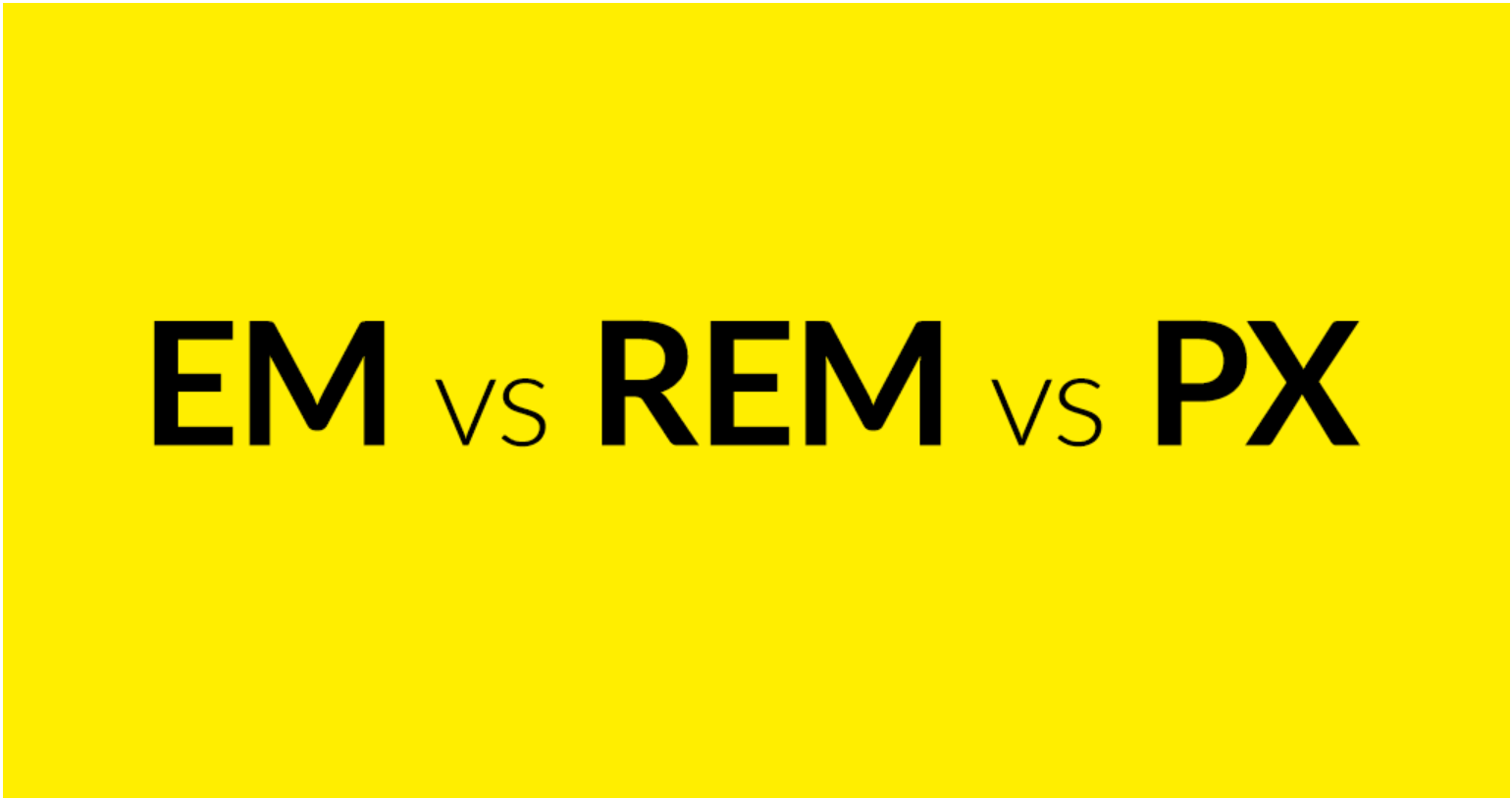
EM vs REM vs PX – Why you shouldn't “just use pixels”

Roughly an 8 minute read by **Simon (/about/simon-willans)**

4th January 2017

Pssst. This post is now over a year old! Things change in a heartbeat on the web so certain aspects of this post may no longer be correct or relevant.



The debate has been had many times - what units of measurement should we use in our CSS?

We, like many others[1][2], were ready to ditch REMs and return to the beloved pixel. We lost track of why we adopted the use of REMs in the first place. The problem doesn’t just revolve around font-sizes - it’s also about accessibility.

