Painless CSS

Build Beautiful, Intuitive Websites

BILL MEI

This is a free sample of the book
Painless CSS: Build Beautiful, Intuitive Websites
by Bill Mei

Learn more at https://www.painlesscss.com/

Chapter 11: The Box Model

In *Chapter 5: Block and Inline Elements*, we talked about how elements are placed one after another as if the browser was a blind bricklayer, and how text wraps around to the next line if it gets to the end of the browser window. In this chapter, we'll surgically dissect one of these bricks to see what it's made of.

Whenever you use the `width`, `height`, `margin`, `padding`, or `border` CSS properties, you are changing the dimensions and behavior of this "brick", or "box". I'll use the terms "box", "brick", and "element" interchangeably. The ***CSS Box Model*** describes the relationships between these five properties. There are two ways to imagine the box model:

1. The popular way (using boxes)
2. The *Painless CSS* way (using fences)

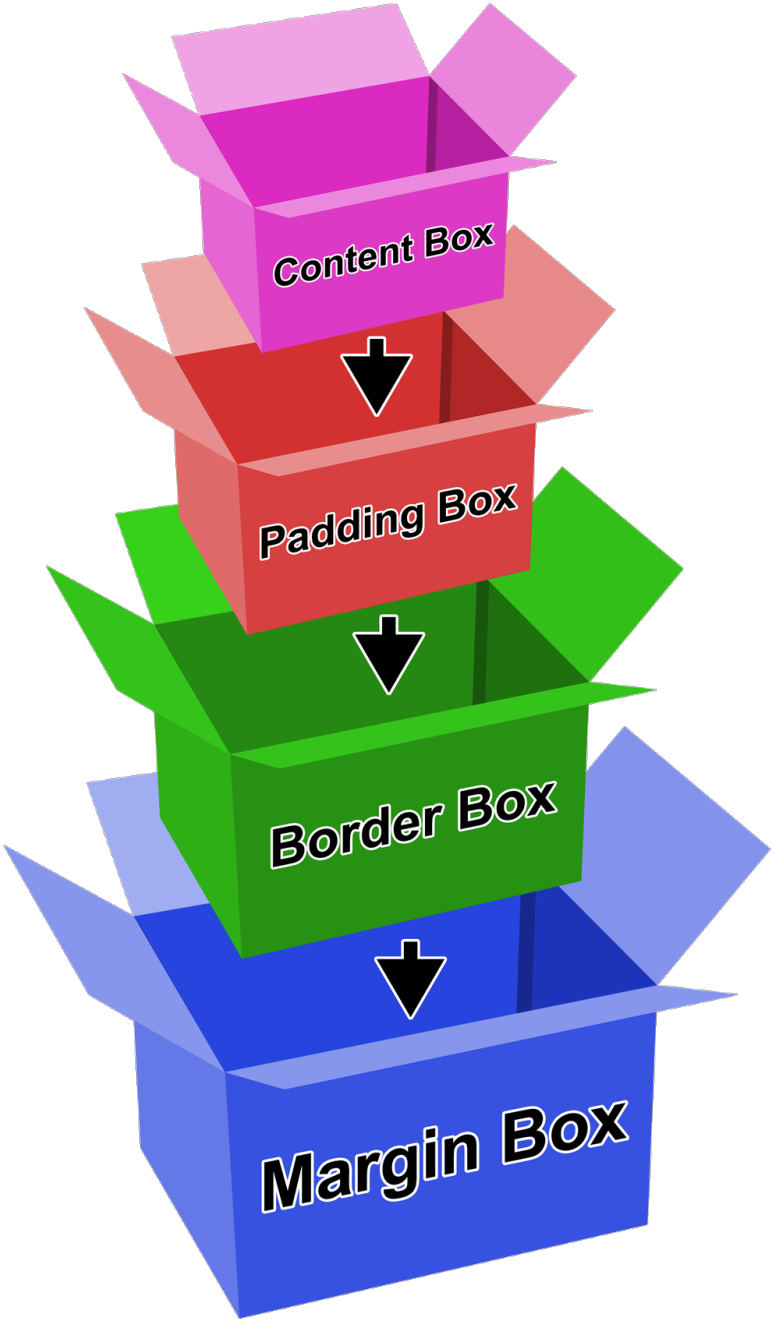
First, I'll describe the classical (popular) way to give you some context. Then, I'll present what I call the ***Painless CSS fence*** which I believe is simpler to understand and will result in less pain when you use this mental model in practice. (I'm doing this for the same reason I contrasted the *Painless CSS cascade* against the classical cascade.)

The popular box model

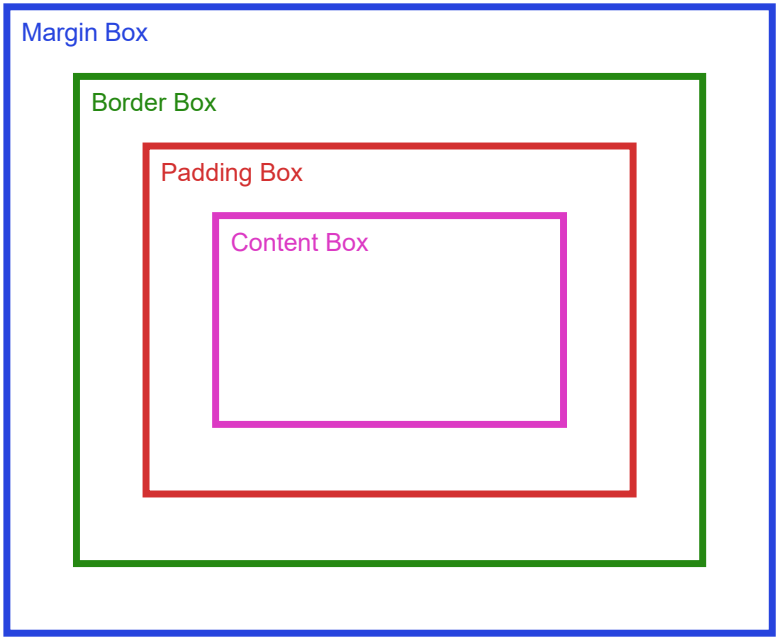
Every HTML element can be thought about as a series of rectangular boxes stacked into each other:

- The content box
- The padding box
- The border box
- The margin box

Side view



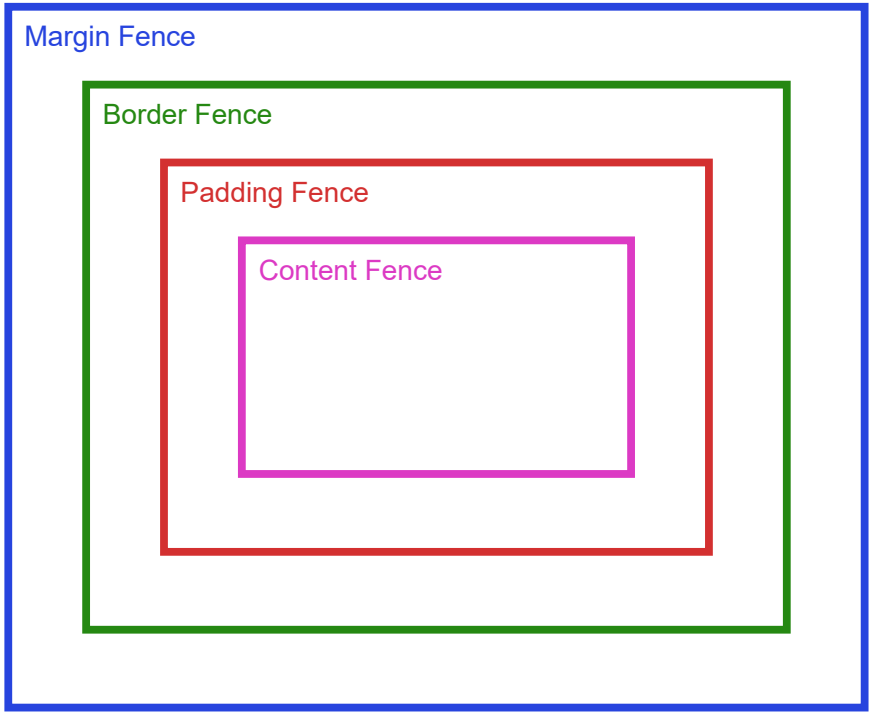
Top view



- The `margin` property tells you how big the Margin Box is
- The `border` property tells you how big the Border Box is
- The `padding` property tells you how big the Padding Box is
- The `width` and `height` properties tell you how big the Content Box is

The Painless CSS Fence

While the traditional "box model" is a fine explanation when you look at an element in isolation, it fails at sufficiently describing what happens when multiple elements interact. Instead of "boxes", I like to imagine each element as a yard surrounded by multiple layers of fences instead:



The fence model makes it more intuitive to imagine what happens when two elements bump up into each other. For example, it still works when you have

negative margins (how can the Margin Box be "negative"?). Using fences also makes the definition of each property more precise.

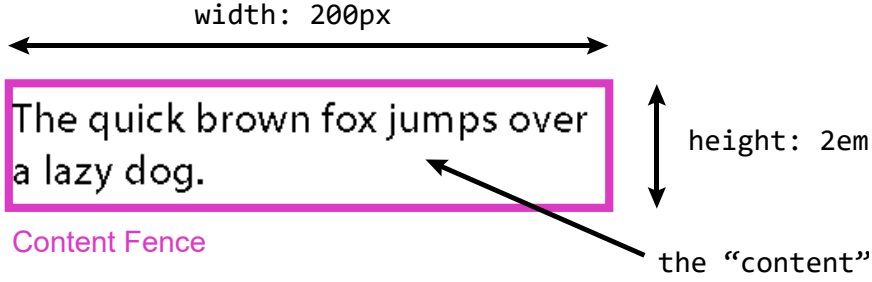
Definition of Content

The `width` and `height` properties tell you how much distance there is between opposite content fences.

- `width` is the distance between the content fences in the x direction
- `height` is the distance between the content fences in the y direction.

Example:

```
width: 200px;
height: 2em;
```



Any children of your HTML element cannot escape the content fence.

If you did not explicitly set a `width` or `height`, then the content fence will become the same size as your child. This means that if your child is "too big" for the content fence, the content fence will grow to accommodate the child. If you explicitly set a `width` but not a `height`, then the content fence will grow in the y direction but not in the x direction, and vice-versa.

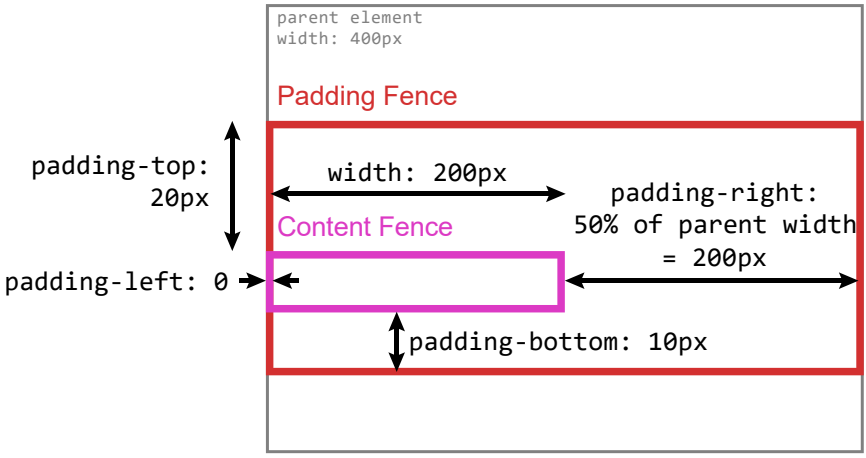
However, if you explicitly set both a `width` and a `height` and the child is too big for both these dimensions, then the child will try its darndest to break out of the content fence. The rules that define what happens during this jailbreak are set by the `overflow` property. We'll talk more about `overflow` in *Bonus 11: Overflow*.

Definition of Padding

The `padding` property tells you how much distance there is between the content fence and the padding fence.

