The box model

Everything in CSS has a box around it, and understanding these boxes is key to being able to create layouts with CSS, or to align items with other items. In this lesson, we will take a proper look at the CSS *Box Model* so that you can build more complex layout tasks with an understanding of how it works and the terminology that relates to it.

Prerequisites:	Basic computer literacy, basic software installed, basic knowledge of working with files, HTML basics (study Introduction to HTML), and an idea of how CSS works (study CSS first steps.)
Objective:	To learn about the CSS Box Model, what makes up the box model and how to switch to the alternate model.

Block and inline boxes

In CSS we broadly have two types of boxes — **block boxes** and **inline boxes**. These characteristics refer to how the box behaves in terms of page flow, and in relation to other boxes on the page:

If a box is defined as a block, it will behave in the following ways:

- The box will extend in the inline direction to fill the space available in its container. In most cases this means that the box will become as wide as its container, filling up 100% of the space available.
- The box will break onto a new line.
- The `width` and `height` properties are respected.
- Padding, margin and border will cause other elements to be pushed away from the box

Unless we decide to change the display type to inline, elements such as headings (e.g. `<h1>`) and `<p>` all use `block` as their outer display type by default.

If a box has an outer display type of `inline`, then:

- The box will not break onto a new line.
- The `width` and `height` properties will not apply.
- Padding, margin and borders will apply but will not cause other inline boxes to move away from the box.

The `<a>` element, used for links, `<span>`, `<em>` and `<strong>` are all examples of elements that will display inline by default.

The type of box applied to an element is defined by `display` property values such as `block` and `inline`, and relates to the **outer** value of `display`.

Aside: Inner and outer display types

At this point, we'd better also explain **inner** and **outer** display types. As mentioned above, boxes in CSS have an *outer* display type, which details whether the box is block or inline.

Boxes also have an *inner* display type, however, which dictates how elements inside that box are laid out. By default, the elements inside a box are laid out in **normal flow**, which means that they behave just like any other block and inline elements (as explained above).

We can, however, change the inner display type by using `display` values like `flex`. If we set `display: flex;` on an element, the outer display type is `block`, but the inner display type is changed to `flex`. Any direct children of this box will become flex items and will be laid out according to the rules set out in the Flexbox spec, which you'll learn about later on.

> **Note**: To read more about the values of display, and how boxes work in block and inline layout, take a look at the MDN guide to Block and Inline Layout.

When you move on to learn about CSS Layout in more detail, you will encounter `flex`, and various other inner values that your boxes can have, for example `grid`.

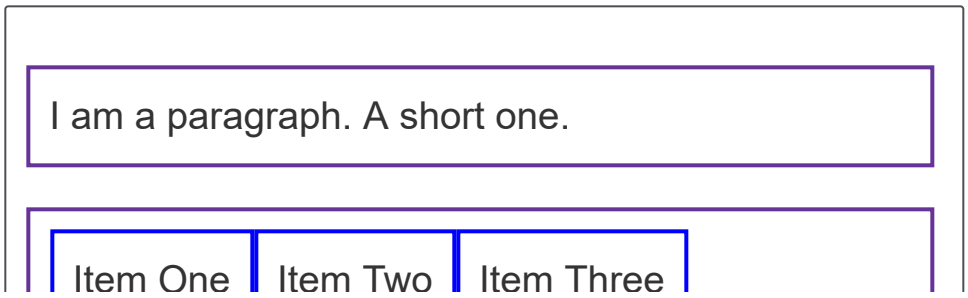
Block and inline layout, however, is the default way that things on the web behave — as we said above, it is sometimes referred to as *normal flow*, because without any other instruction, our boxes lay out as block or inline boxes.

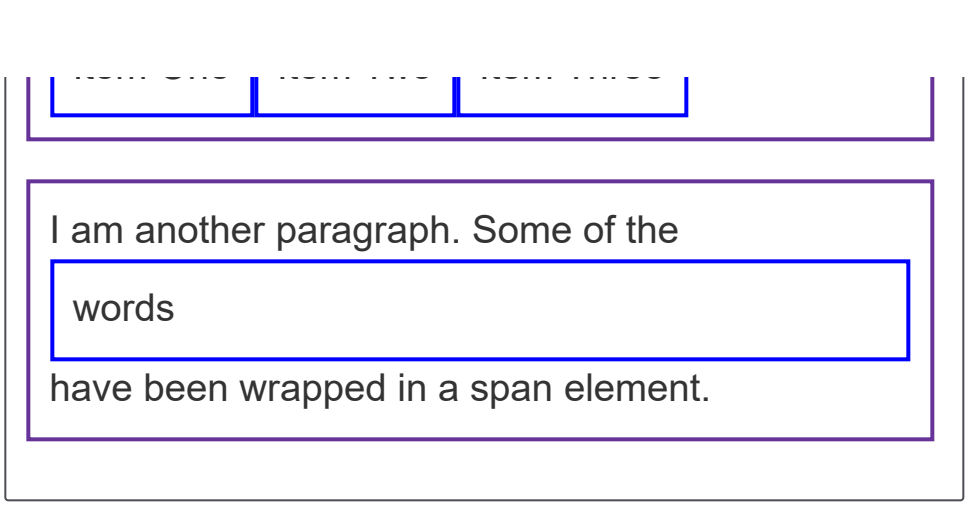
Examples of different display types

Let's move on and have a look at some examples. Below we have three different HTML elements, all of which have an outer display type of `block`. The first is a paragraph, which has a border added in CSS. The browser renders this as a block box, so the paragraph begins on a new line, and expands to the full width available to it.

