What You'll Be Creating

There are quite enough headaches to deal with when coding HTML email. So it hardly seems fair if we also need to keep on top of the new email clients and device sizes that pop up every week. CSS and media query support can vary from app to app making it impossible to avoid being consumed by fear every time you hear that there's a new and exciting mail app that you've yet to test.

But what if you could build an email template that was responsive, even in environments with the poorest support for modern CSS? What if, every time you heard about some new email app that everyone's trying, instead of shaking with fear, you could feel safe and secure in the knowledge that your emails probably look fine?

The method that I espouse below is all about creating a good experience for email clients which have no media query support whatsoever.

It's called the *fluid hybrid* method, sometimes referred to as the *spongy* method of email development. The *fluid* part refers to the fact that we use lots of percentages. The *hybrid* part is because we also use `max-width` to restrict some of our elements on larger screens.

If you're looking for a ready-made, professional solution, grab one of our Best Email Templates. Otherwise, let's begin this tutorial.

Six Major Problems We're Aiming to Solve

1. Gmail's app for Android and iOS is a pain

It's more popular than the default mail app on Android, but Gmail doesn't support media queries, which we traditionally rely on to resize and reformat our emails for small screens. This tutorial will show you how to make emails that are responsive, even in the Gmail app.

2. New email apps are being released all the time

It's difficult to keep track of all the new mail apps that keep appearing. Some of them really care about email rendering and have great CSS and media query support, but some of them focus more on email workflows and don't support media queries at all. This tutorial will show you how to make an email that is *always* responsive, no

matter the level of CSS support.

3. The number of possible device screen sizes is almost infinite

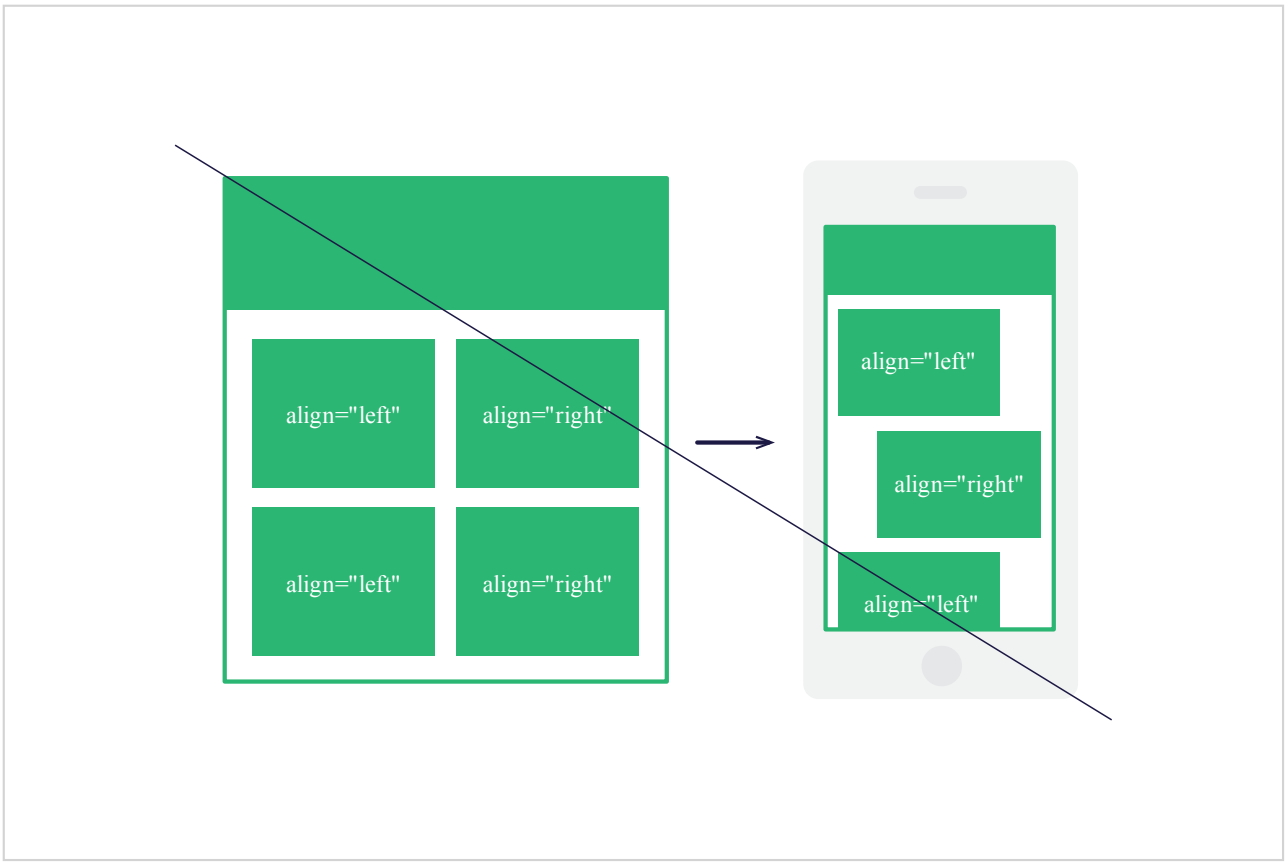
Not only do we have large desktops and tiny smartphones, but we also have large smartphones and tiny laptops. Just because someone is accessing their Gmail via their laptop doesn't mean their screen is big enough to view a 700px wide email, and people using an iPhone 6+ could handle a two-column layout, but instead are usually stuck with a single column. This tutorial shows you how to make a layout that will reformat to fit the space available, even in webmail.

4. Creating a responsive email by stacking `<td>`s on mobile doesn't work everywhere

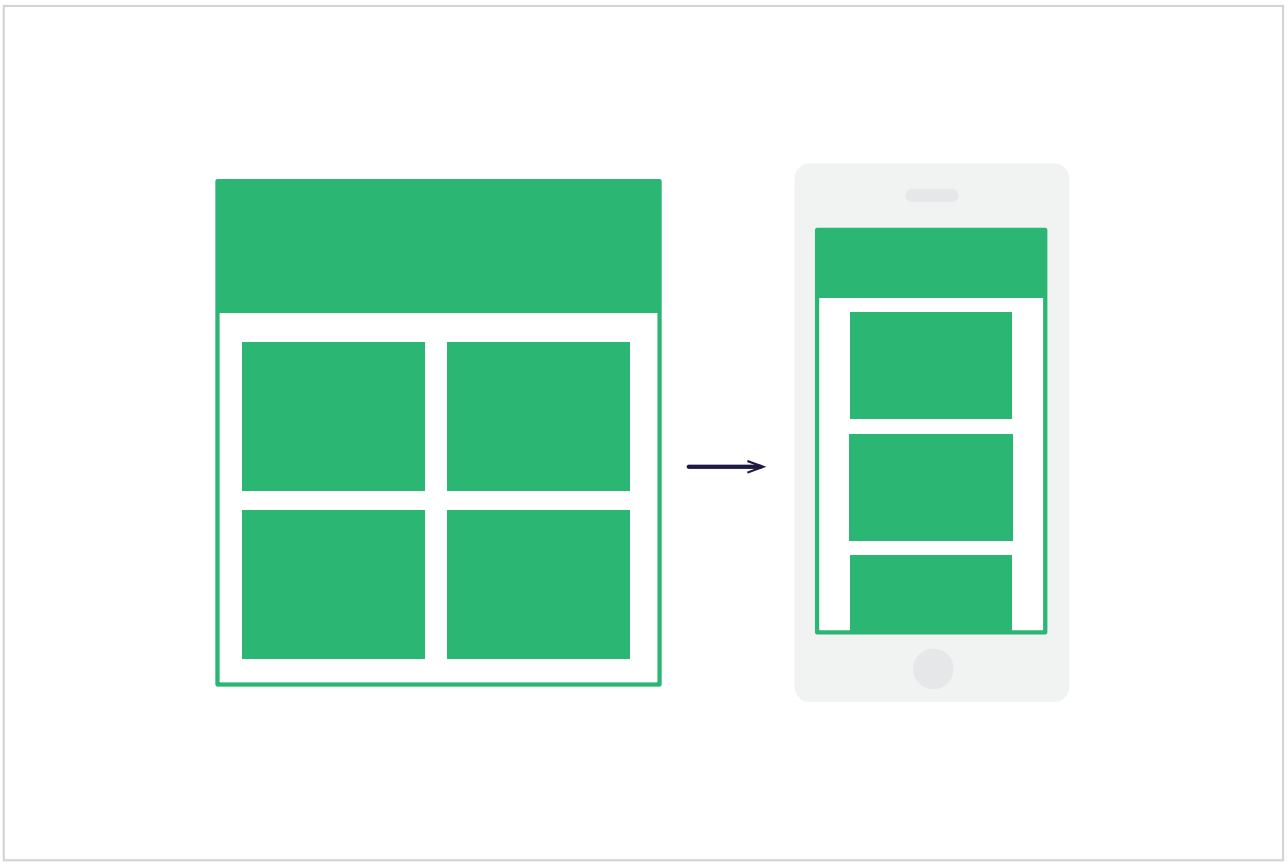
Certain email clients (on iOS and even some of the native mail apps on previous versions of Android) do not properly stack two table cells on the same row; they will only stack two individual tables. This tutorial uses a completely different method that is fully supported across all apps and devices. The usual solution to the problem is to use tables with the `align="left"` or `align="right"` attribute, but that presents another pitfall:

5. When using the *aligned table* method of responsive development, your tables are stuck aligned to the left or right in mobile apps which don't support media queries

The method in this tutorial uses a different approach which ensures that your columns all stack in the centre on mobile, even in the Gmail app. (You can also easily set them to align to the left or right if you prefer.)