TL;DR:

Pixels are ignorant, don’t use them.
Use REMs for sizes and spacing.
Use EMs for media queries.

Pixels

Pixels (px) are what we've all become accustomed to over the years. Everyone knows what a pixel is (although the size of a pixel isn't always the same, but that's for another day). Everyone is comfortable with using pixels. They're easily translatable. Designers typically work in pixels, so it's easy to take sizes directly from Photoshop straight in to build.

So what's wrong with pixels?

Accessibility

I'm a big advocate of accessibility on the web. I'd take accessibility over "pretty" any day.

If you're setting all of your font-sizes, element sizes and spacing in pixels, you're not treating the end user with respect.

In most browsers, a user can set their default browser font-size to be a different size to the default (typically 16px). If the user sets their default to 20px, all font-sizes should scale accordingly.

However, if the website explicitly sets font-sizes in pixels, a heading set at 30px will always be 30px. That might sound great from a designer/developer point of view - but you're not the user, stop making websites for yourself.

Thankfully, setting font-sizes in pixels doesn't completely ruin accessibility. The user can still zoom in and out with ctrl plus +/- (cmd instead of ctrl on OS X). However, we can do better.

REMs

If you're in any way familiar with the web design world, you will undoubtedly have heard of REMs. If you're not, REMs are a way of setting font-sizes based on the font-size of the root HTML element. They also allow you to quickly scale an entire project by changing the root font-size (for example at a certain media query/screen size).

"[The REM] unit represents the font-size of the root element (e.g. the font-size of the <html>element). When used on the font-size on this root element, it represents its initial value."[3]

How to calculate PX from REM

A basic and most common example: html font-size is set to 10px, paragraph is set to 1.6rem - 1.6rem * 10px = 16px.

Setting a root font-size of 10px is the most common scenario when I see people using REMs. It allows for a quick conversion between pixel values to REM values simply by dividing the number by 10.

However, setting the base font-size in pixels still has the same problem as the pixel example above. Accessibility overridden.

While REMs certainly have their uses, I'm willing to bet that most people use REMs because they are seen as cooler than pixels. I rarely see a project where someone actually changes the root HTML font-size depending on screen size; and rightfully so. Unless you've got a very typographically heavy design, you're unlikely to want to scale everything at once.

So how can we un-break our accessibility faux pas?

Set the root HTML font-size as a percentage. That's a percentage of the user's default browser font-size. A typical method is to set the HTML font-size to 62.5%. That's because 62.5% of 16px (typical default browser font-size) is 10px. That would still make 1.6rem = 16px. This now means that if the user's default browser font-size is changed to, for example, 20px, 1.6rem would now equal 20px. So if your user wants bigger fonts, let them. Happy designer. Happy developer. All numbers are still easy to work with.

The ideal scenario would be to leave the HTML font-size as 100%, but that does make the maths a little bit harder. For example, 16px is now 1rem, 20px is 1.25rem, 24px is 1.5rem etc.

Sass/SCSS to the rescue

Working all of these numbers out in your head would be pretty time consuming. Thankfully, if you use Sass/SCSS, LESS, or any other CSS pre-processor, you shouldn't worry. You can use functions to calculate these things for you.

What about EMs?

EMs perform initially in a similar fashion to REMs, until it comes to nesting. I've never been a fan of EMs, especially when it comes to font-sizes. For example, take a div with a font-size of 2em, then add a paragraph with a font-size of 2em. The font-size of that paragraph is now 2ems relative to the div. I quickly lose track of the maths and what size is what, and it quickly becomes unmanageable. This is what REMs solve - the size always refers back to the root.

What about media queries?

So we've established that using pixel values overrides browser default settings, so simply converting all pixel sizes to REMs is the best thing to do, right? Not quite.

This blog post highlights some of the key the differences between using pixels, EMs and REMs in media queries (https://zellwk.com/blog/media-query-units/). Go have a read and then come back.

In summary, both pixels and REMs for media queries fail in various browsers when using browser zoom, and EMs are the best option we have. REMs fail much more than pixels at this point, so we're going to discount them completely.

It does get a bit more tricky though. Both EMs and pixels have their downfalls with media queries when it comes to the difference of a decimal place of that unit. If you happen to use both min and max width media queries in the same block of code, you're going to have a bad time as soon as the user starts to change their browser zoom or default font-size.

Here are some examples:

We use 6 decimal places because certain browsers show no difference between 2-5.d.p.