The second is a list, which is laid out using `display: flex`. This establishes flex layout for the items inside the container, however, the list itself is a block box and — like the paragraph — expands to the full container width and breaks onto a new line.

Below this, we have a block-level paragraph, inside which are two `<span>` elements. These elements would normally be `inline`, however, one of the elements has a class of block, and we have set it to `display: block`.






```
p,
ul {
  border: 2px solid rebeccapurple;
  padding: .5em;
}

.block,
li {
  border: 2px solid blue;
  padding: .5em;
}

ul {
  display: flex;
  list-style: none;
}

.block {
  display: block;
}
```

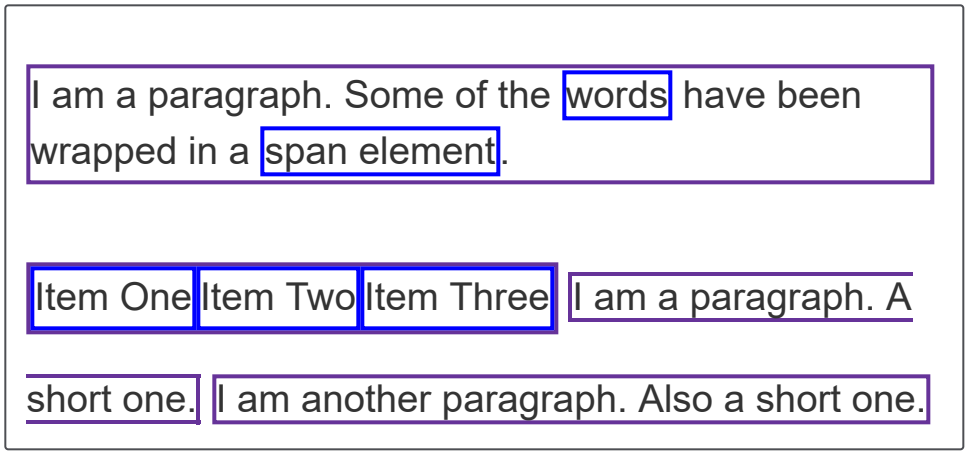
```
<p>I am a paragraph. A short one.</p>
<ul>
  <li>Item One</li>
  <li>Item Two</li>
  <li>Item Three</li>
</ul>
<p>I am another paragraph. Some of the <span
class="block">words</span> have been wrapped in a
<span>span element</span>.</p>
```

We can see how `inline` elements behave in this next example. The `<span>` elements in the first paragraph are inline by default and so do not force line breaks.

We also have a `<ul>` element which is set to `display: inline-flex`, creating an inline box around some flex items.

Finally, we have two paragraphs both set to `display: inline`. The inline flex container and paragraphs all run together on one line rather than breaking onto new lines as they would do if they were displaying as block-level elements.

In the example, you can change `display: inline` to `display: block` or `display: inline-flex` to `display: flex` to toggle between these display modes.




```
p,
ul {
  border: 2px solid rebeccapurple;
}

span,
li {
  border: 2px solid blue;
}

ul {
  display: inline-flex;
  list-style: none;
  padding: 0;
}

.inline {
  display: inline;
}
```

```
<p>
    I am a paragraph. Some of the
    <span>words</span> have been wrapped in a
    <span>span element</span>.
</p>
```

```
<ul>
  <li>Item One</li>
  <li>Item Two</li>
  <li>Item Three</li>
</ul>
<p class="inline">I am a paragraph. A short one.</p>
<p class="inline">I am another paragraph. Also a short
one.</p>
```

Reset

You will encounter things like flex layout later in these lessons; the key thing to remember for now is that changing the value of the `display` property can change whether the outer display type of a box is block or inline, which changes the way it displays alongside other elements in the layout.

In the rest of the lesson, we will concentrate on the outer display type.

What is the CSS box model?

The full CSS box model applies to block boxes, inline boxes only use some of the behavior defined in the box model. The model defines how the different parts of a box — margin, border, padding, and content — work together to create a box that you can see on the page. To add some additional complexity, there is a standard and an alternate box model.