Example:

```
/* parent width: 400px */
width: 200px;
padding-left: 0;
padding-right: 50%;
padding-top: 20px;
padding-bottom: 10px;
```

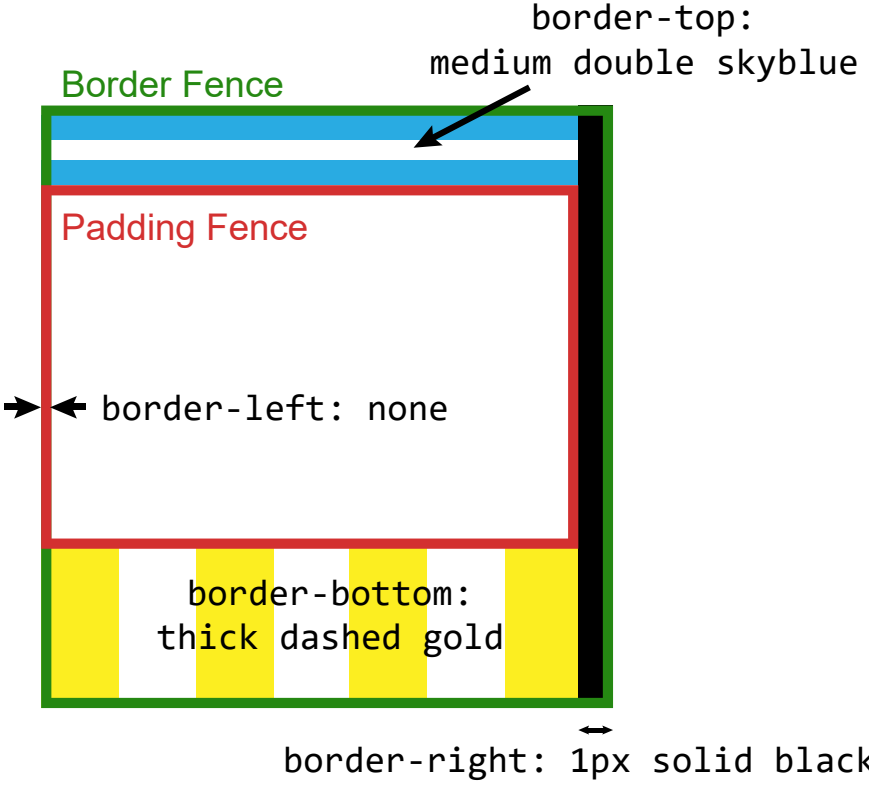


Definition of Border

The `border` property tells you how much distance there is between the padding fence and the border fence.

Example:

```
/* border: [width] [style] [color]; */
border-left: none;
border-right: 1px solid black;
border-top: medium double skyblue;
border-bottom: thick dashed lime;
```



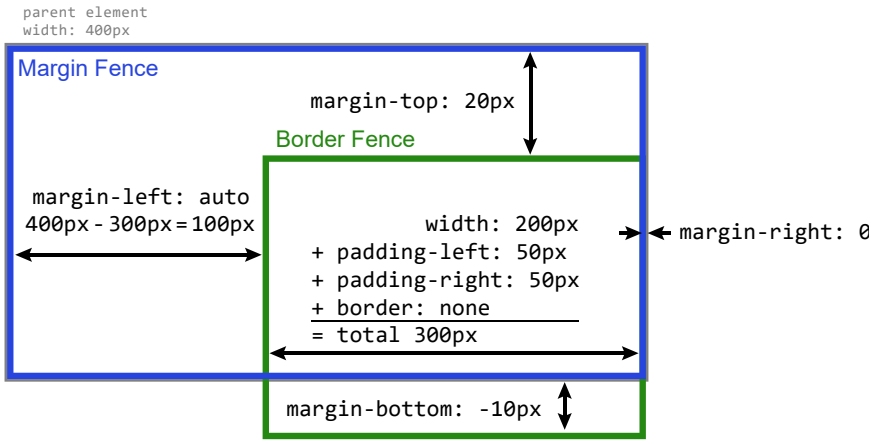
Any background images or background colors on your element cannot escape the border fence, unless you change the `background-clip` property.

Definition of Margin

The `margin` property tells you how much distance there is between the border fence and the margin fence.

Example:

```
/* parent width: 400px */
width: 200px;
padding-left: 50px;
padding-right: 50px;
margin-left: auto;
margin-right: 0;
margin-top: 20px;
margin-bottom: -10px;
```



In this example, it looks like the border fence is protruding outside the bottom of the margin fence, and this is because our `margin-bottom` is negative.

The margin fence is the outer perimeter that defines where two elements are allowed to touch each other. *no element's margin fence is allowed to cross into any other element's margin fence.*

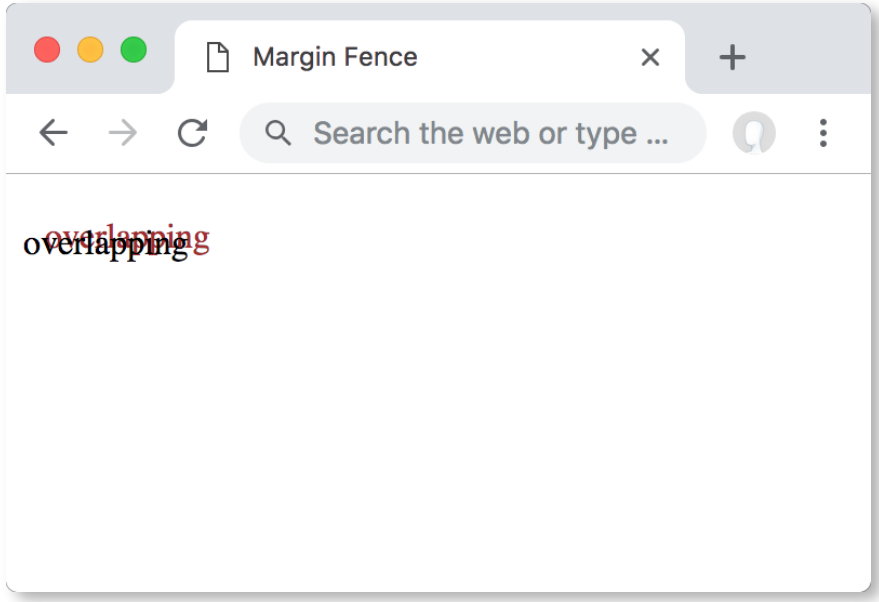
You can think of the margin fence as essentially the outside edge of an individual "brick" in our wall of bricks.

