callmenick.com

CSS Equal Height Columns, Three Different Ways - Call Me Nick

Nick Salloum

Get Source View Demo

Achieving equal height columns with just CSS is such a common scenario with a few simple solutions. In projects that follow a standard content-sidebar layout, it just helps to have both of the containers seemingly stretch to the bottom of the page. It just looks better, cleaner, and more complete. I'm going to highlight three different ways that this can be achieved, using strictly CSS. No JavaScript, no jQuery to calculate the height of one and set it on the other. Just plain old beautiful CSS.

Diving In

It's common for designs/developers to want to achieve this look just purely for aesthetic purposes. It often finishes the page nicer, and makes everything seem more connected. The reason it doesn't work off the bat is because of floated elements. A container that contains floated elements will expand to the height of the tallest element, only if that container is cleared (I use the common `clearfix` method, so should you). This means that setting the column heights to 100% will have no effect, because the containing element doesn't actually have a height. Fear not though, simple solutions await us. Let's first look at the markup we'll be using for the first two examples.

```
<div class="main">
    <div class="container clearfix">
        <div class="content">
            <section>
                <h1>This is the Main Content</h1>
                <hr>
                <h2>A sub heading</h2>
                <p>Lorem ipsum dolor sit amet,
consectetur...</p>
            </section>
        </div>
        <div class="sidebar">
            <aside>
                <h2>This is a sidebar</h2>
                Sign up to the newsletter!
            </aside>
        </div>
    </div>
</div>
```

Now, let's dig into each of our three examples.

1) Equal Height Columns Using Margins, Paddings, and Overflow

This first method uses margins, paddings, and overflow to force the columns to be equal heights. The methodology entails setting a big enough padding at the bottom of each floated element, and countering it with an equal negative margin at the bottom of the same elements. The trick is to set the overflow on the parent container to hidden. We're going to assume a fixed sidebar and fluid content container for this example. Here's our CSS:

```
.main .container {
    padding-right: 330px;
    overflow: hidden;
}
.content {
    float: left;
    width: 100%;
    background-color: #fff;
}
.sidebar {
    float: right;
    margin-right: -330px;
    width: 300px;
    background-color: #fff;
}
.content,
.sidebar {
    padding-bottom: 99999px;
    margin-bottom: -99999px;
}
section,
aside {
    padding: 30px
}
```

Simple and easy! This could be extended to multiple rows for a more grid-like layout instead of just two columns. You can also use fluid width columns if you want. This method can extend across all types of grid work, and I use it right here on my site on the home and category pages to layout the articles/posts in a grid format with equal heights. Some simple media queries will set us up for a responsive flow. Here they are:

```
@media all and (max-width: 840px) {
    .main .container {
        padding: 0 30px;
        overflow: visible;
    }
    .content {
        float: none
    }
    .sidebar {
        float: none;
        margin-right: 0;
        width: 100%;
    }
    .content,
    .sidebar {
        padding-bottom: 0;
        margin-bottom: 0;
    }
    .content {
        margin-bottom: 30px
    }
}
```

Simple and easy, right? Now, onto solution number 2.

2) Equal Height Columns Using CSS Pseudo Classes and Positioning

This version makes use of the pseudo class `:after`. It also makes use of the fact that this pseudo class places content right after the element. It requires a hint of math to make things line up, but it reacts appropriately when you resize the browser etc. The parent container will have a relative positioning, and the pseudo elements will take on an absolute positioning, with 100% heights. We'll only position the pseudo elements the required distance from the left and right of the container though. The container will have a hidden overflow on it too. Here's the CSS:

```
.main .container {
    padding: 0 360px 0 30px;
    position: relative;
    overflow: hidden;
}
.content {
    float: left;
    width: 100%;
    background-color: #fff;
}
.sidebar {
    float: right;
    margin-right: -330px;
    width: 300px;
    background-color: #fff;
}
.content:after,
.sidebar:after {
    display: block;
    position: absolute;
    height: 100%;
    content: "";
    background-color: #fff;
}
.content:after {
    left: 30px;
    right: 360px;
}
.sidebar:after {
    right: 30px;
    width: 300px;
}
section,
aside {
    padding: 30px
}
```