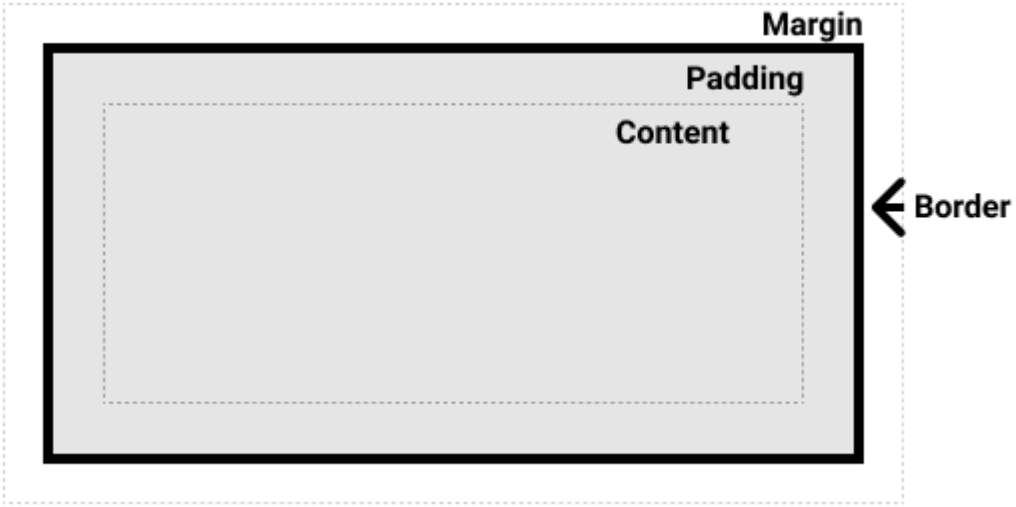
Parts of a box

Making up a block box in CSS we have the:

- **Content box**: The area where your content is displayed, which can be sized using properties like `width` and `height`.
- **Padding box**: The padding sits around the content as white space; its size can be controlled using `padding` and related properties.

- **Border box**: The border box wraps the content and any padding. Its size and style can be controlled using `border` and related properties.
- **Margin box**: The margin is the outermost layer, wrapping the content, padding and border as whitespace between this box and other elements. Its size can be controlled using `margin` and related properties.

The below diagram shows these layers:



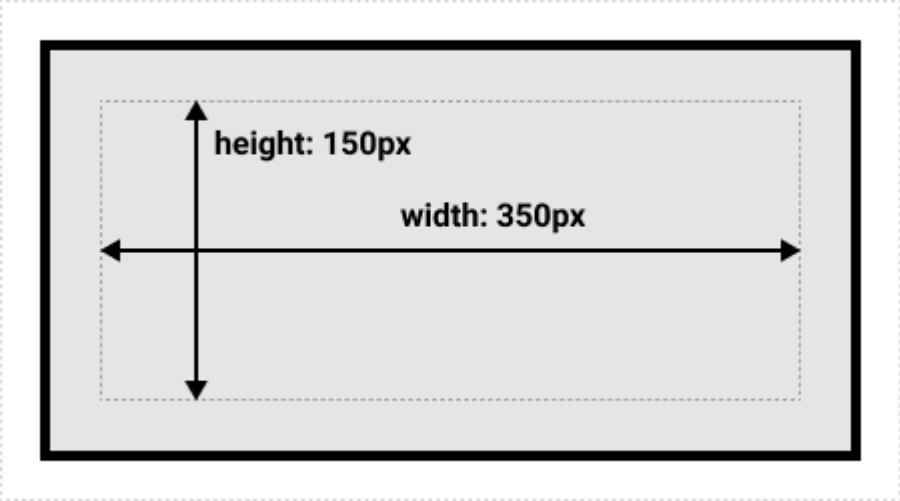
The standard CSS box model

In the standard box model, if you give a box a `width` and a `height` attribute, this defines the width and height of the *content box*. Any padding and border is then added to that width and height to get the total size taken up by the box. This is shown in the image below.

If we assume that the box has the following CSS defining `width`, `height`, `margin`, `border`, and `padding`:

```
.box {
  width: 350px;
  height: 150px;
  margin: 10px;
  padding: 25px;
  border: 5px solid black;
}
```

The space taken up by our box using the standard box model will actually be 410px (350 + 25 + 25 + 5 + 5), and the height 210px (150 + 25 + 25 + 5 + 5), as the padding and border are added to the width used for the content box.



Note: The margin is not counted towards the actual size of the box — sure, it affects the total space that the box will take up on the page, but only the space outside the box. The box's area stops at the border — it does not extend into the margin.

The alternative CSS box model

You might think it is rather inconvenient to have to add up the border and padding to get the real size of the box, and you would be right! For this reason, CSS had an alternative box model introduced some time after the standard box model. Using this model, any width is the width of the visible box on the page, therefore the content area width is that width minus the width for the padding and border. The same CSS as used above would give the below result (width = 350px, height = 150px).

By default, browsers use the standard box model. If you want to turn on the alternative model for an element you do so by setting `box-sizing: border-box` on it. By doing this you are telling the browser to take the border box as the area defined by any size you set.

```
.box {
  box-sizing: border-box;
}
```

If you want all of your elements to use the alternative box model, and this is a common choice among developers, set the `box-sizing` property on the `<html>` element, then set all other elements to inherit that value, as seen in the snippet below. If you want to understand the thinking behind this, see the CSS Tricks article on box-sizing.