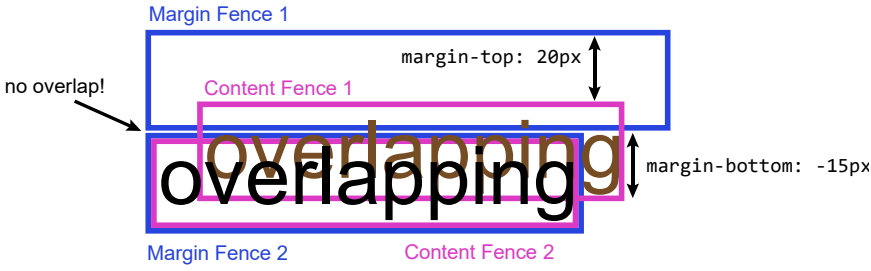
This means when two bricks are placed beside each other, they touch at their margin fences:

Source Code: chapter11/margin-fence.html

```
.margin-fence-1 {
  color: brown;
  margin-left: 10px;
  margin-right: 10px;
  margin-top: 20px;
  margin-bottom: -15px;
}

<div class="margin-fence-1">overlapping</div>
<div>overlapping</div>
```





Although it looks like these two elements are overlapping (their content fences are crossing each other), the two margin fences never overlap! The negative margin on

the first element makes the content fence protrude outside of the margin fence, which gives the illusion of overlapping, but the margin fences never cross!

Also, margin fences are *always* touching. You can't push elements further apart by adding empty space between them—the margin fence *is* that empty space! This means the most basic way to push two elements apart is to increase the size of the margin fence, and the most basic way to bring two elements closer together is to reduce the size of the margin fence.

Now that we've learned about the boundaries of each brick, let's see what happens when we place multiple bricks together on a page.

Chapter 12:
Document Flow

In *Chapter 3: Rendering*, we showed how the browser renders elements on the screen from left to right. Then, in *Chapter 5: Block and Inline Elements*, we examined how elements are placed one after another as if the browser were a blind bricklayer, and how text wraps around to the next line if it gets to the end of the browser window. Finally, in *Chapter 11: The Box Model*, we investigated the internals of each brick (or box).

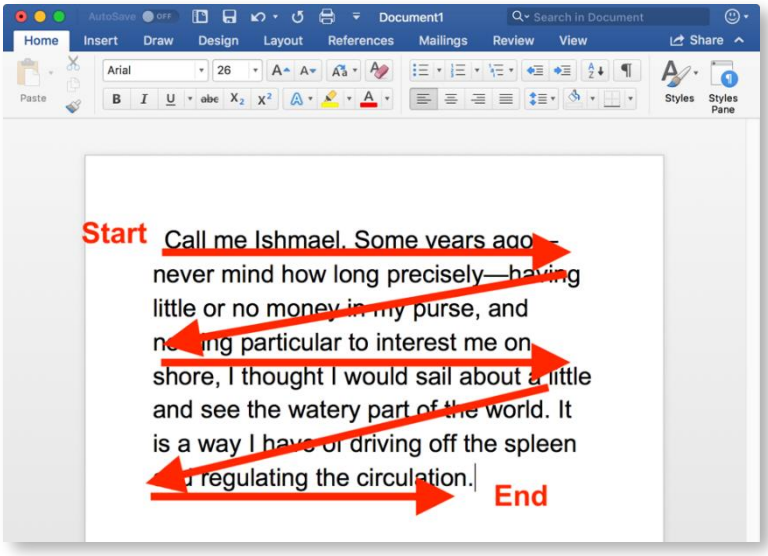
We now have the tools we need to understand how to place HTML elements on a page. First, let's look at the default placement rules, and then let's use CSS to break these rules.

The sequence of blind bricklaying is also called the browser's ***Document Flow***, or ***Normal Flow***, because this is how elements are positioned by default.

