Floating and Positioning Elements

> *The float property is a valuable and powerful asset to any web designer/developer working with HTML and CSS.* —Noah Stokes

This chapter covers

- Learning how elements flow down the page
- Interrupting the normal flow by floating elements
- Using floats to create drop caps and pull quotes
- Interrupting the normal flow by positioning elements

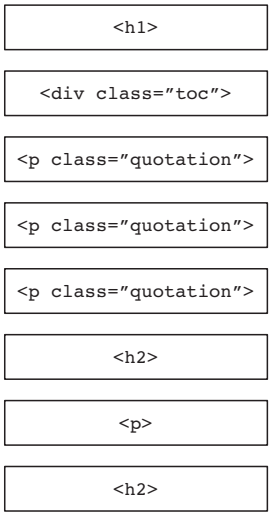
Left to its own devices, the web browser imposes an inflexible structure on your web pages, and your site is in danger of becoming boring (at least from a design perspective). To avoid that fate, you need to take control of your page elements and free them from the web browser's fixed ideas about how things should be laid out. You do that by wielding two of the most powerful CSS tools in the web designer's arsenal: floating and positioning. With these tools, you can break out of the browser's default element flow and build interesting, creative pages that people will be itching to visit. This chapter tells you everything you need to know.

Understanding the Default Page Flow

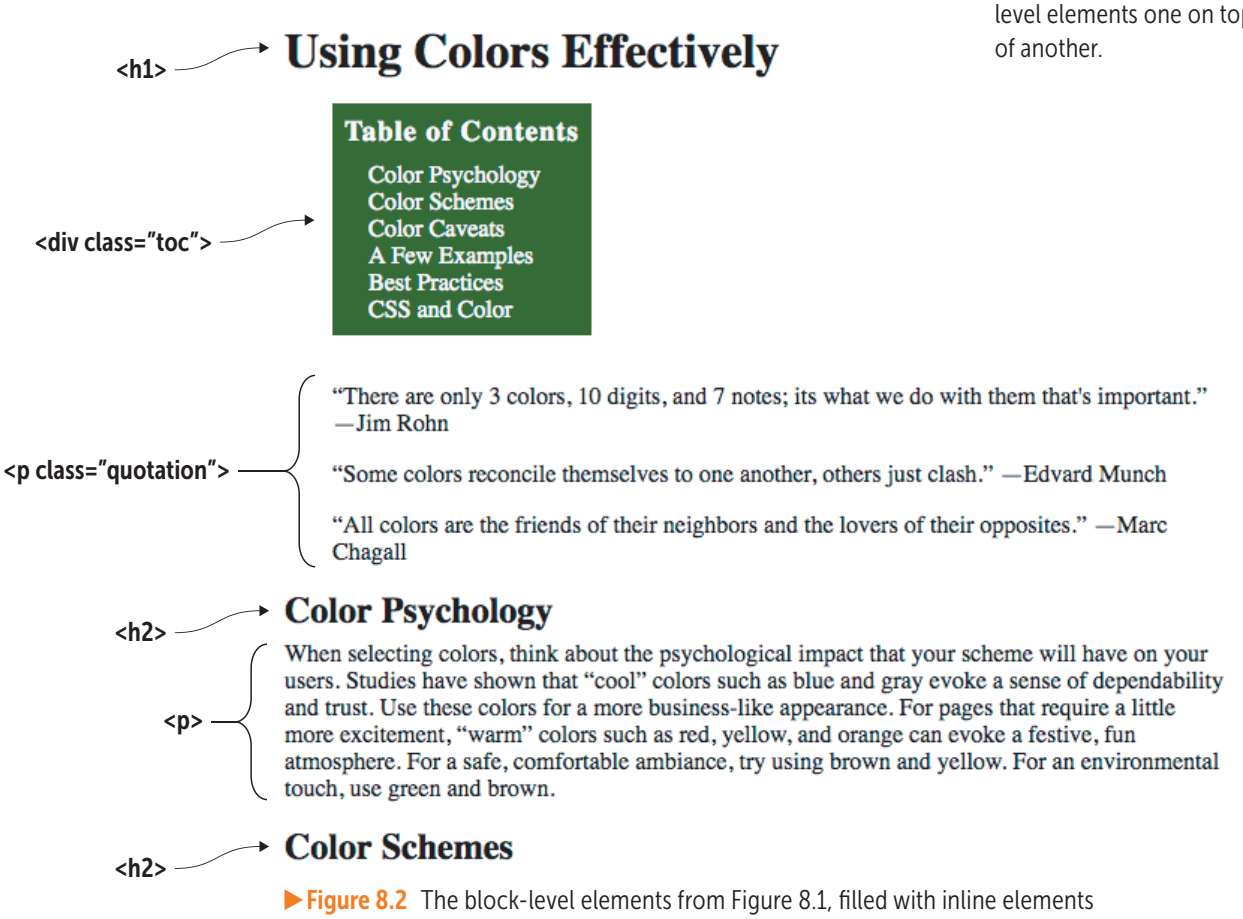
When you add elements to a web page, the browser lays out those elements in the order in which they appear in the HTML file according to the following rules:

- Block-level elements are stacked vertically, with the first element on top, the second element below it, and so on.
- Each inline element is rendered from left to right (in English and other left-to-right languages) within its parent block element.

Figure 8.1 shows a schematic diagram of a few block-level elements, stacked as the browser would render them. Figure 8.2 shows the corresponding web page with inline elements added.



▶ Figure 8.1
The browser stacks block-level elements one on top of another.



▶ Figure 8.2 The block-level elements from Figure 8.1, filled with inline elements

Lesson 8.1: *Floating Elements*

Covers: The `float` property

Online: wdpg.io/8-1-0

You can interrupt the top-to-bottom flow of elements by *floating* one or more elements to the left or right. *Floating* means that the browser takes the element out of the usual flow and places it as far as possible to the left or to the right (depending on the value you provide) and as high as possible (depending on other content) in its parent element. Then the rest of the page content flows around the floated element.

You float an element by setting its `float` property:

```
element {
    float: left|right|none;
}
```

In Figure 8.2, for example, the page would look nicer and make better use of space if the table of contents could be pushed to the right with the quotations flowing around it. That's readily done with the `float` property, as shown in the following example.

MASTER

Because the nearby nonfloated page elements wrap around the floated element, you should ensure that adequate whitespace exists between them by adding a margin around the floated element.

REMEMBER

Unlike with a nonfloated element, the top and bottom margins of a floated element do not collapse. See Chapter 9 to learn more about collapsing margins.

▶ *Example*

Online: wdpg.io/8-1-1

This example uses the `float` property to float the table of contents to the right.

WEB PAGE

Using Colors Effectively

"There are only 3 colors, 10 digits, and 7 notes; its what we do with them that's important." —Jim Rohn

"Some colors reconcile themselves to one another, others just clash." —Edvard Munch

"All colors are the friends of their neighbors and the lovers of their opposites." —Marc Chagall

Table of Contents

- Color Psychology
- Color Schemes
- Color Caveats
- A Few Examples
- Best Practices
- CSS and Color
- Color Resources

Color Psychology

When selecting colors, think about the psychological impact that your scheme will have on your users. Studies have shown that "cool" colors such as blue and gray evoke a sense of dependability and trust. Use these colors for a more business-like appearance. For pages that require a little more excitement, "warm" colors such as red, yellow, and orange can evoke a festive, fun atmosphere. For a safe, comfortable ambiance, try using brown and yellow. For an environmental touch, use green and brown.

Color Schemes

continued

CSS

```
.toc {
    float: right;
    margin-left: 2em;
    margin-bottom: 2em;
    etc.
}
```

The float property applied to the toc class

HTML

```
<h1>Using Colors Effectively</h1>
<div class="toc">
    <h3>Table of Contents</h3>
    <div>Color Psychology</div>
    <div>Color Schemes</div>
    <div>Color Caveats</div>
    <div>A Few Examples</div>
    <div>Best Practices</div>
    <div>CSS and Color</div>
    <div>Color Resources</div>
</div>
<p class="quotation">
"There are only 3 colors, 10 digits, and 7 notes; its what we do
with them that's important." —Jim Rohn
</p>
etc.
```

This tag uses the toc class.

BEWARE

If you float an inline element, be sure to give it a width so that the browser knows how much space to give the element.

