Sizing items in CSS

- The natural or intrinsic size of things
- Setting a specific size
  - Using percentages
  - Percentage margins and padding
- min- and max- sizes
  - max-width on images
- Viewport units
- Summary

In the various lessons so far, you have come across several ways to size items on a web page using CSS. Understanding how big the different features in your design will be is important. So, in this lesson, we will summarize the various ways elements get a size via CSS and define a few terms about sizing that will help you in the future.

Prerequisites:	HTML basics (study Basic HTML syntax), CSS basic syntax, CSS selectors.
Learning outcomes:	• Understand the concept of intrinsic size. • Setting absolute and percentage sizes. • Setting maximum and minimum width and height. • Understand viewport units, and why they are useful.

The natural or intrinsic size of things

HTML Elements have a natural size, set before they are affected by any CSS. A straightforward example is an image. An image file contains sizing information, described as its **intrinsic size**. This size is determined by the image *itself*, not by any formatting we happen to apply.

If you place an image on a page and do not change its height or width, either by using `<img>` attributes or CSS, it will be displayed using that intrinsic size. We have given the image in the example below a border so that you can see the extent of its size as defined in its file.

```
<img
  alt="star"
  src="https://mdn.github.io/shared-assets/images/examples/big-star.png" />
```

```
img {
  border: 5px solid darkblue;
}
```

An empty div, on the other hand, has no size of its own. If you add a div to your HTML with no content, then give it a border as we did with the image, you will see a line on the page. This is the `<div>`'s collapsed border — there is no content to hold it open.

In our example below, that border covers the entire width of the container, because it is a block-level element, a behavior that should be starting to become familiar to you. It has no height (or size in the block dimension) because it has no content.

```
<div class="box"></div>
```

```
.box {
  border: 5px solid darkblue;
}
```

In the example above, try adding some text inside the empty element. You'll see the border open up because the height of the element is defined by the content. Again, this is the intrinsic size of the element — its size is defined by its content.

Setting a specific size

We can, of course, give elements in our design a specific size. When a size is given to an element (the content of which then needs to fit into that size) we refer to it as an **extrinsic size**.

In the next example, we give two `<div>`s specific width and height values, and they will now have that size no matter what content is placed inside them. As the right-hand `<div>` demonstrates, a set height can cause content to overflow if there is more content than will fit inside its containing element (you'll learn more about overflow in a subsequent lesson).

```
<div class="wrapper">
  <div class="box"></div>
  <div class="box">
    These boxes both have a height set, this box has content in it which will
    need more space than the assigned height, and so we get overflow.
  </div>
</div>
```

```
body {
  font: 1.2em sans-serif;
}
.wrapper {
  display: flex;
}

.wrapper > * {
  margin: 20px;
}

.box {
  border: 5px solid darkblue;
  height: 100px;
  width: 200px;
}
```

Due to this problem of overflow, fixing the height of elements with lengths or percentages is something we need to do very carefully on the web.

Using percentages

In many ways, percentages act like length units, and as we discussed in the lesson on values and units, they can often be used interchangeably with lengths. When using a percentage, you need to be aware of what it is a percentage *of*. In the case of a box inside another container, if you give the child box a percentage width, it will be a percentage of the width of the parent container.

```
<div class="container">
  <div class="box">I have a percentage width.</div>
</div>
```

```
body {
  font: 1.2em sans-serif;
}

.box {
  border: 5px solid darkblue;
  width: 50%;
}
```

This is because percentages resolve against the size of the containing block. With no percentage applied, our `box` `<div>` takes up `100%` of the available space, as it is a block-level element. If we give it a percentage width, this becomes a percentage of the space it would normally fill.

Try editing the above example:

1. Remove the `box` `<div>`'s `width` declaration to verify that it takes up `100%` of the available `width` by default.
2. Revert your previous change — give the `box` `<div>` a `width` of `50%` again.
3. Now give the `container` `<div>` a `width` of `50%`. You'll see that the `box` `<div>`'s `width` gets smaller, because it is relative to the `width` of its container.

Percentage margins and padding

If you set `margins` and `padding` as a percentage, you may notice some strange behavior.

In the below example, we have a box, on which we've set a margin of 10% and a padding of `10%`. The padding and margin on the top and bottom of the box are the same size as the padding and margin on the left and right.

```
<div class="box">I have margin and padding set to 10% on all sides.</div>
```

```
body {
  font: 1.2em sans-serif;
}
.box {
  border: 5px solid darkblue;
  width: 200px;
  margin: 10%;
  padding: 10%;
}
```

You might expect the percentage top and bottom margins to be a percentage of the element's height, and the percentage left and right margins to be a percentage of the element's width. However, this is not the case!