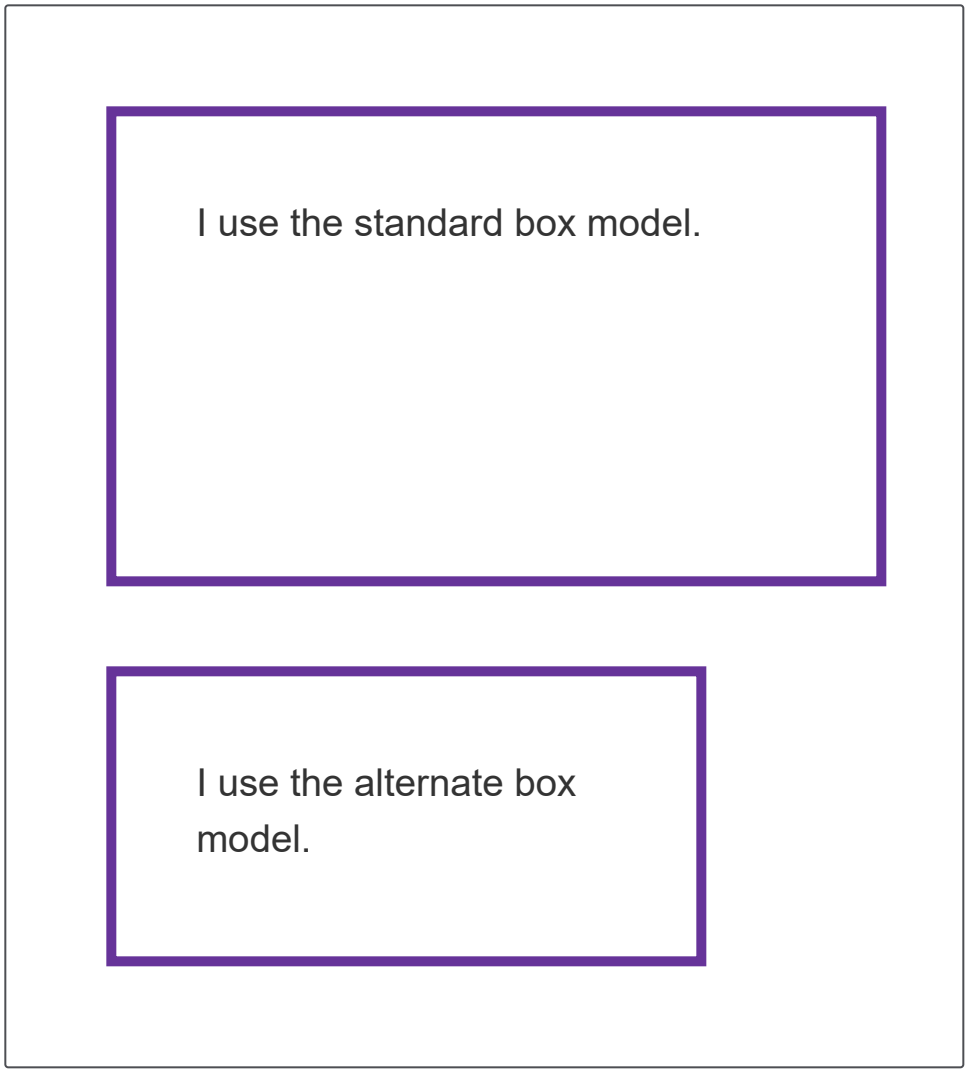
```
html {
  box-sizing: border-box;
}
*, *::before, *::after {
  box-sizing: inherit;
}
```

Note: An interesting bit of history — Internet Explorer used to default to the alternative box model, with no mechanism available to switch.

Playing with box models

In the below example, you can see two boxes. Both have a class of `.box`, which gives them the same `width`, `height`, `margin`, `border`, and `padding`. The only difference is that the second box has been set to use the alternative box model.

Can you change the size of the second box (by adding CSS to the `.alternate` class) to make it match the first box in width and height?




```
.box {
  border: 5px solid rebeccapurple;
  background-color: lightgray;
  padding: 40px;
  margin: 40px;
  width: 300px;
  height: 150px;
}

 alternate {
```

```
.alternate {
  box-sizing: border-box;
}
```

