

HTML

CSS

JS

Result

Current html font size: 16px

Using 'px' unit

Using 'rem' unit

Using 'em' unit

Resources

1×

0.5×

0.25×

Advertisement

WEB DESIGN > RESPONSIVE WEB DESIGN

Comprehensive Guide: When to Use Em vs. Rem

by [Kezz Bracey](#) 21 Jul 2015Difficulty: Intermediate Length: Long Languages: English

Responsive Web Design

Web Typography



You may have come to terms with using flexible units of measurement, but you still might not fully understand when to use `rem` and when to use `em`. This tutorial will help you figure it out!

Both `em` and `rem` are flexible, scalable units which are translated by the browser into pixel values, depending on the font size settings in your design. If you use a value of `1em` or `1rem`, it could translate in the browser as anything from `16px` to `160px` or

any other value.

StylesComputedEvent ListenersDOM BreakpointsPropertiesAccessibility Properties

.sixteen {

font-size: 16px;

padding: ▶ 1em;

}

CSS padding set to 1em

StylesComputedEvent ListenersDOM BreakpointsPropertiesAccessibility Properties

☐ Show inherited properties

display: block;

font-size: 16px;

height: 24.4444446563721px;

padding-bottom: 16px;

padding-left: 16px;

padding-right: 16px;

padding-top: 16px;

width: 1654.27087402344px;

Computes to 16px

StylesComputedEvent ListenersDOM BreakpointsPropertiesAccessibility Properties

.onesixty {

font-size: 160px;

padding: ▶ 1em;

}

CSS padding set to 1em

StylesComputedEvent ListenersDOM BreakpointsPropertiesAccessibility Properties

☐ Show inherited properties

display: block;

font-size: 160px;

height: 486.666687011719px;

padding-bottom: 160px;

padding-left: 160px;

padding-right: 160px;

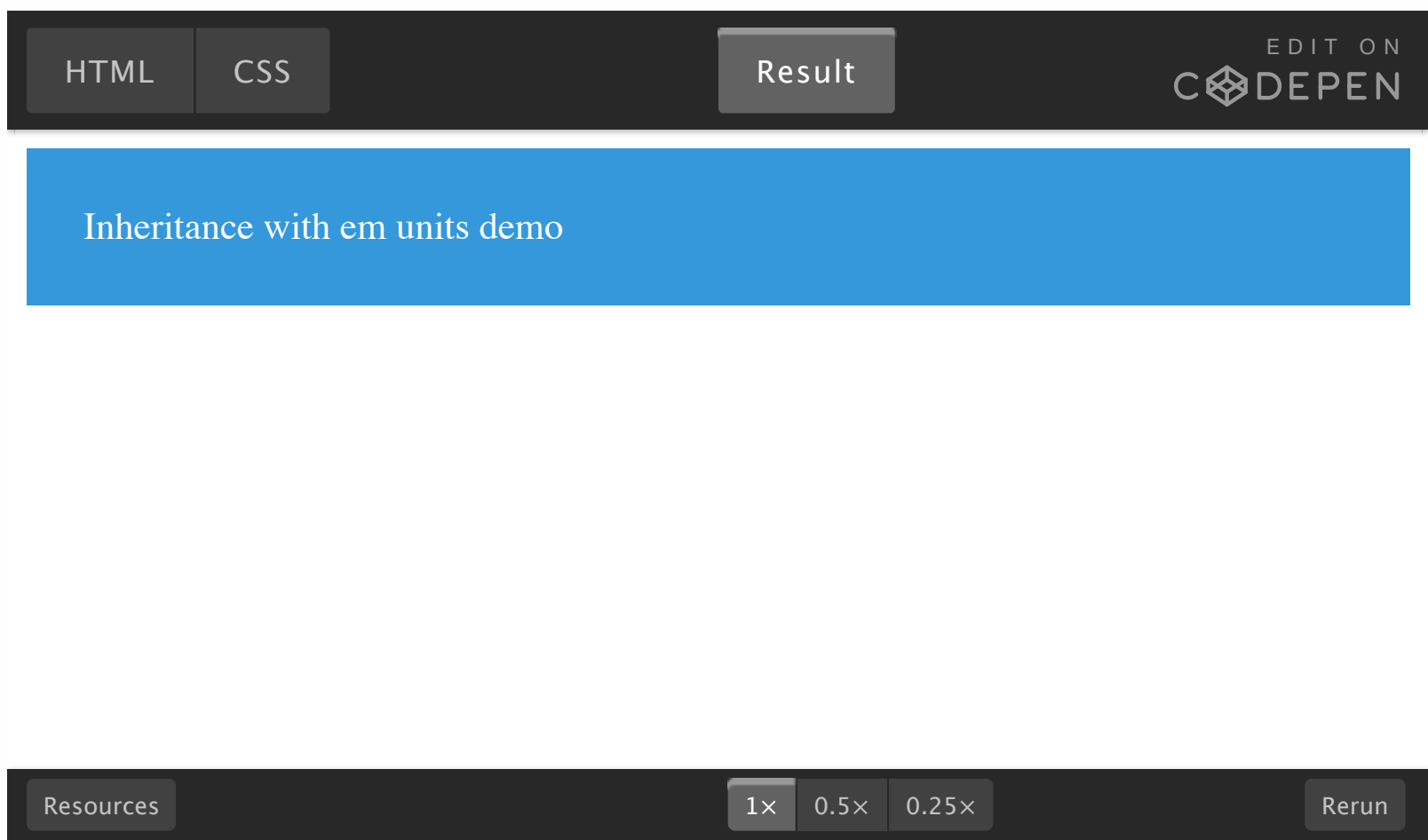
padding-top: 160px;