When you use margin and padding set in percentages, the value is calculated from the **inline size** of the containing block — therefore, the width when working in a horizontal language. In our example, all of the margins and padding are `10%` of the width. This means you can have equal-sized margins and padding all around the box. This is a fact worth remembering if you use percentages in this way.

min- and max- sizes

In addition to giving things a fixed size, we can ask CSS to give an element a minimum or a maximum size. If you have a box that might contain a variable amount of content, and you always want it to be *at least* a certain height, you could set the min-height property on it. The box will always be at least this height, but will then grow taller if there is more content than the box has space for at its minimum height.

In the next example, you can see two boxes, both with a defined `min-height` of 100 pixels. The box on the left is 100 pixels tall; the box on the right has content that needs more room, and so it has grown taller than 100 pixels.

```
<div class="wrapper">
  <div class="box"></div>
  <div class="box">
    These boxes both have a min-height set. This box has content in it, which
    will need more space than the assigned height, and so it grows from the
    minimum.
  </div>
</div>
```

```
body {
  font: 1.2em sans-serif;
}
.wrapper {
  display: flex;
  align-items: flex-start;
}

.wrapper > * {
```

```
  margin: 20px;
}

.box {
  border: 5px solid darkblue;
  min-height: 100px;
  width: 200px;
}
```

This is very useful for avoiding overflow when dealing with variable amounts of content.

`max-width` on images

A common use of max-width is to cause images to scale down if there is not enough space to display them at their intrinsic width, while making sure they don't become larger than that width.

As an example, if you were to set `width: 100%` on an image, and its intrinsic width was smaller than its container, the image would be forced to stretch and become larger, causing it to look pixelated.

If you instead use `max-width: 100%`, and its intrinsic width is smaller than its container, the image will not be forced to stretch and become larger, thus preventing pixelation.

In the example below, we have embedded the same image three times:

- The first image has been given `width: 100%` and is in a container which is larger than it, therefore it stretches to the container width.
- The second image has `max-width: 100%` set on it and therefore does not stretch to fill the container.
- The third box contains the same image again, also with `max-width: 100%` set; in this case you can see how it has scaled down to fit into the box.

```
<div class="wrapper">
  <div class="box">
    <img
      alt="star"
      class="width"
      src="https://mdn.github.io/shared-assets/images/examples/big-star.png" />
  </div>
  <div class="box">
    <img
      alt="star"
      class="max"
```

```
      src="https://mdn.github.io/shared-assets/images/examples/big-star.png" />
  </div>
  <div class="mini-box">
    <img
      alt="star"
      class="max"
      src="https://mdn.github.io/shared-assets/images/examples/big-star.png" />
  </div>
</div>
```

```
.wrapper {
  display: flex;
  align-items: flex-start;
}

.wrapper > * {
  margin: 20px;
}

.box,
.mini-box {
  border: 5px solid darkblue;
}
```

```
.box {
  width: 200px;
}
.mini-box {
  width: 30px;
}
.width {
  width: 100%;
}
.max {
  max-width: 100%;
}
```

This technique is used to make images *responsive*, so that when viewed on a smaller device they scale down appropriately. You should, however, not use this technique to load really large images and then scale them down in the browser. Images should be appropriately sized to be no larger

than they need to be for the largest size they are displayed in the design. Downloading overly large images will cause your site to become slow, and it can cost users more money if they pay for data by the megabyte.

Viewport units

The viewport — which is the visible area of your page in the browser you are using to view a site — also has a size. In CSS, we have units that relate to the size of the viewport — the `vw` unit for viewport width, and `vh` for viewport height. Using these units, you can size something relative to the viewport of the user.

`1vh` is equal to `1%` of the viewport height, and `1vw` is equal to `1%` of the viewport width. You can use these units to size boxes, but also text. In the example below we have a box which is sized as `20vh` and `20vw`. The box contains a letter `A`, which has been given a font-size of `10vh`.

```
<div class="box">A</div>
```

```
body {
  font-family: sans-serif;
}

.box {
  border: 5px solid darkblue;
  width: 20vw;
  height: 20vh;
  font-size: 10vh;
}
```

Changing the `vh` and `vw` values will change the size of the box and font, respectively; changing the viewport size will also change the box and font sizes because they are sized relative to the viewport. To see the box and text size change when you adjust the viewport size, vw-vh and resize the browser window.

Sizing things according to the viewport can be useful in your designs. For example, if you want a full-page banner to show before the rest of your content, making that part of your page `100vh` high will push the rest of the content below the viewport, meaning that it will only appear once the document is scrolled.

Summary

This lesson has given you a rundown of some key issues that you might run into when sizing things on the web. When you move on to CSS Layout, sizing will become very important in mastering the different layout methods, so it is worth understanding the concepts here before moving on.

In the next article, we'll give you some tests that you can use to check how well you've understood and retained the information we've provided on CSS sizing.

□□□ #3
□ □□□ □□□ 2026-10-31 03:27:53 UTC
□ cocobook-sync □□□ 2026-11-04 17:32:57 UTC