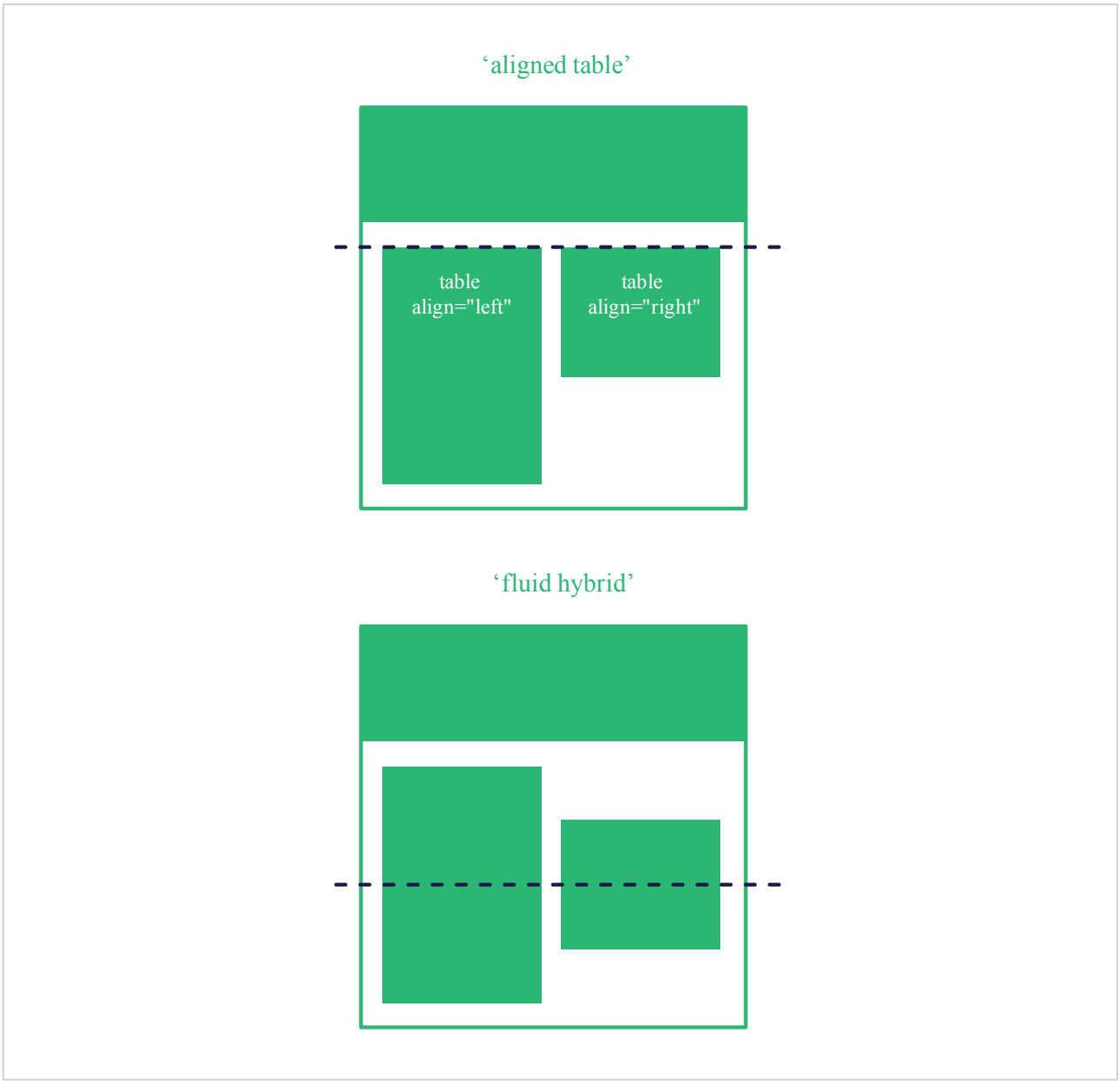
Tables aligned to the left or right remain in their positions in mobile apps that don't support media queries. You can't adjust them using mobile-specific CSS



This tutorial will show you how to have your columns stack in the center, even in apps that don't support media queries

6. When you use the *aligned table* method of responsive development, you lose the ability to vertically align content in adjacent columns

This tutorial will also show you how to vertically align two columns on the same row to the top or the middle, as if they were all table cells on the same row that were using the `valign` attribute.



This tutorial will teach you the 'fluid hybrid' method, which allows you to vertically align your columns to the top, middle or bottom

1. Getting Started

Start with a blank file and save it as index.html, then copy and paste this code:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <!--[if !mso]><!-->
        <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <!--<![endif]-->
    <meta name="viewport" content="width=device-width, initial-scale=1.
    <title></title>
    <link rel="stylesheet" type="text/css" href="styles.css" />
    <!--[if (gte mso 9)|(IE)]>
    <style type="text/css">
        table {border-collapse: collapse;}
    </style>
    <![endif]-->
</head>
<body>
    <center class="wrapper">
        <div class="webkit">
            [content goes here]
        </div>
    </center>
</body>
</html>
```

Let's quickly run through all of the elements in the code above:

`!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"` is the DOCTYPE I like best—I find it yields the fewest quirks.

`<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />` gives us support for all Unicode characters in our document.

`<meta http-equiv="X-UA-Compatible" content="IE=edge" />` is used here so that Windows Phones will display our mobile version correctly. It is hidden inside a conditional comment that hides it from any 'mso' (Microsoft Outlook) products to prevent a problem where Windows Live Mail will not display images when this tag is used. (Big thanks go to Don Braithwaite for this fix.)

We'll include `<title></title>` although it's best to leave it empty. A title tag is required for valid XHTML, but some native Android mail clients will display this title just before the preheader in the inbox preview, which usually isn't ideal.

You'll see that I'm using an external stylesheet for this tutorial, but the approach you take is up to you. To follow along, you should now create a new document called `styles.css` and save it in the same directory as your HTML file. We'll be inlining our CSS at the end.

Next, between `<!--[if (gte mso 9)|(IE)]>` and `<![endif]-->` we have some conditional CSS for Outlook to force it to collapse borders on all tables and prevent unwanted gaps. This conditional expression targets all versions of Microsoft Outlook (mso) greater than or equal to version 9 (which is all of them since version 9 is the earliest: Outlook 2000) as well as versions of Outlook that use Internet Explorer to render (being Outlook 2000–2003).

Under the body, first we have a `<center>` tag, to center its contents and act as our bearer of a few useful global CSS properties (since the body tag often gets stripped in webmail clients). We also have `<div class="webkit">` for earlier versions of WebKit-powered mail clients (basically only Apple Mail 6 and below, and Outlook 2011 in some cases). These earlier versions only support `max-width` on block-level elements and the easiest way to ensure our layout displays at the correct size is to wrap it in this div, which saves us having to set the widths in a media query as per my previous tutorial. (Thanks go to zerocents for this fix.)

2. Initial Styles

Next, create a blank CSS file called styles.css. Into your newly-created file, paste the following:

```
/* Basics */
body {
    margin: 0 !important;
    padding: 0;
    background-color: #ffffff;
}
```

