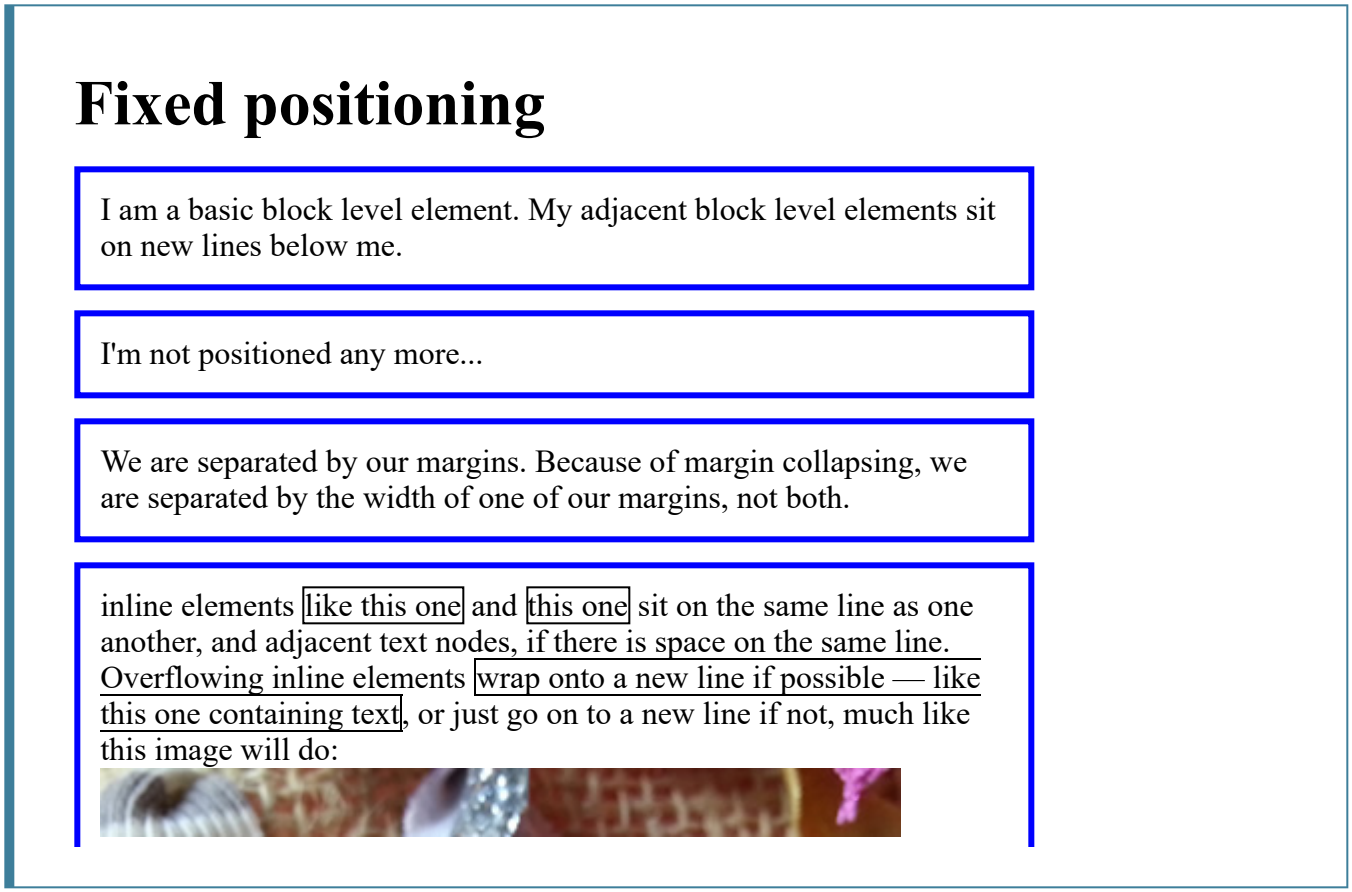
```
h1 {
  position: fixed;
  top: 0;
  width: 500px;
  margin: 0 auto;
  background: white;
  padding: 10px;
}
```

The `top: 0;` is required to make it stick to the top of the screen; we then give the heading the same width as the content column and use the faithful old `margin: 0 auto;` trick to center it. We then give it a white background and some padding, so the content won't be visible underneath it.