FAQ

Can I float only block-level elements? *No, you can also apply the* `float` *property to an inline element, such as a* `span`*. When you do, however, the browser takes the element out of the normal flow, turns it into a block-level element, and then floats it.*

Clearing Floated Elements

In the preceding example, notice that not only do the three quotations wrap around the floated table of contents; so do the first `h2` element ("Color Psychology") and part of the paragraph that follows it. That behavior normally is what you want. But what if, for aesthetic or other reasons, you prefer that the `h2` element and its text do *not* wrap around the table of contents?

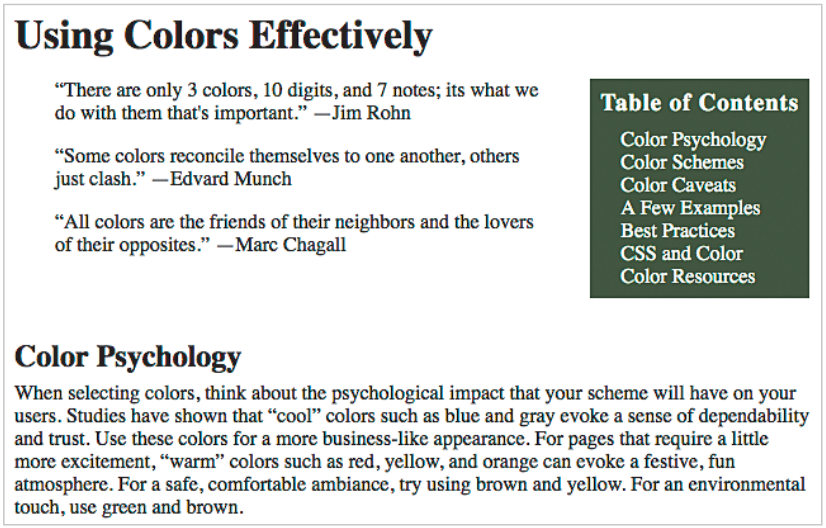
You can do that by telling the browser that you want the `h2` element to *clear* the floated element. *Clearing* a floated element means that the browser renders the element after the end of the floated element. You clear an element by setting its `clear` property:

```
element {
    clear: left|right|both|none;
}
```

You use `left` to clear element of any elements that have been floated left, `right` to clear element of any elements that have been floated right, or `both` to clear element of both left- and right-floated elements. To clear the `h2` element in the example, I'd use the following code:

```
h2 {
    clear: right;
}
```

Figure 8.3 shows the page with the `h2` (Color Psychology) now clearing the floated table of contents.



▶ Figure 8.3
The Color Psychology **`h2`** element now clears the floated table of contents.

Preventing Container Collapse

Floated elements have a few gotchas that you need to watch for. The biggest one is that under certain circumstances, a floated element will overflow or drop right out of its parent container. To see what I mean, take a look at the following code (see Figure 8.4), which has two `<p>` tags in a `<div>` container that has been styled with a light blue background and a red border:

PLAY
You can float multiple elements. ⇨ **Online:** wdpg.io/8-1-3

CSS:

```
div {
    border: 1px solid red;
    background-color: lightcyan;
}
```

HTML:

```
<div>
    <p>
        If you float two consecutive elements, the second floated
element will always appear either beside the first floated element
or below it.
    </p>
    <p>
        For example, if you float the elements left, the second
will appear to the right of the first. If there isn't enough room
to the right, it will appear below the first element.
    </p>
</div>
```

▶ **Figure 8.4**
Two `<p>` elements inside a `<div>` container

If you float two consecutive elements, the second floated element will always appear either beside the first floated element or below it.

For example, if you float the elements left, the second will appear to the right of the first. If there isn't enough room to the right, it will appear below the first element.

Figure 8.5 shows the result when I style the `<p>` tags with a width and float them to the left:

CSS:

```
.col {
    float: left;
    width: 300px;
}
```

HTML:

```
<p class="col">
```

▶ **Figure 8.5**
When I float the `<p>` elements, the `<div>` container collapses on itself.

The has collapsed.

If you float two consecutive elements, the second floated element will always appear either beside the first floated element or below it.

