Box Model

- Content and sizing
- The areas of the box model
- A useful analogy
- Debugging the box model
- Controlling the box model
- Resources
  - User agent stylesheets

Say you have this bit of HTML:

```
<p>I am a paragraph of text that has a few words in it.</p>
```

Then you write this CSS for it:

```
p {
  width: 100px;
  height: 50px;
  padding: 20px;
  border: 1px solid;
}
```

The content would break out of your element and it would be 142px wide, rather than 100px. Why is that? The box model is a core foundation of CSS and understanding how it works, how it is affected by other aspects of CSS and importantly, how you can control it will help you to write more predictable CSS.

https://codepen.io/web-dot-dev/pen/WNRemxN

A really important thing to remember when writing CSS, or working on the web as a whole, is that everything displayed by CSS is a box. Whether that's a box that uses `border-radius` to look like a circle, or even just some text: the key thing to remember is that it's all boxes.

Content and sizing

Boxes have different behavior based on their `display` value, their set dimensions, and the content that lives within them. This content could be even more boxes—generated by child elements—or plain text content. Either way, this content will affect the size of the box by default.

You can control this by using **extrinsic sizing**, or, you can continue to let the browser make decisions for you based on the content size, using **intrinsic sizing**.

Let's quickly look at the difference, using a demo to help us.

> https://codepen.io/web-dot-dev/pen/abpoMBL
>
> Notice that when the box is using extrinsic sizing, there's a limit of how much content you can add before it overflows out of the box's bounds. This makes the word, "awesome", overflow.

The demo has the words, "CSS is awesome" in a box with fixed dimensions and a thick border. The box has a width, so is extrinsically sized. It controls the sizing of its child content. The problem with this though, is that the word "awesome" is too large for the box, so it overflows outside of the parent box's **border box** (more on this later in the lesson). One way to prevent this overflow is to allow the box to be intrinsically sized by either unsetting the width, or in this case, setting the `width` to be `min-content`. The `min-content` keyword tells the box to only be as wide as the intrinsic minimum width of its content (the word "awesome"). This allows the box to fit around "CSS is awesome", perfectly.

Let's look at something more complex to see the impact of different sizing on real content:

> https://codepen.io/web-dot-dev/pen/wvgwOJV

Toggle intrinsic sizing on and off to see how you can gain more control with extrinsic sizing and let the content have more control with intrinsic sizing. To see the effect that intrinsic and extrinsic sizing has, add a few sentences of content to the card. When this element is using extrinsic sizing, there's a limit of how much content you can add before it overflows out of the element's bounds, but this isn't the case when intrinsic sizing is toggled on.

By default, this element has a set `width` and `height`—both `400px`. These dimensions give strict bounds to everything inside the element, which will be honored unless the content is too large for the box in which case visible overflow will happen. You can see this in action by changing the content of the caption, under the flower picture to something that exceeds the height of the box, which is a few lines of content.

When content is too big for the box it is in, we call this overflow. You can manage how an element handles overflow content, using the `overflow` property.

When you switch to intrinsic sizing, you are letting the browser make decisions for you, based on the box's content size. It's much more difficult for there to be overflow with intrinsic sizing because our box will resize with its content, rather than try to size the content. It's important to remember that this is the default, flexible behavior of a browser. Though extrinsic sizing gives more control on the surface, intrinsic sizing provides the most flexibility, most of the time.

The areas of the box model

Boxes are made up of distinct box model areas that all do a specific job.

> *A diagram showing the four main areas of the box model - content box, padding box, border box and margin box*
>
> The four main areas of the box model: content box, padding box, border box and margin box.

You start with **content box**, which is the area that the content lives in. As you learned before: this content can control the size of its parent, so is usually the most variably sized area.

The **padding box** surrounds the content box and is the space created by the `padding` property. Because padding is inside the box, the background of the box will be visible in the space that it creates. If our box has overflow rules set, such as `overflow: auto` or `overflow: scroll`, the scrollbars will occupy this space too.