width: 1366.25px;

Computes to 160px

On the other hand `px` values are used by the browser as is, so `1px` will always display as exactly `1px`.

Try the slider out on this [CodePen example](#) to see how the value of `rem` and `em` units can translate into different pixel values, while explicitly set `px` units stay fixed in size:



The Big Question

Using `em` and `rem` units gives us flexibility in our designs, and the ability to scale elements up and down, instead of being stuck with fixed sizes. We can use this flexibility to make our designs easier to adjust during development, more responsive, and to allow browser users to control the overall scale of sites for maximum readability.

Both `em` and `rem` units provide this flexibility and work in similar ways, so the big question is, when should we use `em` values and when should we use `rem` values?

Crucial Difference

The difference between `em` and `rem` units is how the browser determines the `px` value they translate into. Understanding this difference is the key to determining when to use each unit.

We're going to begin by going over how `rem` and `em` units work from the ground up to make sure you know every detail. Then we'll move on to *why* you should use `em` or

rem units.

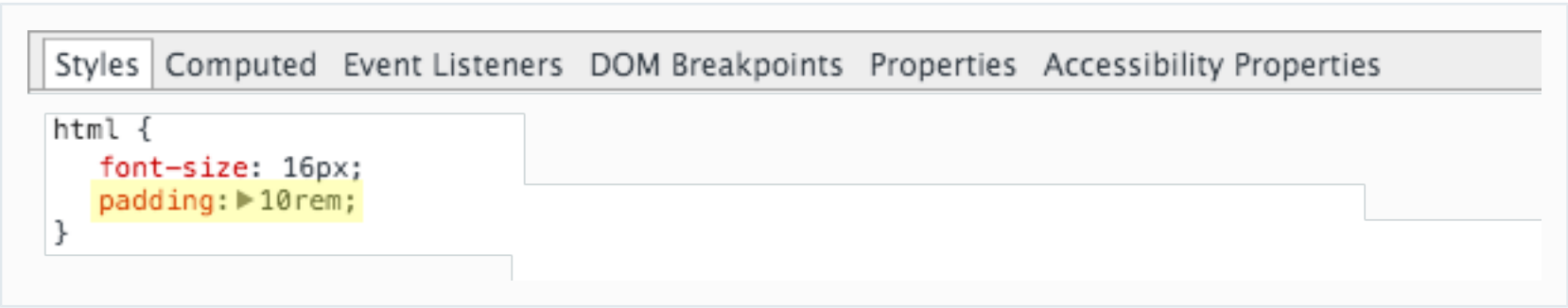
Finally we'll look at the practical application of exactly which elements of a typical design you should use each type of unit on.



How rem Units Translate to Pixel Values

When using rem units, the pixel size they translate to depends on the font size of the root element of the page, i.e. the html element. That root font size is multiplied by whatever number you're using with your rem unit.

For example, with a root element font size of 16px, 10rem would equate to 160px, i.e. 10 x 16 = 160.