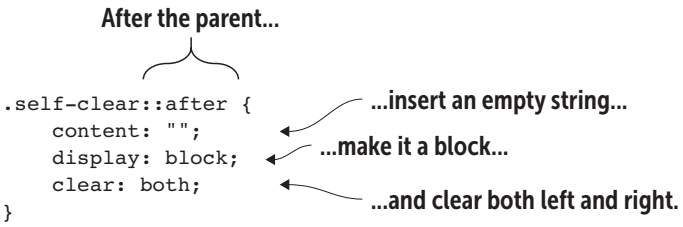
For example, if you float the elements left, the second will appear to the right of the first. If there isn't enough room to the right, it will appear below the first element.

MASTER
Some web developers prefer to use a more semantic name for the class, such as `group`.

REMEMBER
This solution is sometimes called a clearfix *hack.*

Bizarrely, the `<div>` container nearly disappears! That red line across the top is all that's left of it. What happened? When I floated the `<p>` elements, the browser took them out of the normal flow of the page. The `<div>` container saw that it no longer contained anything, so it collapsed on itself. This always occurs when a parent element contains only floated child elements.

To fix this problem, you can tell the parent element to clear its own child elements, thus preventing it from collapsing. Figure 8.6 shows a class that does this.



▶ **Figure 8.6**
A class that enables a parent element to clear its own child elements

This class tells the browser to insert an empty string, rendered as a block-level element, and have it clear both left- and right-floated elements. The following example shows the fix in action and the full code.

▶ *Example*

⇨ **Online:** wdpg.io/8-1-5

This example fixes the collapsing parent problem by telling the parent to self-clear its own floated child elements.

WEB PAGE

If you float two consecutive elements, the second floated element will always appear either beside the first floated element or below it.

For example, if you float the elements left, the second will appear to the right of the first. If there isn't enough room to the right, it will appear below the first element.

CSS

```
div {
    border: 1px solid red;
    background-color: lightcyan;
    width: 675px;
}
.col {
    float: left;
    width: 300px;
}
.self-clear::after {
    content: "";
    display: block;
    clear: both;
}
```

This rule styles the div element.

This class adds a width and floats the element.

This class prevents the parent from collapsing.

continued

HTML

```
<div class="self-clear">
<p class="col">
If you float two consecutive elements, the second floated element
will always appear either beside the first floated element or
below it.
</p>
<p class="col">
For example, if you float the elements left, the second will
appear to the right of the first. If there isn't enough room to the
right, it will appear below the first element.
</p>
</div>
```

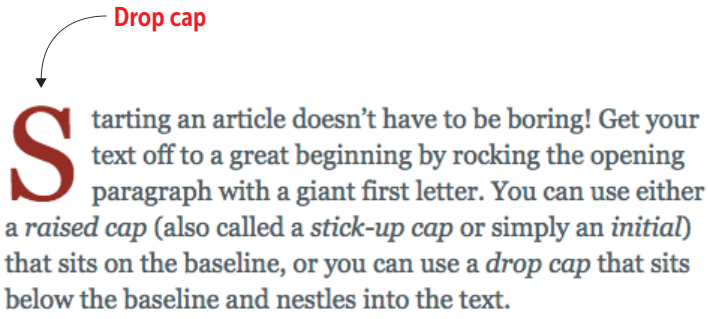
Floating a Drop Cap

Floats have many uses, but one of my favorites is creating a *drop cap*, which is a paragraph's large first letter that sits below the baseline and "drops" a few lines into the paragraph. The trick is to select the opening letter by using the `::first-letter` pseudo-element and float that letter to the left of the paragraph. Then you mess around with font size, line height, and padding to get the effect you want, as shown in the following example.

▶ *Example* Online: wdpg.io/8-1-6

This example uses `float` *and the* `::first-letter` *pseudo-element to create a drop cap.*

WEB PAGE



CSS

```
.first-paragraph::first-letter {
   float: left;
   padding-top: .1em;
   padding-right: .1em;
   color: darkred;
   font-size: 5em;
   line-height: .6em;
}
```

1. Select the first letter.
2. Float it to the left.
3. Style to taste.

HTML

```
<p class="first-paragraph">
Starting an article doesn't have to be boring! Get your text off
to a great beginning by rocking the opening paragraph with a giant
first letter. You can use either a <i>raised cap</i> (also called
a <i>stick-up cap</i> or simply an <i>initial</i>) that sits on
the baseline, or you can use a <i>drop cap</i> that sits below the
baseline and nestles into the text.
</p>
```

Floating a Pull Quote

Another great use for floats is to add a pull quote to an article. A *pull quote* is a short but important or evocative excerpt from the article that's set off from the regular text. A well-selected and well-designed pull quote can draw in a site visitor who might not otherwise read the article.