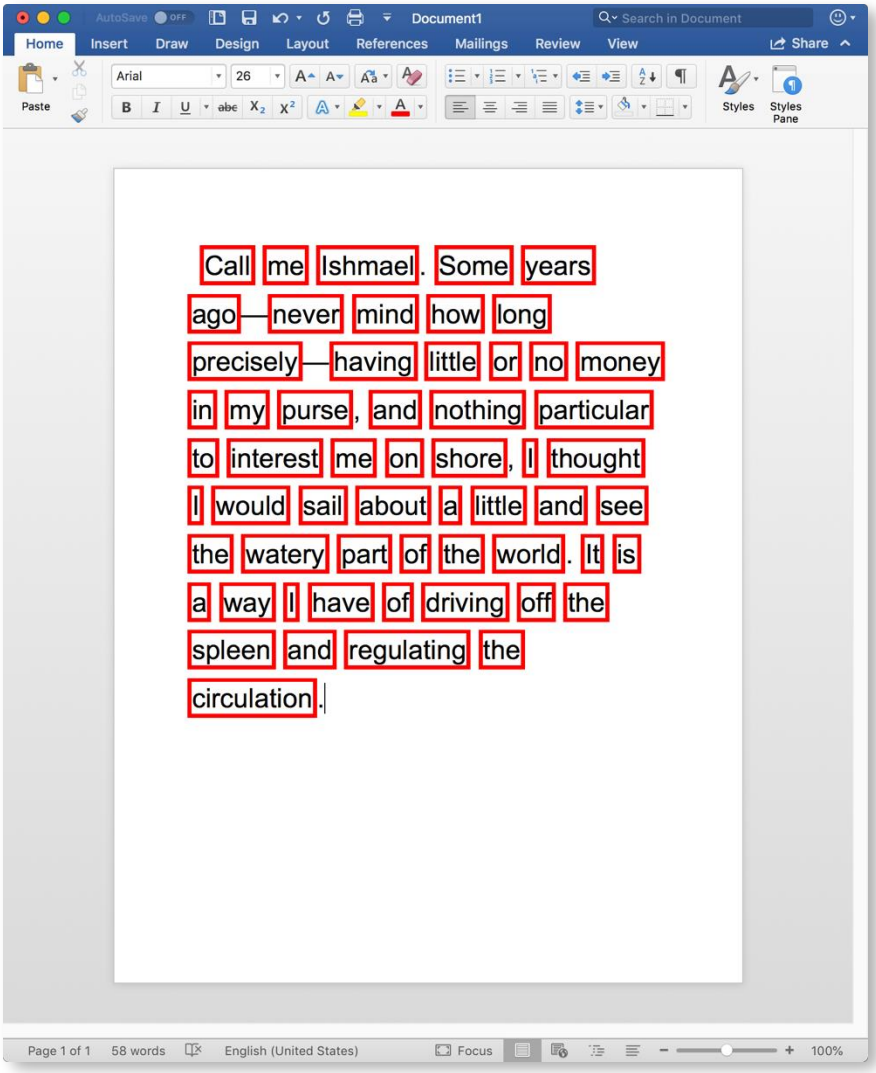
Inside the Document Flow

Normal Flow

When we say that something is "following the document flow" or "inside the normal flow", we refer to the fact that every new element is placed in the default left-to-right, top-to-down order. This is just like how in a Word document (or typewriter), your cursor starts at the top-left of the page and eventually ends up at the bottom-right of the page when you finish typing.

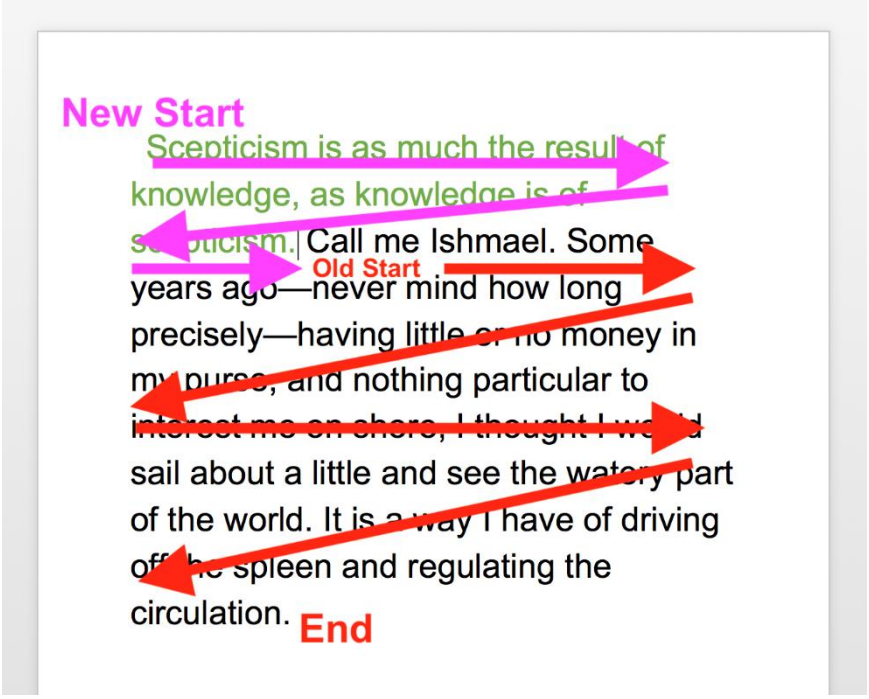


If we add a border to each brick (or box, as we learned in *Chapter 11*) in this document, it becomes clear where the "brick wall" analogy originates:



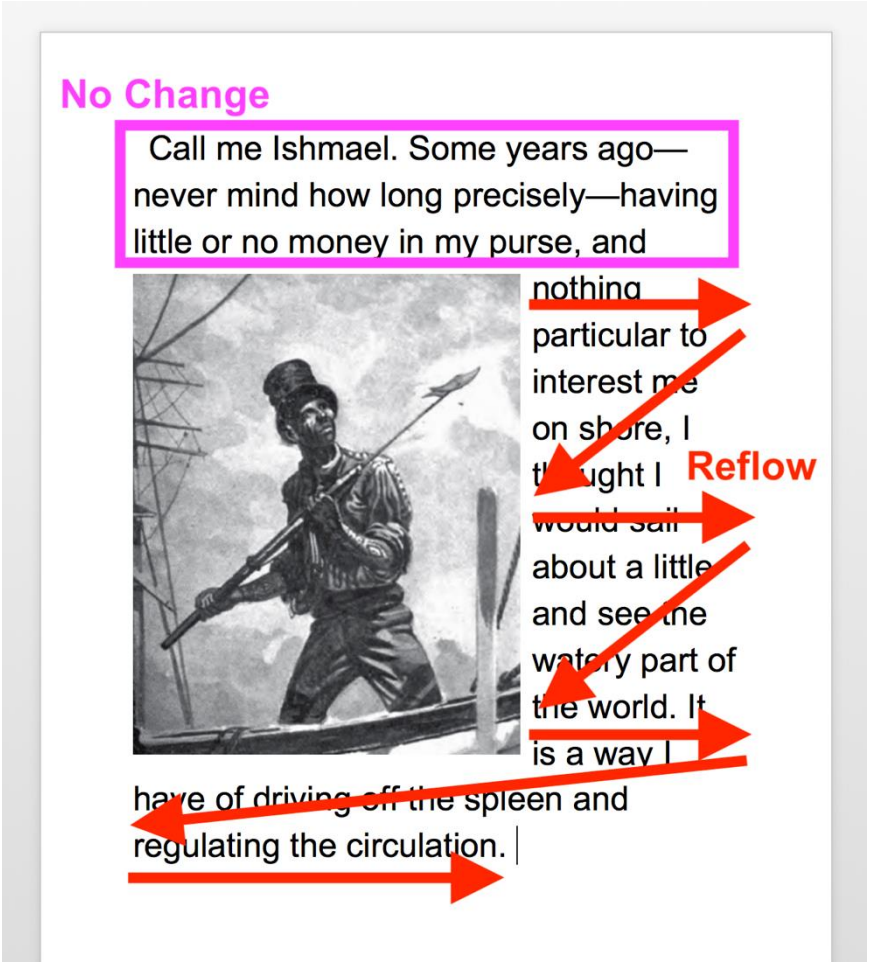
If we add a new element to the end of the document, it *must* go to the end of the document flow. We can't arbitrarily place new elements anywhere on the page. To do that, we need to put the new element *outside* the document flow.

