@adamfschwartz
Code on GitHub

THE MAGIC OF CSS

CHAPTER 3

Tables

`table-layout`

In the previous chapter we discussed layout. But what we meant by that is the construction of your content from a design perspective—how you structure your app geometrically to make sense for your use case. Think the Two Pane App potion.

But `layout` has a specific meaning in various contexts. In the context of tables, it means how the browser decides to size columns and rows of a `table` element based on the CSS applied by the user agent and you, and the content within each table cell.

This process is truly *magical*.

A complex layout algorithm is used for both the horizontal and vertical. And these algorithms fork early based on the `table-layout` you specify, of which there are two options:

`auto`	*The default*. I attempt to size columns relatively to each other by the widest cell in each column, unless you give me specific widths, at which point I use the

widths you specify to make relative comparisons.
(CSS spec)

`fixed`	I attempt to size columns evenly, unless you give me specific widths in `px`, at which point I attempt to honor your sizing exactly, unless I can't because your math doesn't work out. (CSS spec)

These are very rough definitions, and definitely not complete. I highly recommend you read through the spec at some point to get a better understanding. But nothing is better than playing with live code, so let's look at some examples to get a clearer picture.

Example 1: No widths

`table-layout: auto`

This is the title of some object	Action
This is the title of another object	Action

`table-layout: fixed`

This is the title of some object	Action
This is the title of another object	Action

Notice how in the `fixed` case, the columns are sized evenly since no widths are specified, but in the `auto` cased they're sized proportionally by the width of the cell contents.

Example 2: Percentage widths

Now let's look at the same example with column widths set to `20%` and `50%`, respectively.

`table-layout: auto`

This is the title of some object	Action
This is the title of another object	Action

`table-layout: fixed`

This is the title of some object	Action
This is the title of another object	Action

☐ Toggle `white-space: nowrap`

In both cases, our widths are being taken into account, but only relatively. This is always true with `auto` but it's additionally true here with `fixed` because the widths are specified in percentages. The browser says, "20% is 2/7ths out of the total 20+50%", so when the table is `1000px` wide, the first column ends up at `284px` and the second column at `714px`, roughly a ratio of `2:5`. (It won't be perfectly 2:5 due to `cell-spacing`, `cell-padding`, `border`, `border-spacing`, `border-collapse`, etc., rounding, and other constraints.)

Notice that with `white-space: nowrap` applied to each cell, the auto case compensates but the fixed case lets the text overflow.

Challenge question to think about: *why is first column slightly wider in the* `fixed` *case?*

Example 3: Mixed unit widths

Now let's look at the same example with column widths set to `400px` and `70%`, respectively.

`table-layout: auto`

This is the title of some object	Action
This is the title of another object	Action

`table-layout: fixed`

This is the title of some object	Action
This is the title of another object	Action

Ok.... Since the width of each table is `555px` , there's no way for the browser to fit the columns `400px` and `70% × 555px` into a `555px` –wide table. So it does the best it can.

In the `auto` case, our widths are being taken into account, but only relatively. It compares `400px / 555px` to `70% × 555px` and does the best it can. (The behavior here varies from browser to browser.)

In the `fixed` case, the `400px` is honored, since fixed values are prioritized over percentage based values, and so the second column gets the remainder.

Tabular data

This is a CSS course, so I won't spend much time here. But the main reason to use tables in an application is to display *tabular data*. Tabular data means anything you might display in a spreadsheet. A content matrix.

When it comes to styling tables with tabular data, there are some good general rules to follow:

- Wide tables should be tiger-striped or use a `:hover` background color (or similar) to help the eye associate cells-of-the-same-row.
- Columns of numerical data should be right aligned so that the digits line up.
- Right-most columns may need to be right aligned to avoid a ragged right edge (think `text-align: justify`).
- When possible, row heights should be identical to make vertical scanning easier. (This general principle is known in the biz as "vertical rhythm", and it's very important.)

Check out the table styling potion for an example table design which follows these rules.

Tables as a layout tool

In the previous chapter on layout, we showed that tables can be used to center vertically content of arbitrary height. Until `flex` is widely supported, you should feel comfortable using tables for this. But other than that, if you catch yourself using a `table` to implement layout constructs that don't have to do with tabular data, *you're probably doing it wrong*.

If your browser support is IE10+, then use `flex` [1]. Phillip Walton has a great tutorial[2] on vertical centering with flexbox.

Table gotchas

There are *sooo*[3] many[4] reasons[5] why you shouldn't use tables for anything other than tabular data or vertical centering (as discussed). But to drive that point home, here are some extremely common *gotchas* that make tables frustrating to work with.

Gotcha 1: Table cells do not respect overflows (`table-layout: auto` , Firefox, IE)

This means that even if you use `table-layout: fixed` and specify a pixel width, `overflow: hidden` isn't going to actually work on a table cell in every browser. (If you use `table-layout: auto` , overflows won't be respected in any browser.)

`table-layout: auto`

I'm being told to be `100px` wide and `20px` tall, but I ain't listening.	I'm just some text

Gotcha 2: Table cells don't respect relative positioning (Firefox)

Yup. You heard me correctly. You go apply `position: relative` to a table cell, place a `position: absolute` element inside, and in Firefox, the absolute element will be positioned relative to the earliest positioned parent of the table instead. Bummer.

The bug was reported in 2000.[6]

	I'm just some text

I am `position: absolute` with `right: 40px`

Gotchas tl;dr

If, after evaluating the options, you believe that using a `table` element is the right way to go, just make sure you wrap the contents of every table cell with a `div`. This way you have all of the styling control you need for each cell while still being able to utilize the extremely powerful—albeit confusing—table layout engine.

Further reading

- w3: CSS2 Tables specification
- Drewish: Vertical rhythm tool

Citations

1. Can I use: flex
2. Solved by Flexbox: Vertical Centering
3. Seybold Seminars: Why tables for layout is stupid
4. Smashing Magazine: Table Layouts vs. Div Layouts: From Hell to… Hell?
5. Vaneso Design: Are CSS Tables Better Than HTML Tables?
6. Mozilla Bugzilla: relative positioning of table cells doesn't work

Chapter 2: Layout Chapter 4: Color