What is a CSS Unit?

A CSS unit determines the size of a property you're setting for an element or its content. For example, if you wanted to set the property `margin` of a paragraph, you would give it a specific value. This value includes the CSS unit.

Let's look at a small example:

```
p {
  margin: 20px;
}
```

Copy

In this case, `margin` is the property, `20px;` is the value, and `px` (or "pixel") is the CSS unit.

Even though it's common to see units like `px` used, the big question is often, "What's the best unit to use here?"

Here are some considerations to make when picking a unit type, and example use cases:

Absolute vs. Relative Units

When considering all the options for which units to use, it's important to consider the two categories of units: absolute and relative.

Absolute Units

Units that are “absolute” are the same size regardless of the parent element or window size. This means a property set with a value that has an absolute unit will be that size when looked at on a phone or on a large monitor (and everything in between!)

Absolute units can be useful when working on a project where responsiveness is not being considered. For example, desktop apps that can't be resized can be styled for the default dimensions. If the window doesn't scale, you don't need the content to either.

Hint: Absolute units can be less favourable for responsive sites because they don't scale when the screen size changes.

Absolute Unit	Description	Example
px	1/96 of 1 inch (96px = 1 inch)	`font-size: 12px;`
pt	1/72 of 1 inch (72pt = 1 inch)	`font-size: 12pt;`
pc	12pt = 1pc	`font-size: 1.2pc;`
cm	centimeter	`font-size: 0.6cm;`

mm	millimeter (10 mm = 1 cm)	font-size: 4mm;
in	inches	font-size: 0.2in;

Pixels (`px`) are typically the most popular absolute unit for screens. Centimeters, millimeters, and inches are more common for print and you may not have even known they were options!

Relative Units

Relative units are useful for styling responsive sites because they scale relative to the parent or window size (depending on the unit).

As a general rule, relative units can be used as the default for responsive sites. This can help you avoid having to update styles for different screen sizes.

Relative units can be a little more difficult than absolute units in determining which to use, so let's go through your options in detail.

Relative Unit	Description
%	Relative to the parent element's value for that property
em	Relative to the current font-size of the element
rem	Relative to the font-size of the root (e.g. the `<html>` element). "rem" = "root em"
ch	Number of characters (1 character is equal to the width of the current font's 0/zero)
vh	Relative to the height of the viewport (window or app size). 1vh = 1/100 of the viewport's height
vw	Relative to the width of viewport. 1vw = 1/100 of the viewport's width.
vmin	Relative to viewport's smaller dimension (e.g. for portrait orientation, the width is smaller than the height so it's relative to the width). 1vmin = 1/100 of viewport's smaller dimension.
vmax	Relative to viewport's larger dimension (e.g. height for portrait orientation). 1vmax = 1/100 of viewport's larger dimension.

ex	Relative to height of the current font's lowercase "x".

It's not always clear which of these options is best to use for each type of CSS property. For example, `%` is usually more appropriate for layout-related properties like `width` than it would be for `font-size`.

Here are some examples of when you would use each relative unit.

%: You want a child element to have 10% of the parent's width as a margin so it never fills the whole parent element. If the parent's size changes, the margin will update too.

```
.child {
  margin: 10%;
}
```

em: You want the font of a child element to be half the size of its parent's font-size (e.g. the paragraph under a section's title).

```
.child {
  font-size: 0.5em;
}
```

rem: The font-size should be twice the size as the root element's font. This could be how you size your headers because they should all be the same size regardless of the parent container.

```
.header {
  font-size: 2rem;
}
```

ch: You have a mono-spaced font (the characters are always the same width) and you only have space for 10 characters.

```
.small-text {
```

```
  width: 10ch;
}
```

vh: Your landing page should always be the height of the viewport/window.

```
.wrapper {
  height: 100vh;
}
```

vw: You have a section with text that should be half as wide as the viewport/window.

```
.half-size {
  width: 50vw;
```

vmin: You have an image that should always be as wide as the viewport's smaller dimension. On a phone being held in portrait mode, the image will be as wide as the viewport's width.

```
.min-width {
  width: 100vmin;
}
```

vmax: You don't care if an image gets cut off because you want it to completely fill the larger dimension of the viewport. For example, if an image of a pattern is used as a background.

```
.max-width {
  width: 100vmax;
}
```

ex: You probably won't come across `ex` very often but it's generally a good measure of a font's mid-section. Let's say you want to a font's `line-height` to be double the height of the font's "x".

```
.double-x {
  line-height: 2ex;
```