Sizing and dimensions

Sizing and dimensions

We have already touched on the size properties in the various discussions about display and positioning. But here we'll cover them properly and add a few more.

Default behavior

The default sizing behavior depends upon the display property for an element.

inline

Inline elements take the size of their content plus any padding. Additionally, inline elements ***ignore*** any explicit sizing properties (`width`, `height`, etc.) unless they are also `position:absolute` or `position:fixed`. This leads to a lot of confusion when newbies are working with inline elements. If you have an inline element whose size you want to indicate explicitly, you should probably change it to inline-block.

inline-block

Inline-block elements also take the size of their content, plus padding. However, they respect any explicit sizing properties. This is handy.

block

By default when no sizing properties are used, block level elements take the width of their parent and the height of their content. Block level elements respect any explicit sizing properties.

The "width of parent" aspect of block level elements occasionally surprises developers who might not expect that each animal in a modest list of pets extends all the way to the right edge of the browser.

These display states are covered in the display section.

images - aspect ratio preserving

Images have an interesting behavior in that if only one dimension is set, the other is automatically calculated so that the original aspect ratio of the image is preserved. This is true for both decorative CSS images and `<img>` tags.

sizing properties

There are six sizing properties. They are

- `width`
- `min-width`
- `max-width`
- `height`
- `min-height`
- `max-height`

The width and height properties are a simple way to explicitly set the width or height of an element. It is set directly, and the element maintains that dimension (unless it is inline and ignores these properties). And certainly, when dealing with images, there is little reason to pursue anything but the most straightforward approach.

However, if you look again at the descriptions for inline-block and block level elements above, you will notice that inline-block elements are *sized (height and width) to their content*. And block level elements take the *width of their parent* and the *height of their content*. So these elements are fundamentally **variably** sized, and this variability is one of the most powerful and useful aspects of these elements.

However, when we use an explicit width or height property, we remove that variability from the element. This makes it less powerful and less useful.

The `min-width` and `min-height` properties allow us to set a minimum boundary for that variability, but otherwise the variable sizing of the element is unimpeded. So if we have `min-width:300px;` that means the element will be 300 pixels or possibly wider.

Likewise the `max-width` and `max-height` properties allow us to set a maximum boundary for the variability.

As we move into flexbox based layouts, variability in our design will become very important.

Best practice

Unless you have good cause, try to avoid using explicit dimension properties like `width` and `height`. If you must control the dimensions, consider using the min- or max- variants.

Cropping and scrolling: overflow

If the element's dimensions are overdetermined by the sizing properties, then its content may not fit. In the example below, the width and height of the paragraph have been set too small for its content. As a result, the content overflows the rectangle of the paragraph. We've made this easier to see by adding a border and background color to the paragraph.

This default behavior, that content that doesn't fit is shown anyway, can be surprising if you weren't expecting it.

CSS	**Result**

`p {`
`  width:  250px;`
`  height: 200px;`
`  border: 1px solid black;`
`  background-color: lightblue;`
` }`
	For a long time I used to go to bed early. Sometimes, when I had put out my candle, my eyes would close so quickly that I had not even time to say "I'm going to sleep." And half an hour later the thought that it was time to go to sleep would awaken me; I would try to put away the book which, I imagined, was still in my hands, and to blow out the light; I had been thinking all the time, while I was asleep, of what I had just been reading, but my thoughts had run into a channel of their own, until I myself seemed actually to have become the subject of my book: a church, a quartet, the rivalry between François I and Charles V. This impression would persist for some moments after I was awake; it did not disturb my mind, but it lay like scales upon my eyes and prevented them from registering the fact that the candle was no longer burning. Then it would begin to seem unintelligible, as the thoughts of a former existence must be to a reincarnate spirit; the subject of my book would separate itself from me, leaving me free to choose whether I would form part of it or no; and at the same time my sight would return and I would be astonished to find myself in a state of darkness, pleasant and restful enough for the eyes, and even more, perhaps, for my mind, to which it appeared incomprehensible, without a cause, a matter dark indeed.

overflow

The overflow properties govern this situation. There are three related properties: `overflow`, `overflow-x`, and `overflow-y`.

```
p { overflow: auto; }
```

With common text, overflowing normally only occurs in the vertical direction (like in the example above). But if your element contains images, extremely long words, or has adjusted CSS white spacing properties, then content can overflow horizontally as well. The `overflow` property applies a common policy to both situations, and the `overflow-x` and `overflow-y` properties let you assign different policies for horizontal versus vertical overflow.

There are five possible values: `unset`, `auto`, `visible`, `hidden`, and `scroll`. In the example below, the paragraphs are limited to a height of 100 pixels.

1. `unset` is both the default value when overflow has not been set and a value that can be explicitly set.
2. The interpretation for the `auto` value may vary from browser to browser. Typically, if a scroll bar is needed, it is shown, but if it is not needed, no scroll bar is shown. In the example below, no horizontal scroll bar is needed, so none is shown. If there was less content in the paragraph, then no scroll bar would be shown at all.
3. When the value is `scroll`, then the scroll bars are *always* shown, whether they are needed or not.