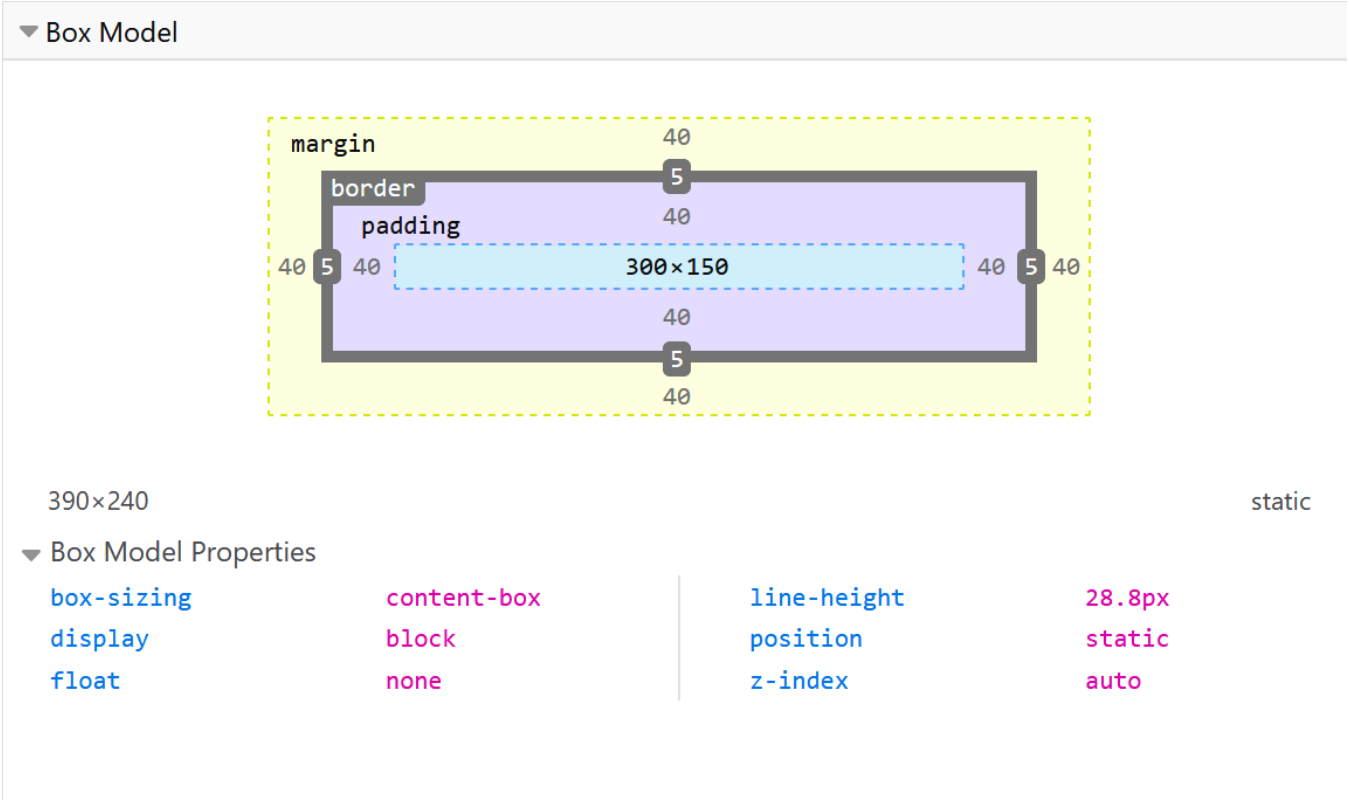
```
<div class="box">I use the standard box model.</div>
<div class="box alternate">I use the alternate box model.
</div>
```

Reset

Note: You can find a solution for this task here.

Use browser DevTools to view the box model

Your browser developer tools can make understanding the box model far easier. If you inspect an element in Firefox's DevTools, you can see the size of the element plus its margin, padding, and border. Inspecting an element in this way is a great way to find out if your box is really the size you think it is!



Margins, padding, and borders

You've already seen the `margin`, `padding`, and `border` properties at work in the example above. The properties used in that example are **shorthands** and allow us to set all four sides of the box at once. These shorthands also have equivalent longhand properties, which allow control over the different sides of the box individually.

Let's explore these properties in more detail.

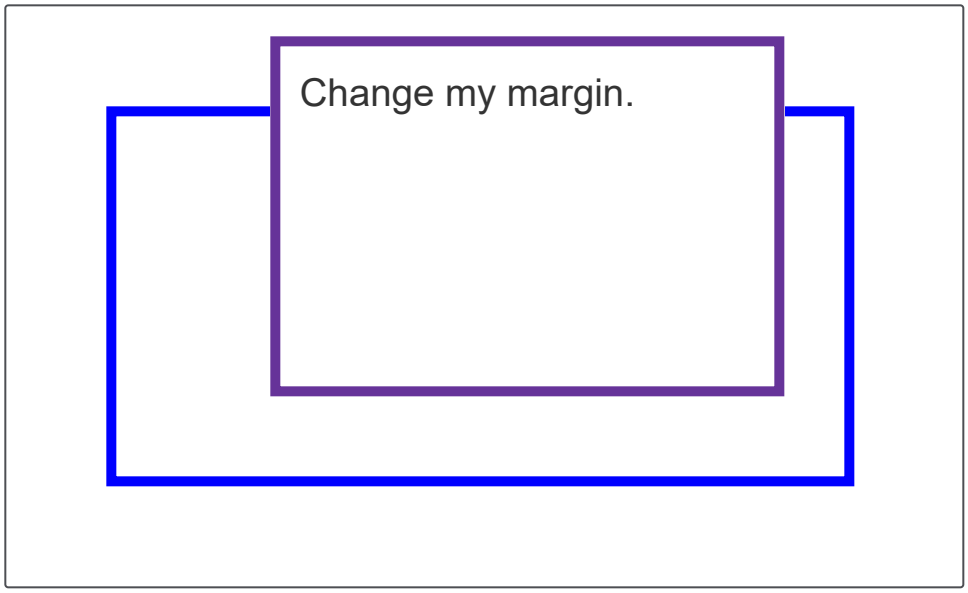
Margin

The margin is an invisible space around your box. It pushes other elements away from the box. Margins can have positive or negative values. Setting a negative margin on one side of your box can cause it to overlap other things on the page. Whether you are using the standard or alternative box model, the margin is always added after the size of the visible box has been calculated.