CSS padding set to 10rem

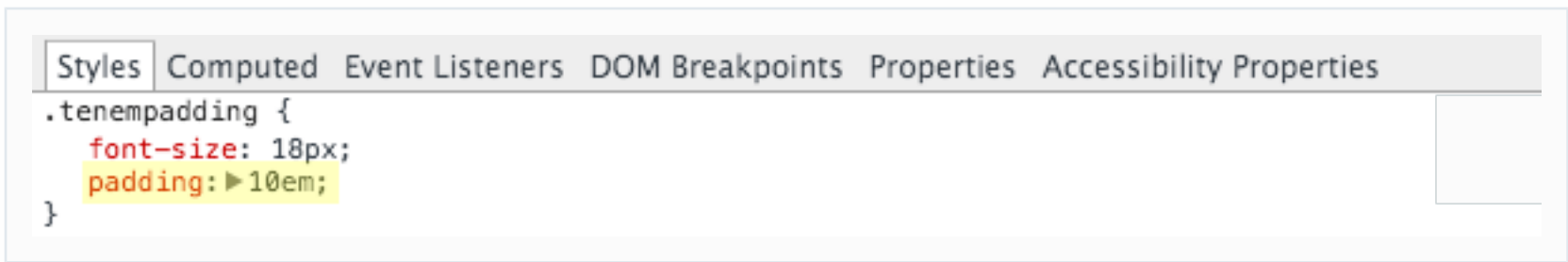


Computes to 160px

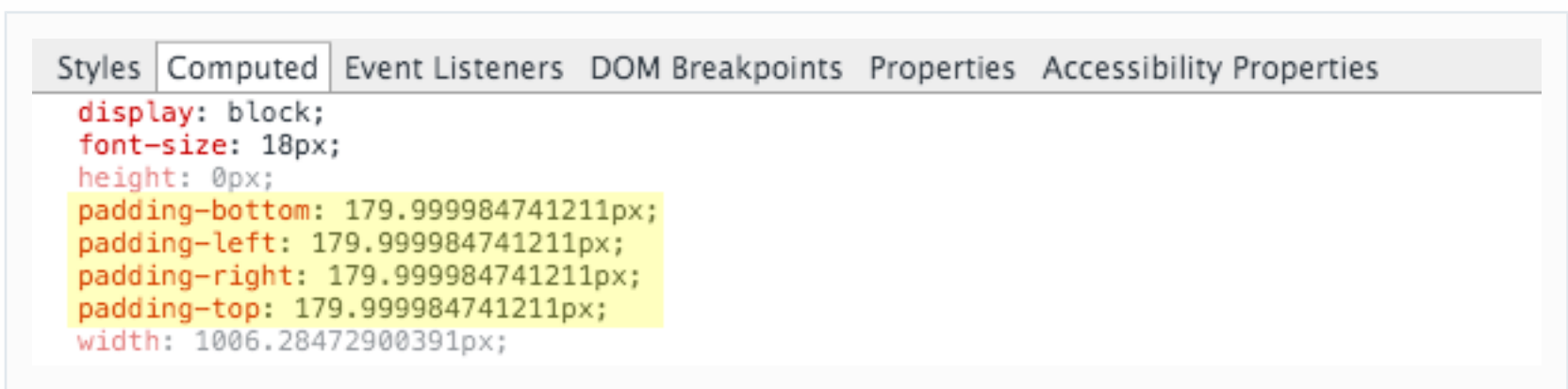
How em Units Translate to Pixel Values

When using em units, the pixel value you end up with is a multiplication of the font size on the element being styled.

For example, if a div has a font size of 18px, 10em would equate to 180px, i.e. $10 \times 18 = 180$.



CSS padding set to 10em



Computes to 180px (or close enough to it)

Important to Know:

It's a somewhat widespread misconception that em units are relative to the font size of the parent element. In fact, as per the W3 spec, they are relative to the font size “of the element on which they are used”.

Parent element font sizes *can* effect `em` values, but when that happens it's solely because of inheritance. Let's take a look at why, and see how this works in action.

The Effect of Inheritance on `em` Units

Working with `em` units can start to get tricky when it comes to inheritance, because every element automatically inherits the font size of its parent. The effect of inheritance can only be overridden by explicitly setting a font size with a unit not subject to inheritance, such as `px` or `vw`.

The font size of the element on which `em` units are used determines their size. But that element may have inherited a font size from its parent, which inherited a font size from its parent, and so on. As such it's possible for any `em` value to be effected by the font size of any of its parents.