You create a pull quote by surrounding the excerpted text in an element such as a `span` and then floating that element, usually to the right. Now style the element as needed to ensure that it stands apart from the regular text: top and/or bottom margins, a different font size, style, or color, and so on. Following is an example.

MASTER

If you prefer a raised cap to a drop cap, you can modify the example code to accommodate this preference. You need to remove the `float` *declaration and the* `padding-top` *and* `padding-right` *declarations.*

▶ Example

Online: wdpg.io/8-1-7

This example uses `float` *to create a pull quote.*

WEB PAGE

A *pull quote* is a short excerpt or an important phrase or quotation that has been copied ("pulled") from a piece of text and displayed as a separate element between or, more often, to one side of the regular text. It's important that the pull quote be styled in a way that not only makes it stand apart from the regular text (with, for example, a different font size, style, or color), but also makes it stand out for the reader. After all, it's the job of the pull quote to entice the would-be reader and create a desire to read the article.

> *It's the job of the pull quote to entice the would-be reader.*

Pull quote

CSS

```
.pullquote {
   float: right;
   width: 50%;
   margin: 1.25em 0 1em .25em;
   padding-top: .5em;
   border-top: 1px solid black;
   border-bottom: 1px solid black;
   font-size: 1.05em;
   font-style: italic;
   color: #666;
}
.pullquote::before {
   content: "\0201c";
   float: left;
   padding: .1em .2em .4em 0;
   font-size: 5em;
   line-height: .45em;
}
```

This code floats the element.

This code styles the pull quote.

Creates an optional large quotation mark.

HTML

```
<p>
A <i>pull quote</i> is a short excerpt or an important phrase or
quotation that has been copied ("pulled") from a piece of text and
displayed as a separate element between or, more often, to one
side of the regular text.
<span class="pullquote">
It's the job of the pull quote to entice the would-be reader.
</span>
It's important that the pull quote be styled in a way that not only
makes it stand apart from the regular text (with, for example, a
different font size, style, or color), but also makes it stand out
for the reader. After all, it's the job of the pull quote to entice
the would-be reader and create a desire to read the article.
</p>
```

The pull quote element

Despite head-scratching behaviors such as parent collapse, floating elements are useful for breaking them out of the default flow to achieve interesting layouts and effects. Floats get the browser to do most of the work, but if you want even more control of the look of your pages, you need to get more involved by specifying the positions of your elements.

Lesson 8.2: *Positioning Elements*

Covers: The `position` property

Online: wdpg.io/8-2-0

I mentioned earlier in this chapter that the default layout the browser uses for page elements renders the elements in the order in which they appear in the HTML file, stacking block-level elements and allowing inline elements to fill their parent blocks left to right. This system rarely produces a compelling layout, so another technique you can use (besides floating elements) to break out of the default flow is positioning one or more elements yourself, using the CSS `position` property combined with one or more of the CSS offset properties:

```
element {
    position: static|relative|absolute|fixed;
    top: measurement|percentage|auto;
    right: measurement|percentage|auto;
    bottom: measurement|percentage|auto;
    left: measurement|percentage|auto;
    z-index: integer|auto;
}
```

For the first four offset properties—`top`, `right`, `bottom`, and `left`—you can use any of the CSS measurement units you learned about in Chapter 7, including `px`, `em`, `rem`, `vw`, and `vh`. You can also use a percentage or `auto` (the default). The `z-index` property sets the element's position in the *stacking context*, which defines how elements are layered "on top" of and "under" one another when they overlap. An element with a higher `z-index` value appears layered over one with a lower value.

For the `position` property, here's a quick summary of the four possibilities:

- `static`—Ignores the offset properties (this is the default positioning used by the browser)
- `relative`—Positions the element offset from its default position while keeping the element's default position within the page flow
- `absolute`—Positions the element at a specific place within the nearest ancestor that has a nonstatic position while removing the element from the page flow
- `fixed`—Positions the element at a specific place within the browser viewport while removing the element from the page flow

The next few sections give you a closer look at relative, absolute, and fixed positioning.