Example 1: Browser zoom set to 100%, browser width set to 640px

```
min-width: 64em = Hit
max-width: 63.99em = Miss
max-width: 63.999999em = Hit
min-width: 640px = Hit
max-width: 639px = Miss
max-width: 639.999999px = Miss
```

Example 1b: Browser zoom set to 100%, browser width set to 639px

```
min-width: 64em = Miss
max-width: 63.99em = Hit
max-width: 63.999999em = Hit
min-width: 640px = Miss
max-width: 639px = Hit
max-width: 639.999999px = Hit
```

So we can't use 6dp for EMs, because it hits both media queries.

Example 2: Browser zoom set to 110%, browser width set to 704px (because 640px * 110% = 704px)

```
min-width: 64em = Miss
max-width: 63.99em = Miss
max-width: 63.999999em = Hit
min-width: 640px = Miss
max-width: 639px = Miss
max-width: 639.999999px = Hit
```


Example 2b: Browser zoom set to 110%, browser width set to 705px

```
min-width: 64em = Hit
max-width: 63.99em = Miss
max-width: 63.999999em = Miss
min-width: 640px = Hit
max-width: 639px = Miss
max-width: 639.999999px = Miss
```

So we can't use 2dp for EMs. So all we're left with at this stage is 6dp pixels. This works with browser zoom. However...

Example 3: Browser zoom set to 100%, browser font size set to 20px, browser width set to 800 (because 640 * 125% = 800)

```
min-width: 64em = Hit
max-width: 63.99em = Miss
max-width: 63.999999em = Hit
```

So again, 6dp EMs are still out. And we can't use pixels for media queries at all, because they still fire at 640px/639px because they ignore EMs/REMs.

So what's the solution?

Unfortunately, there isn't an answer. Until browsers sort out rounding issues, we're a bit stuck.

The simplest acceptable option, is that we never create a min-width and max-width overlap in the same block. Example:

```
body {
        @media screen and (max-width: 63.99em) {
                background: blue;
        }

        @media screen and (min-width: 64em) {
                background: blue;
        }
}
```

The problem with the above is that there are certain scenarios where both media queries are hit, or both are ignore. So the safe bet would be to write something like:

```
body {
        background: blue;

        @media screen and (min-width: 64em) {
                background: blue;
        }
}
```

Why isn't there a real solution? I don't believe enough people care. Any statistics saying that people don't change their default browser font-size are definitely skewed. The options in Chrome to change the default font-size is now very buried in the advanced browser options.

Pitfalls

There are some pitfalls to this approach:

If you're used to writing both min-width and max-width media queries in the same code block, it will take longer.
It increases complexity as you'll have to override CSS one direction or the other, rather than telling the browser to use option A or B.
It will increases file size due to these overrides. Not by much, but it's worth considering.

Depending on the project, who's developing it, and various other factors such as project budgets (we have to be realistic here), pixels might be easier and quicker. It also might be easier and quicker to ignore accessibility.

Remember who you're building websites for.

It's also worth noting that this blog post is accurate at the time of writing, but with the constant updates to the way browsers work, this issue could become a thing of the past.

Just use pixels (https://benfrain.com/just-use-pixels/)

R.I.P. REM, Viva CSS Reference Pixel! (https://mindtheshift.wordpress.com/2015/04/02/r-i-p-rem-viva-css-reference-pixel/)

Length - Mozilla Developer Network (https://developer.mozilla.org/en/docs/Web/CSS/length)

Simon (/about/simon-willans) spends the majority of his time building responsive websites, focusing heavily on large-scale frameworks.

You should also read...

7th February 2018

We've joined the Tofoo revolution!

Food, glorious food! We're super excited to get our teeth stuck into this one...

(https://engageinteractive.co.uk/blog/weve-joined-the-tofoo-revolution-1)

23rd January 2018

Bonding over boards with our new partner, Lawcris!

Find out more about our new Partner, Lawcris, as we welcome them to the team.

(https://engageinteractive.co.uk/blog/bonding-over-boards-with-our-new-partner-lawcris)

15th January 2018

How to create Low Poly Art in Adobe Illustrator

It's a process you may need to practice, but with this tutorial, you've got all you need to know!

(https://engageinteractive.co.uk/blog/how-to-create-low-poly-art-in-adobe-illustrator)

1st Floor, Munro House

Duke Street

Leeds

LS9 8AG

0113 457 4090 (tel:01134574090)

hello@engageinteractive.co.uk (mailto:hello@engageinteractive.co.uk)

(https://instagram.com/engageinteractive)

(https://facebook.com/engageinteractive.co.uk)

(https://twitter.com/engagetweet)

Attention to digital™

RAR Recommended

Premier Google Partner