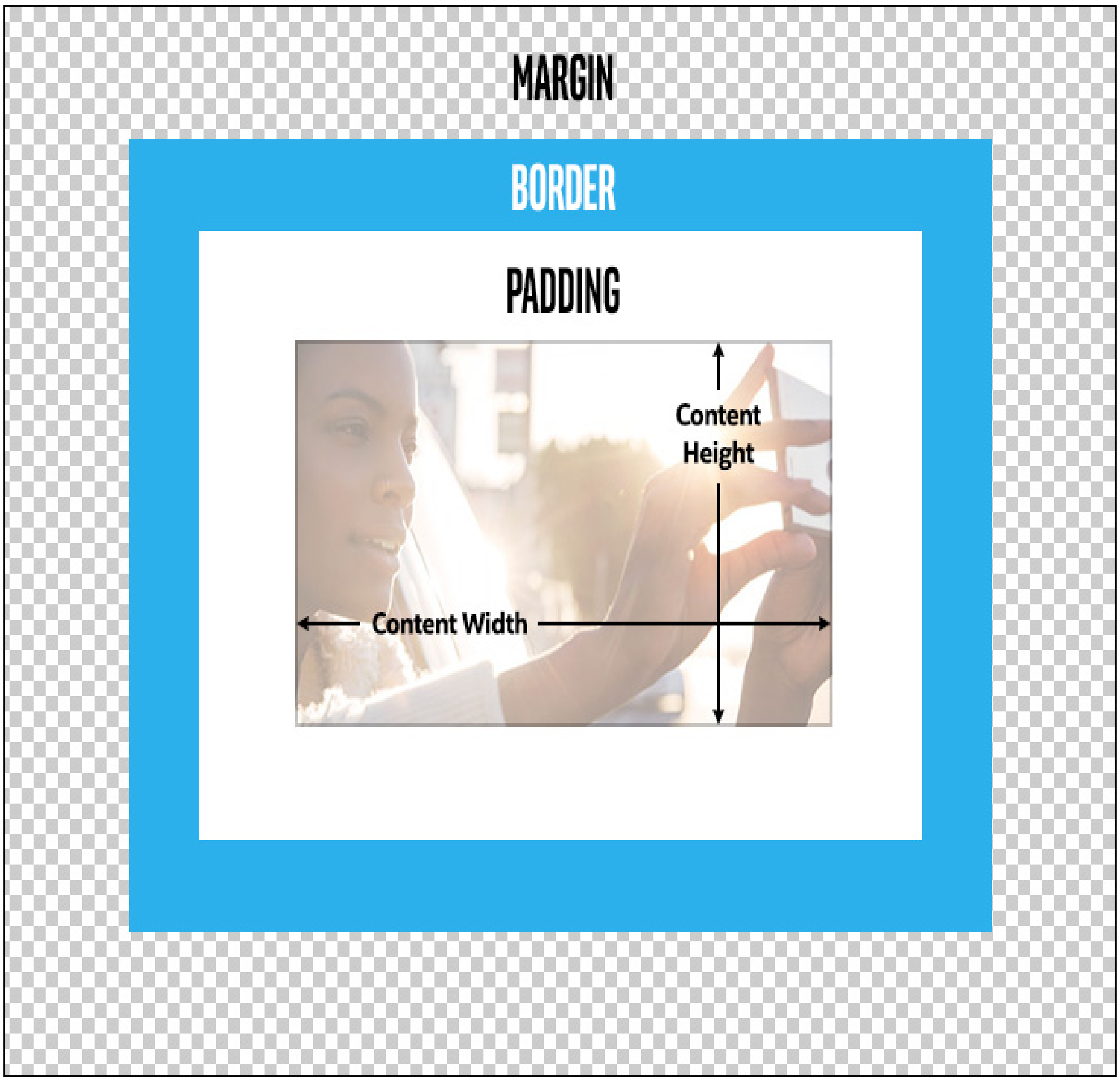
unset	**auto**	**visible**	**hidden**	**scroll**
For a long time I used to go to bed early. Sometimes, when I had put out my candle, my eyes would close so quickly that I had not even time to say "I'm going to sleep."	For a long time I used to go to bed early. Sometimes	For a long time I used to go to bed early. Sometimes, when I had put out my candle, my eyes would close so quickly that I had not even time to say "I'm going to sleep."	For a long time I used to go to bed early. Sometimes, when I had put	For a long time I used to go to bed early.

The box model and box-sizing

So, if we say that some block level element is supposed to have a height and width of 100 pixels, does that include the border or the padding? This is an excellent question, worthy of some experimentation. The reader is encouraged to explore this.

The answer is that the default behavior of the browser is that the sizing properties govern the size of the content area and any padding or borders are "extra". But, if this is not the desired behavior, you can change it.

Every element has several "boxes" it manages: its own content, padding, border, and margins. In CSS parlance, this question is about the "Box Model" of the element. Here is an illustration of how the different boxes are organized (innermost to outermost).



box-sizing

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

The `box-sizing` property determines how the sizing properties are applied. It has two values: `content-box` and `border-box`.

The `content-box` value is default and simply means that the height or width properties affect the content box of the element and any padding or border is "additional".

When `border-box` is used, the sizing properties are used to set the "whole" size of the element, and the content size is likely to be less.

This is making my head hurt! How important is this?

Another good question. If you are manipulating items with JavaScript, then it may be important. If you are using any of the "older" methods for CSS layout (like floats, tables, etc.), then managing the box model is of paramount importance.

However, if you are using the flexbox layout (which we begin in the next section), then the box-model is not that important. The rule of thumb is that the more you are directly managing the size of items, the more likely you will need to change the `box-sizing` property to be `border-box`.