Relative Positioning

When you position an element relatively, the element's default position remains in the normal page flow, but the element is shifted by whatever value or values you specify as the offset:

- If you supply a `top` value, the element is shifted down.
- If you supply a `right` value, the element is shifted from the right.
- If you supply a `bottom` value, the element is shifted up.
- If you supply a `left` value, the element is shifted from the left.

Having the element's default page-flow position maintained by the browser can lead to some unusual rendering, as shown in the following example.

REMEMBER

These shifts assume that you supply positive values to each property. Negative values are allowed (and are used often in web-design circles) and result in shifts in the opposite direction. A negative `top` *value shifts the element up, for example.*

▶ *Example*

⇨ **Online:** wdpg.io/8-2-1

This example sets the `span` *element to relative positioning with a* `top` *offset.*

WEB PAGE

Relative positioning shifts an element out of its default position while preserving the element's original space in the page flow. This can cause page weirdness. For example, if you set the top property, the element shifts down. This leaves a gap where the element would have been, which can look odd.

Gap where the span element would have been

The shifted span element

CSS

```
span {
    position: relative;
    top: 3em;
    border: 2px solid blue;
}
```

Applies relative positioning and a top offset to the span element

HTML

```
<div>
Relative positioning shifts an element out of its default position
while preserving the element's original space in the page flow.
This can cause page weirdness. For example, if you set the top
property, the element <span>shifts down</span>. This leaves a gap
where the element would have been, which can look odd.
</div>
```

The span element

You probably won't use relative positioning much for laying out page elements directly, but as you see in the next section, it comes in handy when you want to prepare elements to use absolute positioning.

PLAY

Use relative positioning to add watermark text to a paragraph.

⇨ **Online:** wdpg.io/8-2-2

Absolute Positioning

When you position an element absolutely, the browser does two things: It takes the element out of the default page flow, and it positions the element with respect to its nearest nonstatic (that is, positioned) ancestor. Figuring out this ancestor is crucial if you want to get absolute positioning right:

- Move up the hierarchy to the element's parent, grandparent, and so on. The first element you come to that has had its `position` property set to something other than `static` is the ancestor you seek.
- If no such ancestor is found, the browser uses the viewport, meaning that the element's absolute position is set with respect to the browser's content area.

With the ancestor found, the browser sets the element's absolute position with respect to that ancestor as follows:

- If you supply a `top` value, the element is moved down from the ancestor's top edge.
- If you supply a `right` value, the element is moved left from the ancestor's right edge.
- If you supply a `bottom` value, the element is moved up from the ancestor's bottom edge.
- If you supply a `left` value, the element is moved right from the ancestor's left edge.

REMEMBER

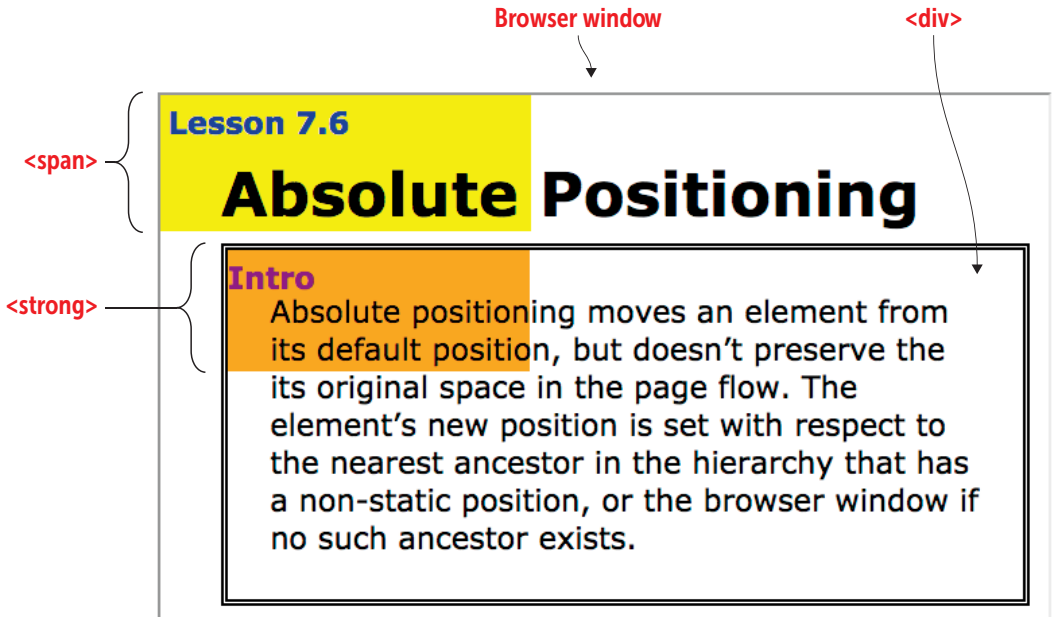
As with relative positioning, negative values are allowed and position the element in the opposite direction. A negative `left` value moves the element left with respect to the ancestor's left edge, for example.