If you save and refresh now, you'll see a fun little effect whereby the heading stays fixed, and the content appears to scroll up and disappear underneath it. But we could improve this more — at the moment some of the content starts off underneath the heading. This is because the positioned heading no longer appears in the document flow, so the rest of the content moves up to the top. We need to move it all down a bit; we can do this by setting some top margin on the first paragraph. Add this now:

```
p:nth-of-type(1) {
  margin-top: 60px;
}
```

You should now see the finished example:



> **Note**: You can see the example at this point live at `6_fixed-positioning.html` (see source code).

position: sticky

There is another position value available called `position: sticky`, which is somewhat newer than the others. This is basically a hybrid between relative and fixed position, which allows a positioned element to act like it is relatively positioned until it is scrolled to a certain threshold point (e.g. 10px from the top of the viewport), after which it becomes fixed. This can be used to for example cause a navigation bar to scroll with the page until a certain point, and then stick to the top of the page.

```
.positioned {
  position: sticky;
  top: 30px;
  left: 30px;
}
```

Sticky positioning

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla luctus aliquam dolor, eu lacinia lorem placerat vulputate. Duis felis orci, pulvinar id metus ut, rutrum luctus orci. Cras porttitor imperdiet nunc, at ultricies tellus laoreet sit amet. Sed auctor cursus massa at porta. Integer ligula ipsum, tristique sit amet orci vel, viverra egestas ligula. Curabitur vehicula tellus neque, ac ornare ex malesuada et. In vitae convallis lacus. Aliquam erat volutpat. Suspendisse ac imperdiet turpis. Aenean finibus sollicitudin eros pharetra

An interesting and common use of `position: sticky` is to create a scrolling index page where different headings stick to the top of the page as they reach it. The markup for such an example might look like so:

```
<h1>Sticky positioning</h1>

<dl>
    <dt>A</dt>
    <dd>Apple</dd>
    <dd>Ant</dd>
    <dd>Altimeter</dd>
    <dd>Airplane</dd>
    <dt>B</dt>
    <dd>Bird</dd>
    <dd>Buzzard</dd>
    <dd>Bee</dd>
    <dd>Banana</dd>
    <dd>Beanstalk</dd>
    <dt>C</dt>
    <dd>Calculator</dd>
    <dd>Cane</dd>
    <dd>Camera</dd>
    <dd>Camel</dd>
    <dt>D</dt>
    <dd>Duck</dd>
    <dd>Dime</dd>
    <dd>Dipstick</dd>
```

```
    <dd>Drone</dd>
    <dt>E</dt>
    <dd>Egg</dd>
    <dd>Elephant</dd>
    <dd>Egret</dd>
</dl>
```

The CSS might look as follows. In normal flow the `<dt>` elements will scroll with the content. When we add `position: sticky` to the `<dt>` element, along with a `top` value of 0, supporting browsers will stick the headings to the top of the viewport as they reach that position. Each subsequent header will then replace the previous one as it scrolls up to that position.

```
dt {
  background-color: black;
  color: white;
  padding: 10px;
  position: sticky;
  top: 0;
  left: 0;
  margin: 1em 0;
}
```

Note: You can see this example live at `7_sticky-positioning.html` (see source code).

Summary

I'm sure you had fun playing with basic positioning; while it is not a method you would use for entire layouts, as you can see there are many tasks it is suited for.

Previous | Overview: CSS layout | Next

See also

- The `position` property reference.
- Practical positioning examples, for some more useful ideas.

In this module

- Introduction to CSS layout
- Normal Flow
- Flexbox
- Grid
- Floats
- Positioning
- Multiple-column Layout
- Legacy Layout Methods
- Supporting older browsers
- Fundamental Layout Comprehension Assessment