```
table {
    border-spacing: 0;
    font-family: sans-serif;
    color: #333333;
}
td {
    padding: 0;
}
img {
    border: 0;
}
div[style*="margin: 16px 0"] {
    margin:0 !important;
}
.wrapper {
    width: 100%;
    table-layout: fixed;
    -webkit-text-size-adjust: 100%;
    -ms-text-size-adjust: 100%;
}
.webkit {
    max-width: 600px;
    margin: 0 auto;
}
```

Here I am zeroing the margins and padding on the body, table and table cells and zeroing any borders that would appear on linked images. The body margin is set to `!important` to help with an alignment issue on Android KitKat (more on that below). We have set our background color because it's always a good idea to explicitly do so, even if it's pure white, to avoid background colour glitches in Outlook or Lotus Notes.

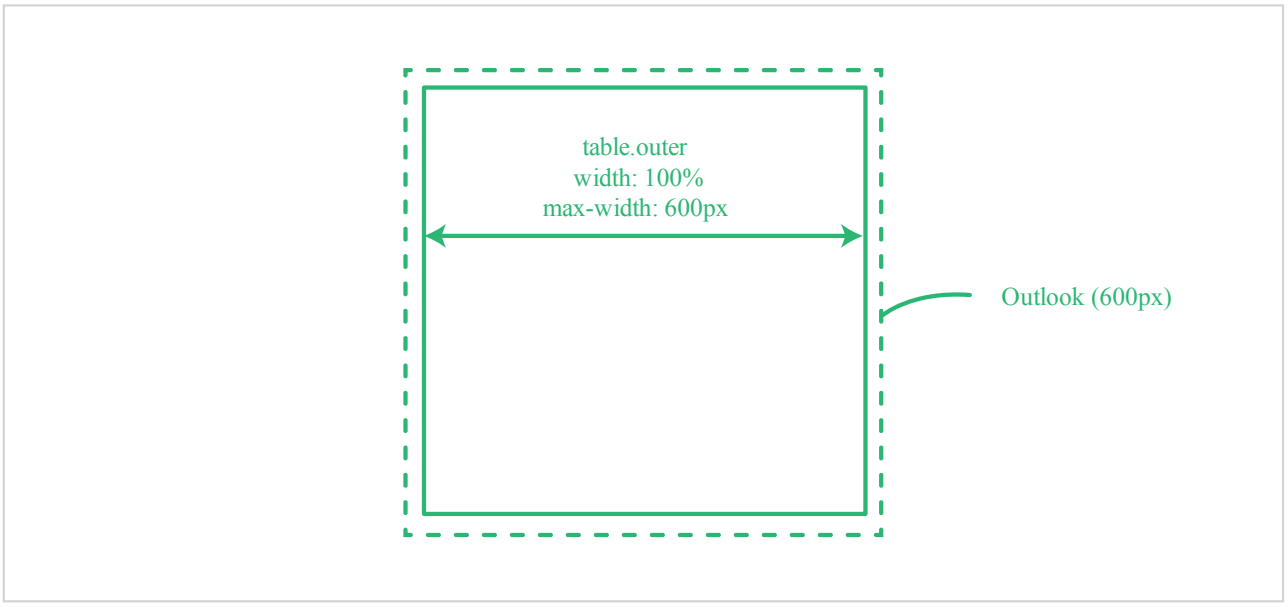
Our styles for the `table` and `td` tags are to take the place of the `cellpadding` and `cellspacing` HTML attributes. It's entirely up to you if you prefer to use the HTML attributes instead; in the past I've always supported using an HTML attribute instead of a CSS property where possible, however as I have worked on more and more large-scale projects I have found that it can be more manageable to specify these in the CSS, especially if you are usually working in a platform that automatically handles CSS inlining for you.

The `div[style*="margin: 16px 0"]` rule is used in conjunction with `!important` on our body margin to zero some unwanted padding in Android 4.4's native mail client.You can read more about this on James White's blog where he explains this clever fix.

There are also some styles for `.wrapper` with a few properties to stop text resizing in a weird way on Windows Phones and iOS, as well as `table-layout: fixed` to ensure our centered content will actually be centered in Yahoo mail. We're setting a `max-width` of 600px wide on a div called `.webkit` that is used to contain everything in Apple Mail 6 (and below) and Outlook 2011, and giving it a `margin` of `0 auto` so that it will be centered in Yahoo mail too.

3. Creating the Structural Outer Container

We'll start with one of the key building blocks for this method: a conditional table for Outlook that is hidden to all other clients. We need these because we're going to be using the `max-width` property which Outlook doesn't support. Therefore we need to create special Outlook-only tables with explicit pixel widths to contain everything in Outlook.



Our conditional table for Outlook is used because Outlook doesn't support the max-width property

So in our HTML file, let's delete the `[content goes here]` placeholder and paste in the following code. I tend to keep my conditional code tags all left-aligned at the same indentation level for readability, but how you format them is up to you.

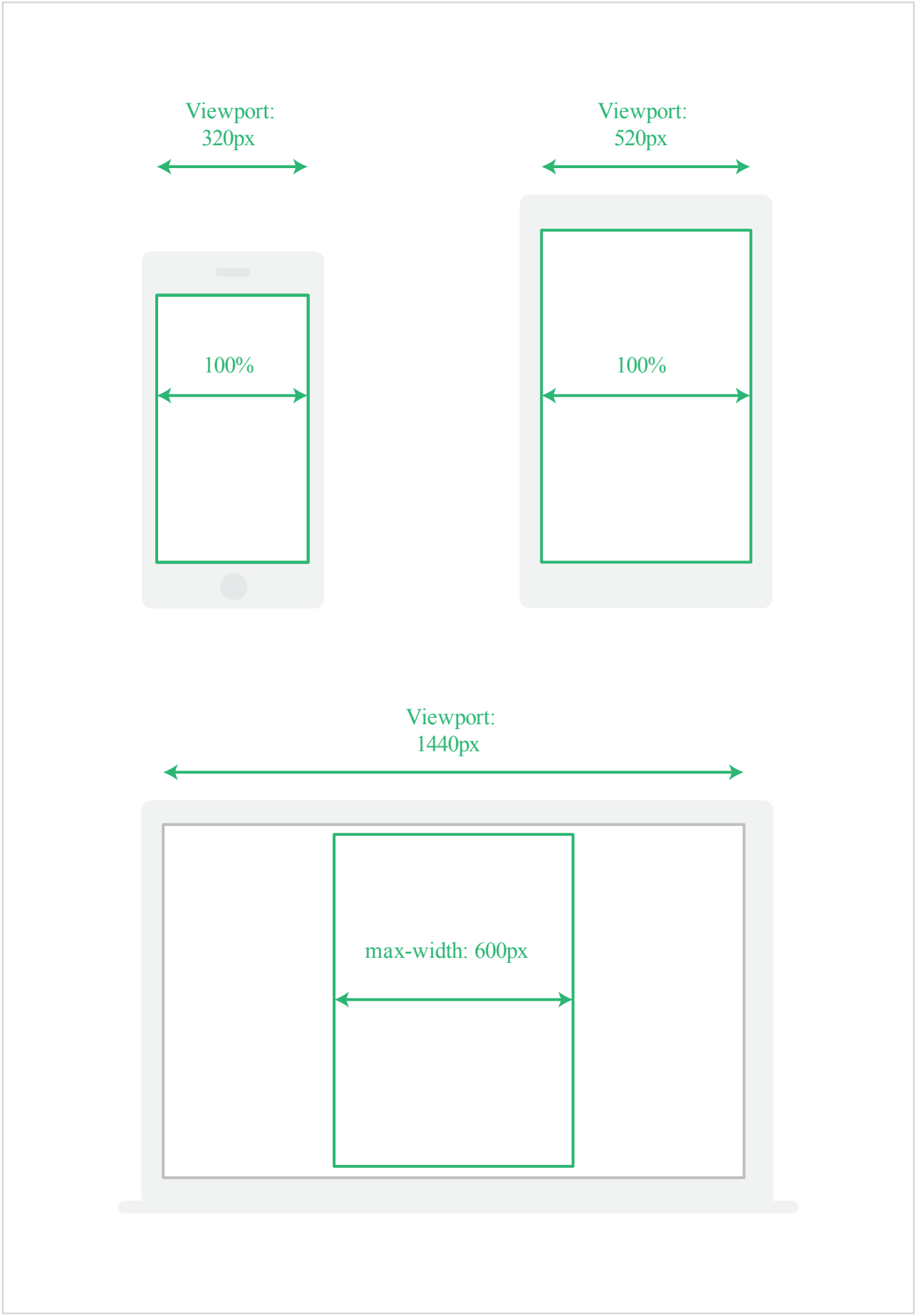
```
<!--[if (gte mso 9)|(IE)]>
<table width="600" align="center">
```

```
<tr>
<td>
<![endif]-->
<table class="outer" align="center">
<tr>
        <td>
            [content goes here]
        </td>
    </tr>
</table>
<!--[if (gte mso 9)|(IE)]>
</td>
</tr>
</table>
<![endif]-->
```

Note: there is no styling on my conditional table tags. I'm going to be using Campaign Monitor's inliner tool, inliner.cm, which also inlines styles onto conditional tables. If you're going to use a different inliner then it may not do this, so be sure to add `cellpadding="0" cellspacing="0" border="0"` to your conditional Outlook tables.

Inside our conditional table, you'll see that we have a `<table class="outer">` which is our key outer building block for every client except Outlook.

We want this outer table to be 100% wide on small screens but on larger screens we only want it to be a maximum of 600px wide. Therefore we're going to set its width to 100% and give it a max-width of 600px.



Our table is set to be 100% wide until it reaches a maximum of 600px

Handy tip: For a quick and easy buffer on mobile, without having to fiddle around with padding or media queries, change the `width` of your table from 100% to 95%.

So let's paste these styles into our styles.css file:

```
.outer {
Margin: 0 auto;
    width: 100%;
    max-width: 600px;
}
```

We also have `Margin: 0 auto;` set here to center our table in Yahoo in Chrome. Though this use of margin is for Yahoo's sake, I always capitalise `Margin` so that Outlook.com won't strip it out–a neat little hack thanks to Wiktor's comment on this blog post.

Now we have our outer structure, it's time to start adding some content.

4. Adding a Full-width Banner Image

First, download the tutorial files and move the /images directory so that it's in the same folder as your index.html file.

Now let's add a class of `full-width-image` to the `td` inside our `.outer` table and then we'll replace our [content goes here] placeholder with an image tag, so that the entire table now looks like this:

```
<table class="outer" align="center">
    <tr>
        <td class="full-width-image">
            <img src="images/header.jpg" width="600" alt="" />
        </td>
    </tr>
</table>
```

We have specified a pixel width for our image in the HTML so that Outlook will display it correctly, but we are going to override it with a width of 100% in the CSS so

that it will resize freely in other clients:

```
.full-width-image img {
    width: 100%;
    max-width: 600px;
    height: auto;
}
```

We are also setting the max-width to match the pixel width that we set in the HTML (600px) because Windows Phone does not play nice when the max-width is also set to 100%. We have set height to auto to ensure that our image doesn't end up with the wrong aspect ratio anywhere. (Huge thanks go to Courtney Fantinato who developed this excellent method for using high-resolution images with the fluid hybrid approach.)

Now you can preview your HTML file and see how the image fluidly resizes based on the size of the viewport.

5. Adding a Single Column Layout

Add another row to the `.outer` table with this markup:

```
<tr>
<td class="one-column">
        <table width="100%">
            <tr>
                <td class="inner contents">
                    <p class="h1">Lorem ipsum dolor sit amet</p>
                    <p>Maecenas sed ante pellentesque, posuere leo id,
                </td>
            </tr>
        </table>
    </td>
</tr>
```

And add this styling to your CSS file:

```
.inner {
padding: 10px;
```

```
}
p {
    Margin: 0;
}
a {
    color: #ee6a56;
    text-decoration: underline;
}
.h1 {
    font-size: 21px;
    font-weight: bold;
    Margin-bottom: 18px;
}
.h2 {
    font-size: 18px;
    font-weight: bold;
    Margin-bottom: 12px;
}

/* One column layout */
.one-column .contents {
    text-align: left;
}
.one-column p {
    font-size: 14px;
    Margin-bottom: 10px;
}
```

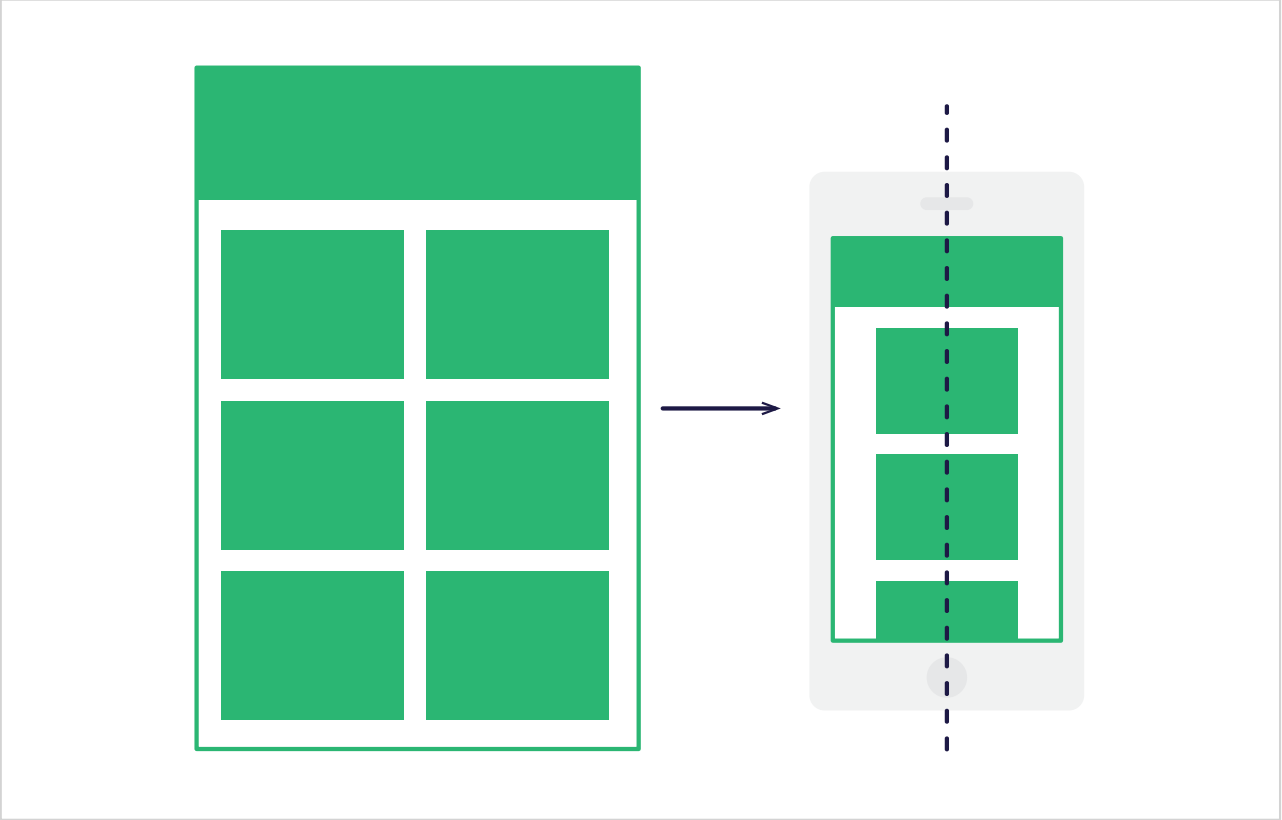
You'll notice that I've used the `<p>` tag and a set of classes for styling them. I like using paragraphs to style text, and you can actually manage them really easily because of the capital-M `Margin` hack that I mentioned earlier. I also use `<p class="h1">` instead of `<h1>` because Outlook.com has some h1, h2 and h3 styles that *always* override your styling.

So in the CSS above we've set up `10px` padding for our column, we've reset all the margins on `<p>`, set some basic styles for links and my .h1 and .h2 classes, then ensured the contents of our column are left-aligned with styled paragraphs.

Now onto the exciting stuff.. multiple columns!

6. Adding a Two-column Layout*

*which will be centered when stacked.



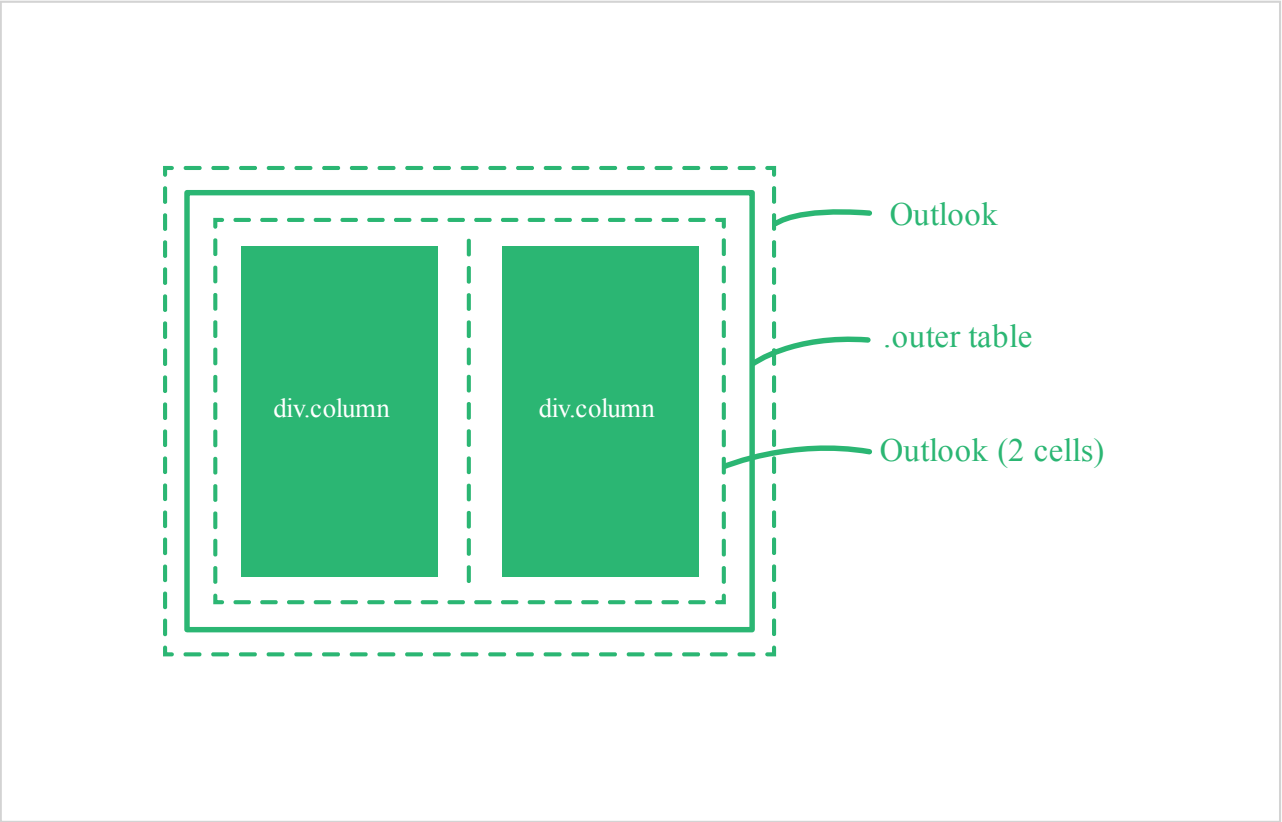
We are going to create a two-column layout on desktop that stacks to a single, centered column on mobile

First add this new row to the `.outer` table. It contains a cell with the class `.two-column` and, inside that, a conditional table for Outlook with two 50% wide columns:

```
<tr>
<td class="two-column">
        <!--[if (gte mso 9)|(IE)]>
        <table width="100%">
        <tr>
        <td width="50%" valign="top">
        <![endif]-->
        [column to go here]
        <!--[if (gte mso 9)|(IE)]>
        </td><td width="50%" valign="top">
        <![endif]-->
        [column to go here]
        <!--[if (gte mso 9)|(IE)]>
        </td>
        </tr>
        </table>
        <![endif]-->
    </td>
</tr>
```

These conditional columns are important, because without them Outlook will not let

our two floating tables sit neatly side-by-side. As Outlook doesn't support `max-width` either, these columns help restrict each column to the correct size.



Visualising what our two-column structure will look like

Now replace each of the `[column to go here]` placeholders with this:

```
<div class="column">
<table width="100%">
        <tr>
            <td class="inner">
                [content goes here]
            </td>
        </tr>
    </table>
</div>
```

The way we are going to get two columns to float side by side on desktop, but stack in the centre on mobile, is by using a combination of `text-align: center` and `display: inline-block`. All inline and inline-block elements obey the text-align property. Therefore, if we wrap our tables in a div that is set to inline-block, we can very easily set their alignment when they stack by setting the text-align property on

their container. You can choose from left, center or right text alignment, and your inline-block divs will obey. You can indeed simply set the table itself to be `display: inline-block`, but only if you aren't going to be nesting any more tables inside. Things can start behaving weirdly if you nest tables inside inline-block tables, so if you need to nest, always ensure the inline-block container is a div.

Let's style our container cell `.two-column` with our chosen alignment. We're also going to add `font-size: 0` to get rid of any gaps between our columns inside this cell.