Let's look at an example. Feel free to try this out in [CodePen](#) for yourself as we step through it. As you go along, use Chrome Developer Tools or Firebug for Firefox to inspect the pixel values our `em` units are computed into.

Example of `em` Inheritance

If we have a page with a root font size of `16px` (usually the default) and a child div inside it with padding of `1.5em`, that div will inherit the font size of `16px` from the root element. Hence its padding will translate to `24px`, i.e. $1.5 \times 16 = 24$.

HTML

CSS

Result

EDIT ON
CODEPEN

Inheritance with em units demo

Resources

1x

0.5x

0.25x

Rerun

Then what if we wrap another div around the first and set its `font-size` to `1.25em`?

HTML

CSS

Result

EDIT ON
CODEPEN

Inheritance with em units demo

Resources

1x

0.5x

0.25x

Rerun

Our wrapper div inherits the root font size of `16px` and multiplies that by its own `font-size` setting of `1.25em`. This sets the div to have a font size of `20px`, i.e. $1.25 \times 16 = 20$.

Now our original div is no longer inheriting directly from the root element, instead it's inheriting a font size of `20px` from its new parent div. Its padding value of `1.5em` now equates to `30px`, i.e. $1.5 \times 20 = 30$.

This scaling effect can be compounded even further if we add an `em` based font size to our original div, let's say `1.2em`.

HTML

CSS

Result

EDIT ON
CODEPEN

Inheritance with em units demo

Resources

1x

0.5x

0.25x

Rerun

The div inherits the `20px` font size from its parent, then multiplies that by its own `1.2em` setting, giving it a new font size of `24px`, i.e. $1.2 \times 20 = 24$.

The `1.5em` padding on our div will now change in size again with the new font size, this time to `36px`, i.e. $1.5 \times 24 = 36$.

Finally, to further illustrate that `em` units are relative to the font size of the element they're used on, (not the parent element), let's see what happens to our padding of `1.5em` if we explicitly set the div to use a font size of `14px`, a value not subject to inheritance.

HTML

CSS

Result

EDIT ON
CODEPEN

Heading 1

Heading 2

Resources

1x0.5x0.25x

Rerun

Now, our padding has dropped down to `21px`, i.e. $1.5 \times 14 = 21$. It is unaffected by the font size of the parent element.

With all this potential for complication, you can see why its important to know how to use `em` units in a manageable way.

The Effect of Browser Settings on the HTML Element Font Size

By default browsers usually have a font size of `16px`, but this can be changed to anywhere from `9px` to `72px` by the user.

Important to Know:

The root `html` element inherits its font size from the settings in the browser, unless overridden with an explicitly set fixed value.

So while the font size on the `html` element is what directly determines `rem` values, that font size may have first come from browser settings.

Thus browser font size settings can effect the value of every `rem` unit used in a design, as well as every `em` unit via inheritance.

Browser Setting Effect When No HTML Font Size is Set

Unless overridden, the `html` element it will inherit whatever the default font size setting in the browser is. For example, let's take a site where no `font-size` property is set on the `html` element.

If a user has their browser at the default font size of 16px, the root font size will be 16px. In Chrome Developer Tools you can see what an element has inherited by checking **Show inherited properties** under the **Computed** tab.

In this case `10rem` equates to `160px`, i.e. $10 \times 16 = 160$.

If the user bumps their browser font size up to 18px, the root font size becomes 18px. Now `10rem` translates to `180px`, i.e. $10 \times 18 = 180$.

Browser Setting Effect with `em` Unit HTML Font Size

When an `em` based font size is set on the `html` element, the pixel value it translates to will be a multiple of the browser font size setting.

For example, if the site's `html` element had a `font-size` property set to `1.25em`, the root font size would be 1.25 times the browser font size setting.

If the browser font size was set to `16px`, the root font size would come out as `20px`, i.e. $1.25 \times 16 = 20$.

In this case `10rem` would equal `200px`, i.e. $10 \times 20 = 200$.

However, if the browser font size was set to `20px`, the root font size would instead translate to `25px`, i.e. $1.25 \times 20 = 25$.

Now `10rem` would equal `250px`, i.e. $10 \times 25 = 250$.