However, if we wanted to add text to the beginning of the document flow, then we can still do this in the normal flow, it's just that every other element after it must "reflow" to accommodate this new element.



In our bricklaying analogy, this is the difference between adding a brick onto the end of a wall, versus destroying the entire wall just to add a brick at the very beginning and re-laying all the bricks on top of the first brick all over again.

If you've ever experimented with inserting an image into the middle of a Microsoft Word document, you know what this is like. The image doesn't change any text that was before it, but everything after it must be repositioned all over again to accommodate.[62]



[62] Image attribution: https://en.wikipedia.org/wiki/File:Queequeg.JPG

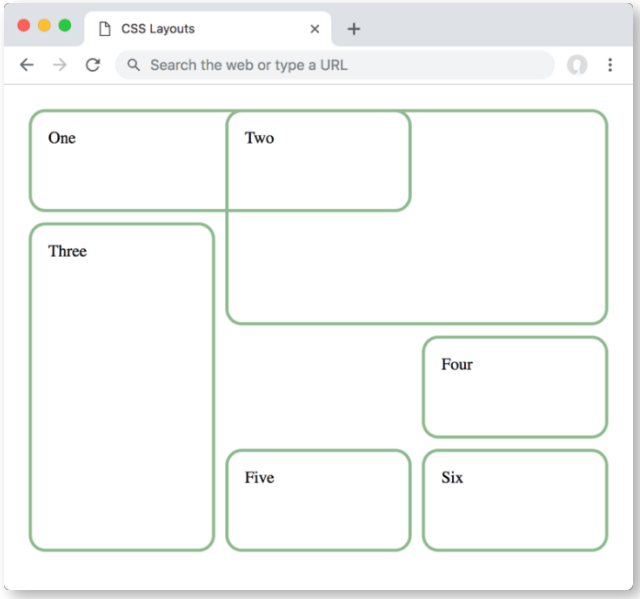
Changing the Normal Flow

Grid Layout

The normal flow can be modified for more advanced control over how elements are placed. For example, the `display: grid` declaration allows you to put more rigid guardrails around where elements start and end, instead of relying solely on the `width` and `height` properties of your `block` and `inline-block` elements. This is also known as a ***Grid Layout***. See the documentation[63] on `display: grid` for more information.

`Source Code: chapter12/grid-layout.html`

```
.grid-layout {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-gap: 10px;
  grid-auto-rows: minmax(100px, auto);
}
```



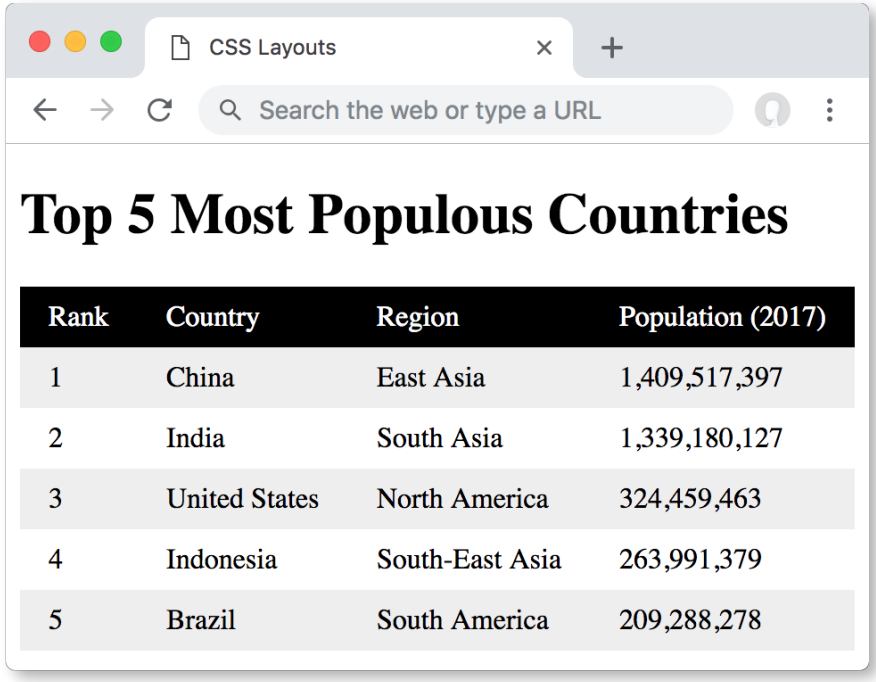
63 https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Grids

Table Layout

There is also a ***Table Layout*** enabled with `display: table` which makes the document flow behave like a spreadsheet. This is meant for showing tabular data on a page. See the documentation[64] on `display: table` for more information.

`Source Code: chapter12/table-layout.html`

```
.table-layout {
  display: table;
}
.country, .category {
  display: table-row;
}
.country div, .category div {
  display: table-cell;
}
```



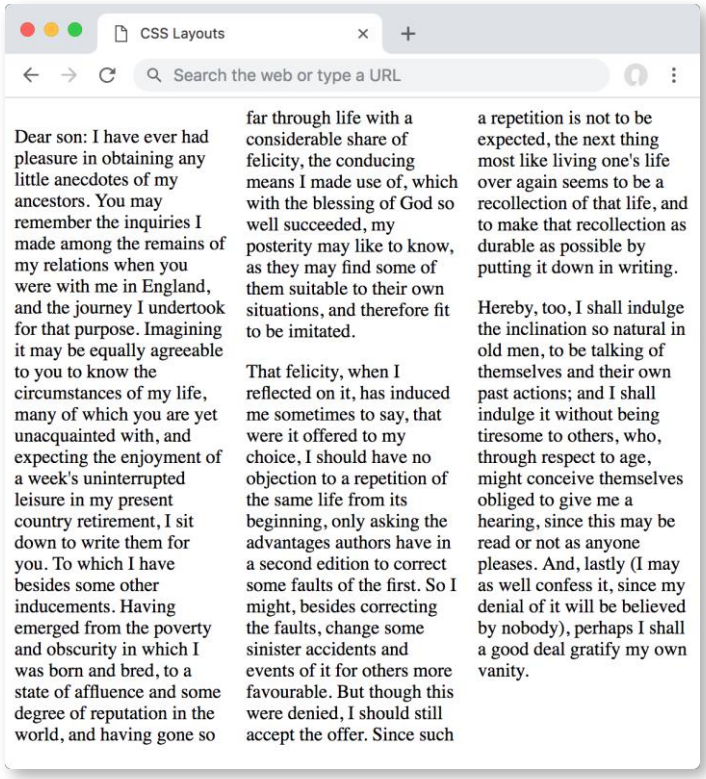
[64] https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Introduction

Multi-Column Layout

Finally, there is a ***Multi-Column Layout*** enabled with the `column-count` and `column-width` properties which makes the document flow behave as in a multi-column newspaper article. See the documentation[65] on `column-count` and `column-width` for more information.