We can control all margins of an element at once using the `margin` property, or each side individually using the equivalent longhand properties:

- `margin-top`
- `margin-right`
- `margin-bottom`
- `margin-left`

In the example below, try changing the margin values to see how the box is pushed around due to the margin creating or removing space (if it is a negative margin) between this element and the containing element.




```
.box {
  margin-top: -40px;
  margin-right: 30px;
  margin-bottom: 40px;
  margin-left: 4em;
}
```

```
<div class="container">
  <div class="box">Change my margin.</div>
</div>
```

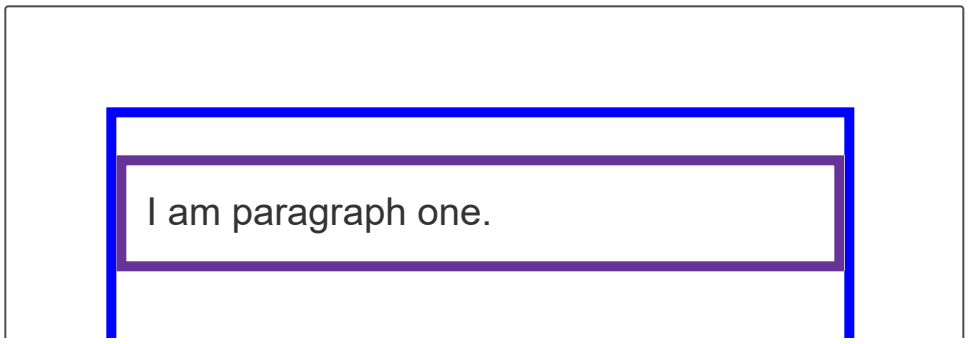
Reset

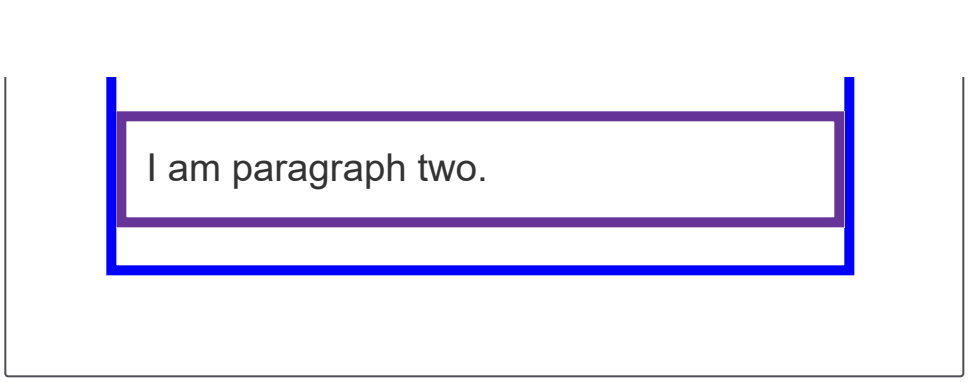
Margin collapsing

A key thing to understand about margins is the concept of margin collapsing. If you have two elements whose margins touch, and both margins are positive, those margins will combine to become one margin, which is the size of the largest individual margin. If one or both margins are negative, the amount of negative value will subtract from the total.

In the example below, we have two paragraphs. The top paragraph has a `margin-bottom` of 50 pixels. The second paragraph has a `margin-top` of 30 pixels. The margins have collapsed together so the actual margin between the boxes is 50 pixels and not the total of the two margins.

You can test this by setting the `margin-top` of paragraph two to 0. The visible margin between the two paragraphs will not change — it retains the 50 pixels set in the `bottom-margin` of paragraph one. If you set it to -10px, you'll see that the overall margin becomes 40px — it subtracts from the 50px.






```
.one {
  margin-bottom: 50px;
}

.two {
  margin-top: 30px;
}
```

```
<div class="container">
  <p class="one">I am paragraph one.</p>
  <p class="two">I am paragraph two.</p>
</div>
```

Reset

There are a number of rules that dictate when margins do and do not collapse. For further information see the detailed page on mastering margin collapsing. The main thing to remember

for now is that margin collapsing is a thing that happens. If you are creating space with margins and don't get the space you expect, this is probably what is happening.

Borders

The border is drawn between the margin and the padding of a box. If you are using the standard box model, the size of the border is added to the `width` and `height` of the box. If you are using the alternative box model then the size of the border makes the content box smaller as it takes up some of that available `width` and `height`.

For styling borders, there are a large number of properties — there are four borders, and each border has a style, width and color that we might want to manipulate.

You can set the width, style, or color of all four borders at once using the `border` property.

To set the properties of each side individually, you can use:

- `border-top`
- `border-right`
- `border-bottom`
- `border-left`

To set the width, style, or color of all sides, use the following:

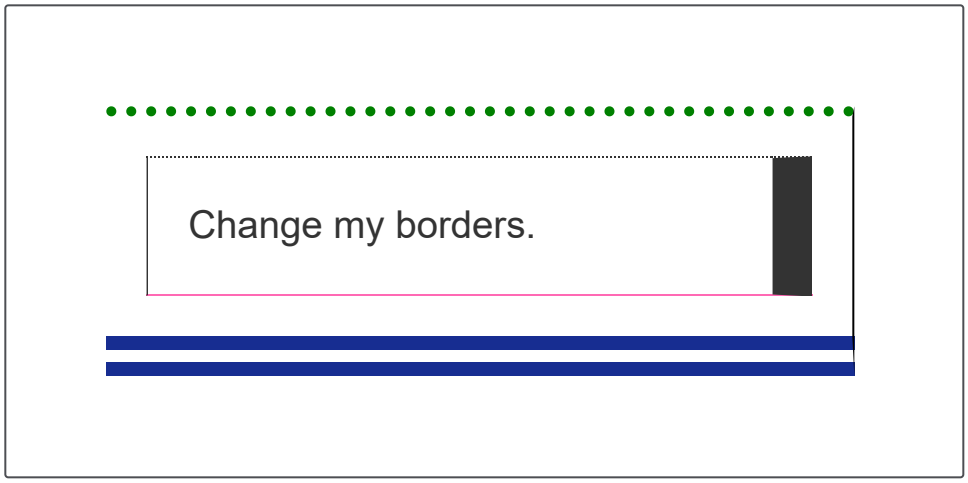
- `border-width`
- `border-style`
- `border-color`

To set the width, style, or color of a single side, you can use one of the most granular longhand properties:

- `border-top-width`
- `border-top-style`
- `border-top-color`
- `border-right-width`
- `border-right-style`

- `border-right-color`
- `border-bottom-width`
- `border-bottom-style`
- `border-bottom-color`
- `border-left-width`
- `border-left-style`
- `border-left-color`

In the example below we have used various shorthands and longhands to create borders. Have a play around with the different properties to check that you understand how they work. The MDN pages for the border properties give you information about the different styles of border you can choose from.




```
.container {
  border-top: 5px dotted green;
  border-right: 1px solid black;
  border-bottom: 20px double rgb(23,45,145);
}

.box {
  border: 1px solid #333333;
  border-top-style: dotted;
  border-right-width: 20px;
  border-bottom-color: hotpink;
}
```

```
<div class="container">
  <div class="box">Change my borders.</div>
</div>
```

Reset

Padding

The padding sits between the border and the content area. Unlike margins you cannot have negative amounts of padding, so the value must be 0 or a positive value. Any background applied to your element will display behind the padding, and it is typically used to push the content away from the border.

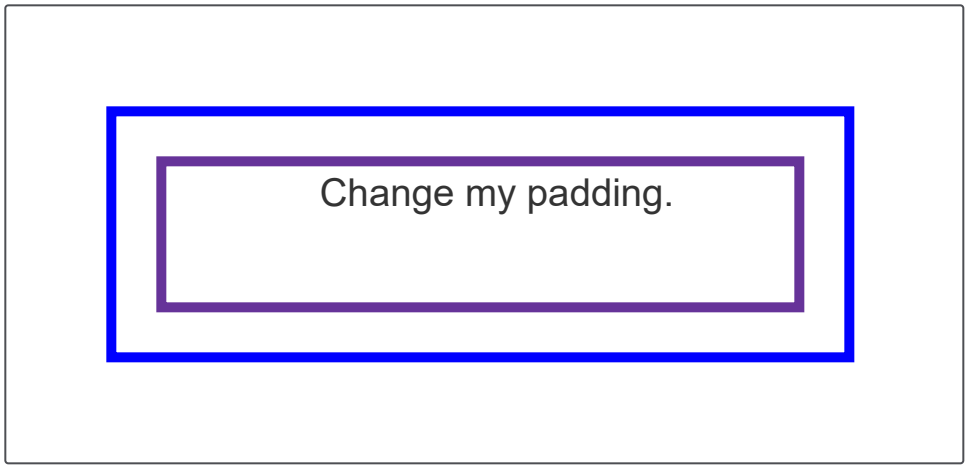
We can control the padding on each side of an element individually using the `padding` property, or each side individually using the equivalent longhand properties:

- `padding-top`
- `padding-right`
- `padding-bottom`
- `padding-left`

If you change the values for padding on the class `.box` in the example below you can see that this changes where the text begins in relation to the box.

You can also change the padding on the class `.container,` which will make space between the container and the box. Padding can be changed on any element, and will

make space between its border and whatever is inside the element.




```
.box {
  padding-top: 0;
  padding-right: 30px;
  padding-bottom: 40px;
  padding-left: 4em;
}

.container {
  padding: 20px;
}
```

```
<div class="container">
  <div class="box">Change my padding.</div>
</div>
```

Reset

The box model and inline boxes

All of the above applies fully to block boxes. Some of the properties can apply to inline boxes too, such as those created by a `<span>` element.

In the example below, we have a `<span>` inside a paragraph and have applied a `width`, `height`, `margin`, `border`, and `padding` to it. You can see that the width and height are ignored. The margin, padding, and border are respected but they do not change the relationship of other content to our inline box and so the padding and border overlaps other words in the paragraph.