▶ Example

Online: wdpg.io/8-2-3

This example sets both a `span` element and a `strong` element to absolute positioning.

WEB PAGE



CSS

```
h1, div {
    position: relative;
    z-index: 2;
}
span {
    position: absolute;
    top: 0;
    left: 0;
    z-index: 1;
    padding: 0.25em 6em 3em 0.25em;
    background-color: yellow;
    color: blue;
}
strong {
    position: absolute;
    top: 0;
    left: 0;
    z-index: -1;
    padding: 0.25em 5em 2.5em 0;
    background-color: orange;
    color: purple;
}
```

The div element is nonstatic.

The span and strong elements are positioned absolutely.

continued

HTML

```
<h1>
Absolute Positioning
</h1>
<div>
Absolute positioning moves an element from its default position,
but doesn't preserve the its original space in the page flow. The
element's new position is set with respect to the nearest ancestor
in the hierarchy that has a non-static position, or the browser
window if no such ancestor exists. <strong>Intro</strong>
</div>
<span>Lesson 8.6</span>
```

The span element

The strong element

SEE IT

To see an animation of how the browser positions the elements in this example, open the example in the Web Design Playground and click the See It button.

Online: wdpg.io/8-2-3

MASTER

This example also demonstrates the `z-index` *property. The* `h1` *and* `div` *elements have been given a* `z-index` *value of 2. The* `span` *element is given a* `z-index` *of 1; therefore, it appears "behind" the* `h1`*. The* `strong` *element is given a* `z-index` *of -1; therefore, it appears "behind" the* `div`*.*

In this example, two elements are positioned absolutely:

- `span`—This element has no nonstatically positioned ancestor, so it's positioned with respect to the browser window. When you set both `top` and `left` to 0, the `span` element moves to the top-left corner of the window.
- `strong`—This element is nested inside a `div` element that's positioned relatively. Therefore, the `strong` element's absolute position is with respect to the `div`. In this case, when you set both `top` and `left` to 0, the `strong` element moves to the top-left corner of the `div`.

Fixed Positioning

The final `position` property value that I'll consider is `fixed`. This value works just like `absolute`, except for two things:

- The browser always computes the position with respect to the browser window.
- The element doesn't move after it has been positioned by the browser, even when you scroll the rest of the page content.

As you might imagine, this value would be useful for adding a navigation bar that's fixed to the top of the screen or a footer that's fixed to the bottom. You see an example of the latter in Chapter 15.

Summary

- In the default page flow, block-level elements are stacked vertically, and inline elements are rendered from left to right within their parent blocks.
- To pull an element out of the default page flow, set its `float` property to `left` or `right`.
- To position an element, set its `position` property to `relative`, `absolute`, or `fixed`; then specify the new position with `top`, `right`, `bottom`, and `left`.
- Set an element's position within the stacking context by using the `z-index` property, which layers higher-value elements over smaller-value elements.

FAQ

Why did you use -1 for the strong element's *z-index? The* `strong` *element is a descendant of the* `div` *element, and in CSS, the only way to make a descendant appear lower in the stacking context than its ancestor is to give the descendant a negative* `z-index` *value.*

PLAY

You can use absolute positioning to add tooltips *(pop-up descriptions) to your links.* ⇨ **Online:** wdpg.io/8-2-4