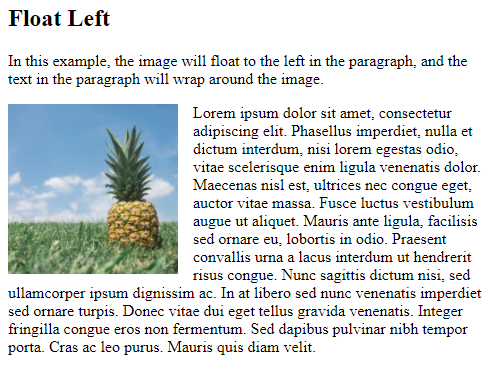
The main thing I wanted to cover today was the CSS property called "float". Our book gives a good explanation of it in Chapter 15. But, I'll give a basic rundown of it here, and also try to paint a picture of how this property fits into the larger CSS "ecosystem".

The float property was originally developed to wrap text around images. The float would be applied on the image, and as a result, any neighboring text would float around this image. Here’s an illustration:



And here’s the relevant code that makes up the above example (I abbreviated the non-relevant bits with ellipses):

<style>

img {

float: left;

}

</style>

…

<p>

<img src="pineapple.jpg">

Lorem ipsum dolor sit amet…

</p>

The important thing to note here is that the float property is applied on the image, NOT the text / paragraph. Yet, it is the text (which doesn’t have the float) which wraps around the image. A float will affect its neighboring elements.

The float property has two basic values: *left* and *right* (there’s also *none*, which is the default, and *inherit* which inherits the float value from the parent element, but we don’t need to concern ourself too much with these). In the above example, if we changed *float: left* to *float: right* on the img selector, then the image would be pushed all the way to the right, and the text would be hugging the picture from the left. Try to modify the above code to achieve this effect (don’t forget to add standard HTML tags, like html, head, body, etc.).

Now, although floats were meant to be used with text surrounding images primarily, people have come to use them to position and align all kinds of elements. Say, for example, that you have a main content container, which takes up 80% of the width, and a sidebar, which takes up 20% of the width. You want them to appear side-by-side: main content on the left, sidebar on the right. Well, you could use *display: inline-block* to achieve this, but then you have to deal with the strange quirk of about 4px of spacing being added between the two elements, which will cause them to break onto separate lines (20% + 80% + 4px is more than 100%). In this particular case, it’s actually easier to use floats.

See this JSFiddle for the code: <https://jsfiddle.net/juliandev/k1rLc5p9/> (JSFiddle is a site where you can share small coding projects with people).

Observe that both elements, main and aside have been given *float: left*. This changes their default display behavior. Normally, any element with *display: block* would occupy the whole width of its parent element and will not align side-by-side with another element. However, the *width* property in combination with *float: left* informs these block-level elements that they should only occupy as much space as specified, and allow neighboring elements to float around them on the right. The *aside* element is actually wrapping around the *main* element, much like text around the image in the previous example.

Note that just because floats are a simpler and preferred way of aligning our two elements side-by-side in this case, it does not mean that the approach we covered previously, *display: inline / inline-block*, is obsolete. There are many contexts in which the display property would be preferrable. How do you know whether to use floats or display? To a large extent, you have to train this ability through experience. But, to give an example, if having a gap of 4px between elements is of no consequence, then the display property may provide the better approach. Broadly speaking, I tend to pick the solution that requires less code or less complexity.

Moving on…

To understand floats intuitively, imagine that elements on which the float property is applied produce their own field of gravity, so that any neighboring elements are attracted to them. If an element has *float: right*, then it will be pushed to the right while nearby elements will gravitate / be pulled towards it from the left.

Now, if you think far ahead, you may anticipate an issue here: where does this “field of gravity” end? At what point does a floating element stop pulling elements towards it? This is a good question. To understand the problem, do this:

Create three DIVs, put some text inside them (“Box 1”, “Box 2”, “Box 3”), and assign them IDs (“div1”, “div2”, “div3”). Let’s say we want to align the first 2 boxes side-by-side using floats. Give the first two DIVs the property *float: left* (but not the third one). This should achieve our goal of aligning the first two boxes side-by-side. But, there’s an issue: the 3rd box is also a wrapping onto the same line. It’s attracted by the gravity of the first two boxes. What if we want the 3rd box to remain unaffected?

CSS anticipates this problem and provides us with a property called *clear* to address it. This property accepts three basic values: left, right, and both. *Left* will clear *float: left*, right will clear *float: right*, and *both* will clear floats on the left and the right. But what does it mean to “clear” a float? In simple terms, the clear property informs CSS where to stop floating.

To see this in action, assign the property *clear: left* to the third DIV, and refresh the page. Now, the third box should appear on a separate line, because it ended the float.

This solution, often called “clearfix”, also solves another problem.

Go ahead a remove the clear property. Then, wrap the three DIVs in a containing DIV. Give all of the three boxes a height property (of 100px, let’s say), and a border (*1px solid black*) in order to better see their dimensions. Now, open up DevTools in Chrome, and select the containing DIV element. It should highlight blue on the page when you’re hovering over the tag. Take note of the size of the three boxes. They overflow outside the blue area being highlighted. In other words, the parent element is shorter than its children elements! This a problem, because if we, for example, wanted to apply a background color to this parent element, the color wouldn’t extend to match the height of the boxes it contains.

This quirk is a side-effect of the float property. When an element is floated, it removes it from the normal flow of the page, and as a result, its dimensions stop being imparted on its parent elements, which would otherwise expand to contain the dimensions of its children. Fortunately, applying the clearfix solves this problem for us.

Add *clear: left* back in to the 3rd box, and observe that the parent element’s height now contains the height of its children elements.

Aside from the float exercises that freeCodeCamp may provide, try your hand at this challenge to practice your understanding of floats: <https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Floats_skills>

By the way, as I said before, alignment / positioning tends to be the hardest part of CSS, since its fairly complex, and has many strange quirks (as we’ve just seen). However, once we cover this topic, almost everything else will be down-hill, and we’ll be ready to start building real website layouts.