Summarizing `em` vs. `rem` Difference

What all the above boils down to is this:

- Translation of `rem` units to pixel value is determined by the font size of the `html` element. This font size is influenced by inheritance from the browser font size setting unless explicitly overridden with a unit not subject to inheritance.
- Translation of `em` units to pixel values is determined by the font size of the element they're used on. This font size is influenced by inheritance from parent elements unless explicitly overridden with a unit not subject to inheritance.

Why Use `rem` Units:

The greatest power that `rem` units offer is not just that they give consistent sizing regardless of element inheritance. Rather, it's that they give us a way to have user font size settings influence every single aspect of a site's layout by using `rem` units where we used to use `px` units.

For this reason the primary purpose of using `rem` units should be to ensure that whatever default font size a user has their browser set to, the layout will adjust to accommodate the size of text within it.

A site can be designed initially focusing on the most common default browser font size of 16px.

Browser font size 16px

But by using `rem` units, if a user increases their font size, the integrity of the layout will be preserved, and the text won't get squashed up in a rigid space meant for smaller text.

Browser font size 34px

And if the user decreases their font size, the entire layout scales down, and they won't be left with a tiny smattering of text in a wasteland of white space.

Browser font size 9px

Users change font size settings for all kinds of reasons, from strained eyesight to choosing optimum settings for devices that can be vastly different in size and

viewing distance. Accommodating these settings allows for much better user experiences.

Important to Know:

Some designers use `rem` based layouts in conjunction with a fixed `px` unit `font-size` setting on the `html` element. This is typically done so that a change of font size on the `html` element can scale overall the design up or down.

I strongly advise against this as it overrides the font size the `html` element inherits from the user's browser settings. Hence this prevents the settings from having any effect, and removes the user's ability to optimize for best viewing.

If you do want to change the font size on the `html` element, do so with an `em` or `rem` value as the root font size will then still be a multiple of the user's browser font size setting.

This will still allow you to scale your design up or down by changing your `html` element's font size, but you'll preserve the effect of user browser settings.

Why Use `em` Units

The key value `em` units offer is they allow sizing values to be determined by a font size other than that of the `html` element.

For this reason, the primary purpose of `em` units should be to allow for scalability *within the context of a specific design element*.

For example, you might set the `padding`, `margin` and `line-height` around a navigation menu item to use `em` values.

This way if you change the menu's font size the spacing around the menu items will scale proportionately, independently of the rest of the layout.

Menu with 1.2rem font size

In the earlier section on inheritance you saw how quickly keeping track of `em` units can get out of hand. For this reason, I recommend *only* using `em` units if you identify a clear need for them.

Practical Application

There may be some debate among web designers and I'm sure different people have different preferred approaches, however my recommendation is as follows.

Use `em` Units For:

Any sizing that should *scale depending on the font-size of an element other than the root*.

Generally speaking, the only reason you'll need to use `em` units is to scale an element which has non default font sizing.

As per our example above, design components like menu items, buttons, and headings may have their own explicitly stated font sizes. If you change these font sizes, you want the entire component to scale proportionately.

Common properties this guideline will apply to are `margin`, `padding`, `width`, `height` and `line-height` settings, when used on elements with non default font sizing.

I recommend that when you do employ `em` units, the font size of the element they're used on should be set in `rem` units to preserve scalability but avoid inheritance confusion.

Typically Don't Use `em` Units for Font Sizes

It's quite common to see `em` units used for font sizing, particularly in headings, however I would suggest that designs are more manageable if `rem` units are typically used for font sizing.

The reason headings often use `em` units is they're a better choice than `px` units, being relative to regular text size. However `rem` units can achieve this goal equally well. If any font size adjustment on the `html` element is made, the heading sizes will still scale too.

Try changing the `em` font size on the `html` element in this [CodePen](#) to see for yourself:

More often than not, we don't want our font sizes to scale based on any element other than the root, with only a few exceptions.

One example where we might want `em` based font sizing could be a drop down menu, where we have second level menu item text sized depending on the font size

of the first level. Another example might be a font icon used inside a button, where the icon's size should relate to the button's text size.

However most elements in a web design will tend not to have this type of requirement, so you'll generally want to use `rem` units for font sizing, with `em` units only where specifically needed.

Use `rem` units for:

Any sizing that doesn't need `em` units for the reasons described above, and that should *scale depending on browser font size settings*.

This accounts for almost everything in a standard design including most heights, most widths, most padding, most margins, border widths, most font sizes, shadows, basically almost every part of your layout.