I a raph and this is a span inside that paragraph. A span is an inline element and so does not respect width and height.

```
span {
  margin: 20px;
  padding: 20px;
  width: 80px;
  height: 50px;
  background-color: lightblue;
  border: 2px solid blue;
}
```

```
<p>
    I am a paragraph and this is a <span>span</span>
inside that paragraph. A span is an inline element and so
does not respect width and height.
</p>
```

Reset

Using display: inline-block

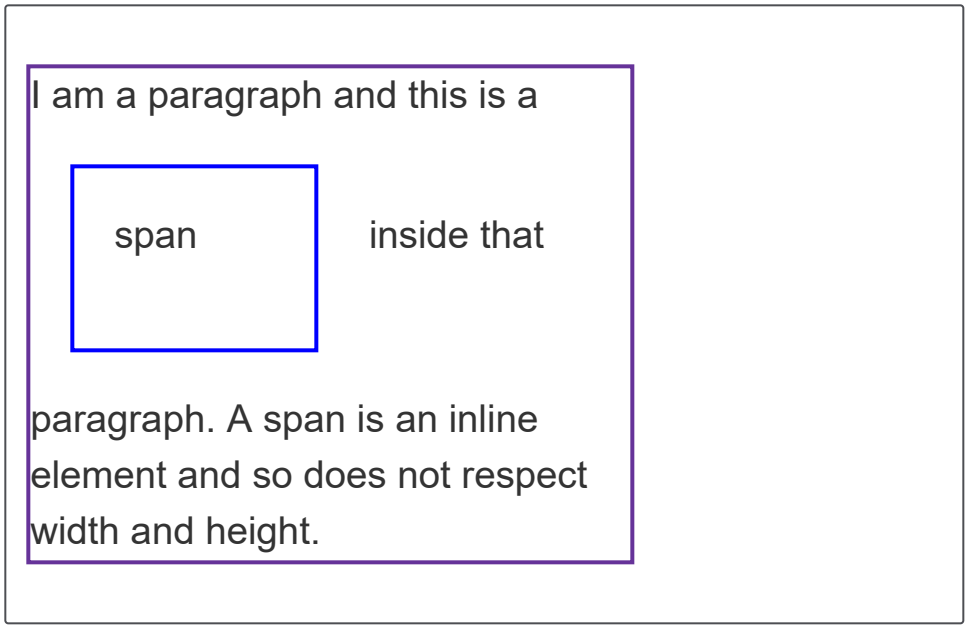
There is a special value of `display`, which provides a middle ground between `inline` and `block`. This is useful for situations where you do not want an item to break onto a new line, but do want it to respect `width` and `height` and avoid the overlapping seen above.

An element with `display: inline-block` does a subset of the block things we already know about:

- The `width` and `height` properties are respected.
- `padding`, `margin`, and `border` will cause other elements to be pushed away from the box.

It does not, however, break onto a new line, and will only become larger than its content if you explicitly add `width` and `height` properties.

In this next example, we have added `display: inline-block` to our `<span>` element. Try changing this to `display: block` or removing the line completely to see the difference in display models.




```
span {
  margin: 20px;
  padding: 20px;
  width: 80px;
  height: 50px;
  background-color: lightblue;
  border: 2px solid blue;
  display: inline-block;
}
```

```
<p>
    I am a paragraph and this is a <span>span</span>
inside that paragraph. A span is an inline element and so
does not respect width and height.
</p>
```

Reset

Where this can be useful is when you want to give a link to a larger hit area by adding `padding`. `<a>` is an inline element like `<span>`; you can use `display: inline-block` to allow padding to be set on it, making it easier for a user to click the link.

You see this fairly frequently in navigation bars. The navigation below is displayed in a row using flexbox and we have added padding to the `<a>` element as we want to be able to change the `background-color` when the `<a>` is hovered. The padding appears to overlap the border on the `<ul>` element. This is because the `<a>` is an inline element.

Add `display: inline-block` to the rule with the `.links-list a` selector, and you will see how it fixes this issue by causing the padding to be respected by other elements.


```
.links-list a {
  background-color: rgb(179,57,81);
  color: #fff;
  text-decoration: none;
  padding: 1em 2em;
}

.links-list a:hover {
  background-color: rgb(66, 28, 40);
  color: #fff;
}
```

```
<nav>
  <ul class="links-list">
    <li><a href="">Link one</a></li>
    <li><a href="">Link two</a></li>
    <li><a href="">Link three</a></li>
  </ul>
</nav>
```

Reset

Summary

That's most of what you need to understand about the box model. You may want to return to this lesson in the future if you ever find yourself confused about how big boxes are in your layout.

In the next lesson, we will take a look at how backgrounds and borders can be used to make your plain boxes look more interesting.

Previous | Overview: Building blocks | Next

In this module

Last modified: Jan 15, 2020, by MDN contributors

Related Topics

Complete beginners start here!

- Getting started with the Web

HTML — Structuring the Web

- Introduction to HTML
- Multimedia and embedding
- HTML tables
- HTML forms

CSS — Styling the Web

- CSS first steps
- CSS building blocks

  CSS building blocks overview

  Cascade and inheritance

  CSS selectors

- The box model
- Backgrounds and borders
- Handling different text directions
- Overflowing content
- Values and units
- Sizing items in CSS
- Images, media, and form elements
- Styling tables
- Debugging CSS
- Organizing your CSS

- Styling text
- CSS layout

JavaScript — Dynamic client-side scripting

- JavaScript first steps
- JavaScript building blocks
- Introducing JavaScript objects
- Asynchronous JavaScript
- Client-side web APIs

Accessibility — Make the web usable by everyone

- Accessibility guides
- Accessibility assessment

Tools and testing

- Cross browser testing

Server-side website programming

- First steps
- Django web framework (Python)
- Express Web Framework (node.js/JavaScript)

Further resources

- Common questions

How to contribute