`Source Code: chapter12/column-layout.html`

```
.column-layout {
  column-count: 3;
}
```



65 https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Multiple-column_Layout

Note that in the above examples, although we've changed the normal flow, we haven't changed the basic rules, since the base principle of putting elements one after the other in a left-to-right, top-to-down order still reigns. These modifications merely add some additional rules that help shape our brick wall in a certain way, but it hasn't let us build a levitating wall in midair, and we are still beholden to the document flow.

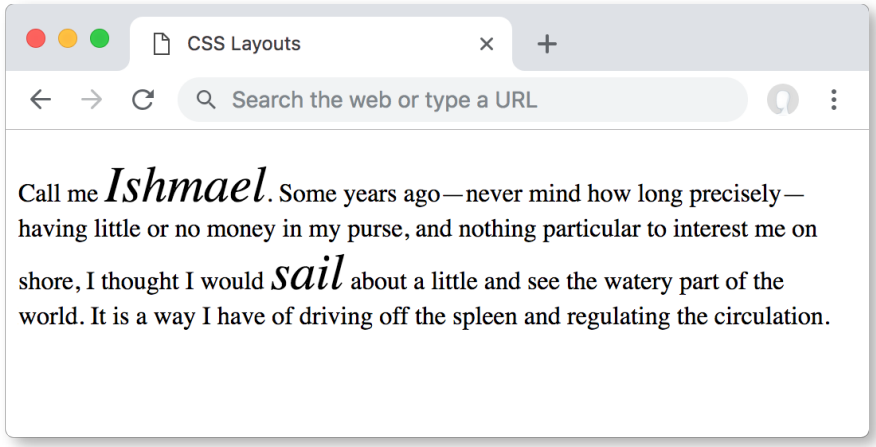
Changing the Bricks

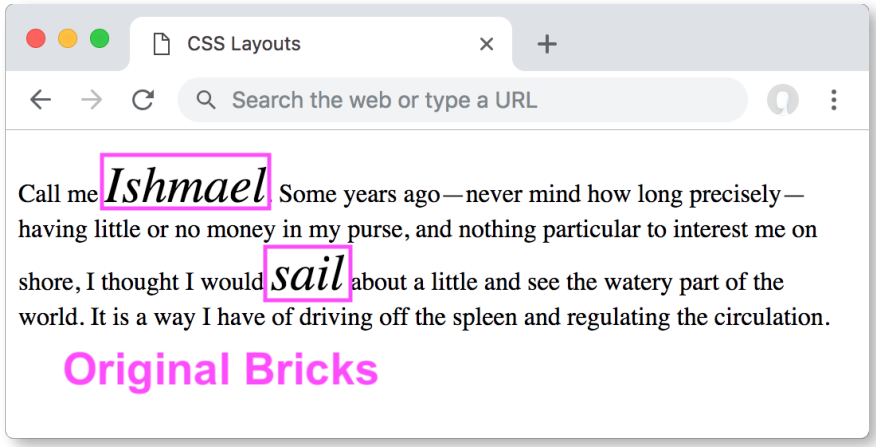
If you change any `display: block` elements to `display: inline`, you'll notice that some elements move around on the page, and the layout changes. However, this is not because we've broken or modified the document flow in some way. The document flow stays the same—we've only changed the shape of the bricks (elements), so any subsequent bricks in the document flow must reflow to accommodate the new brick. But we are not picking up a brick and putting it in a different place in the wall.

The same thing is true for properties like `padding`, `margin`, `width`, `height`, and `transform`. Even though these properties are often used to move elements around on the page, they only change the shape of the element, and not the document flow. Here's an example:

`Source Code: chapter12/changing-bricks.html`

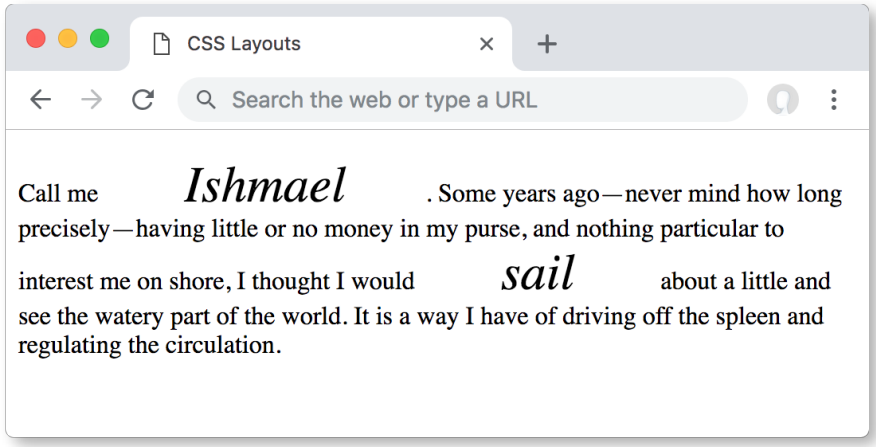
```
.changed-brick {
  margin: 0;
}
```

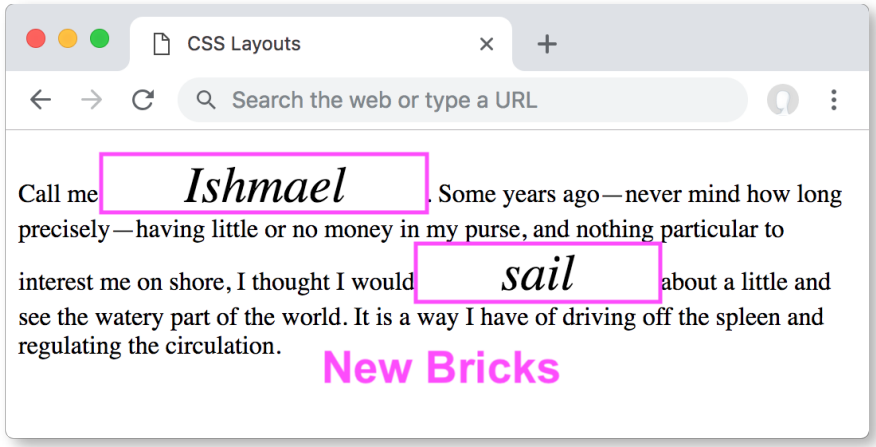




If I put extra margins around the highlighted brick, the content will appear to be farther apart, because I made the "brick" bigger by making the margin larger, so the only way to fit larger bricks into the same sized wall is to have fewer bricks per row.

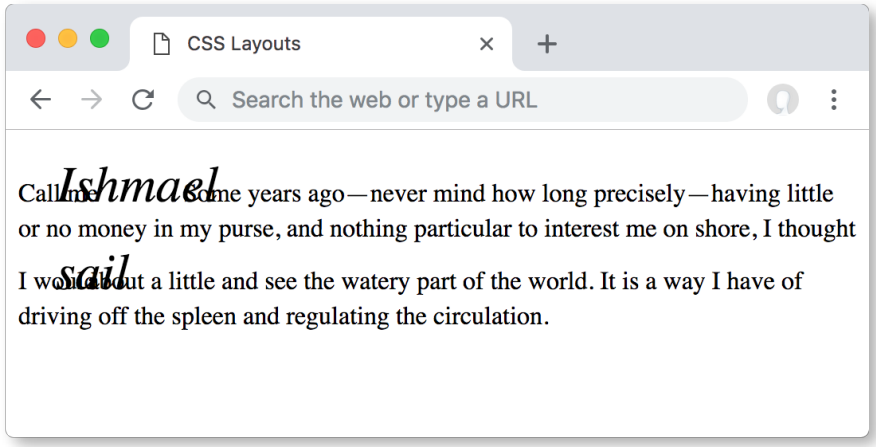
```
.changed-brick {
  margin-left: 50px;
  margin-right: 50px;
}
```

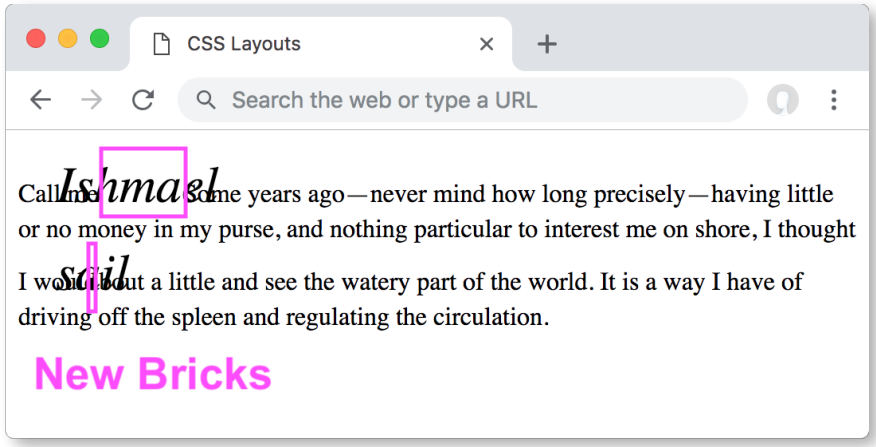




What makes this confusing is that you could have *negative margins*. If you make the margin negative, then it's the same thing in reverse, you're *subtracting* from the total space that the brick takes up in a row, so you can fit more bricks into the same row. From the user's perspective, it'll look like the bricks are overlapping each other. But if you think about it, they're not *really* overlapping each other; I haven't changed or broken the document flow rules in any way, it's just that I've made the brick have "negative size", allowing more bricks to fit into the wall.

```
.changed-brick {
  margin-left: -30px;
  margin-right: -30px;
}
```





Outside the Document Flow

What if you want to arbitrarily place an element anywhere on the page? The document flow doesn't allow you to do this, so you'll need to place it *outside* the document flow. There are two main ways of breaking the document flow, which we'll discuss in the next two chapters:

- Option 1: Set the element's `float` property (*Chapter 13: Floats*)
- Option 2: Change the element's `position` property (*Chapter 14: Position*)

Finally, there is the `flexbox` property. Flexbox allows you to change the basic rules of the document flow to position elements in a manner that isn't normally possible. However, flexbox does not *remove* the document flow, or place any elements *outside of it*—all flexbox elements are still subject to *a* document flow with *some* order, it's just not in the regular left-to-right, top-to-down order. Thus, while flexbox is not technically "outside" the document flow, it behaves so differently from the normal flow that it needs its own investigation.

- Option 3: Use flexbox

Flexbox is a huge topic that is applicable only when you want to control more advanced layouts. For this reason, it's outside the scope of this book; if you'd like to learn more about flexbox, there are two resources I can recommend for this.

The first is the MDN Guide to Flexbox[66]:

https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Flexible_Box_Layout/Basic_Concepts_of_Flexbox

The second is Chris Coyier's Guide to Flexbox[67]:

https://css-tricks.com/snippets/css/a-guide-to-flexbox/

If you pick a third-party library such as Bootstrap to manage your layouts, you generally won't need to worry about the internal details of this too much. Once you find yourself wanting a more advanced layout, you can come back and read more of the flexbox resources. Let's continue by talking about floats.

---

66 Archived link: https://web.archive.org/web/20181031212705/https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Flexible_Box_Layout/Basic_Concepts_of_Flexbox

67 Archived link: https://web.archive.org/web/20181108135212/https://css-tricks.com/snippets/css/a-guide-to-flexbox/