The left and right positioning of the content and sidebar are 30px to compensate for the padding on the parent container. Let's dig into some media queries to make it responsive:

```
@media all and (max-width: 840px) {
    .main .container {
        padding: 0 30px;
        overflow: visible;
    }
    .content {
        float: none;
        margin-bottom: 30px;
    }
    .sidebar {
        float: none;
        margin-right: 0;
        width: 100%;
    }
    .content:after,
    .sidebar:after {
        display: none
    }
}
```

Nice and easy, huh? Onto number 3.

3) Equal Height Columns Using Table Display

Ah, the good old table layout. We won't actually use tables in this solution, but we'll set our CSS properties to have table displays. This is probably the simplest solution, but still my least favourite. I try to stay away from table layouts a lot these days, especially if using the CSS properties. You need to be mindful of browser support and different browser interpretations. Nonetheless, it's still an elegant, simple, valid solution, and if it works for you, then go for it. The HTML structure is a bit different this time too, so let's first take a look at that:

```
<div class="main">
    <div class="container">

        <div class="table">
            <div class="row">
                <div class="col content">
                    <h1>Section 1</h1>
                    <hr>
                    <p>Lorem ipsum dolor sit
amet, consectetur adipisicing elit. Ullam,
laborum, repudiandae, soluta sapiente architecto
et fugit laboriosam exercitationem error quisquam
doloribus veritatis consequatur ipsum consectetur
ad maxime obcaecati quod ipsa.</p>
                    <p>Lorem ipsum dolor sit
amet, consectetur adipisicing elit. Numquam,
sunt, soluta, vero, illo in dolore cum velit
aperiam eius quidem ad quasi inventore rerum ab
rem itaque et ullam earum.</p>
                    <p>I hope your site is more
interesting than this demo page.</p>
                </div>
                <div class="col sidebar">
                    <h2>Sidebar</h2>
                    <p>Subscribe to the
newsletter!</p>
                </div>
            </div>
        </div>

    </div>
</div>
```

We'll set the parent to `display: table`, we'll have a table row as well, and our two columns inside. I reset the container padding to 0 to compensate for the border-spacing property on the table. Here's our CSS:

```
.main .container {
    padding: 0 0
}
.table {
    display: table;
    border-collapse: separate;
    border-spacing: 30px 0;
}
.row {
    display: table-row
}
.col {
    display: table-cell;
    background-color: #fff;
    padding: 30px;
}
.col.sidebar {
    width: 300px
}
```

Let's round it off with some media queries to make it responsive. We need to reset the table[-row, column] display to block display, and add our container padding back. Not too much work at all. Here's the CSS:

```
@media all and (max-width: 840px) {

    .main .container {
        padding: 0 30px
    }
    .table {
        display: block
    }
    .row {
        display: block
    }
    .col {
        display: block
    }
    .col.content {
        margin-bottom: 30px
    }
    .col.sidebar {
        width: 100%
    }
}
```

Round 3, nice and easy.

Flexbox Incoming

Support for the CSS flexible box layout (flexbox) is on the rise, and that's awesome. The flexbox layout mode is stocked full of options, including vertical and horizontal distribution of boxes in a container. I'd recommend acquainting yourself with it from now, because it's going to speed up your workflow immensely. Not having to clear floats? Check. Vertically centred fixed dimension boxes? Check. Get on it, and wait for support to increase a bit more.

Wrap Up

There are probably more ways than above out there to achieve equal height columns. The above demonstrates just how easy it can be with plain old CSS, of different varieties. The best solution is always what works best for you though! I hope you enjoyed this tutorial, and feel free to leave any feedback below!

Get Source View Demo