```
/*Two column layout*/
.two-column {
text-align: center;
    font-size: 0;
}
```

Now we'll style our inline-block `div` which acts as our column:

```
.two-column .column {
width: 100%;
    max-width: 300px;
    display: inline-block;
    vertical-align: top;
}
```

We're using a width of 100% up to a max-width of 300px so that this particular column will be 100% wide on viewports that are smaller than 300px wide.

You can set your `vertical-align` to whatever you like: top, center or bottom. Setting `vertical-align` to `top` means that each column acts as though it is a table cell using the HTML property `valign="top"`, while setting it to `middle` is as though it has `valign="middle"`. Note that you can have many rows of these divs within the same cell and the vertical alignment will always dictate the vertical alignment on a row-by-row basis. It's pretty nifty! Also ensure that whatever you choose here matches the `valign` that you've set on your Outlook conditional table because Outlook doesn't support `vertical-align`. If you're having a mismatch with alignment in Outlook, forgetting to set the `valign` on the conditional tables is usually the culprit.

Next we'll add a table with two rows to each column. This is so that when everything stacks on mobile, each image has its corresponding text directly underneath.

So let's replace our two `[content goes here]` placeholders with the following:

```
<table class="contents">
    <tr>
            <td>
                    <img src="images/two-column-01.jpg" width="280" alt=
            </td>
    </tr>
    <tr>
            <td class="text">
                    Maecenas sed ante pellentesque, posuere leo id, elei
            </td>
    </tr>
</table>
```

Each column is 300px wide with 10px of padding on either side, leaving 280px for our image.

Next we'll style the `.contents` class by giving it a width of 100%:

```
.contents {
width: 100%;
}
```

And then let's add our styling under the two-column layout to set our font size and text-alignment, ensure our images display at 100% wide, and to give our text underneath a little bit of padding:

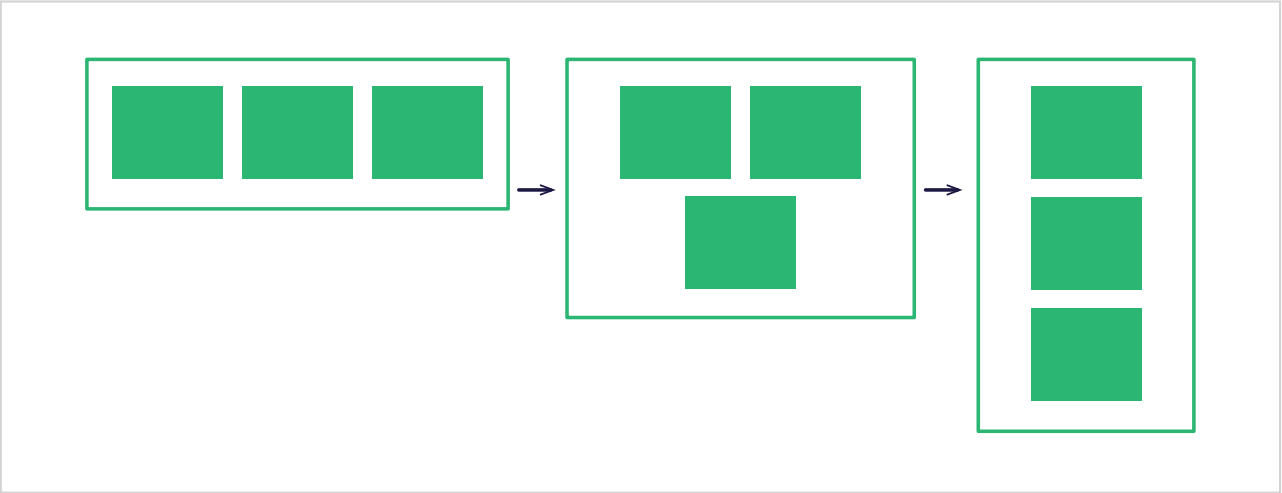
```
.two-column .contents {
font-size: 14px;
    text-align: left;
}
.two-column img {
    width: 100%;
    max-width: 280px;
    height: auto;
}
.two-column .text {
    padding-top: 10px;
}
```

Now you should have a two-column layout which stacks vertically when you resize your browser and shrinks as appropriate when you make the viewport narrower than 300px.

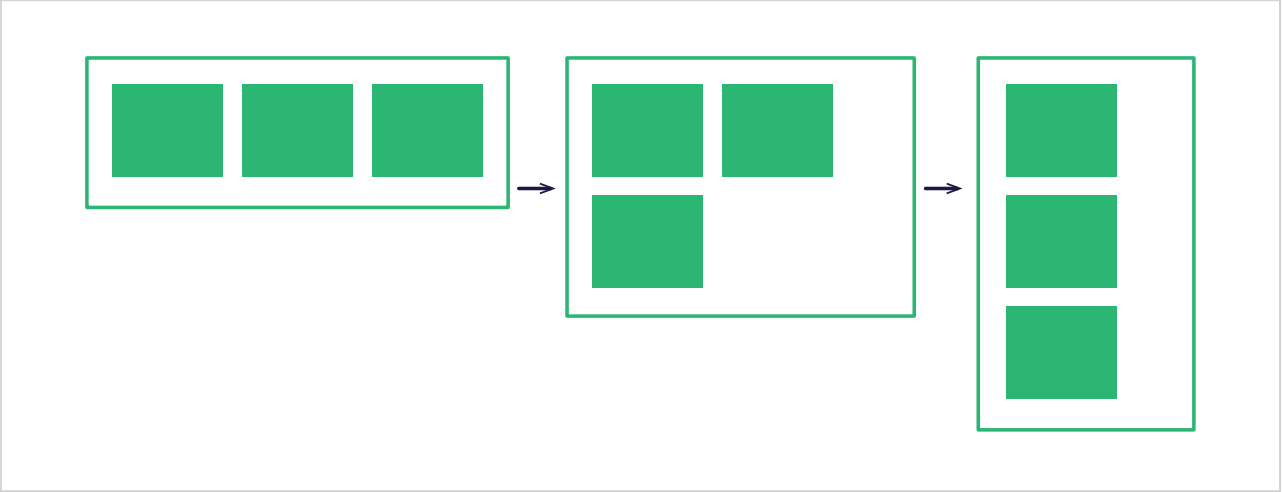
7. Adding a Three-column Layout

Again we are going to create side-by-side columns that stack on mobile by using the combination of `text-align: center` and `display: inline-block`.

We are going to be using `text-align: center` so that our column stacks in the center, but you can also always use left or right text alignment. Here is an example of how center and left-aligned elements will stack:



Example of how three columns will stack, using `text-align: center` on container



Example of how three columns will stack, using `text-align: left` on container

So we'll repeat the two-column process with an additional column. Add this new row to the `.outer` table. (Usually I prefer to use percentage widths for the cells in my conditional Outlook tables, but in this case it's easier to set the width of each to 200.)

```
<tr>
<td class="three-column">
        <!--[if (gte mso 9)|(IE)]>
        <table width="100%">
        <tr>
        <td width="200" valign="top">
        <![endif]-->
        [column to go here]
        <!--[if (gte mso 9)|(IE)]>
        </td><td width="200" valign="top">
        <![endif]-->
        [column to go here]
        <!--[if (gte mso 9)|(IE)]>
        </td><td width="200" valign="top">
        <![endif]-->
        [column to go here]
        <!--[if (gte mso 9)|(IE)]>
        </td>
        </tr>
        </table>
        <![endif]-->
    </td>
</tr>
```

Now add the following CSS to give some additional padding to this row, as well as set up all the properties we used in the two-column layout to make our columns behave how we want them to. It will also set up our styles for the div columns that

we are about to add, which will be 200px wide in this case.