> https://codepen.io/web-dot-dev/pen/BaReoEV

The **border box** surrounds the padding box and its space is occupied by the `border` value. The border box is the bounds of your box and the **border edge** is the limit of what you can visually see. The `border` property is used to visually frame an element.

The final area, the **margin box**, is the space around your box, defined by the `margin` rule on your box. Properties such as `outline` and `box-shadow` occupy this space too because they are painted on top, so they don't affect the size of our box. You could have an `outline-width` of `200px` on our box and everything inside and including the border box would be exactly the same size.

> https://codepen.io/web-dot-dev/pen/XWprGea

A useful analogy

The box model is complex to understand, so let's recap what you've learned with an analogy.

> *Three photo frames*

In this diagram, you have three photo frames, mounted to a wall, next to each other. The diagram has labels that associate elements of the frame with the box model.

To break this analogy down:

- The content box is the artwork.
- The padding box is the white matte, between the frame and the artwork.
- The border box is the frame, providing a literal border for the artwork.
- The margin box is the space between each frame.
- The shadow occupies the same space as the margin box.

Debugging the box model

Browser DevTools provide a visualisation of a selected box's box model calculations, which can help you understand how the box model works and importantly, how it is affecting the website you're working on.

Go ahead and try this in your own browser:

1. Open DevTools
2. Select an element
3. Show the box model debugger

Controlling the box model

To understand how to control the box model, you first need to understand what happens in your browser.

Every browser applies a user agent stylesheet to HTML documents. The CSS used varies between each browser, but they provide sensible defaults to make content easier to read. They define how elements should look and behave if there's no CSS defined. It is in the user agent styles where a box's default `display` is set, too. For example, if we are in a normal flow, a `<div>` element's default `display` value is `block`, a `<li>` has a default `display` value of `list-item`, and a `<span>` has a default `display` value of `inline`.

An `inline` element has block margin, but other elements won't respect it. Use `inline-block`, and those elements will respect the block margin, while the element maintains most of the same behaviors it had as an `inline` element. A `block` item will, by default, fill the available **inline space** , whereas a `inline` and `inline-block` elements will only be as large as their content.

Alongside an understanding of how user agent styles affect each box, you also need to understand `box-sizing`, which tells our box how to calculate its box size. By default, all elements have the following user agent style: `box-sizing: content-box;`.

Having `content-box` as the value of `box-sizing` means that when you set dimensions, such as a `width` and `height`, they will be applied to the **content box**. If you then set `padding` and `border`, these values will be added to the content box's size.

The actual width of this box will be 260px. As the CSS uses the default `box-sizing: content-box`, the applied width is the width of the content, `padding` and `border` on both sides are added to that. So 200px for the content + 40px of padding + 20px of border makes a total visible width of 260px.

You *can* control this, though, by making the following modification to use the alternative box model, `border-box`:

```
.my-box {
  box-sizing: border-box;
width: 200px;
border: 10px solid;
padding: 20px;
}
```

This alternative box model tells CSS to apply the `width` to the border box instead of the content box. This means that our `border` and `padding` get *pushed in*, and as a result, when you set `.my-box` to be `200px` wide: it actually renders at `200px` wide.

Check out how this works in the following interactive demo. Notice that when you toggle the `box-sizing` value it shows—via a blue background—which CSS is being applied *inside* our box.

https://codepen.io/web-dot-dev/pen/oNBvVpM

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

This CSS rule selects every element in the document and every `::before` and `::after` pseudo element and applies `box-sizing: border-box`. This means that every element will now have this alternative box model.

Because the alternative box model can be more predictable, developers often add this rule to resets and normalizers, like this one.

Resources

- Introduction to the box model
- What are browser developer tools?

User agent stylesheets

- Chromium
- Firefox
- Webkit

□□□ #3
□ □□□ □□□ 2026-10-31 22:45:22 UTC
□ cocobook-sync □□□ 2026-11-04 17:39:07 UTC