In a nutshell, everything that can be made scalable with `rem` units, should be.

Tip

When creating layouts it's often easier to think in pixels but output in `rem` units.

You can have pixel to `rem` calculations done automatically via a preprocessor like [Stylus](#) / [Sass](#) / [Less](#), or a postprocessor like [PostCSS](#) with the [PXtoRem](#) plugin.

Alternatively, you can use [PXtoEM](#) to manually do your conversions.

Always Use `rem` Media Queries

Importantly, when using `rem` units to create a uniformly scalable design, your media queries should also be in `rem` units. This will ensure that whatever a user's browser font size, your media queries will respond to it and adjust your layout.

For example, if a user scales up text very high, your layout may need to snap down from a two columns to a single column, just as it might on a smaller screened mobile device.

If your breakpoints are at fixed pixel widths, only a different viewport size can trigger them. However with `rem` based breakpoints they will respond to different font sizing too.

Don't Use `em` or `rem` For:

Multi Column Layout Widths

Column widths in a layout should typically be `%` based so they can fluidly fit unpredictably sized viewports.

However single columns should still generally incorporate `rem` values via a `max-width` setting.

For example:

```
css .container { width: 100%; max-width: 75rem; }
```

This keeps the column flexible and scalable, but prevents it from becoming too wide for the text therein to be comfortably readable.

When an Element Should be Strictly Unscalable

In a typical web design there won't be many parts of your layout that can't be designed for scalability. However occasionally you will come across elements that really do need to use explicit fixed values for the purpose of preventing scaling.

The precondition for employing fixed sizing values should be that if the element in question were scaled it would break. This really doesn't come up often, so before you're tempted to whip out those `px` units, ask yourself if using them is an absolute necessity.

Wrapping Up

Let's have a quick bullet point recap of what we've covered:

- `rem` and `em` units are computed into pixel values by the browser, based on font sizes in your design.
 - `em` units are based on the font size of the element they're used on.
 - `rem` units are based on the font size of the `html` element.
 - `em` units can be influenced by font size inheritance from any parent element
 - `rem` units can be influenced by font size inheritance from browser font settings.
-
- Use `em` units for sizing that should scale depending on the font size of an element other than the root.
 - Use `rem` units for sizing that doesn't need `em` units, and that should scale depending on browser font size settings.
 - Use `rem` units unless you're sure you need `em` units, including on font sizes.
 - Use `rem` units on media queries
 - Don't use `em` or `rem` in multi column layout widths - use `%` instead.
 - Don't use `em` or `rem` if scaling would unavoidably cause a layout element to break.

I hope you've now built a robust and complete picture of exactly how `em` and `rem` units work, and through that know how to best leverage them in your designs.

I encourage you to try the usage guidelines contained in this tutorial for yourself, and enjoy the fully fledged scalability and responsiveness of the layouts they'll enable you to create.

Advertisement

Kezz Bracey

Hi there. I'm a designer & coder who works in the areas of web design / development, game development and digital art. In the web space I'm a front end all rounder but I have a particular specialization in theme creation, no matter the platform. I also love finding the latest most efficient, user focused design and dev techniques of the day. In game development I'm addicted to playing with every different engine, toolset and framework I can find. In digital art I love everything from painting to vector work to pixel art to 3D modelling. In short, if it's creative and you can make it digitally, I love it.

 [kezzbracey](#)

 FEED

 LIKE

 FOLLOW

Weekly email summary

Subscribe below and we'll send you a weekly email summary of all new Web Design tutorials. Never miss out on learning about the next big thing.

Update me weekly



Bör du överväga att pensionera dig utomlands?

Om du har en förmögenhet på
mer än 3 miljoner kronor finns
bra tips om pensionering i den
här guiden och få regelbundna
uppdateringar.

LÄS MER HÄR!

FISHER INVESTMENTS NORDEN*

Advertisement

Translations

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

[Translate this post](#)

Powered by



FISHER INVESTMENTS NORDEN*

Vid vilken ålder kan du gå i pension?

45 – 54

55 – 64

65 – 74

75+

[Utforska Här >](#)

Beställ din guide och få regelbundna uppdateringar.

Advertisement

ENVATO TUTORIALS



JOIN OUR COMMUNITY



HELP



28,152

Tutorials

1,263

Courses

39,815

Translations

Facebook

Twitter

Pinterest