```
/*Three column layout*/
.three-column {
    text-align: center;
    font-size: 0;
    padding-top: 10px;
    padding-bottom: 10px;
}
.three-column .column {
    width: 100%;
    max-width: 200px;
    display: inline-block;
    vertical-align: top;
}
.three-column .contents {
    font-size: 14px;
    text-align: center;
}
.three-column img {
    width: 100%;
    max-width: 180px;
    height: auto;
}
.three-column .text {
    padding-top: 10px;
}
```

Let's now insert our columns by replacing the `[column to go here]` placeholders with one div each:

```
<div class="column">
    <table width="100%">
        <tr>
            <td class="inner">
                <table class="contents">
                    <tr>
                        <td>
                            <img src="images/three-column-01.jpg" width
                        </td>
                    </tr>
                    <tr>
                        <td class="text">
                            Scelerisque congue eros eu posuere. Praesen
                        </td>
                    </tr>
                </table>
            </td>
        </tr>
```

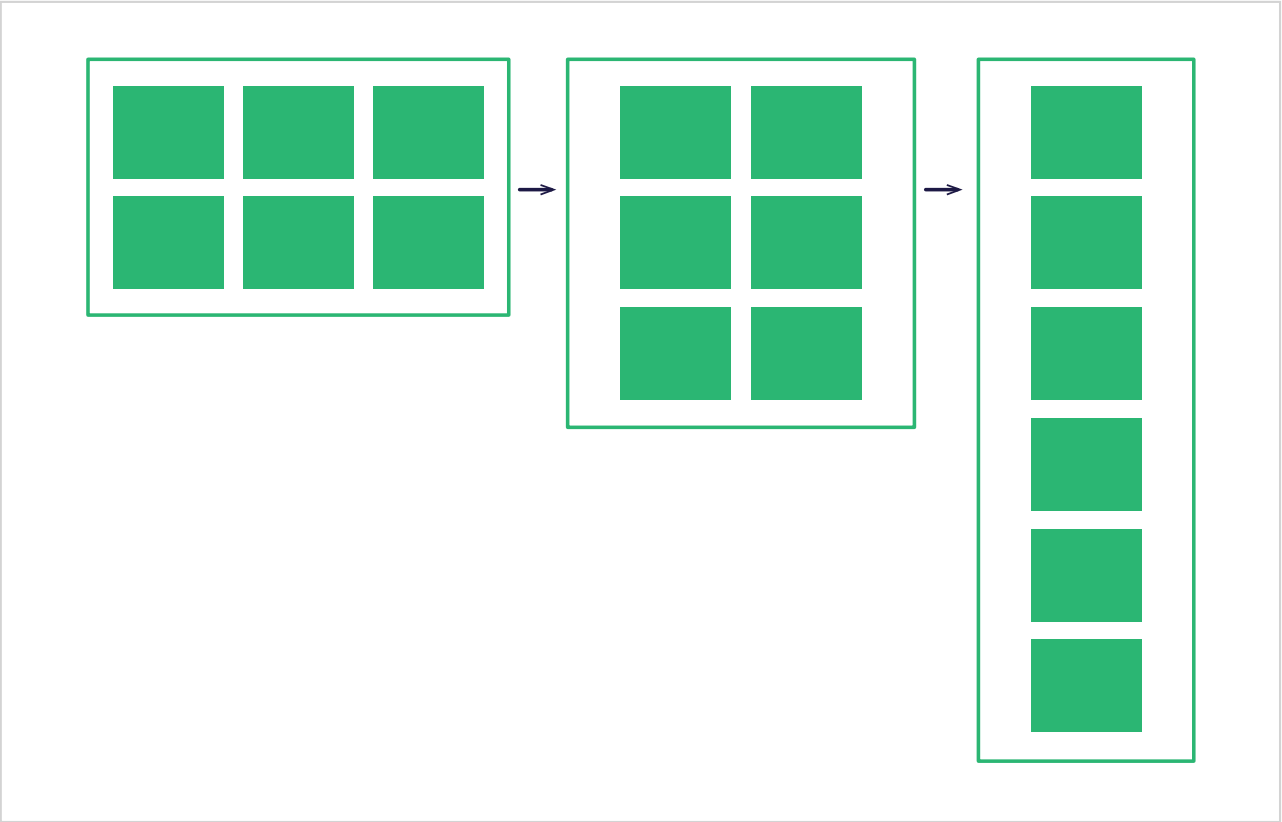
```
    </table>
</div>
```

And that's it! Now you should have a three-column layout, where the columns will stack on narrower viewports.

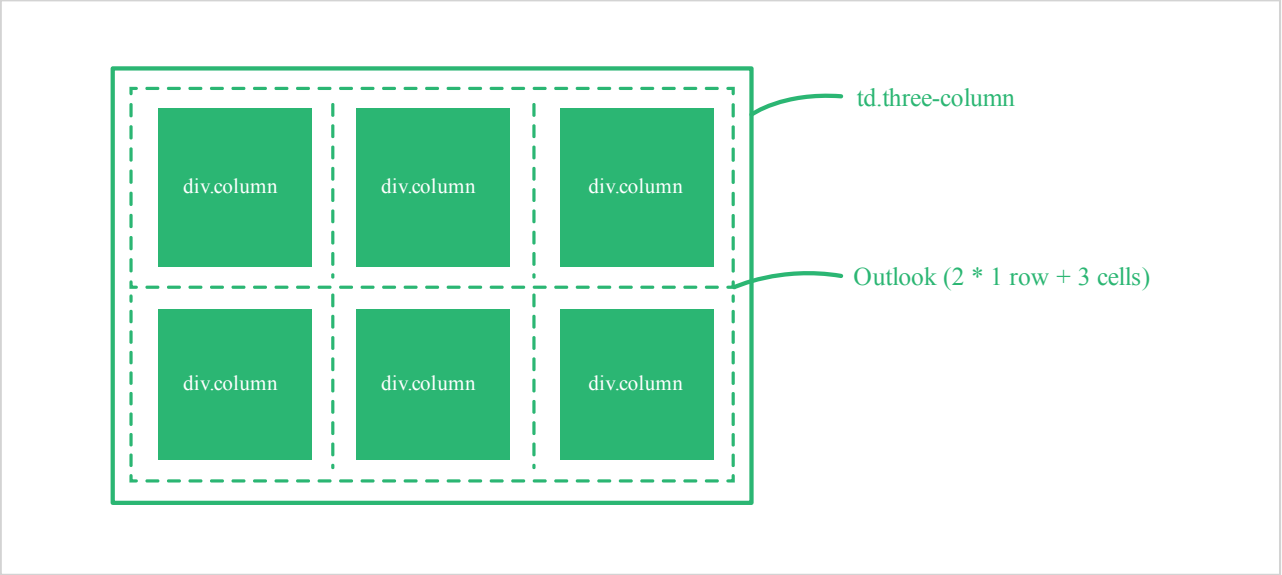
Owing to the fact that this layout has an uneven number of columns, sometimes you can find yourself with two columns displaying on the top row, with only one column beneath. While I think you can really make this look great if you design for this scenario, it can sometimes look a little unbalanced. Often the best way to get around this is to either use left alignment, or use multiple rows of three columns so that when the content stacks on mid-sized resolutions there are still an even number of columns per row.

Adding a three-column layout with multiple rows

When you want to add *more* rows in your multi-column layouts, you can add as many inline-block elements to a single container cell as you like. This way, when the viewport becomes too narrow to fit all of the columns, they simply reflow to fit the space available.



While you don't need to separate the rows of divs for most clients, you do need to add additional `<tr>`s to your conditional table for Outlook.



This is how our conditional table works in Outlook to keep our rows and columns separate

Let's try this out by starting a new row in our `.outer` table with the class of `.three-column`. Here we have a new three-column row with a conditional table inside. You'll see that there are three columns in our conditional table and then we end the conditional table's row with `</tr>` and open a new `<tr>`, which contains another three 200px wide cells.

```
<tr>
<td class="three-column">
        <!--[if (gte mso 9)|(IE)]>
        <table width="100%">
        <tr>
        <td width="200" valign="top">
        <![endif]-->
        [column to go here]
        <!--[if (gte mso 9)|(IE)]>
        </td><td width="200" valign="top">
        <![endif]-->
        [column to go here]
        <!--[if (gte mso 9)|(IE)]>
        </td><td width="200" valign="top">
        <![endif]-->
        [column to go here]
        <!--[if (gte mso 9)|(IE)]>
        </td>
        </tr>
        <tr>
```

```
        <td width="200" valign="top">
        <![endif]-->
        [column to go here]
        <!--[if (gte mso 9)|(IE)]>
        </td><td width="200" valign="top">
        <![endif]-->
        [column to go here]
        <!--[if (gte mso 9)|(IE)]>
        </td><td width="200" valign="top">
        <![endif]-->
        [column to go here]
        <!--[if (gte mso 9)|(IE)]>
        </td>
        </tr>
        </table>
        <![endif]-->
    </td>
</tr>
```

Next we'll add a div like this to each conditional cell, replacing the `[column to go here]` placeholder:

```
<div class="column">
    <table width="100%">
        <tr>
            <td class="inner contents">
                <p class="h2">Heading</p>
                <p>Class eleifend aptent taciti sociosqu ad litora torqu
                <p><a href="#">Read more</a></p>
            </td>
        </tr>
    </table>
</div>
```

Now if you resize your window, you'll again see that the columns stack to fill the available space. Three columns will reduce to two-columns with three rows, until they collapse finally to a single column with six rows.

Adding even more columns

We won't add any more multi-column layouts in this tutorial, but you are free to have as many columns in a layout as you wish. All you need to do is ensure that your content blocks all add up to the width of your container, and that your conditional table for Outlook has the correct number of cells and rows to contain them.

8. Adding a Two-column 'Sidebar' Layout

Next we'll create a two column layout with a 500px wide column and then a narrower 100px wide sidebar for an icon.

Firstly we'll add a row and cell with the class of `.left-sidebar` and inside that we'll pop our conditional table for Outlook which has a single row and two uneven columns:

```
<tr>
<td class="left-sidebar">
        <!--[if (gte mso 9)|(IE)]>
        <table width="100%">
        <tr>
        <td width="100">
        <![endif]-->
        [column to go here]
        <!--[if (gte mso 9)|(IE)]>
        </td><td width="500">
        <![endif]-->
        [column to go here]
        <!--[if (gte mso 9)|(IE)]>
        </td>
        </tr>
        </table>
        <![endif]-->
    </td>
</tr>
```

Then in each column, replacing the `[column to go here]` placeholder, we'll add a div. This time we will call one `.column .left` and one `.column .right` because they have different widths.

Note: here I'm using multiple classes on a single element. Some inliner tools and/or ESPs may not support doing this, so check with your system or platform first. As mentioned above, I use inliner.cm to inline my CSS, which does support multiple classes.

In the first column, add the left-hand div which contains our icon:

```
<div class="column left">
    <table width="100%">
```

```
        <tr>
            <td class="inner">
                <img src="images/sidebar-01.jpg" width="80" alt="" />
            </td>
        </tr>
    </table>
</div>
```

Then in the second column, add the right-hand div with the text and link:

```
<div class="column right">
    <table width="100%">
        <tr>
            <td class="inner contents">
                Praesent laoreet malesuada cursus. Maecenas scelerisque
            </td>
        </tr>
    </table>
</div>
```

Next let's style the container and set up the columns:

```
/* Left sidebar layout */
.left-sidebar {
text-align: center;
    font-size: 0;
}
.left-sidebar .column {
    width: 100%;
    display: inline-block;
    vertical-align: middle;
}
.left-sidebar .left {
    max-width: 100px;
}
.left-sidebar .right {
    max-width: 500px;
}
.left-sidebar .img {
    width: 100%;
    max-width: 80px;
    height: auto;
}
```

And finally let's set up some text styling and a link colour:

```
.left-sidebar .contents {
font-size: 14px;
    text-align: center;
}
.left-sidebar a {
    color: #85ab70;
}
```

Now you should have your left sidebar layout and when you resize your browser to make it smaller, the icon will jump above the text and sit in the centre.

9. Adding a Reversed 'Sidebar' Layout

Now we want to duplicate our sidebar layout, and this time we want the icon to be on the right-hand side to create some visual interest on desktop. But on mobile we want the content to stack in exactly the same order as our previous sidebar layout, so that the icon is on top of the text.

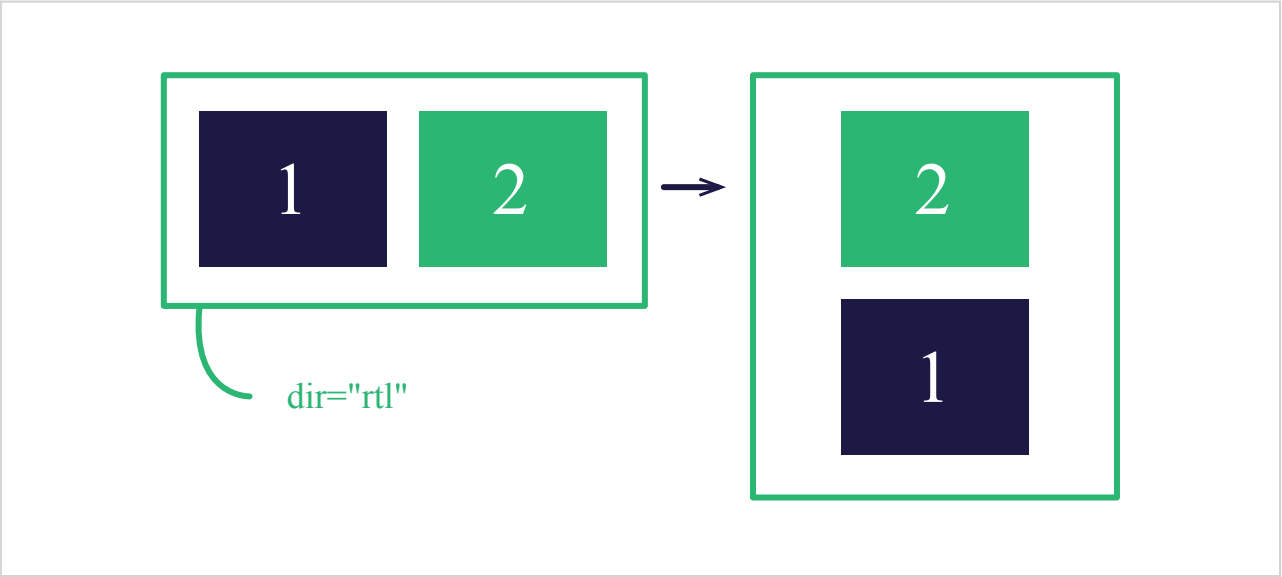
Firstly, we're going to copy and paste the entire row with our `.left-sidebar` cell, and the *only* thing we are going to change is the class of the container cell from `.left-sidebar` to `.right-sidebar`:

```
<tr>
    <td class="right-sidebar">
        <!--[if (gte mso 9)|(IE)]>
        <table width="100%">
        <tr>
        <td width="100">
        <![endif]-->
        <div class="column left">
            <table width="100%">
                <tr>
                    <td class="inner contents">
                        <img src="images/sidebar-02.jpg" width="80" alt
                    </td>
                </tr>
            </table>
        </div>
        <!--[if (gte mso 9)|(IE)]>
        </td><td width="500">
        <![endif]-->
        <div class="column right">
            <table width="100%">
```

```
                <tr>
                    <td class="inner contents">
                        Maecenas sed ante pellentesque, posuere leo id,
                    </td>
                </tr>
            </table>
        </div>
        <!--[if (gte mso 9)|(IE)]>
        </td>
        </tr>
        </table>
        <![endif]-->
    </td>
</tr>
```

Absolutely everything else is exactly the same.

What we're going to do is use `dir="rtl"` (meaning direction is right-to-left) to our advantage. This property is for denoting alphabets that run right-to-left, such as Arabic. But in our case it's simply going to tell each email client to render our elements in the opposite order.



Our elements will stack in the opposite order if `dir="rtl"` is set on their container

First, to the container (`.right-sidebar`), you need to add `dir="rtl"`. This is telling it to render our floating columns inside from right to left. So our opening tag should now look like this:

```
<td class="right-sidebar" dir="rtl">
```

Then, in the Outlook conditional code, we also need to add `dir="rtl"` to the `<table>` because we are telling the table to render the `<td>`s in the opposite order.

So our opening conditional comment should now look like this:

```
<!--[if (gte mso 9)|(IE)]>
<table width="100%" dir="rtl">
<tr>
<td width="100">
<![endif]-->
```

And finally, we need to add `dir="ltr"` to our .column-left and .column-right divs, because they have the content inside them, and since that content is in English, it needs to run left-to-right. If we don't set this on these elements, they'll inherit the right-to-left direction from their parent elements.

Our `.column-left` should now look like this:

```
<div class="column left" dir="ltr">
    <table width="100%">
        <tr>
            <td class="inner contents">
                <img src="images/sidebar-02.jpg" width="80" alt="" />
            </td>
        </tr>
    </table>
</div>
```

And our .column-right should now look like this:

```
<div class="column right" dir="ltr">
    <table width="100%">
        <tr>
            <td class="inner contents">
                Maecenas sed ante pellentesque, posuere leo id, eleifend
            </td>
        </tr>
    </table>
</div>
```

So to recap, our entire row should now look like this:

```
<tr>
    <td class="right-sidebar" dir="rtl">
        <!--[if (gte mso 9)|(IE)]>
        <table width="100%" dir="rtl">
        <tr>
        <td width="100">
        <![endif]-->
        <div class="column left" dir="ltr">
            <table width="100%">
                <tr>
                    <td class="inner contents">
                        <img src="images/sidebar-02.jpg" width="80" alt
                    </td>
                </tr>
            </table>
        </div>
        <!--[if (gte mso 9)|(IE)]>
        </td><td width="500">
        <![endif]-->
        <div class="column right" dir="ltr">
            <table width="100%">
                <tr>
                    <td class="inner contents">
                        Maecenas sed ante pellentesque, posuere leo id,
                    </td>
                </tr>
            </table>
        </div>
        <!--[if (gte mso 9)|(IE)]>
        </td>
        </tr>
        </table>
        <![endif]-->
    </td>
</tr>
```

And finally, let's add our styling, which is all exactly the same as for our `.left-sidebar`, except for the link colour:

```
/* Right sidebar layout */
.right-sidebar {
text-align: center;
    font-size: 0;
}
.right-sidebar .column {
```

```
    width: 100%;
    display: inline-block;
    vertical-align: middle;
}
.right-sidebar .left {
    max-width: 100px;
}
.right-sidebar .right {
    max-width: 500px;
}
.right-sidebar .img {
    width: 100%;
    max-width: 80px;
    height: auto;
}
.right-sidebar .contents {
    font-size: 14px;
    text-align: center;
}
.right-sidebar a {
    color: #70bbd9;
}
```

There we go! So now we have two sidebars on opposite sides, but when everything stacks on mobile both sidebars appear above the text.

10. Adding Progressive Enhancement with Media Queries

Now you have a complete email template that is responsive everywhere without a single media query in sight! But of course there are quite a few email clients that do indeed support media queries, so now we can go about progressively enhancing our already fabulous template with a few tweaks to make everything look beautiful in a client like iOS Mail.

Firstly, we'll make all of our columns appear at 100% the width of the viewport for anything up to 400px. We'll also limit our three-column images to 50% wide so that they don't blow up too large and we'll override our pixel `max-width` on two-column images so that they take up the whole viewport. On the columns all we have to do is override the max-width because, as you'll recall, they all already have a width of 100% and it's just the `max-width` that is restricting them.

So add this to your CSS file:

```
/*Media Queries*/
@media screen and (max-width: 400px) {
.two-column .column,
    .three-column .column {
        max-width: 100% !important;
    }
    .two-column img {
        max-width: 100% !important;
    }
    .three-column img {
        max-width: 50% !important;
    }
}
```

Then, between 401px and 600px wide, we'll add the following to make the two- and three-column layouts appear as they would on desktop, but shrunk down to fit.

