Why won't this work?

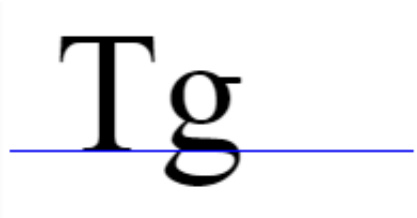
When newbie developers are groping around CSS blindly, they often stumble upon a variety of CSS properties that could be used to alter the positioning or size of an element such as `left`, `top` and`margin`. But when using the properties, these developers become bewildered when they don't behave consistently. Sometimes the properties work, sometimes they don't, sometimes they do the*opposite* of what it is doing in a different rule. We have not covered properties like `left` and `top` yet, but we have introduced `margin`and the intrepid reader may have already discovered in their exercises that `margin` can have some unexpected behavior. Why is that?

The answer has to do with the two CSS properties `display` and`position`. The `display` property, in particular, has different default values for different tags, some tags start with`display:block`, and others are `display:inline`. They behave very differently. These two properties (`display` and `position`) will often change how an element responds to certain other layout properties. And when this is not understood, then it may just seem random to a developer struggling to get stuff to work.

So, let's begin by understanding the very important difference between block and inline display. And that begins with the baseline.

BASELINE

The text "baseline" is a key concept to understanding how the browser makes it layout decisions.



In the image above we see two text characters set next to each other, the blue line indicates the baseline. The baseline determines how and where the characters are positioned. Note that the the tail of the "g" hangs below the baseline.

The baseline is never drawn by the browser, it is not exposed directly to you as a

developer, and CSS only arguably has any properties concerning it, but it governs the placement of all the inline elements.

DISPLAY: BLOCK VERSUS INLINE

As the browser is rendering your page, every time it encounters the next tag it has a simple question: "Do I give this element it's own line?" For example, every <p> tag gets a new line, but <a> tags do not.

This is the key distinction between "block" level elements (like the <p> tag) and "inline" elements (like the <a> tag). Here is a quick table of the default values for some of the tags we've learned already.

Block	Inline
• p • h1 • div • blockquote • ul • ol • li	• a • span • q • i • b

Here are some differences between block level and inline elements.

Block Level

- appears below and to the left of their block level neighbors (like a carriage return on a typewriter taking to the next new line)
- **will expand to fill the width of the parent container by default**
- respects all margin properties
- can have its `width` property set, which will make it narrower and cause its children to wrap, but not crop. (We'll cover this later)
- takes on the height of all its children (pending certain exceptions) so long as its

own `height` is unset. (We will cover setting the height later)

- ignores the `vertical-align` property

Inline Elements

- simply appear to the right of their preceding inline neighbor. They do not drop to the next line unless they must to "wrap"
- **by default, the width is simply the width of the content of the element, plus any padding**
- **ignore top and bottom margin settings**
- **ignore `width` and `height` properties**
- are subject to `vertical-align` property as well as CSS `white-space` settings
- support `padding`, but any `padding-top` or `padding-bottom`does **not** contribute to the calculation of the height of the text line it sits upon
- inline element cleave to the baseline where they are being placed.

The last bullet about inline elements is one of the most important to understand. *Inline elements cleave to the baseline*. This is very important to understand why inline elements are positioned vertically the way they are. It also contributes to why they ignore top and bottom margins. Note the making an inline element "bigger" with padding will certainly keep its neighbors away horizontally. But if there is a neighboring text line above or below, it can only be kept at bay with the `line-height` property, not margins or padding.

Below we see a span that has padding, margin-top and background-color applied, but no extra room is being made for it above or below, so its background is overlapping the lines above and below.

CSS	Result
`span { margin-top: 15px; /* ignored */ padding:15px; background-color: lightblue; }`	Nothing could hinder it but her love of extremes, and her insistence on regulating life according to notions which might cause a wary man to hesitate before he made her an offer, or even might lead her at last to refuse all offers.

So here we prevent the overlap by setting the `line-height` of the span. However, this solution should not be considered optimal. Better is to change the span to be `display:inline-block`, which is discussed next.

CSS	Result
`span { margin-top: 15px; /* still ignored */ padding:15px; background-color: lightblue; line-height: 42px; /* fix */ }`	Nothing could hinder it but her love of extremes, and her insistence on regulating life according to notions which might cause a wary man to hesitate before he made her an offer, or even might lead her at last to refuse all offers.

inline-block

The astute reader may have spotted an obvious omission from the table of block and inline elements above: <img> . Is <img> a block level element or inline? If you venture to experiment you may conclude "both", and you will be right.

For historic reasons, the <img> tag defaults to `display:inline` in most browsers. If you inspect using the browsers inspector, that's what you will see. However, it does not follow the same rules as other inline elements. In fact, regardless of what the inspector says, images are special cased and are inline-block.

Inline-block elements still cleave to the text baseline of the line they are on. If top or bottom margins or paddings are used, then the entire line is adjusted to make room. (So the line-height does not need to be used)

- `inline-block` elements respect `margin-top` and `margin-bottom`
- the vertical padding for inline-block elements contributes to the calculation of the height of the line it falls on
- inline-block elements respect `width` and `height` properties

In some browsers, some of the form elements default to inline block (like `<button>`, `<select>`, and `<input>`)

Here is the overlapping background style presented again, but this time instead of using line-height to solve the problem, we simply make the span element `display:inline-block`. Note that the`margin-top` is also respected.

CSS	Result
`span { margin-top: 15px; /* no longer ignored */ padding:15px; background-color: lightblue; display:inline-block; /* fix */ }`	Nothing could hinder it but her love of extremes, and her insistence on regulating life according to notions which might cause a wary man to hesitate before he made her an offer, or even might lead her at last to refuse all offers.

THE DISPLAY PROPERTY

At long last we arrive at the `display` property. We have now seen three of its possible values: `block`, `inline`, and `inline-block`. There are others (like `none` and `flex`) and we will cover them later.

```
.name { display: inline-block; }
```

A key to not getting confounded by the display property is to have a grasp on which elements default to which display value and appreciating the differences between block, inline and inline-block display.