```
@media screen and (min-width: 401px) and (max-width: 620px) {
.three-column .column {
        max-width: 33% !important;
    }
    .two-column .column {
        max-width: 50% !important;
    }
}
```

These are fairly arbitrary adjustments to demonstrate what's possible—you can fiddle and play around with this as much as you like to achieve your desired outcome across a whole range of device sizes that support media queries.

11. Inlining Your Code

If your email sending platform doesn't take care of inlining for you then you'll need to do it manually. First, delete the link tag `<link rel="stylesheet" type="text/css" href="styles.css" />` in the `<head>` of your document and replace it with `<style type="text/css">`. Copy the contents of style.css and paste it underneath, then close the style tag with `</style>`. Finally, copy and paste the entire file into the box at inliner.cm and wait for the results. Once processed, copy the contents out of the

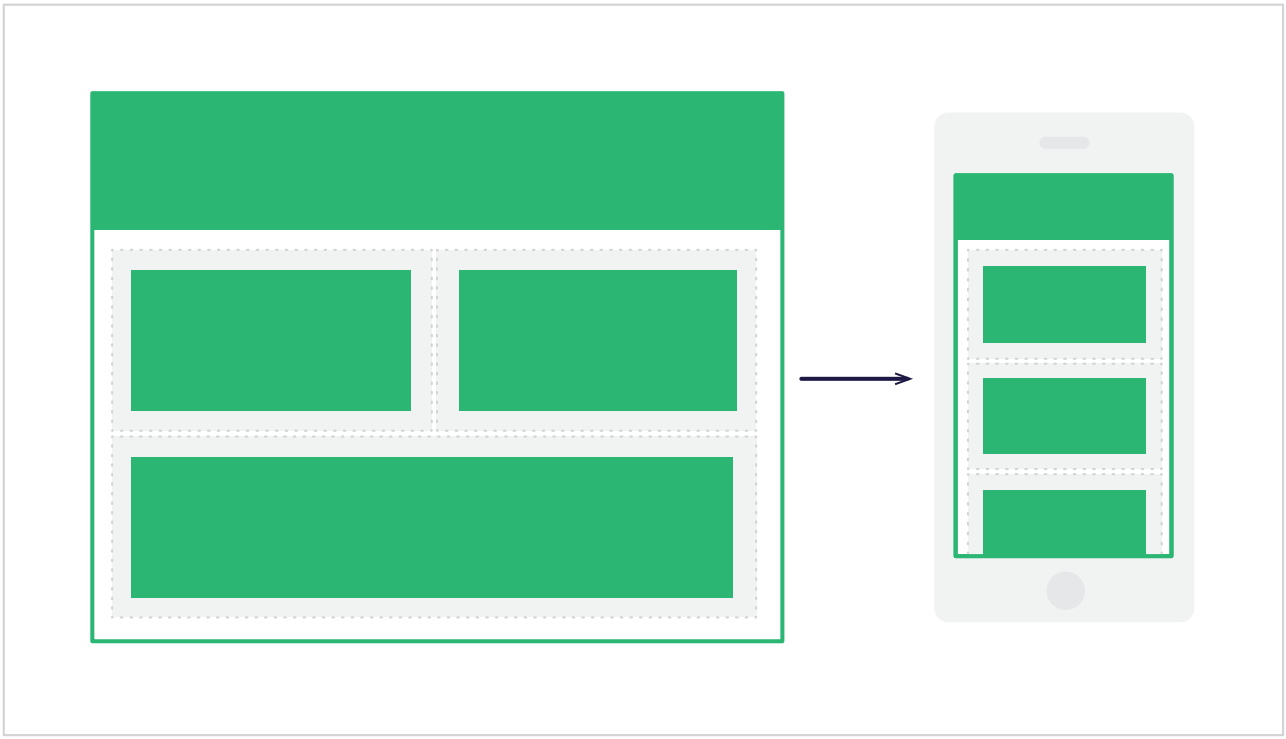
box and you're ready to go!

And There You Have it!

Well done! We now have a fully-functioning responsive HTML email using fewer than twenty lines of media queries.

A few final notes and tips

- I have found with this method of email development that simplicity and uniformity is key. Things start to get out of hand when you have different padding on different columns. For this tutorial you'll notice that I've kept padding uniform at 10px on the `.inner` class which applies to every single layout. This is important because when everything stacks on mobile you want the padding to be uniform.



Always have the same amount of padding on the left and right sides of your elements so that padding is uniform on mobile in clients with no media query support

- Sometimes you may need to calculate outwards to establish what the outer width of your layout is going to be, based on what your best uniform padding

value is, rather than deciding on the outer width and then working your way inwards.

- It's often a good idea to design in the browser with this method, since often it can be quite a challenge to retrofit certain designs to work with this approach.
- When using this method it's quite safe to start designing emails that are a lot wider than usual because you know the layout is going to rearrange itself to fit smaller viewports, even in webmail. The only sticking point is Outlook, which still requires those conditional tables so that everything will look right, and therefore Outlook users are always going to see the entire email at its widest. Even though you still need to bear this lowest common denominator in mind, there's definitely some room to move beyond the usual 550-600px restriction that we usually place on email.

The icons

Thanks again go to Pierre Borodin for all the icons used in my tutorials.

Addendum: iOS9 iPad Mail App Bug

With the release of iOS9, Apple appear to have reintroduced an old WebKit bug to the Mail app on iPad. The bug sometimes causes a gap between inline-block elements if there is any whitespace between the two elements in the HTML markup.

Note: The code from this tutorial is unaffected but you may encounter this bug if you alter it or use these methods in another template.

One way to tackle the issue is to allow a few spare pixels in your column container so that any small gaps don't result in your columns stacking.

The other solution is simply to get rid of any empty whitespace between the `<div>` tags in your HTML. You would adjust the following code:

```
<!--[if (gte mso 9)|(IE)]><table width="100%">
<tr>
<td width="50%" valign="top">
<![endif]-->
<div class="column">
```

```
    [content]
</div>
<!--[if (gte mso 9)|(IE)]>
</td><td width="50%">
<![endif]-->
<div class="column">
    [content]
</div>
<!--[if (gte mso 9)|(IE)]>
</td>
</tr>
</table>
<![endif]-->
```

and change it to this:

```
<!--[if (gte mso 9)|(IE)]><table width="100%">
<tr>
<td width="50%" valign="top">
<![endif]-->
<div class="column">
    [content]
</div><!--[if (gte mso 9)|(IE)]></td><td width="50%"><![endif]--><div c
    [content]
</div>
<!--[if (gte mso 9)|(IE)]>
</td>
</tr>
</table>
<![endif]-->
```

As you can see, the closing of the first `<div>` tag, the conditional code for Outlook, and the opening of the next `<div>` tag are all on the same line without any spaces between them. This will stop any gaps from rendering in iOS9 Mail on iPad.

Thanks to Rémi Parmentier for working out the fix!

If your about to kickstart a new project browse through our Best Email Templates.

Difficulty:
Advanced

Length:
Long

Categories:

Email **HTML** **CSS** **Email Design** **Responsive Web Design**

Translations:
Español
Русский

Envato Tuts+ tutorials are translated into other languages by our community members—you can be involved too!

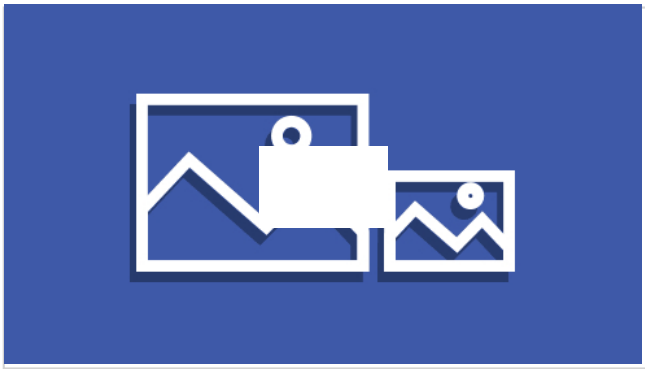
Translate this post Powered by native

View Online Demo View on Github

About Nicole Merlin

Email designer and developer and lover of all things email. Owner of Email Wizardry, an email design and development studio in Victoria, Australia. Occasional blogger and slightly more frequent tweeter.

Suggested Envato Tuts+ Course



Understanding Responsive Images $9

Related Tutorials

Create a Dashboard Module in OpenCart

Code

Building a CMS: Structure and Styling

Code

Magento Theme Development: Product Page, Part 1

Code

Envato Market Item